OpenLink Virtuoso Universal Server Documentation

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

OpenLink Virtuoso Universal Server Documentation

OpenLink Software Documentation Team <virtuoso.docs@openlinksw.com >

<virtuoso.docs@openlinksw.com >

Preface

1. Conventions

A note on the typographical conventions that have been used in this text:

This is the normal font type used for ordinary text.

```
This monospaced font is used to describe program code.

This monospaced font is used to describe screen output,
differentiated from code if required.
```

This See Also:
is a tip used for cross-referencing material either within the documentation, or externally.

This Note:
is a note used for revealing a point of interest or special feature

Special formatting is used to highlight `functions()` and their *`parameters`* above the rest of the text.

Important keywords are *emphasized* using bolding (or strong character typing) and text that is

Important
very important and must be noticed

is displayed in a traditional red warning color

Chapter 1. Overview

Abstract

A quick overview on Virtuoso providing answers to simple questions that may already be in mind.

1.1. What is Virtuoso?

OpenLink Virtuoso is the first CROSS PLATFORM Universal Server to implement Web, File, and Database server functionality alongside Native XML Storage, and Universal Data Access Middleware, as a single server solution. It includes support for key Internet, Web, and Data Access standards such as: XML, XPATH, XSLT, SOAP, WSDL, UDDI, WebDAV, SMTP, SQL-92, ODBC, JDBC, and OLE-DB. Virtuoso currently supports the following Operating systems - Windows 95/98/NT/2000, Linux (Intel, Alpha, Mips, PPC), Solaris, AIX, HP-UX, Unixware, IRIX, Digital UNIX, DYNIX/PTX, FreeBSD, SCO, MacOS X.

Virtuoso is a revolutionary, next generation, high-performance virtual database engine for the Distributed Computing Age. It is a core universal data access technology set to accelerate our advances into the emerging Information Age.

Virtuoso provides transparent access to your existing data sources, which are typically databases from different database vendors.

Through a single connection, Virtuoso will simultaneously connect your ODBC, JDBC, UDBC, OLE-DB client applications and services to data within Oracle, Microsoft SQL Server, DB/2, Informix, Progress, CA-Ingres and other ODBC compliant database engines. All your databases are treated as single logical unit.

The diagram below depicts how applications that are built in conformance with industry standards (such as ODBC, JDBC, UDBC, and OLE-DB) only need to make a single connection via Virtuoso's Virtual Database Engine and end up with concurrent and real-time access to data within different database types.

Further still, Virtuoso exposes all of its functionality to Web Services. This means that your existing infrastructure can be used support Web Services directly without any hint of replacement.

1.2. Why Do I Need Virtuoso?

You need Virtuoso because Knowledge is power or competitive advantage (depending on how you choose to exploit it). All Knowledge comes from Information. Information is produced from Data.

The Internet is reducing the cost of accessing Information, thereby increasing the appetite and rates at which Information is produced and consumed. Unfortunately data required for the production of Information simply does not reside within one database engine within your organization.

Whether you know it or not it is highly probable that the quest for critical information within your organization actually requires traversing several data sources served by numerous database engines from different database vendors.

Virtuoso simply reduces the cost of bringing together data from different data sources with the view to accelerating the production of information by your Query Tools, Web & Internet Application Development Environments, Traditional Application Development Tools, and Desktop Productivity Tools.

Virtuoso enables you to compete effectively in the Information Age.

One of the biggest challenges facing the uptake of XML is the availability of key XML Data itself. Virtuoso simplifies the process of creating XML data from existing HTML, syndicated XML, and SQL databases. Virtuoso enables real-time creation of Dynamic XML documents (DTD or XML Schema based) from homogeneous or heterogeneous SQL Databases "on the fly".

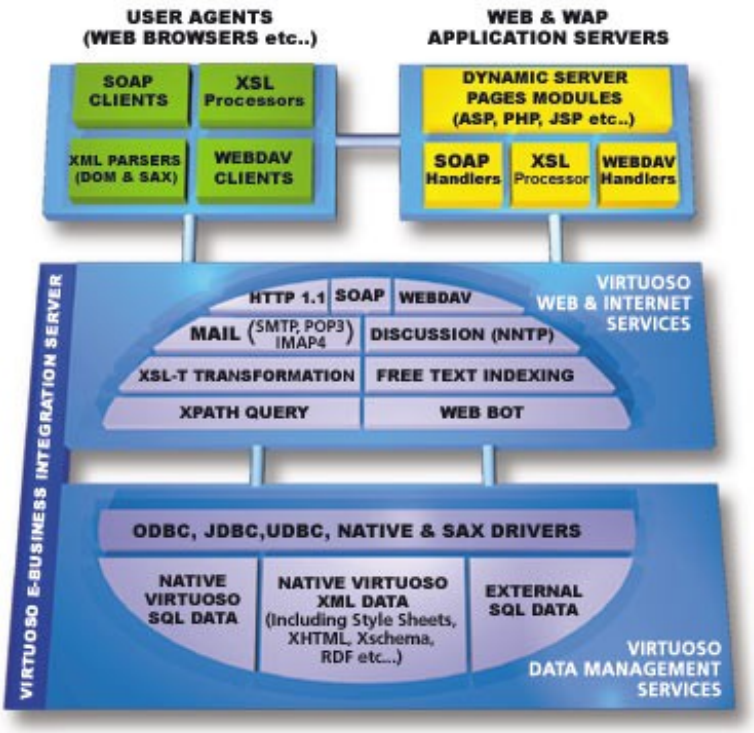
By implementing a number of protocols in a single server solution, Virtuoso provides you with a unifying foundation upon which next generation eBusiness solutions can be developed and deployed. Virtuoso reduces the cost of bringing together data from different data sources and leverages this into increased effectiveness of your Query Tools, Web & Internet Application Development Environments, Traditional Application Development Tools, and Desktop Productivity Tools. Virtuoso enables you compete effectively in the Information Age.

OpenLink Software is an acclaimed technology innovator and leading vendor of High-Performance & CROSS PLATFORM eBusiness solutions that adhere to a broad range of industry standards that include: ODBC, JDBC, OLE DB, SQL, WebDAV, HTTP, XML, SOAP, UDDI, WSDL, SMTP, NNTP, POP3, LDAP amongst others.

Our product & services portfolio includes a suite of High-Performance Universal Data Access Drivers for ODBC, OLE DB and JDBC, Internet Data Integration Servers, Virtual and Federated Database Engines, Embeddable SQL-Database Engines, Application Servers, Enterprise Portal Servers and professional services expertise capable of handling the most demanding eBusiness application development, deployment, and integration challenges.

1.3. Key Features of Virtuoso

Figure 1.1. OpenLink Virtuoso Product Architecture



1.3.1. XML Document Storage & Creation

Virtuoso enables you to develop eBusiness solutions that use *XML* as a common data access foundation layer that provides transparent access to structured and unstructured data. XML Data documents can be created internally, or imported from around the Web and then stored in Virtuoso. You can also create dynamic XML documents by transforming SQL to XML on the fly, leveraging data that resides within homogeneous and/or heterogeneous database(s). *XPATH 2.0* query language support enables you to query entire XML Documents using and industry standard query language. The Virtuoso Server provides some basic support for the *XQuery 1.0* XML Query Language specification. There is *XML Schema* support for extending Virtuoso Data types used by SOAP Services.

1.3.2. Web Page Hosting

Virtuoso has an integrated HTTP web server, for static HTML pages, or dynamic content using Virtuoso Server Pages (*VSP*) . Hosting and execution of *PHP4* scripts is supported via Virtuoso Server Extensions Interface (VSEI) for Zend.

1.3.3. Web Services Creation & Hosting

Enables the creation of *SOAP* compliant Web Services from SQL Stored Procedures, these procedures may be native to Virtuoso or resident in third party databases that support ODBC or JDBC. Virtuoso automatically generates *WSDL* files for the Stored Procedures that it exposes as Web Services. As a *UDDI* server (registry) all of your Web Services can be stored for access across the internet or within an intranet. It can also synchronize data with other UDDI servers.

1.3.4. WebDAV Compliant Web Store

WebDAV support enables Virtuoso to act as the Web Content Store for all of your eBusiness data, this includes Text, Graphics and Multimedia files. WebDAV support also enables Virtuoso to play the familiar roles of a FILE & WEB SERVER, hosting entire Web sites within a single database file, or across multiple database files.

1.3.5. Content Replication & Synchronization

Virtuoso's sophisticated data replication and synchronization engine enables the automated distribution and updating of SQL and Web Content across distributed Virtuoso servers.

1.3.6. Transparent Access To Heterogeneous Data

Virtuoso's Virtual Database Engine enables you to produce Dynamic Web Content from any major database management system. This enables dynamic, real-time HTML and XML generation from any number of different database engines concurrently.

1.3.7. Mail Delivery & Retrieval Services

Virtuoso can act as an *SMTP* , *POP3* , and *IMAP4* proxy to any email client. This enables the development and deployment of sophisticated database driven email solutions.

1.3.8. NNTP Aggregation & Serving

Virtuoso supports the Network News Transfer Protocol used by Internet newsgroup forums. *NNTP* servers manage the global network of collected newsgroup postings and represent a vast repository of targeted information archives. As an NNTP aggregator, Virtuoso enables integration of multiple news forums around the world. All news content in Virtuoso is dynamically indexed to provide keyword searches, enabling rapid transformation of disparate text data into information. Virtuoso also acts as an NNTP server, enabling creation of new Internet and Intranet News Forums to leverage the global knowledgebase into eBusiness Intelligence.

1.4. Virtuoso FAQ

We have received various inquiries on high-end metadata stores. We will here go through some salient questions. The requested features include:

- Scaling to trillions of triples
- Running on clusters of commodity servers
- Running in federated environments, possibly over wide-area networks
- Built-in inference
- Transactions
- Security
- Support for extra triple level metadata, such as security attributes

Questions:

1.4.1. What is the storage cost per triple?

This depends on the index scheme. If indexed 2 ways, assuming that the graph will always be stated in queries, this is 31 bytes.

With 4 indices, supporting queries where the graph can be left unspecified (i.e., triples from any graph will be considered in query evaluation), this is 39 bytes. The numbers are measured with the LUBM validation data set of 121K triples, with no full-text index on literals.

With 4 indices and a full text index on all literals, the Billion Triples Challenge data set, 1115M triples, is about 120 GB of database pages. The database file size is larger due to space in reserve and other factors. 120 GB is the number to use when assessing RAM-to-disk ratio, i.e., how much RAM the system ought to have in order to provide good response. This data set is a heterogeneous collection including social network data, conversations harvested from the Web, DBpedia, Freebase, etc., with relatively numerous and long text literals.

The numbers do not involve any database page stream compression such as gzip. Using such compression does not save in terms of RAM because cached pages must be kept uncompressed but will cut the disk usage to about half.

1.4.2. What is the cost to insert a triple (for the insertion itself, as well as for updating any indices)?

The more triples are inserted at a time, the faster this goes. Also, the more concurrent triple insertions are going on, the better the throughput. When loading data such as the US Census, a cluster of 2 commodity servers can insert up to 100,000 triples per second.

A single 4-core machine can load 1 billion triples of LUBM data at an average rate of 36K triples per second. This is limited by disk.

1.4.3. What is the cost to delete a triple (for the deletion itself, as well as for updating any indices)?

The delete cost is similar to insert cost.

1.4.4. What is the cost to search on a given property?

If we are looking for equality matches, a single 2GHz core can do about 250,000 single triple random lookups per second as long as disk reads are not involved. If each triple requires a disk seek the number is naturally lower.

Parallelism depends on the query. With a query like counting all x and y such that x knows y and y knows x, we get up to 3.4 million single-triple lookups-per-second on a cluster of 2 8-core Xeon servers. With complex nested sub-queries the parallelism may be less.

Lookups involving ranges of values, such as ranges of geographical coordinates or dates use an index, since quads are indexed in a manner that collates in the natural order of the data type.

1.4.5. What data types are supported?

Virtuoso supports all RDF data types, including language-tagged and XML schema typed strings as native data types. Thus there is no overhead converting between RDF data types and types supported by the underlying DBMS.

1.4.6. What inferencing is supported?

Subclass, subproperty, identity by inverse-functional properties, and owl:sameAs are processed at run time if an inference context option is specified in the query.

There is a general-purpose transitivity feature that can be used for a wide variety of graph algorithms. For example:

```
SELECT ?friend
WHERE
  {
    <alice> foaf:knows ?friend option (transitive)
```

```
}
```

would return all the people directly or indirectly known by <alice>.

1.4.7. Is the inferencing dynamic, or is an extra step required before inferencing can be used?

The mentioned types of inferencing are enabled by a switch in the query and are done at run-time, with no step for materialization of entailed triples needed. The pattern:

```
{?s a <type>}
```

would iterate over all the RDFS subclasses of <type> and look for subjects with this type.

The pattern:

```
{<thing> a ?class}
```

will, if the match of ?class has superclasses, also return the superclasses even though the superclass membership is not physically stored for each superclass.

Of course, one can always materialize entailed triples by running SPARQL/SPARUL statements to explicitly add any implied information.

If two subjects have the same inverse functional property with the same value, they will be considered the same. For example, if two people have the same email address, they will be considered the same.

If two subjects are declared to be owl:sameAs, either directly or through a chain of x owl:sameAs y, y owl:sameAs z, and so on, they will be considered the same.

These features can be individually enabled and disabled. They all have some run time cost, hence they are optional. The advantage is that no preprocessing of the data itself is needed before querying, and the data does not get bigger. This is important, especially if the database is very large and queries touch only small parts of it. In such cases, materializing implied triples can be very costly. See discussion at E Pluribus Unum

1.4.8. Do you support full-text search?

Virtuoso has an optional full-text index on RDF literals. Searching for text matches using the SPARQL regex feature is very inefficient in the best of cases. This is why Virtuoso offers a special *bif:contains* predicate similar to the SQL *contains* predicate of many relational databases. This supports a full-text query language with proximity, and/or/and-not, wildcards, etc.

While the full-text index is a general-purpose SQL feature in Virtuoso, there is extra RDF-specific intelligence built into it. One can, for example, specify which properties are indexed, and within which graphs this applies.

1.4.9. What programming interfaces are supported? Do you support standard SPARQL protocol?

Virtuoso supports the standard SPARQL protocol.

Virtuoso offers drivers for the Jena, Sesame, and Redland frameworks. These allow using Virtuoso's store and SPARQL implementation as the back end of Jena, Sesame, or Redland applications. Virtuoso will then do the query optimization and execution. Jena and Sesame drivers come standard; contact us about Redland.

Virtuoso SPARQL can be used through any SQL call level interface (CLI) supported by Virtuoso (i.e., ODBC, JDBC, OLE-DB, ADO.NET, XMLA). All have suitable extensions for RDF specific data types such as IRIs and typed literals. In this way, one can write, for example, PHP web pages with SPARQL queries embedded, just using the SQL tools. Prefixing a SQL query with the keyword "sparql" will invoke SPARQL instead of SQL, through any SQL client API.

1.4.10. How can data be partitioned across multiple servers?

Virtuoso Cluster partitions each index of all tables containing RDF data separately. The partitioning is by hash. The result is that the data is evenly distributed over the selected number of servers. Immediately consecutive triples are generally in the same partition, since the low bits of IDs do not enter in into the partition hash. This means that key compression works well.

Since RDF tables are in the end just SQL tables, SQL can be used for specifying a non-standard partitioning scheme. For example, one could dedicate one set of servers for one index, and another set for another index. Special cases might justify doing this.

With very large deployments, using a degree of application-specific data structures may be advisable. See "Does Virtuoso support property tables".

1.4.11. How many triples can a single server handle?

With free-form data and text indexing enabled, 500M triples per 16G RAM can be a ballpark guideline. If the triples are very short and repetitive, like the LUBM test data, then 16G per one billion triples is a possibility. Much depends on the expected query load. If queries are simple lookups, then less memory per billion triples is needed. If queries will be complex (analytics, join sequences, and aggregations all over the data set), then relatively more RAM is necessary for good performance.

The count of quads has little impact on performance as long as the working set fits in memory. If the working set is in memory, there may be 15-20% difference between a million and a billion triples. If the database must frequently go to disk, this degrades performance since one can easily do 2000 random accesses in memory in the time it takes to do one random access from disk. But working-set characteristics depend entirely on the application.

Whether the quads in a store all belong to one graph or any number of graphs makes no difference. There are Virtuoso instances in regular online use with hundreds of millions of triples, such as DBpedia and the Neurocommons databases.

1.4.12. What is the performance impact of going from the billion to the trillion triples?

Performance dynamics change when going from a single server to a cluster. If each partition is around a billion triples in size, then the single triple lookup takes the same time, but there is cluster interconnect latency added to the mix.

On the other hand, queries that touch multiple partitions or multiple triples in a partition will do this in parallel and usually with a single message per partition. Thus throughput is higher.

In general terms, operations on a single triple at a time from a single thread are penalized and operations on hundreds or more triples at a time win. Multiuser throughput is generally better due to more cores and more memory, and latency is absorbed by having large numbers of concurrent requests.

See a sample of SPARQL scalability .

1.4.13. Do you support additional metadata for triples, such as time-stamps, security tags etc?

Since quads (triple plus graph) are stored in a regular SQL table with special data types, changing the table layout to add a column is possible. This column would not however be visible to SPARQL without some extra tuning. For coarse grain provenance and security information, we recommend doing this at the graph level, where triples that belong together are tagged with the same provenance or security are in the same graph. The graph can then have the relevant metadata as its properties.

If tagging at the single triple level is needed, this will most often not be needed for all triples. Hence altering the table for all triples may not be the best choice. Making a special table that has the graph, subject, predicate and object of the tagged triple as a key and the tag data as a dependent part may be more efficient. Also, this table could be more easily accessed from SPARQL.

Using the RDF reification vocabulary is not recommended as a first choice but is possible without any alterations.

Alterations of this nature are possible but we recommend contacting us for specifics. We can provide consultancy on the best way to do this for each application. Altering the storage layout without some extra support from us is not recommended.

1.4.14. Should we use RDF for our large metadata store? What are the alternatives?

If the application has high heterogeneity of schema and frequent need for adaptation, then RDF is recommended. The alternative is making a relational database.

Making a custom non-RDF object-attribute-value representation on Virtuoso or some other RDBMS is possible but not recommended.

The reason for this is that this would miss many of the optimizations made specifically for RDF, use of the SPARQL language, inference, compatibility with diverse browsers and front end tools, etc. Not to mention interoperability and joinability with the body of linked data. Even if the application is strictly private, using entity names and ontologies from the open world can still have advantages.

If some customization to the quad (triple plus graph layout) is needed, we can provide consultancy on how to do this while staying within the general RDF framework and retaining all the interoperability benefits.

1.4.15. How multithreaded is Virtuoso?

All server and client components are multithreaded, using pthreads on Unix/Linux, Windows native on Windows. Multithread/multicore scalability is good; see BSBM

In the case of Virtuoso Cluster, in order to have the maximum number of threads on a single query, we recommend that each server on the cluster be running one Virtuoso process per 1.2 cores.

1.4.16. Can multiple servers run off a single shared disk database?

This might be possible with some customization but this is not our preferred way. Instead, we can store selected indices in duplicate or more copies inside a clustered database. In this way, all servers can have their own disk. Each key of each index will belong to one partition but each partition will have more than one physical copy, each on a different server. The cluster query logic will perform the load balancing. On the update side, the cluster will automatically do a distributed transaction with two phase commit to keep the duplicates in sync.

1.4.17. Can Virtuoso run on a SAN?

Yes. Unlike Oracle RAC, for example, Virtuoso Cluster does not require a SAN. Each server has its own database files and is solely responsible for these. In this way, having shared disk among all servers is not required. Running on a SAN may still be desirable for administration reasons. If using a SAN, the connection to the SAN should be high performance, such as Infiniband.

1.4.18. How does Virtuoso join across partitions?

Partitioning is entirely transparent to the application. Virtuoso has a highly optimized message-flow between cluster nodes that combines operations into large batches and evaluates conditions close to the data. See a sample of RDF scaling.

1.4.19. Does Virtuoso support federated triple stores? If there are multiple SPARQL end points, can Virtuoso be used to do queries joining between these?

This is a planned extension. The logic for optimizing message flow between multiple end-points on a wide-area network is similar to the logic for message-optimization on a cluster. This will allow submitting a query with a list of end-points. The query will then consider triples from each of the end points, as if the content of all the end points were in a single store.

End-point meta information, such as voiD descriptions of the graphs in the end-points, may be used to avoid sending queries to end points that are known not to have a certain type of data.

1.4.20. How many servers can a cluster contain?

There is no fixed limit. If you have a large cluster installed, you can try Virtuoso there. Having an even point-to-point latency is desirable.

1.4.21. How do I reconfigure a cluster, adding and removing machines, etc?

We are working on a system whereby servers can be added and removed from a cluster during operation and no repartitioning of the data is needed.

In the first release, the number of server processes that make up the cluster is set when creating the database. These processes with their database files can then be moved between machines but this requires stopping the cluster and updating configuration files.

1.4.22. How will Virtuoso handle regional clusters?

Performance of a cluster depends on the latency and bandwidth of the interconnect. At least dual 1Gbit ethernet is recommended for each node. Thus a cluster should be on a single local or system area network.

If regional copies are needed, we would replicate between clusters by asynchronous log shipping. This requires some custom engineering.

When a transaction is committed at one site, it is logged and sent to the subscribing sites if they are online. If there is no connection, the subscribing sites will get the data from the log. This scheme now works between single Virtuoso servers, and needs some custom development to be adapted to clusters.

If replicating all the data of one site to another site is not possible, then application logic should be involved. Also, if consolidated queries should be made against large, geographically-separated clusters, then it is best to query them separately and merge the results in the application. All depends on the application level rules on where data resides.

1.4.23. Is there a mechanism for terminating long running queries?

Virtuoso SPARQL and SQL offer an "anytime" option that will return partial results after a configurable timeout.

In this way, queries will return in a predictable time and indicate whether the results are complete or not, as well as give a summary of resource utilization.

This is especially useful for publishing a SPARQL endpoint where a single long running query could impact the performance of the whole system. This timeout significantly reduces the risk of denial of service.

This is also more user-friendly than simply timing-out a query after a set period and returning an error. With the anytime option, the user gets a feel for what data may exist, including whether any data exists at all. This feature works with arbitrarily complex queries, including aggregation, GROUP BY, ORDER BY, transitivity, etc.

Since the Virtuoso SPARQL endpoint supports open authentication (OAuth), the authentication can be used for setting timeouts, so as to give different service to different users.

It is also possible to set a timeout that will simply abort a query or an update transaction if it fails to terminate in a set time.

Disconnecting the client from the server will also terminate any processing on behalf of that client, regardless of timeout settings.

The SQL client call-level interfaces (ODBC, JDBC, OLE-DB, ADO.NET, XMLA) each support a cancel call that can terminate a long running query from the application, without needing to disconnect.

1.4.24. Can the user be asynchronously notified when a long running query terminates?

There is no off-the-shelf API for this but making an adaptation of the SPARQL endpoint that could proceed after the client disconnected and, for example, could send results by email is trivial. Since SOAP and REST Web services can be programmed directly in Virtuoso's stored procedure language, implementing and exposing this type of application logic is easy.

1.4.25. How many concurrent queries can Virtuoso handle?

There is no set limit. As with any DBMS, response times get longer if there is severe congestion.

For example, having 2 or 3 concurrent queries per core is a good performance point which will keep all parts of the system busy. Having more than this is possible but will not increase overall throughput.

With a cluster, each server has both HTTP and SQL listeners, so clients can be evenly spread across all nodes. In a heavy traffic Web application, it is best to have a load balancer in front of the HTTP endpoints to divide the connections among the servers and to keep some cap on the number of concurrently running requests, enforcing a maximum request-rate per client IP address, etc.

1.4.26. What is the relative performance of SPARQL queries vs native relational queries?

This is application dependent. In Virtuoso, SPARQL and SQL share the same query execution engine, query optimizer, and cost model. If data is highly regular (i.e., a good fit for relational representation), and if queries typically access most of the row, then SQL will be more efficient. If queries are unpredictable, data is ragged, schema changes frequent, or inference is needed, then RDF will do relatively better.

The recent Berlin SPARQL Benchmark shows some figures comparing Virtuoso SQL and SPARQL and SPARQL in front of relational representation. However, the test workload is heavily biased in favor of relational. See also BSBM: MySQL vs Virtuoso.

With the TPC-H workload, relationally stored data, and SPARQL mapped to SQL, we find that with about half the queries there is no significant cost to SPARQL. With some queries there is additional overhead because the mapping does not produce a SQL query identical to that specified in the benchmark.

1.4.27. Does Virtuoso Support Property Tables?

For large applications, we would recommend RDF whenever there is significant variability of schema, but would still use an application-specific, relational style representation for those parts of the data that are regular in format. This is possible without loss of flexibility for the variable-schema part. However, this will introduce relational-style restrictions on the regular data; for example, a person could only have a single date-of-birth by design. In many cases, such restrictions are quite acceptable. Querying will still take place in SPARQL, and the representation will be transparent.

A relational table where the primary key is the RDF subject and where columns represent single-valued properties is usually called a property table. These can be defined in a manner similar to defining RDF mappings of relational tables.

1.4.28. What performance metrics does Virtuoso offer?

There is an extensive array of performance metrics. This includes:

- Cluster status summary with thread counts, CPU utilization, interconnect traffic, clean and dirty cache pages, virtual memory swapping warning, etc. This is either a cluster total or a total with breakdown per cluster node.
- Disk access, lock contention, general concurrency, and access count per index
- Statistics on memory usage for disk caching index-by-index, cache replacement statistics, disk random and sequential read times
- Count of random, sequential index access, disk access, lock contention, cluster interconnect traffic per query/client
- Detailed query-execution plans are available through the `explain` function

1.4.29. What support do you provide for concurrent/multithreaded operation? Is your interface thread-safe?

All client interfaces and server-side processes are multithreaded. As usual, each thread of an application should use a different connection to the database.

1.4.30. What level of ACID properties is supported?

Virtuoso supports all 4 isolation levels from dirty read to serializable, for both relational and RDF data.

The recommended default isolation is read-committed, which offers a clean historical read of data that has uncommitted updates. This mode is similar to the Oracle default isolation and guarantees that no uncommitted data is seen, and that no read will block waiting for a lock held by another client.

There is transaction logging and roll forward recovery, with two phase commit used in Virtuoso Cluster if an update transaction modifies more than one server.

For RDF workloads which typically are not transactional and have large bulk loads, we recommend running in a "row autocommit" mode without transaction logging. This virtually eliminates log contention but still guarantees consistent results of multithreaded bulk loads.

Setting this up requires some consultancy and custom development but is well worthwhile for large projects.

1.4.31. Do you provide the ability to atomically add a set of triples, where either all are added or none are added?

Yes. Doing this with millions of triples per transaction may run out of rollback space. Also, there is risk of deadlock if multiple such inserts run at the same time. For good concurrency, the inserts should be of moderate size. As usual, deadlocks are resolved by aborting one of the conflicting transactions.

1.4.32. Do you provide the ability to add a set of triples, respecting the isolation property (so concurrent accessors either see none of the triple values, or all of them)?

Yes. The reading client should specify serializable isolation and the inserting client should perform the insert as a transaction, no row autocommit mode.

1.4.33. What is the time to start a database, create/open a graph?

Starting a Virtuoso server process takes a few seconds. Making a new graph takes no time beyond the time to insert the triples into it. Once the server process(es) are running, all the data is online.

With high-traffic applications, reaching cruising speed may sometimes take a long time, specially if the load is random-access intensive. Filling gigabytes of RAM with cached disk pages takes a long time if done a page at a time. To alleviate this, Virtuoso pre-reads 2MB-sized extents instead of single pages if there is repeated access to the same extent within a short time. Thus cache warm-up times are shortened.

1.4.34. What sort of security features are built into Virtuoso?

For SQL, we have the standard role-based security and an Oracle-style row-level security (policy) feature.

For SPARQL, users may have read or update roles at the level of the quad store.

With RDF, a graph may be owned by a user. The user may specify read and write privileges on the graph. These are then enforced for SPARUL (the SPARQL update language) and SPARQL.

When an RDF graph is based on relationally stored data in Virtuoso or another RDBMS through Virtuoso's SQL federation feature (i.e., if the graph is an Linked Data View of underlying SQL data), then all relational security controls apply.

Further, due to the dual-nature of Virtuoso, sophisticated ontology-based security models are feasible. Such models are not currently used by default, but they are achievable with our consultancy.

1.5. Tips and Tricks

1.5.1. How Can I convert triples with geo properties to geometries to use spartial query?

Assuming a Named Graph with the following triples:

```
...
<http://linkedgeodata.org/triplify/node454640663> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http
<http://linkedgeodata.org/triplify/node454640663> <http://www.georss.org/georss/point> "53.2752338 -9.044
<http://linkedgeodata.org/triplify/node454640663> <http://www.w3.org/2003/01/geo/wgs84_pos#long> "-9.0443
<http://linkedgeodata.org/triplify/node454640663> <http://www.w3.org/2003/01/geo/wgs84_pos#lat> "53.27523

<http://linkedgeodata.org/triplify/node280886720> <http://www.w3.org/2000/01/rdf-schema#label> "Abbey Hou
<http://linkedgeodata.org/triplify/node280886720> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http
<http://linkedgeodata.org/triplify/node280886720> <http://www.georss.org/georss/point> "53.2874983 -9.070
<http://linkedgeodata.org/triplify/node280886720> <http://www.w3.org/2003/01/geo/wgs84_pos#long> "-9.0702
<http://linkedgeodata.org/triplify/node280886720> <http://www.w3.org/2003/01/geo/wgs84_pos#lat> "53.28749
```

```
...
```

In order to convert these triples (with geo properties) to geometries to use spartial query, you need to run the `DB.DBA.RDF_GEO_FILL ()` function to populate the special RDF_GEO table with the point information as detailed at Creating Geometries From RDF Data documentation section.

Then the geo-spatail information will be available and can be queried.

Note: RDF_GEO and associated indexes are created when the database is first created and thus just needs to be populated with the geometry data in the RDF triples, which is what the `DB.DBA.RDF_GEO_FILL ()` function does.

See Also:

- Programmatic Manipulation of Geometries in RDF
- GEO Spatial Examples
- GEO Spatial Tutorials:
  - ROUND
  - DESCRIBE
  - CONSTRUCT
  - ASK
  - UNION
  - COUNT
  - FILTER
  - Find Distance Variant I
  - Find Distance Variant II
  - Querying Time and Space Variant I
  - Querying Time and Space Variant II

1.5.2. How Can I execute SPARQL queries containing '$' character using ISQL?

Assuming a SPARQL query should filter on the length of labels:

```
SELECT ?label
FROM <http://mygraph.com>
WHERE
  {
    ?s ?p ?label
    FILTER(regex(str(?label), "^.{1,256}$") )
  }
```

ISQL uses '*$* ' character as a prefix for macro names of its preprocessor. When '*$* ' character is used in SPARQL query to be executed in ISQL, the character should be replaced with '*$$* ' notation or an escape char + numeric code:

```
SQL> SPARQL
SELECT ?label
FROM <http://mygraph.com>
WHERE
  {
    ?s ?p ?label
    FILTER(REGEX(str(?label), "^.{1,256}$$") )
  }
```

Note also that the FILTER written in this way, finds *?label-s* with length less than 256.

To achieve fast results, *REGEX* should be replaced with the *bif:length* function:

```
SQL> SPARQL
SELECT ?label
FROM <http://mygraph.com>
WHERE
  {
```

```
    ?s ?p ?label
    FILTER (bif:length(str(?label))<= 256)
  }
```

In this way the SPARQL query execution can work much faster if the interoperability is not required and *?label-s* are numerous.

1.5.3. How can I find on which table deadlocks occur?

One possible way to find on which table deadlocks occur is to execute the following statement:

```
SELECT TOP 10 *
FROM SYS_L_STAT
ORDER BY deadlocks DESC
```

1.5.4. How Can I configure parameters to avoid out of memory error?

In order to avoid out of memory error, you should make sure the values for the paramaters *NumberOfBuffers* and *MaxCheckpointRemap* are not set with the same values.

For example, the following configuration will cause an error out of memory:

```
# virtuoso.ini

...
[Parameters]
NumberOfBuffers = 246837
MaxDirtyBuffers = 18517
MaxCheckpointRemap = 246837
...
```

Changing the value of the parameter *MaxCheckpointRemap* with let's say 1/4 of the DB size will resolve the issue.

1.5.5. What are "Generate RDB2RDF triggers" and "Enable Data Syncs with Physical Quad Store" Linked Data Views options?

These Linked Data Views options basically persist the triples in the transient View Graph in the Native Quad Store. The Data Sync is how you keep the transient views in sync with the persisted triples.

Without this capability you cannot exploit faceted browsing without severe performance overhead when using Linked Data based conceptual views over ODBC or JDBC accessible data sources.

Note: Using these options when the RFViews have already been created is not currently possible via the Conductor UI. Instead you should be able to add them manually from isql:

1. Drop the Linked Data View graph and Quad Map.
2. Create it again with the RDB2RDF Triggers options enabled.

See Also:
RDB2RDF Triggers

1.5.6. How to Manage Date Range SPARQL queries?

The following examples demonstrate how to manage date range in a SPARQL query:

Example with date range

```
SELECT ?s ?date
FROM <http://dbpedia.org>
WHERE
  {
    ?s ?p ?date . FILTER ( ?date >= "19450101"^^xsd:date && ?date <= "19451231"^^xsd:date )
  }
LIMIT 100
```

Example with bif:contains

Suppose there is the following query using bif:contains for date:

If ?date is of type xsd:date or xsd:dateTime and of valid syntax then bif:contains(?date, '"1945*"') will not found it, because it will be parsed at load/create and stored as SQL DATE value.

So if data are all accurate and typed properly then the filter is:

```
(?date >= xsd:date("1945-01-01") && ?date < xsd:date("1946-01-01"))
```

i.e. the query should be:

```
SELECT DISTINCT ?s ?date
FROM <http://dbpedia.org>
WHERE
  {
    ?s ?p ?date . FILTER( ?date >= xsd:date("1945-01-01") && ?date < xsd:date("1946-01-01")  && (str(?p)
  }
LIMIT 10
```

If data falls, then the free-text will be OK for tiny examples but not for "big" cases because *bif:contains(?date, '"1945*"')* would require that less than 200 words in the table begins with 1945. Still, some data can be of accurate type and syntax so range comparison should be used for them and results aggregated via UNION.

If dates mention timezones then the application can chose the beginning and the end of the year in some timezones other than the default.

1.5.7. How can I see which quad storages exist and in which quad storage a graph resides?

Let's take for example a created Linked Data View from relational data in Virtuoso . The RDF output therefor should have two graphs which reside in a quad storage named for ex.:

```
http://example.com/rdfv_demo/quad_storage/default
```

Also the RDF is accessible over the SPARQL endpoint with the following query:

```
define input:storage <http://example.com/rdfv_demo/quad_storage/default>
SELECT *
WHERE
  {
    ?s ?p ?o
  }
```

Now one could ask is there a way to define internally (once) that the quad storage should be included in queries to the SPARQL endpoint? So that the user does not have to define the *input:storage* explicitly in each query, like this:

```
http://example.com/sparql?query=select * where { ?s ?p ?o }&default-graph-uri=NULL&named-graph-uri=NULL
```

All metadata about all RDF storages are kept in "system" graph <http://www.openlinksw.com/schemas/virtrdf#> (namespace prefix *virtrdf:*). Subjects of type *virtrdf:QuadStorage* are RDF storages. There are three of them by default:

```
SQL> SPARQL SELECT * FROM virtrdf: WHERE { ?s a virtrdf:QuadStorage };
s
VARCHAR
_______________________________________________________________________________

http://www.openlinksw.com/schemas/virtrdf#DefaultQuadStorage
http://www.openlinksw.com/schemas/virtrdf#DefaultServiceStorage
http://www.openlinksw.com/schemas/virtrdf#SyncToQuads

3 Rows. -- 3 msec.
```

- *virtrdf:DefaultQuadStorage*

is what's in use if no input:storage specified.
- *virtrdf:DefaultServiceStorage*

  will be used for SPARQL federation.
- *virtrdf:SyncToQuads*

  is to keep the list of Linked Data Views that are translated into RDB2RDF triggers.

There are two ways of using the Linked Data View from above in SPARQL endpoint without *define input:storage* :

1. Create Linked Data View right in

   virtrdf:DefaultQuadStorage

   or add the view in other storage and then copy it from there to

   virtrdf:DefaultQuadStorage

   .

   - In any of these two variants, use:

     ```
     SPARQL ALTER QUAD STORAGE virtrdf:DefaultQuadStorage . . .
     ```
2. Use

   SYS_SPARQL_HOST

   table as described here and set

   SH_DEFINES

   so it contains your favorite

   define input:storage

   .

1.5.8. Can I drop and re-create the DefaultQuadStorage?

Currently system metadata consist of three "levels":

1. *QuadMapFormats*

   are used to describe transformations of individual SQL values (or types of SQL values),
2. *QuadMaps*

   refers to

   QuadMapFormats

   (via

   QuadMapValues

) and describe some "minimal" independent RDB2RDF transformations,
3. *QuadStorages*

   organizes

   QuadMaps

.

QuadStorages contains only "symlinks" to maps, if you drop a storage you don't drop all mappings inside. If you drop the *DefaultQuadStorage* or some other built-in thing, it can be safely recovered by `DB.DBA.RDF_AUDIT_METADATA` , with first parameter set to 1. This will keep your own data intact. However we recommend to write a script that declares all your formats, Linked Data Views and storages, to be able to reproduce the configuration after any failures.

1.5.9. How to display only some information from RDF graph?

Virtuoso supports graph-level security, as described here but not subject-level or predicate-level. When exposing data that needs protected access, triples should be confined to private name graphs which are protected by ACLs using WebID.

Note, how you can use WebID to protect Virtuoso SPARQL endpoints .

See Also:
RDF Graphs Security

1.5.10. Is it possible to have the SPARQL endpoint on a different port than the Conductor?

Virtuoso Web Server has the capability to create extra listeners using the Conductor interface .

1. At install time you have your HTTP Server port in your virtuoso.ini set to 8890, which you want to keep in your local network as this contains ALL the endpoints that you have registered in Virtuoso. So as long as you do not open this port in your firewall, you can only get at it from the local machine.
2. Next you should create a new vhost entry using the EXTERNAL name of your machine and use port 80 (or a higher port if you do not want to run as root) for ex.:

```
Interface: 0.0.0.0
Port: 8080
Http Host:  my.example.com
```

3. Next you add a "New directory to this line", click on "Type" radio button and choose "Sparql access point" from the drop-down list and press Next button. Set "Path" to /sparql and press the "Save Changes" button to store.
4. At this point you have created: http://my.example.com:8080/sparql which functions exactly the same as your internal http://example.com/sparql. You can now open your firewall and allow outside machines to connect to port 8080 so people can use your SPARQL endpoint without access to any other endpoint on your Virtuoso installation.
5. You should probably also change your virtuoso.ini so:

```
[URIQA]
DefaultHost = my.example.com:8080
```

6. If you use port 80, you do not have to add :80 at the end of this setting, although it should not make any difference.
7. You can now add other directories / endpoints to the new

   my.example.com

   interface you just created e.g. a nice / directory that points to a index.html which describes your site etc.

See Also:
Internet Domains

1.5.11. How to enable the Virtuoso Entity Framework 3.5 ADO.Net Provider in Visual Studio 2010?

The Virtuoso Entity Framework 3.5 ADO.Net Provider is current only list as a Visible control in the Visual Studio 2008 IDE as the current installers only create the necessary registry entries for Visual Studio 2008. To make it visible in the Visual Studio 2010's IDE the following registry settings need to be manually updated and manual addtions to some of the VS 2010 XML configuration files:

1. Export following registry keys to .reg files and using a text editor, such as Wordpad, edit the Visual Studio version numbers from 8.0 or 9.0 to 10.0:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\VisualStudio\8.0\DataProviders\{EE00F82B-C5A4-4073-8FF1-33F8
- and -
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\VisualStudio\8.0\DataSources\{90FBCAF2-8F42-47CD-BF1A-88FF41
```

2. Once edited, save and double click them to create the new registry entries under:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\VisualStudio\10.0....
```

3. In addition, locate the file C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\CONFIG\machine.config, locate the node, then locate the

   <add name="VirtuosoClient3? Data Provider" ...

   node within it:

```
<add name="VirtuosoClient3 Data Provider" invariant="OpenLink.Data.Virtuoso"
    description=".NET Framework Data Provider for Virtuoso" type="OpenLink.Data.Virtuoso.VirtuosoC
```

and copy is to the equivalent C:\WINDOWS\Microsoft.NET\Frameworks\v4.0.30128\CONFIG\machine.config location.

Visual Studio 2010 will then have the necessary information to locate and load the Virtuoso ADO.Net provider in its IDE.

The registry should typically contain the following entries for Visual Studio 2010 as a result:

```
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\VisualStudio\10.0\DataProviders\{0886A2BB-03D0-4E00-8A
@=".NET Framework Data Provider for Virtuoso"
"AssociatedSource"="{4D90D7C5-69A6-43EE-83ED-59A0E442D260}"
"CodeBase"="C:\\Windows\\assembly\\GAC_MSIL\\virtado3\\6.2.3128.1__391bf132017ae989\\virtado3.dll"
"Description"="Provider_Description, OpenLink.Data.Virtuoso.DDEX.Net3.DDEXResources, virtado3, Version=6.
"DisplayName"="Provider_DisplayName, OpenLink.Data.Virtuoso.DDEX.Net3.DDEXResources, virtado3, Version=6.
"InvariantName"="OpenLink.Data.Virtuoso"
"PlatformVersion"="2.0"
"ShortDisplayName"="Provider_ShortDisplayName, OpenLink.Data.Virtuoso.DDEX.Net3.DDEXResources, virtado3,
"Technology"="{77AB9A9D-78B9-4ba7-91AC-873F5338F1D2}"

[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\VisualStudio\10.0\DataProviders\{0886A2BB-03D0-4E00-8A

[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\VisualStudio\10.0\DataProviders\{0886A2BB-03D0-4E00-8A
@="Microsoft.VisualStudio.Data.Framework.DSRefBuilder"
"Assembly"="Microsoft.VisualStudio.Data.Framework, Version=10.0.0.0, Culture=neutral, PublicKeyToken=b03f

[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\VisualStudio\10.0\DataProviders\{0886A2BB-03D0-4E00-8A

[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\VisualStudio\10.0\DataProviders\{0886A2BB-03D0-4E00-8A

[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\VisualStudio\10.0\DataProviders\{0886A2BB-03D0-4E00-8A
@="OpenLink.Data.Virtuoso.DDEX.Net3.VirtuosoDataConnectionProperties"
"Assembly"="virtado3, Version=6.2.3128.1, Culture=neutral, PublicKeyToken=391bf132017ae989"

[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\VisualStudio\10.0\DataProviders\{0886A2BB-03D0-4E00-8A
@="Microsoft.VisualStudio.Data.Framework.AdoDotNet.AdoDotNetConnectionSupport"
"Assembly"="Microsoft.VisualStudio.Data.Framework, Version=10.0.0.0, Culture=neutral, PublicKeyToken=b03f

[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\VisualStudio\10.0\DataProviders\{0886A2BB-03D0-4E00-8A
@="OpenLink.Data.Virtuoso.DDEX.Net3.VirtuosoDataConnectionUIControl"
"Assembly"="virtado3, Version=6.2.3128.1, Culture=neutral, PublicKeyToken=391bf132017ae989"

[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\VisualStudio\10.0\DataProviders\{0886A2BB-03D0-4E00-8A
@="OpenLink.Data.Virtuoso.DDEX.Net3.VirtuosoDataConnectionProperties"
"Assembly"="virtado3, Version=6.2.3128.1, Culture=neutral, PublicKeyToken=391bf132017ae989"

[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\VisualStudio\10.0\DataProviders\{0886A2BB-03D0-4E00-8A

[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\VisualStudio\10.0\DataProviders\{0886A2BB-03D0-4E00-8A
@="OpenLink.Data.Virtuoso.DDEX.Net3.VirtuosoDataObjectIdentifierResolver"
"Assembly"="virtado3, Version=6.2.3128.1, Culture=neutral, PublicKeyToken=391bf132017ae989"

[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\VisualStudio\10.0\DataProviders\{0886A2BB-03D0-4E00-8A
@="Microsoft.VisualStudio.Data.Framework.DataObjectSupport"
"Assembly"="Microsoft.VisualStudio.Data.Framework, Version=10.0.0.0, Culture=neutral, PublicKeyToken=b03f
"XmlResource"="OpenLink.Data.Virtuoso.DDEX.Net3.VirtuosoObjectSupport"

[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\VisualStudio\10.0\DataProviders\{0886A2BB-03D0-4E00-8A
```

```
@="OpenLink.Data.Virtuoso.DDEX.Net3.VirtuosoDataSourceInformation"
"Assembly"="virtado3, Version=6.2.3128.1, Culture=neutral, PublicKeyToken=391bf132017ae989"

[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\VisualStudio\10.0\DataProviders\{0886A2BB-03D0-4E00-8A

[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\VisualStudio\10.0\DataProviders\{0886A2BB-03D0-4E00-8A
@="Microsoft.VisualStudio.Data.Framework.DataViewSupport"
"Assembly"="Microsoft.VisualStudio.Data.Framework, Version=10.0.0.0, Culture=neutral, PublicKeyToken=b03f
"XmlResource"="OpenLink.Data.Virtuoso.DDEX.Net3.VirtuosoViewSupport"

[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\VisualStudio\10.0\DataSources\{4D90D7C5-69A6-43EE-83ED
@="OpenLink Virtuoso Data Source"
"DefaultProvider"="{0886A2BB-03D0-4E00-8A3D-F235A5DC0F6D}"

[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\VisualStudio\10.0\DataSources\{4D90D7C5-69A6-43EE-83ED

[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\VisualStudio\10.0\DataSources\{4D90D7C5-69A6-43EE-83ED
"Description"="DataSource_Description, OpenLink.Data.Virtuoso.DDEX.Net3.DDEXResources, virtado3, Version=
```

The next Virtuoso releases, 6.3+ will support this new Visual Studio 2010 release out of the box.

1.5.12. How Can I Control the normalization of UNICODE3 accented chars in free-text index?

The normalization of UNICODE3 accented chars in free-text index can be controlled by setting up the configuration parameter *XAnyNormalization* in the virtuoso.ini config file, section [I18N]. This parameter controls whether accented UNICODE characters should be converted to their non-accented base variants at the very beginning of free-text indexing or parsing a free-text query string. The parameter's value is an integer that is bitmask with only 2 bits in use atm:

1. 0: the default behavior, do not normalize anything, so "Jose" and "José" are two distinct words.
2. 2: Any combined char is converted to its (smallest known) base. So "é" will lose its accent and become plain old ASCII "e".
3. 3: This is equl to 1|2 and when set then performs both conversions. As a result, pair of base char and combinig char loses its second char and chars with accents will lose accents.

If the parameter is required at all, the needed value is probably 3. So the fragment of virtuoso.ini is:

```
[I18N]
XAnyNormalization=3
```

In some seldom case the value of 1 can be appropriate. The parameter should be set once before creating the database. If changed on the existing database, all free-text indexes that may contain non-ASCII data should be re-created. On a typical system, the parameter affects all text columns, XML columns, RDF literals and queries.

Strictly speaking, it affects not all of them but only items that use default "x-any" language or language derived from x-any such as "en" and "en-US" but if you haven't tried writing new C plugins for custom languages you should not look so deep.

As an example, with *XAnyNormalization=3* once can get the following:

```
SQL>SPARQL

INSERT IN <http://InternationalNSMs/>
   { <s> <sp> "Índio João Macapá Júnior Tôrres Luís Araújo José" ;
     <ru> "&#1054;&#1085; &#1076;&#1086;&#1073;&#1072;&#1074;&#1080;&#1083; &#1082;&#1072;&#1088;&#1090;&

INSERT INTO <http://InternationalNSMs/>, 2 (or less) triples -- done

SQL> DB.DBA.RDF_OBJ_FT_RULE_ADD (null, null, 'InternationalNSMs.wb');

Done. -- 0 msec.

SQL>vt_inc_index_db_dba_rdf_obj();

Done. -- 26 msec.

SQL>SPARQL
SELECT *
```

```
FROM <http://InternationalNSMs/>
WHERE
  {
    ?s ?p ?o
  }
ORDER BY ASC (str(?o))

s  sp  Índio João Macapá Júnior Tôrres Luís Araújo José
s  ru  &#1054;&#1085; &#1076;&#1086;&#1073;&#1072;&#1074;&#1080;&#1083; &#1082;&#1072;&#1088;&#1090;&#108

2 Rows. -- 2 msec.

SQL> SPARQL
SELECT *
FROM <http://InternationalNSMs/>
WHERE
  {
    ?s ?p ?o . ?o bif:contains "'Índio João Macapá Júnior Tôrres Luís Araújo José'" .
  }

s  sp  Índio João Macapá Júnior Tôrres Luís Araújo José

1 Rows. -- 2 msec.

SQL>SPARQL
SELECT *
FROM <http://InternationalNSMs/>
WHERE
  {
    ?s ?p ?o . ?o bif:contains "'Indio Joao Macapa Junior Torres Luis Araujo Jose'" . }

s  sp  Índio João Macapá Júnior Tôrres Luís Araújo José

1 Rows. -- 1 msec.

SQL> SPARQL
SELECT *
FROM <http://InternationalNSMs/>
WHERE
  {
    ?s ?p ?o . ?o bif:contains "'&#1087;&#1086;&#1089;&#1090;&#1072;&#1074;&#1080;&#1083; &#1072;&#1082;&

s  ru  &#1054;&#1085; &#1076;&#1086;&#1073;&#1072;&#1074;&#1080;&#1083; &#1082;&#1072;&#1088;&#1090;&#108
```

There was also request for function that normalizes characters in strings as free-text engine will do with XAnyNormalization=3 , the function will be provided as a separate patch and depends on this specific patch.

See Also:
Virtuoso Configuration File

1.5.13. How Can I define graph with virt:rdf_sponger option set to "on"?

Suppose we have the following scenario:

1. Create Virtuoso user using Conductor for ex. with name "john" and pwd 1.
2. Create for the user a RDF Sink folder for ex. with name "MySinkFolder" from type "RDF Upload Folder" or use the

   rdf_sink

   folder created automatically for your user.
3. In the properties page of the RDF sink folder add in the WebDAV section this property

   virt:rdf_graph

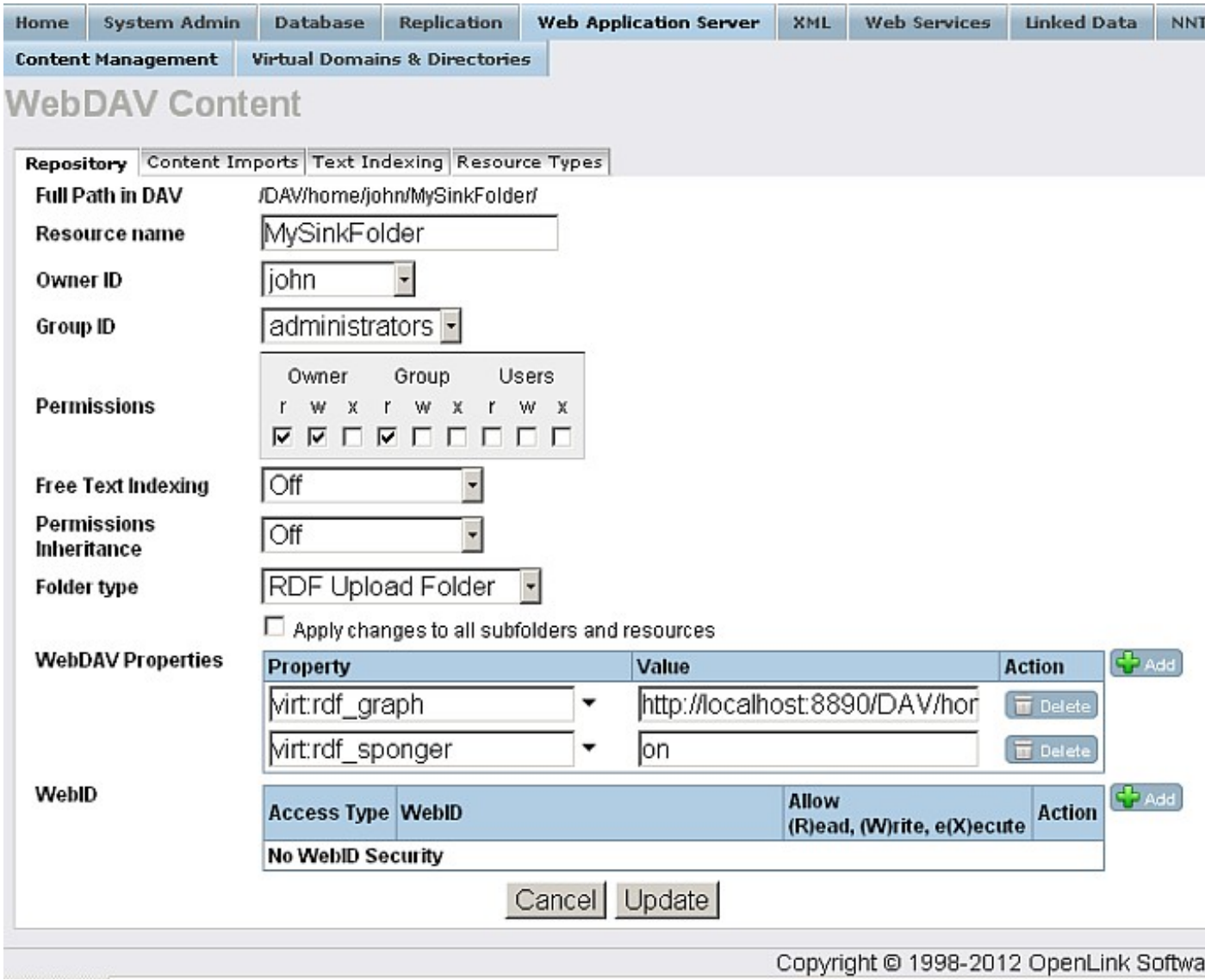
   with value:

   ```
   http://host:port/DAV/home/<user-name>/<rdf-sink-folder>/
   ```

```
-- So in our example it should be:
http://example.com/DAV/home/john/MySinkFolder/
```

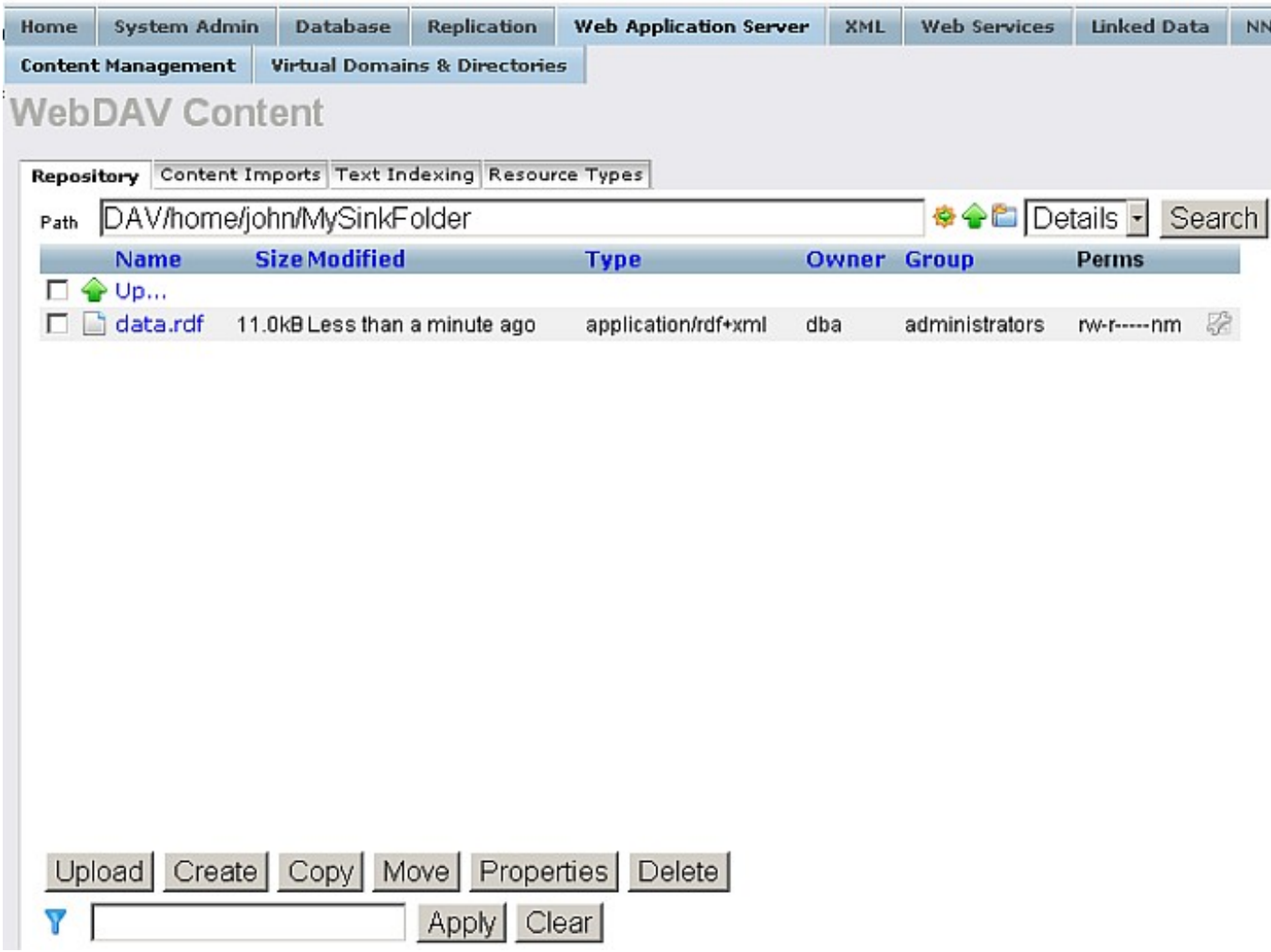
4. Add another property virt:rdf_sponger with value "on".

Figure 1.2.



5. Upload RDF files to the RDF Sink folder "MySinkFolder", for ex. upload file with name "data.rdf":

Figure 1.3.



6. As result the RDF data should be stored in graph depending on your folder name etc.:

```
http:///local.virt/DAV/home/<user-name>/<rdf-sink-folder>/<resource>

-- So in our example it will be:
http:///local.virt/DAV/home/john/MySinkFolder/data.rdf
```

   a. Go to http://host:port/sparql ;
   b. Execute simple query to view the graph triples:

```
SELECT  *
FROM <http://local.virt/DAV/home/john/MySinkFolder/data.rdf>
WHERE
  {
    ?s ?p ?o
  }
```

Figure 1.4.


Virtuoso SPARQL Query Editor
About | Namespace Prefixes | Infer
Default Data Set Name (Graph IRI)
Query Text
SELECT *
FROM <http://local.virt/DAV/home/john/MySinkFolder/data.rdf>
WHERE
{
?s ?p ?o
}
Sponging:
Use only local data (including data retrieved before), but do not retrieve more
Results Format:
HTML
(The CXML output is disabled, see details)
Execution timeout:
0
milliseconds (values less than 1000 are ignored)
Options:
Strict checking of void variables
(The result can only be sent back to browser, not saved on the server, see details)

Figure 1.5.

s	p
http://rdfweb.org/people/danbri/rdfweb/danbri-foaf.rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://rdfweb.org/people/danbri/rdfweb/danbri-foaf.rdf	http://xmlns.com/wot/0.1/assurance
http://rdfweb.org/people/danbri/rdfweb/libby.gif	http://purl.org/dc/elements/1.1/title
http://rdfweb.org/people/danbri/rdfweb/libby.gif	http://purl.org/dc/elements/1.1/description
http://rdfweb.org/people/danbri/rdfweb/libby.gif	http://purl.org/dc/elements/1.1/format
http://rdfweb.org/people/danbri/rdfweb/libby.gif	http://xmlns.com/foaf/0.1/thumbnail
http://www.ilrt.bristol.ac.uk/people/cmdjb/events/dc7/orig/eric.png	http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://www.ilrt.bristol.ac.uk/people/cmdjb/events/dc7/orig/eric.png	http://purl.org/dc/elements/1.1/title
http://www.ilrt.bristol.ac.uk/people/cmdjb/events/dc7/orig/eric.png	http://purl.org/dc/elements/1.1/description
http://www.ilrt.bristol.ac.uk/people/cmdjb/events/dc7/orig/eric.png	http://purl.org/dc/elements/1.1/format
http://www.ilrt.bristol.ac.uk/people/cmdjb/events/dc7/orig/eric.png	http://purl.org/dc/elements/1.1/thumbnail
http://ioctl.org/jan/test/wizard.jpg	http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://ioctl.org/jan/test/wizard.jpg	http://purl.org/dc/elements/1.1/title
http://ioctl.org/jan/test/wizard.jpg	http://purl.org/dc/elements/1.1/description
http://ioctl.org/jan/test/wizard.jpg	http://purl.org/dc/elements/1.1/format
http://ioctl.org/jan/test/wizard.jpg	http://purl.org/dc/elements/1.1/thumbnail
http://www.blogspace.com/rss/rss10	http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://www.blogspace.com/rss/rss10	http://www.w3.org/2000/01/rdf-schema#seeAlso
http://www.blogspace.com/rss/rss10	http://purl.org/rss/1.0/title
http://website.lineone.net/~steve_c-t/Scientology/Pickets/10-03-2001/damien.jpg	http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://website.lineone.net/~steve_c-t/Scientology/Pickets/10-03-2001/damien.jpg	http://purl.org/dc/elements/1.1/description
http://website.lineone.net/~steve_c-t/Scientology/Pickets/10-03-2001/damien.jpg	http://purl.org/dc/elements/1.1/thumbnail

7. In order to define any graph you want with the options from above, you should execute:

```
SQL> DAV_PROP_SET ('/DAV/home/<user-name>/<rdf-sink-folder>/',  'virt:rdf_graph', iri, <user-name>

-- So in our example it should be:
SQL> DAV_PROP_SET ('/DAV/home/john/MySinkFolder/',  'virt:rdf_graph', 'http://mydata.com', 'john',
```

   - Note: calling this function uses the given IRI as the graph IRI when sponging stuff put in <your-rdf-sink-folder>.

8. Finally you should execute the following command to get the RDF data from the new graph:

```
SQL> SELECT DAV_PROP_GET ('/DAV/home/<user-name>/<your-rdf-sink-folder>/', 'virt:rdf_graph',<user-

-- So in our example it should be:
SQL> SELECT DAV_PROP_GET ('/DAV/home/john/MySinkFolder/', 'virt:rdf_graph','john', '1');

Query result:
DAV_PROP_GET
http://example.com/DAV/home/john/MySinkFolder/

No. of rows in result: 1
```

1.5.14. How do I use SPARUL to change a selection of property values from URI References to Literals?

Assume a given graph where triples are comprised of property values that are mixed across URI References and Typed Literals as exemplified by the results of the query below:

```
SELECT DISTINCT ?sa ?oa
```

```
FROM <http://ucb.com/nbeabase>
WHERE
  {
    ?sa a <http://ucb.com/nbeabase/resource/Batch> .
    ?sa <http://ucb.com/nbeabase/resource/chemAbsNo> ?oa . FILTER regex(?oa, '-','i')
  }
```

You can use the following SPARUL pattern to harmonize the property values across relevant triples in a specific graph, as shown below:

```
SQL> SPARQL
INSERT INTO GRAPH <http://ucb.com/nbeabase>
  {
    ?sa <http://ucb.com/nbeabase/resource/sampleId> `str (?oa)`
  }
WHERE
  {
    ?sa <http://ucb.com/nbeabase/resource/chemAbsNo> ?oa . FILTER regex(?oa, '-','i')
  }
```

1.5.15. How is a Checkpoint performed against a Virtuoso Clustered Server?

The cluster cl_exec() function is used to perform a checkpoint across all node of a Virtuoso cluster as follows:

```
SQL>cl_exec ('checkpoint');
```

This typically needs to be run after loading RDF datasets into a Virtuoso cluster to prevent lose of data when the cluster is restarted.

1.5.16. How can I use CONSTRUCT with PreparedStatements?

Assume a given query which uses pragma *output:format '_JAVA_'* with CONSTRUCT:

```
SPARQL DEFINE output:format '_JAVA_'
   CONSTRUCT { ?s ?p ?o }
WHERE
  {
    ?s ?p ?o .
    FILTER (?s = iri(?::0))
  }
LIMIT 1
```

In order to work correctly, the query should be modified to:

```
SPARQL DEFINE output:format '_JAVA_'
   CONSTRUCT { `iri(?::0)` ?p ?o }
WHERE
  {
    `iri(?::0)` ?p ?o
  }
LIMIT 1
```

Equivalent variant of the query is also:

```
SPARQL DEFINE output:format '_JAVA_'
  CONSTRUCT { ?s ?p ?o }
WHERE
  {
    ?s ?p ?o .
    FILTER (?s = iri(?::0))
  }
LIMIT 1
```

1.5.17. How can perform SPARQL Updates without transactional log size getting exceed?

Since SPARUL updates are generally not meant to be transactional, it is best to run these in:

```
SQL> log_enable (2);
```

mode, which commits every operation as it is done. This prevents one from running out of rollback space. Also for bulk updates, transaction logging can be turned off. If so, one should do a manual checkpoint after the operation to ensure persistence across server restart since there is no roll forward log.

Alternatively, the "*TransactionAfterImageLimit* " parameter can be set in the virtuoso.ini config file to a higher value than its 50MB default:

```
#virtuoso.ini
...
[Parameters]
...
TransactionAfterImageLimit = N bytes default 50000000
...
```

See Also:

Using SPARUL

Virtuoso INI Parameters

1.5.18. How can I write custom crawler using PL?

The following code is an example of loading data via crawler with special function to generate link for downloading:

```
create procedure EUROPEANA_STORE (
  in _host varchar,
  in _url varchar,
  in _root varchar,
  inout _content varchar,
  in _s_etag varchar,
  in _c_type varchar,
  in store_flag int := 1,
  in udata any := null,
  in lev int := 0)
{
   declare url varchar;
   declare xt, xp any;
   declare old_mode int;
   xt := xtree_doc (_content, 2);
   xp := xpath_eval ('//table//tr/td/a[@href]/text()', xt, 0);
   commit work;
   old_mode := log_enable (3,1);
   foreach (any u in xp) do
     {
       u := cast (u as varchar);
       url := sprintf ('http://semanticweb.cs.vu.nl/europeana/api/export_graph?graph=%U&mimetype=default&
       dbg_printf ('%s', u);
         {
           declare continue handler for sqlstate '*' {
             dbg_printf ('ERROR: %s', __SQL_MESSAGE);
           };
           SPARQL LOAD ?:url into GRAPH ?:u;
         }
     }
   log_enable (old_mode, 1);
   return WS.WS.LOCAL_STORE (_host, _url, _root, _content, _s_etag, _c_type, store_flag, 0);
}
```

See Also:

Set Up the Content Crawler to Gather RDF

1.5.19. How Can I Get an exact mapping for a date?

Assume a given attempts to get an exact mapping for the literal "1950" using *bif:contains* :

```
SPARQL
SELECT DISTINCT ?s ?o
FROM <http://dbpedia.org>
WHERE
  {
    ?s ?p ?o .
   FILTER( bif:contains (?o, '"1950"')
           && isLiteral(?o)
           && ( str(?p) ! = rdfs:label  || str(?p) !=  foaf:name
           && ( ?o='1950')
         )
  }
```

As an integer 1950 or date 1950-01-01 are not texts, they are not in free-text index and thus invisible for CONTAINS free-text predicate.

A possible way to make them visible that way is to introduce an additional RDF predicate that will contain objects of the triples in question, converted to strings via str() function.

Thus better results will be approached: if searches about dates are frequent then a new predicate can have date/datetime values extracted from texts, eliminating the need for bif:contains.

Therefor, the query from above should be changed to:

```
SPARQL
SELECT DISTINCT ?s ?o
FROM <http://dbpedia.org>
WHERE
  {
    ?s ?p ?o .
    FILTER (  isLiteral(?o)
              && (  str(?p) != str(rdfs:label) || str(?p) !=  foaf:name )
              && str(?o) in ("1950", "1950-01-01"))
  }
```

1.5.20. How Can I Get certificate attributes using SPARQL?

The SPARQL query should use the *cert:hex* and *cert:decimal* in order to get to the values, so for ex:

```
PREFIX cert: <http://www.w3.org/ns/auth/cert#>
PREFIX rsa: <http://www.w3.org/ns/auth/rsa#>

SELECT ?webid
FROM <http://webid.myxwiki.org/xwiki/bin/view/XWiki/homepw4>
WHERE
  {
    [] cert:identity ?webid ;
             rsa:modulus ?m ;
     rsa:public_exponent ?e .
     ?m cert:hex "b520f38479f5803a7ab33233155eeef8ad4e1f575b603f7780f3f60ceab1\n34618fbe117539109c015c5f9
     ?e cert:decimal "65537"^^xsd:string
  }
```

1.5.21. How can I make Multi Thread Virtuoso connection using JDBC?

See details here .

1.5.22. How Do I Perform Bulk Loading of RDF Source Files into one or more Graph IRIs?

See details here .

1.5.23. How to exploit RDF Schema and OWL Inference Rules with minimal effort?

When you install Virtuoso, it's reasoner and highly scalable inference capabilities may not be obvious. Typical cases involve using *rdfs:subClassOf* predicates in queries and wondering why reasoning hasn't occurred in line with the semantics defined in RDF Schema.

The experience applies when using more sophisticated predicates from OWL such as *owl:equivalentProperty* , *owl:equivalentClass* , *owl:sameAs* , *owl:SymmetricalProperty* , *owl:inverseOf* etc ...

Virtuoso implemented inference rules processing in a loosely coupled manner that allow users to conditionally apply inference context (via rules) to SPARQL queries. Typically, you have to create these rules following steps outlined here .

This tips and tricks note provides a shortcut for setting up and exploring RDF Schema and OWL reasoning once you've installed the Virtuoso Faceted Browser VAD package .

See Also:
Inference Rules and Reasoning

Virtuoso Faceted Browser Installation and configuration

Virtuoso Faceted Web Service

1.5.24. How can I dump arbitrary query result as N-Triples?

Assume the following arbitrary query:

```
SPARQL define output:format "NT"
CONSTRUCT { ?s a ?t }
FROM virtrdf:
WHERE { ?s a ?t };
```

For iteration over result-set of an arbitrary query, use `exec_next()` in a loop that begins with `exec()` with cursor output variable as an argument and ends with `exec_close()` after it is out of data.

1.5.25. How do I bind named graph parameter in prepared statement?

Assume the following SPARQL query:

```
CONSTRUCT
  {
    ?s ?p ?o
  }
FROM ?context
WHERE
  {
    ?s ?p ?o
  }
```

To bind the named graph context of the query from above, the best solution due to performance implications, is to change the syntax of the query as:

```
CONSTRUCT
  {
    ?s ?p ?o
  }
WHERE
  {
    graph `iri(??)` { ?s ?p ?o }
  }
```

Note: In case of using "FROM clause", it needs a constant in order to check at the compile time whether the IRI refers to a graph or a graph group:

- Assume "FROM clause" is used as for ex:

```
CONSTRUCT
  {
    ?s ?p ?o
  }
FROM `iri(??)`
WHERE
  {
    ?s ?p ?o
  }
```

- In this case can be made security checks at the compile time (i.e., once) and not waste time at the run time.

- Assume "FROM clause" is used as for ex:

```
CONSTRUCT
  {
    ?s ?p ?o
  }
FROM iri(??)
WHERE
  {
    ?s ?p ?o
  }
```

  - In this case a compile-time check, a run-time check or 50/50 mix of them can be performed, depending on the security policies.
- FROM without constant would mean that security rules, if they present in the storage, are used at the run time, for every triple that matches every triple pattern in the default graph. This issue can be reproduced if FROM is not specified at all: if security differs from default then the effect is noticeably bad.

1.5.26. How can I insert binary data to Virtuoso RDF storage in plain queries and with parameter binding via ADO.NET calls?

The following example shows different methods for insert binary data to Virtuoso RDF storage in plain queries and with parameter binding via ADO.NET calls:

```
# Test_Bin.cs

using System;
using System.Runtime.InteropServices;
using System.Text;
using System.Data;
using OpenLink.Data.Virtuoso;

#if ODBC_CLIENT
namespace OpenLink.Data.VirtuosoOdbcClient
#elif CLIENT
namespace OpenLink.Data.VirtuosoClient
#else
namespace OpenLink.Data.VirtuosoTest
#endif
{
    class Test_Bin
    {
        [STAThread]
        static void Main(string[] args)
        {
            IDataReader myread = null;
            IDbConnection c;

            c = new VirtuosoConnection("HOST=localhost:1111;UID=dba;PWD=dba;");

            IDbCommand cmd = c.CreateCommand();
            int ros;

            try
            {
                c.Open();

                cmd.CommandText = "sparql clear graph <ado.bin>";
                cmd.ExecuteNonQuery();
```

```
//insert binary as base64Binary
                cmd.CommandText = "sparql insert into graph <ado.bin> { <res1> <attr> \"GpM7\"^^<http://w
                cmd.ExecuteNonQuery();

//insert binary as hexBinary
                cmd.CommandText = "sparql insert into graph <ado.bin> { <res2> <attr> \"0FB7\"^^<http://w
                cmd.ExecuteNonQuery();

//prepare for insert with parameter binding
                cmd.CommandText = "sparql define output:format '_JAVA_' insert into graph <ado.bin> { `ir

//bind parameters for insert binary as base64Binary
                IDbDataParameter param = cmd.CreateParameter();
                param.ParameterName = "p1";
                param.DbType = DbType.AnsiString;
                param.Value = "res3";
                cmd.Parameters.Add(param);

                param = cmd.CreateParameter();
                param.ParameterName = "p2";
                param.DbType = DbType.Int32;
                param.Value = 4;
                cmd.Parameters.Add(param);

                param = cmd.CreateParameter();
                param.ParameterName = "p3";
                param.DbType = DbType.AnsiString;
                param.Value = "GpM7";
                cmd.Parameters.Add(param);

                param = cmd.CreateParameter();
                param.ParameterName = "p4";
                param.DbType = DbType.AnsiString;
                param.Value = "http://www.w3.org/2001/XMLSchema#base64Binary";
                cmd.Parameters.Add(param);

                cmd.ExecuteNonQuery();

                cmd.Parameters.Clear();

//bind parameters for insert binary as hexBinary
                param = cmd.CreateParameter();
                param.ParameterName = "p1";
                param.DbType = DbType.AnsiString;
                param.Value = "res4";
                cmd.Parameters.Add(param);

                param = cmd.CreateParameter();
                param.ParameterName = "p2";
                param.DbType = DbType.Int32;
                param.Value = 4;
                cmd.Parameters.Add(param);

                param = cmd.CreateParameter();
                param.ParameterName = "p3";
                param.DbType = DbType.AnsiString;
                param.Value = "0FB7";
                cmd.Parameters.Add(param);

                param = cmd.CreateParameter();
                param.ParameterName = "p4";
                param.DbType = DbType.AnsiString;
                param.Value = "http://www.w3.org/2001/XMLSchema#hexBinary";
                cmd.Parameters.Add(param);

                cmd.ExecuteNonQuery();

                cmd.Parameters.Clear();

//bind parameters for insert binary as byte[]
                param = cmd.CreateParameter();
                param.ParameterName = "p1";
```

```
                    param.DbType = DbType.AnsiString;
                    param.Value = "res5";
                    cmd.Parameters.Add(param);

                    param = cmd.CreateParameter();
                    param.ParameterName = "p2";
                    param.DbType = DbType.Int32;
                    param.Value = 3;
                    cmd.Parameters.Add(param);

                    param = cmd.CreateParameter();
                    param.ParameterName = "p3";
                    param.DbType = DbType.Binary;
                    byte[] bin_val = {0x01, 0x02, 0x03, 0x04, 0x05};
                    param.Value = bin_val;
                    cmd.Parameters.Add(param);

                    param = cmd.CreateParameter();
                    param.ParameterName = "p4";
                    param.DbType = DbType.AnsiString;
                    param.Value = System.DBNull.Value;
                    cmd.Parameters.Add(param);

                    cmd.ExecuteNonQuery();

                    cmd.Parameters.Clear();

    //execute select and check the results
                    cmd.CommandText = "sparql SELECT ?s ?o FROM <ado.bin> WHERE {?s ?p ?o}"; ;
                    myread = cmd.ExecuteReader();
                    int r = 0;

                    while (myread.Read())
                    {
                        Console.WriteLine("=== ROW === "+r);
                        for (int i = 0; i < myread.FieldCount; i++)
                        {
                            string s;
                            if (myread.IsDBNull(i))
                                Console.Write("N/A|\n");
                            else
                            {
                                object o = myread.GetValue(i);
                                Type t = myread.GetFieldType(i);

                                s = myread.GetString(i);
                                Console.Write(s + "[");
                                if (o is SqlExtendedString)
                                {
                                    SqlExtendedString se = (SqlExtendedString)o;
                                    Console.Write("IriType=" + se.IriType + ";StrType=" + se.StrType + ";Valu
                                    Console.Write(";ObjectType=" + o.GetType() + "]|\n");
                                }
                                else if (o is SqlRdfBox)
                                {
                                    SqlRdfBox se = (SqlRdfBox)o;
                                    Console.Write("Lang=" + se.StrLang + ";Type=" + se.StrType + ";Value=" +
                                    Console.Write(";ObjectType=" + o.GetType() + "]|\n");
                                    object v = se.Value;
                                    if (v is System.Byte[])
                                    {
                                        byte[] vb = (byte[])v;
                                        for (int z = 0; z < vb.Length; z++)
                                        {
                                            Console.WriteLine(""+z+"="+vb[z]);
                                        }
                                    }
                                }
                                else
                                    Console.Write(o.GetType() + "]|\n");
                            }
                        }
                        r++;
```

```
                }
            }
            catch (Exception e)
            {
                Console.WriteLine("{0} Exception caught.", e);
            }
            finally
            {
                //                if (myread != null)
                //                  myread.Close();

                if (c.State == ConnectionState.Open)
                    c.Close();

            }

        }
    }
}
```

Output log for example is in the log.txt:

```
# log.txt
=== ROW === 0
res1[IriType=IRI;StrType=IRI;Value=res1;ObjectType=OpenLink.Data.Virtuoso.SqlExtendedString]|
GpM7[Lang=;Type=http://www.w3.org/2001/XMLSchema#base64Binary;Value=GpM7;ObjectType=OpenLink.Data.Virtuos
=== ROW === 1
res2[IriType=IRI;StrType=IRI;Value=res2;ObjectType=OpenLink.Data.Virtuoso.SqlExtendedString]|
0FB7[Lang=;Type=http://www.w3.org/2001/XMLSchema#hexBinary;Value=0FB7;ObjectType=OpenLink.Data.Virtuoso.S
=== ROW === 2
res3[IriType=IRI;StrType=IRI;Value=res3;ObjectType=OpenLink.Data.Virtuoso.SqlExtendedString]|
GpM7[Lang=;Type=http://www.w3.org/2001/XMLSchema#base64Binary;Value=GpM7;ObjectType=OpenLink.Data.Virtuos
=== ROW === 3
res4[IriType=IRI;StrType=IRI;Value=res4;ObjectType=OpenLink.Data.Virtuoso.SqlExtendedString]|
0FB7[Lang=;Type=http://www.w3.org/2001/XMLSchema#hexBinary;Value=0FB7;ObjectType=OpenLink.Data.Virtuoso.S
=== ROW === 4
res5[IriType=IRI;StrType=IRI;Value=res5;ObjectType=OpenLink.Data.Virtuoso.SqlExtendedString]|
0102030405[System.Byte[]]|
```

1.5.27. How can I insert RDF data from Visual Studio to Virtuoso?

The following example shows how to insert RDF Data from Visual Studio to Virtuoso:

```
using System;
using System.Runtime.InteropServices;
using System.Text;
using System.Data;
using OpenLink.Data.Virtuoso;

#if ODBC_CLIENT
namespace OpenLink.Data.VirtuosoOdbcClient
#elif CLIENT
namespace OpenLink.Data.VirtuosoClient
#else
namespace OpenLink.Data.VirtuosoTest
#endif
{
    class Test_Insert
    {
        [STAThread]
        static void Main(string[] args)
        {
            IDataReader myread = null;
            IDbConnection c;

            c = new VirtuosoConnection("HOST=localhost:1111;UID=dba;PWD=dba;Charset=UTF-8");

            IDbCommand cmd = c.CreateCommand();
            int ros;

            try
            {
```

```
                    c.Open();

                    cmd.CommandText = "sparql clear graph <ado.net>";
                    cmd.ExecuteNonQuery();

                    cmd.CommandText = "sparql insert into graph <ado.net> { <a> <P01> \"131\"^^<http://www.w3
                    cmd.ExecuteNonQuery();

                    cmd.CommandText = "sparql insert into graph <ado.net> { <a> <P02> \"1234\"^^<http://www.w
                    cmd.ExecuteNonQuery();

                    cmd.CommandText = "sparql insert into graph <ado.net> { <a> <P03> \"12345.12\"^^<http://w
                    cmd.ExecuteNonQuery();

                    cmd.CommandText = "sparql insert into graph <ado.net> { <a> <P04> \"123456.12\"^^<http://
                    cmd.ExecuteNonQuery();

                    cmd.CommandText = "sparql insert into graph <ado.net> { <a> <P05> \"123456.12\"^^<http://
                    cmd.ExecuteNonQuery();

                    cmd.CommandText = "sparql insert into graph <ado.net> { <a> <P06> \"01020304\"^^<http://w
                    cmd.ExecuteNonQuery();

                    cmd.CommandText = "sparql insert into graph <ado.net> { <a> <P07> \"01.20.1980T04:51:13\"
                    cmd.ExecuteNonQuery();

                    cmd.CommandText = "sparql insert into graph <ado.net> { <a> <P08> \"01.20.1980\"^^<http:/
                    cmd.ExecuteNonQuery();

                    cmd.CommandText = "sparql insert into graph <ado.net> { <a> <P09> \"01:20:19\"^^<http://w
                    cmd.ExecuteNonQuery();

                    cmd.CommandText = "sparql insert into graph <ado.net> { <a> <P10> \"test\" }";
                    cmd.ExecuteNonQuery();

                    cmd.CommandText = "sparql define output:format '_JAVA_' insert into graph <ado.net> { <b>

    //add Object URI
                    add_triple(cmd, "S01", 1, "test1", null);

    //add Object BNode
                    add_triple(cmd, "S02", 1, "_:test2", null);

    //add Literal
                    add_triple(cmd, "S03", 3, "test3", null);

    //add Literal with Datatype
                    add_triple(cmd, "S04", 4, "1234", "http://www.w3.org/2001/XMLSchema#integer");

    //add Literal with Lang
                    add_triple(cmd, "S05", 5, "test5", "en");

                    add_triple(cmd, "S06", 3, (short)123, null);
                    add_triple(cmd, "S07", 3, 1234, null);
                    add_triple(cmd, "S08", 3, (float)12345.12, null);
                    add_triple(cmd, "S09", 3, 123456.12, null);
                    add_triple(cmd, "S10", 3, new DateTime(2001, 02, 23, 13, 44, 51, 234), null);
                    add_triple(cmd, "S11", 3, new DateTime(2001, 02, 24), null);
                    add_triple(cmd, "S12", 3, new TimeSpan(19, 41, 23), null);

                    add_triple(cmd, "S13", 4, "GpM7", "http://www.w3.org/2001/XMLSchema#base64Binary");
                    add_triple(cmd, "S14", 4, "0FB7", "http://www.w3.org/2001/XMLSchema#hexBinary");
                    byte[] bin_val = { 0x01, 0x02, 0x03, 0x04, 0x05 };
                    add_triple(cmd, "S15", 3, bin_val, null);

                }
                catch (Exception e)
                {
                    Console.WriteLine("{0} Exception caught.", e);
                }
                finally
                {
```

```
                if (c.State == ConnectionState.Open)
                    c.Close();

            }

        }

        static void add_triple(IDbCommand cmd, string sub, int ptype, object val, string val_add)
        {
                cmd.Parameters.Clear();

                IDbDataParameter param = cmd.CreateParameter();
                param.ParameterName = "p1";
                param.DbType = DbType.AnsiString;
                param.Value = sub;
                cmd.Parameters.Add(param);

                param = cmd.CreateParameter();
                param.ParameterName = "p2";
                param.DbType = DbType.Int32;
                param.Value = ptype;
                cmd.Parameters.Add(param);

                param = cmd.CreateParameter();
                param.ParameterName = "p3";
                if (val != null && val.GetType() == typeof (System.String))
                    param.DbType = DbType.AnsiString;
                param.Value = val;
                cmd.Parameters.Add(param);

                param = cmd.CreateParameter();
                param.ParameterName = "p4";
                param.DbType = DbType.AnsiString;
                param.Value = val_add;
                cmd.Parameters.Add(param);

                cmd.ExecuteNonQuery();
        }

    }
}
```

See Also:
RDF Insert Methods in Virtuoso

1.5.28. How does default describe mode work?

The default describe mode works only if subject has a type/class. To get any anonymous subject described CBD mode should be used:

```
$ curl -i -L -H "Accept: application/rdf+xml" http://lod.openlinksw.com/describe/?url=http%3A%2F%2Fwww.mp
HTTP/1.1 303 See Other
Date: Mon, 21 Mar 2011 14:24:36 GMT
Server: Virtuoso/06.02.3129 (Linux) x86_64-generic-linux-glibc25-64  VDB
Content-Type: text/html; charset=UTF-8
Accept-Ranges: bytes
TCN: choice
Vary: negotiate,accept,Accept-Encoding
Location: http://lod.openlinksw.com/sparql?query=%20DESCRIBE%20%3Chttp%3A%2F%2Fwww.mpii.de%2Fyago%2Fresou
2Frdf%2Bxml
Content-Length: 0

HTTP/1.1 200 OK
Date: Mon, 21 Mar 2011 14:24:37 GMT
Server: Virtuoso/06.02.3129 (Linux) x86_64-generic-linux-glibc25-64  VDB
Accept-Ranges: bytes
Content-Type: application/rdf+xml; charset=UTF-8
Content-Length: 467967

<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:rdfs="http://www.w3.org/2000/01/rd
<rdf:Description rdf:about="http://www.mpii.de/yago/resource/MTV%5FMovie%5FAward%5Ffor%5FBest%5FComedic%5
```

```
<rdf:Description rdf:about="http://www.mpii.de/yago/resource/#fact_23536547421"><rdf:subject rdf:resource
<rdf:Description rdf:about="http://www.mpii.de/yago/resource/#fact_23536896725"><rdf:object rdf:resource=
...
```

1.5.29. What should I do if the Virtuoso Server is not responding to HTTP requests?

Assume the Virtuoso server is not responding to HTTP requests although SQL connection is working. In order to determine what activity is being performed that might account for this:

1. Check the status:

```
SQL> status('');
REPORT
VARCHAR
_______________________________________________________________________________

OpenLink Virtuoso VDB Server
Version 06.02.3129-pthreads for Linux as of Mar 16 2011
Registered to Uriburner (Personal Edition, unlimited connections)
Started on: 2011/03/17 10:49 GMT+60

Database Status:
  File size 0, 37598208 pages, 7313125 free.
  1000000 buffers, 993399 used, 76771 dirty 0 wired down, repl age 25548714 0 w. io 0 w/crsr.
  Disk Usage: 2642884 reads avg 4 msec, 30% r 0% w last  1389 s, 1557572 writes,
    15331 read ahead, batch = 79.  Autocompact 308508 in 219226 out, 28% saved.
Gate:  71130 2nd in reads, 0 gate write waits, 0 in while read 0 busy scrap.
Log = virtuoso.trx, 14922248 bytes
VDB: 0 exec 0 fetch 0 transact 0 error
1757362 pages have been changed since last backup (in checkpoint state)
Current backup timestamp: 0x0000-0x00-0x00
Last backup date: unknown
Clients: 5 connects, max 2 concurrent
RPC: 116 calls, -1 pending, 1 max until now, 0 queued, 2 burst reads (1%), 0 second brk=9521074176
Checkpoint Remap 331113 pages, 0 mapped back. 1180 s atomic time.
    DB master 37598208 total 7313125 free 331113 remap 40593 mapped back
   temp  569856 total 569851 free

Lock Status: 52 deadlocks of which 0 2r1w, 86078 waits,
   Currently 1 threads running 0 threads waiting 0 threads in vdb.
Pending:

25 Rows. -- 1274 msec.
SQL>
```

2. Connect with the PL debugger and see what is running currently using the info threads call:

```
$ isql 1111 dba <password> -D
DEBUG> info threads
```

3. This should return the current code being executed by the Sever.
4. Run `txn_killall()` to kill any pending transactions which may enable the server to start responding to HTTP requests again:

```
SQL> txn_killall();

Done. -- 866 msec.
```

1.5.30. What CXML params are supported for the SPARQL URL pattern?

The following options are supported for CXML link behavior in the SPARQL URL Pattern:

1. Local faceted navigation links:

```
# CXML_redir_for_subjs=&CXML_redir_for_hrefs=&:

http://lod.openlinksw.com/sparql?default-graph-uri=&should-sponge=&query=SELECT+DISTINCT+%3Fcafe+%
```

2. External Resource Links:

```
# CXML_redir_for_subjs=&CXML_redir_for_hrefs=121:

http://lod.openlinksw.com/sparql?default-graph-uri=&should-sponge=&query=SELECT+DISTINCT+%3Fcafe+%
```

3. External faceted navigation links:

```
# CXML_redir_for_subjs=&CXML_redir_for_hrefs=LOCAL_PIVOT:

http://lod.openlinksw.com/sparql?default-graph-uri=&should-sponge=&query=SELECT+DISTINCT+%3Fcafe+%
```

4. External description resource(TTL):

```
# CXML_redir_for_subjs=&CXML_redir_for_hrefs=LOCAL_TTL:

http://lod.openlinksw.com/sparql?default-graph-uri=&should-sponge=&query=SELECT+DISTINCT+%3Fcafe+%
```

5. External description resource(CXML):

```
# CXML_redir_for_subjs=&CXML_redir_for_hrefs=LOCAL_CXML:

http://lod.openlinksw.com/sparql?default-graph-uri=&should-sponge=&query=SELECT+DISTINCT+%3Fcafe+%
```

1.5.31. How can I replicate all graphs?

To replicate all graphs (except the system graph http://www.openlinksw.com/schemas/virtrdf#), one should use http://www.openlinksw.com/schemas/virtrdf#rdf_repl_all as graph IRI:

```
SQL> DB.DBA.RDF_RDF_REPL_GRAPH_INS ('http://www.openlinksw.com/schemas/virtrdf#rdf_repl_all');
```

1.5.32. What is best method to get a random sample of all triples for a subset of all the resources of a SPARQL endpoint?

The best method to get a random sample of all triples for a subset of all the resources of a SPARQL endpoint, is decimation in its original style:

```
SELECT ?s ?p ?o
FROM <some-graph>
WHERE
  {
    ?s ?p ?o .
    FILTER ( 1 > bif:rnd (10, ?s, ?p, ?o) )
  }
```

By tweaking first argument of *bif:rnd()* and the left side of the inequality you can tweak decimation ratio from 1/10 to the desired value. What's important is to know that the SQL optimizer has a right to execute *bif:rnd (10)* only once at the beginning of the query, so we had to pass additional three arguments that can be known only when a table row is fetched so *bif:rnd (10, ?s, ?p, ?o)* is calculated for every row and thus any given row is either returned or ignored independently from others.

However, *bif:rnd (10, ?s, ?p, ?o)* contains a subtle inefficiency. In RDF store, graph nodes are stored as numeric IRI IDs and literal objects can be stored in a separate table. The call of an SQL function needs arguments of traditional SQL datatypes, so the query processor will extract the text of IRI for each node and the full value for each literal object. That is significant waste of time. The workaround is:

```
SPARQL
SELECT ?s ?p ?o
FROM <some-graph>
WHERE
  {
    ?s ?p ?o .
    FILTER ( 1>  <SHORT_OR_LONG::bif:rnd>  (10, ?s, ?p, ?o))
  }
```

This tells the SPARQL front-end to omit redundant conversions of values.

1.5.33. How can I replicate all graphs?

To replicate all graphs (except the system graph http://www.openlinksw.com/schemas/virtrdf#), one should use http://www.openlinksw.com/schemas/virtrdf#rdf_repl_all as graph IRI:

```
SQL> DB.DBA.RDF_RDF_REPL_GRAPH_INS ('http://www.openlinksw.com/schemas/virtrdf#rdf_repl_all');
```

1.5.34. How can I use SPARQL to make Meshups?

The following example demonstrates how to use SPARQL in order to make Meshups:

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX rtb: <http://www.openlinksw.com/schemas/oat/rdftabs#>

CONSTRUCT
  {
    ?museum geo:geometry ?museumgeo ;
             rtb:useMarker 'star' ;
             foaf:name ?musname;
             rdfs:comment ?muscomment.
    ?edu geo:geometry ?edugeo ;
          rtb:useMarker 'book' ;
          foaf:name ?eduname;
          rdfs:comment ?educomment.
    ?wh geo:geometry ?whgeo;
            rtb:useMarker '03';
            foaf:name ?whname;
            rdfs:comment ?whcomment.
  }
WHERE
  {
    {
      ?museum a dbo:Museum;
              geo:geometry ?museumgeo;
              foaf:name ?musname;
              rdfs:comment ?muscomment.
      filter (lang(?musname)='en' && lang(?muscomment)='en')
    }
    UNION
    {
      ?edu a dbo:University;
           geo:geometry ?edugeo;
           foaf:name ?eduname;
           rdfs:comment ?educomment.
      filter (lang(?eduname)='en' && lang(?educomment)='en')
    }
    UNION
    {
      ?wh a dbo:WorldHeritageSite;
            geo:geometry ?whgeo;
            rdfs:label ?whname;
            rdfs:comment ?whcomment.
      filter (lang(?whname)='en' && lang(?whcomment)='en')
    }
  }
```

1.5.35. How can I use the net_meter utility before starting the ingestion to a cluster?

The net_meter utility is a SQL procedure that runs a network test procedure on every host of the cluster. The network test procedure sends a message to every other host of the cluster and waits for the replies from each host. After the last reply is received the action is repeated. This results in a symmetrical load of the network, all points acting as both clients and servers to all other points.

```
net_meter (
  in n_threads int,
  in n_batches int,
  in bytes int,
  in ops_per_batch int)
```

The parameters have the following meaning:

- *n_threads*

  : The number of network test instances started on each host. A value of 4 on a cluster of 4 hosts would result in a total of 16 network test procedures spread over 4 processes.
- *n_batches*

  : The number of message exchanges done by each network test procedure. A message exchange consists of sending one request to every other host of the cluster and of waiting for all to have replied.
- *bytes*

  : The number of bytes sent to each host in each message exchange. The reply from each host has the same number of bytes.
- *ops_per_batch*

  : This causes each message batch to contain several operations. In practice this is a multiplier on the number of bytes.

Example

Assume anuser has run 2 sets of tests on a cluster:

The first one was 1 thread, 1000 batches, 1000 bytes per exchange, 1 operation per batch:

```
SQL> net_meter (1, 1000, 1000, 1);

round_trips     MBps
REAL            REAL

_______________________________________

1245.393315542000254  2.489418401123078

1 Rows. -- 39345 msec.
```

resulted in a measured throughput of 2.5 MBps

The second one was 1 thread, 5000 batches, 10000 bytes per exchange, 1 operation per batch:

```
SQL> net_meter (1, 5000, 10000, 1);

round_trips     MBps
REAL            REAL

_______________________________________________

15915.291672080031181  305.017186586494738

1 Rows. -- 15394 msec.
```

resulted in a measured throughput of 305 MBps.

This indicates that the user's network is slow when sending multiple short messages.

When using the ingestion you should check the:

```
status('cluster');
```

command a few times and check the XX KB/s amount which should be around or above the 2500 mark.

1.5.36. How can I use the LOAD command to import RDF data?

SPARQL INSERT can be done using the LOAD command:

```
SPARQL INSERT INTO <..> { .... } [[FROM ...] { ... }]
```

```
SPARQL LOAD <x> [INTO <y>]

-- <ResourceURL> will be the Graph IRI of the loaded data:
SPARQL LOAD <ResourceURL>
```

Examples

Load from Resource URL

In order to load data from resource URL for ex: http://www.w3.org/People/Berners-Lee/card#i , execute from isql the following command:

```
SQL> SPARQL LOAD <http://www.w3.org/People/Berners-Lee/card#i>;
callret-0
VARCHAR
_______________________________________________________________________________

Load <http://www.w3.org/People/Berners-Lee/card#i> into graph <http://www.w3.org/People/Berners-Lee/card#

1 Rows. -- 703 msec.
SQL>
```

Load from file

1. Create DAV collection which is visible to public, for ex: http://example.com/DAV/tmp
2. Upload to the DAV collection a file for ex. with name listall.rq and with the following content:

```
SPARQL
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX sioc: <http://rdfs.org/sioc/ns#>
SELECT ?x ?p ?o
FROM <http://mygraph.com>
WHERE
  {
    ?x rdf:type sioc:User .
    ?x ?p ?o.
    ?x sioc:id ?id .
    FILTER REGEX(str(?id), "^King")
  }
ORDER BY ?x
```

3. Execute from ISQL the following command:

```
SQL>SPARQL LOAD bif:concat ("http://", bif:registry_get("URIQADefaultHost"), "/DAV/tmp/listall.rq"
callret-0
VARCHAR
_______________________________________________________________________________

Load <http://example.com/DAV/tmp/listall.rq> into graph <http://myNewGraph.com> -- done

1 Rows. -- 321 msec.
```

Directly LOAD triples using ISQL

```
SQL>SPARQL INSERT INTO graph <http://mygraph.com>
{
  <http://myopenlink.net/dataspace/Kingsley#this>
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://rdfs.org/sioc/ns#User> .

  <http://myopenlink.net/dataspace/Kingsley#this>
  <http://rdfs.org/sioc/ns#id>
  <Kingsley> .

  <http://myopenlink.net/dataspace/Caroline#this>
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://rdfs.org/sioc/ns#User> .

  <http://myopenlink.net/dataspace/Caroline#this>
```

```
  <http://rdfs.org/sioc/ns#id>
  <Caroline> .
};
```

1.5.37. How can I delete graphs using stored procedure?

The following script demonstrates the use of custom stored procedures for deleting graph(s). It first creates a table *GRAPHS_TO_DELETE* , into which the names of the graphs to be deleted should be inserted, as follows:

```
use MYUSR;

create procedure GRAPHS_TO_DELETE_SP (in gd_iris any)
{
  declare gd_iri iri_id;
  declare dp, row any;
  result_names (gd_iri);
  dp := dpipe (0, '__I2IDN');
  foreach (varchar iri in GD_IRIS) do
    {
      dpipe_input (dp, iri);
    }
  while (0 <> (row := dpipe_next (dp, 0)))
    {
      result (row[0]);
    }
}
;

drop view GRAPHS_TO_DELETE_VIEW;
create procedure view GRAPHS_TO_DELETE_VIEW as MYUSR.DBA.GRAPHS_TO_DELETE_SP (gd_iris) (gd_iri any);

create procedure DELETE_GRAPHS (in g_iris any)
{
  declare g_iids any;
  if (not isvector (g_iris) and g_iris is not null)
    signal ('22023', '.....', 'The input argument must be an array of strings or null');
  if (not length (g_iris))
    return 0;
  delete from DB.DBA.RDF_QUAD where G in (select * from GRAPHS_TO_DELETE_VIEW where gd_iris = g_iris) opt
  return row_count ();
}
;
```

Finally call the procedure DELETE_GRAPHS to perform the deletion of the specified graphs. Note it does not return a result set and can be called as follows:

```
SQL> select MYUSR.DBA.DELETE_GRAPHS (vector ('g1', 'g2', 'g3'));
```

This will return the number of triples removed from the specified graphs.

Note: the procedure only applies to the cluster so to get IRI IDs via partitioned pipe (DAQ). It is not usable on single.

1.5.38. How can I use SPARUL to add missing triples to a Named Graph?

What?

Use of SPARUL to add missing triples to a Named Graph. For example, an ontology/vocabulary extension.

Why?

A lot of ontologies and vocabularies started life prior to emergence of the Linked Data meme. As a result, many do not include rdfs:isDefinedBy relations (via triples) that associate Classes and Properties in an ontology with the ontology itself, using de-referencable URIs. The downside of this pattern is that Linked Data's follow-your-nose pattern isn't exploitable when viewing these ontologies e.g., when using contemporary Linked Data aware browsers.

How?

If SPARUL privileges are assigned to SPARQL or other accounts associated with SPARQL Endpoint. Or via WebID? protected SPARQL endpoint with SPARUL is granted to SPARQL or specific accounts or WebIDs in a group.

```
INSERT INTO <LocalNamedGraphIRI>
  { ?s rdfs:DefinedBy <LocalOntologyEntityURI>.
    ?o rdfs:isDefinedBy <http://www.w3.org/ns/auth/acl>. }
FROM <ExtSourceNamedGraphIRI>
WHERE
  {
    ?s a ?o
  }
```

Example

1. Load Quad Named Graph via Sponger based query:

```
DEFINE get:soft "replace"
SELECT DISTINCT *
FROM <http://www.w3.org/ns/auth/acl#>
WHERE
  {
    ?s ?p ?o
  }
```

2. Added Triples via SPARUL to Ontology Named Graph:

```
INSERT INTO <http://www.w3.org/ns/auth/acl#>
  {  ?s rdfs:DefinedBy <http://www.w3.org/ns/auth/acl>.
     ?o rdfs:DefinedBy <http://www.w3.org/ns/auth/acl>. }
FROM <http://www.w3.org/ns/auth/acl#>
WHERE
  {
    ?s a ?o
  }
```

3. Via Conductor or Command-line iSQL courtesy of SPASQL execute the following statements:

    a. Remove an existing graph:

```
SPARQL
CLEAR GRAPH <http://www.w3.org/ns/auth/acl/> ;

SPARQL
CLEAR GRAPH <http://www.w3.org/ns/auth/acl> ;

SPARQL
CLEAR GRAPH <http://www.w3.org/ns/auth/acl#> ;
```

    b. Load a new graph:

```
SPARQL
LOAD <http://www.w3.org/ns/auth/acl> ;
```

    c. Add missing rdfs:isDefinedBy triples via SPARUL:

```
SPARQL
INSERT INTO <http://www.w3.org/ns/auth/acl>
  {  ?s rdfs:DefinedBy <http://www.w3.org/ns/auth/acl>.
     ?o rdfs:DefinedBy <http://www.w3.org/ns/auth/acl>. }
FROM <http://www.w3.org/ns/auth/acl>
WHERE
  {
    ?s a ?o
  } ;
```

4. Verification: Access the following url:

http://<cname>/describe/?uri=http://www.w3.org/ns/auth/acl

See Also:
SPARUL -- an Update Language For RDF Graphs

See Also:
Virtuoso Sponger ???

1.5.39. How can I use the SPARQL IF operator for SPARQL-BI endpoint?

Assume a SPARQL query is to be executed against the Virtuoso DBpedia SPARQL endpoint (http://dbpedia.org/sparql) to retrieve the decimal longitude of the "NJ Devils' hometown" with cardinal direction, which determines whether the decimal is positive (in the case of East) or negative (in the case of West).

Virtuoso supports SPARQL-BI, an extended SPARQL 1.0 implementation from before SPARQL 1.1 was ratified, thus the "IF" operator is not currently supported, but an equivalent `bif:either` built-in SQL function does exist enabling an equivalent query to be executed:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
SELECT ?team ( (bif:either( ?ew = 'W', -1, 1)) * (?d + (((?m * 60) + ?s) / 3600.0)) as ?v)
  {
    ?team a dbo:HockeyTeam . ?team rdfs:label 'New Jersey Devils'@en .
    ?team dbp:city ?cityname . ?city rdfs:label ?cityname .
    ?city dbp:longd ?d; dbp:longm ?m; dbp:longs ?s; dbp:longew ?ew .
  }
```

Figure 1.6. SPARQL IF operator for SPARQL-BI endpoint

team	v
http://dbpedia.org/resource/New_Jersey_Devils	74.185

See Also:
Supported SPARQL-BI "define" pragmas

SPARQL-BI ???

1.5.40. How can I handle checkpoint condition?

In general, to control checkpoint in order to bypass client timeouts during long checkpoints when inserting triples into the Virtuoso server, one must disable automatic checkpoint by:

```
SQL> checkpoint_interval (0);
```

and also to make sure the AutoCheckpointLogSize is off. Then can be performed checkpoint whenever the client wants using the 'checkpoint' command.

However the need of cache check is not needed unless instance shows regular errors. By default the cache check is disabled.

Virtuoso offers a new option in the Virtuoso ini file to enable the check of page maps: *PageMapCheck* , 1/0 default 0:

```
-- Virtuoso.ini

...
[Parameters]
...
PageMapCheck  = 0
...
```

Also customer can add *CheckpointSyncMode = 0* in order to disable sync during checkpoint to speed the operations.

1.5.41. How can I incorporate Content Negotiation into RDF bulk loaders?

The examples from below demonstrate how to incorporate Content Negotiation into RDF bulk loaders:

- Using the `DB.DBA.RDF_LOAD_RDFXML` function:

```
DB.DBA.RDF_LOAD_RDFXML (http_get ('http://purl.org/ontology/mo/', null, 'GET', 'Accept: applicatio
```

- Using the `DB.DBA.TTLP` function: The call to http client should be modified to ask for appropriate content type as for ex:

```
DB.DBA.TTLP (http_get ('http://purl.org/ontology/mo/', null, 'GET', 'Accept: text/n3', null, null,
```

1.5.42. Virtuoso Linked Data Deployment In 3 Simple Steps?

Injecting Linked Data into the Web has been a major pain point for those who seek personal, service, or organization-specific variants of DBpedia. Basically, the sequence goes something like this:

1. You encounter DBpedia or the LOD Cloud Pictorial.
2. You look around (typically following your nose from link to link).
3. You attempt to publish your own stuff.
4. You get stuck.

The problems typically take the following form:

1. Functionality confusion about the complementary Name and Address functionality of a single URI abstraction.
2. Terminology confusion due to conflation and over-loading of terms such as Resource, URL, Representation, Document, etc.
3. Inability to find robust tools with which to generate Linked Data from existing data sources such as relational databases, CSV files, XML, Web Services, etc.

To start addressing these problems, here is a simple guide for generating and publishing Linked Data using Virtuoso.

Step 1 - RDF Data Generation

Existing RDF data can be added to the Virtuoso RDF Quad Store via a variety of built-in data loader utilities.

Many options allow you to easily and quickly generate RDF data from other data sources:

- Install the Sponger Bookmarklet for the URIBurner service. Bind this to your own SPARQL-compliant backend RDF database (in this scenario, your local Virtuoso instance), and then perform Network Resource Fetch of some HTTP-accessible resources.
- Convert relational DBMS data to RDF using the Virtuoso Linked Data Views Wizard.
- Starting with CSV files, you can:
    - Place them at an HTTP-accessible location, and use the Virtuoso Sponger to convert them to RDF or;
    - Use the CVS import feature to import their content into Virtuoso's relational data engine; then use the built-in Linked Data Views Wizard as with other RDBMS data.
- Starting from XML files, you can:
    - Use Virtuoso's inbuilt XSLT-Processor for manual XML to RDF/XML transformation or;
    - Leverage the Sponger Cartridge for GRDDL, if there is a transformation service associated with your XML data source, or;
    - Let the Sponger analyze the XML data source and make a best-effort transformation to RDF.

Step 2 - Linked Data Deployment

Install the Faceted Browser VAD package (fct_dav.vad) which delivers the following:

1. Faceted Browser Engine UI
2. Dynamic Hypermedia Resource Generator:
    - delivers descriptor resources for every entity (data object) in the Native or Virtual Quad Stores
    - supports a broad array of output formats, including HTML+RDFa, RDF/XML, N3/Turtle, NTriples, RDF-JSON, OData+Atom, and OData+JSON.

Step 3 - Linked Data Consumption & Exploitation

Three simple steps allow you, your enterprise, and your customers to consume and exploit your newly deployed Linked Data --

1. Load a page like this in your browser: http://<cname>[:<port>]/describe/?uri=<entity-uri>
    - <cname>[:<port>] gets replaced by the host and port of your Virtuoso instance
    - <entity-uri> gets replaced by the URI you want to see described -- for instance, the URI of one of the resources you let the Sponger handle.
2. Follow the links presented in the descriptor page.
3. If you ever see a blank page with a hyperlink subject name in the About: section at the top of the page, simply add the parameter "&sp=1" to the URL in the browser's Address box, and hit [ENTER]. This will result in an "on the fly" resource retrieval, transformation, and descriptor page generation.
4. Use the navigator controls to page up and down the data associated with the "in scope" resource descriptor.

1.5.43. What are the differences between create, drop, clear and delete Graph?

In Virtuoso it does not matter whether *CREATE GRAPH* and *DROP GRAPH* are called or not. Their purpose is to provide compatibility with the original SPARUL that was designed for Jena. Some Jena triple stores require explicit creation of each graph (like CREATE TABLE in SQL), they report errors if one tries to create a graph twice and so on.

For Virtuoso, a new graph is not an important system event, it has single quad store shared by all graphs. When a graph is made by *CREATE GRAPH* , its name is placed in an auxiliary table that is used solely to signal appropriate errors on CREATE graph that is CREATE-d already or on DROP of a missing graph; this table is not used in any way in SPARQL or other subsystems. In a perfect world, smart development tools will query that table to give hints to a programmer regarding available graphs, but the reality is not so perfect.

What's more important is a difference between *DELETE FROM g { ?s ?p ?o } FROM g WHERE { ?s ?p ?o }* and *CLEAR GRAPH g* , as both will delete all triples from the specified graph <g> with equal speed, but *CLEAR GRAPH* will also delete free-text index data about occurrences of literals in this specific graph. So *CLEAR GRAPH* will make the database slightly more compact and the text search slightly faster. (Naturally, *DROP GRAPH* makes *CLEAR GRAPH* inside, not just *DELETE FROM ...*)

See Also:
RDF Insert Methods in Virtuoso

1.5.44. How can I perform search for predicate values?

What?

Creation of a stored procedure that enables users to find properties based on their string based token patterns.

Why?

When working with datasets from disparate datasources in a common named graph, there are times when you seek to scope sparql or spasql queries to specific property IRI/URI patterns without embarking upon inefficient regex heuristics.

What?

Make a stored procedure for querying against the main quad store table (rdf_quad). Surface the procedure as a magic predicate using "bif:" prefix. To find all the properties whose predicates start with "http://www.openlinksw.com/", a Virtuoso/PL procedure can be used as listed below:

```
SQL> create procedure PREDICATES_OF_IRI_PATH (
  in path varchar,
  in dump_iri_ids integer := 0)
{
  declare PRED_IRI varchar;
  declare PRED_IRI_ID IRI_ID;
  declare path_head_len integer;

  if (dump_iri_ids)
    result_names (PRED_IRI_ID);
  else
```

```
  result_names (PRED_IRI);

for ( SELECT RP_NAME, RP_ID
      FROM RDF_PREFIX
      WHERE (RP_NAME >= path) AND (RP_NAME < path || chr(255)) ) do
  {
    declare fourbytes varchar;
    fourbytes := '----';
    fourbytes[0] := bit_shift (RP_ID, -24);
    fourbytes[1] := bit_and (bit_shift (RP_ID, -16), 255);
    fourbytes[2] := bit_and (bit_shift (RP_ID, -8), 255);
    fourbytes[3] := bit_and (RP_ID, 255);

    for ( SELECT RI_NAME, RI_ID from RDF_IRI
          WHERE (RI_NAME >= fourbytes) AND (RI_NAME < fourbytes || chr(255)) ) do
      {
        if (exists (SELECT TOP 1 1 FROM RDF_QUAD WHERE P=RI_ID))
          result (case when (dump_iri_ids) then RI_ID else RP_NAME || subseq (RI_NAME, 4) end);
      }
  }

for ( path_head_len := length (path)-1; path_head_len >= 0; path_head_len := path_head_len - 1)
  {
    for ( SELECT RP_NAME, RP_ID from RDF_PREFIX
          WHERE RP_NAME = subseq (path, 0, path_head_len) ) do
      {
        declare fourbytes varchar;
        fourbytes := '----';
        fourbytes[0] := bit_shift (RP_ID, -24);
        fourbytes[1] := bit_and (bit_shift (RP_ID, -16), 255);
        fourbytes[2] := bit_and (bit_shift (RP_ID, -8), 255);
        fourbytes[3] := bit_and (RP_ID, 255);

        for ( SELECT RI_NAME, RI_ID from RDF_IRI
              WHERE (RI_NAME >= fourbytes || subseq (path, path_head_len))
              AND (RI_NAME < fourbytes || subseq (path, path_head_len) || chr(255)) ) do
          {
            if (exists (SELECT TOP 1 1 FROM RDF_QUAD WHERE P=RI_ID))
              result (case when (dump_iri_ids) then RI_ID else RP_NAME || subseq (RI_NAME, 4) end);
          }
      }
  }
}
;

Done. -- 16 msec.
```

Example Usage

1. Execute:

```
set echo on;
```

2. Collect all predicates that start with:

```
-- http://www.openlinksw.com/
SQL>PREDICATES_OF_IRI_PATH ('http://www.openlinksw.com/', 1);
VARCHAR
_____________________________________________________________________________

#i351
#i159
#i10
#i8
...

-- http://www.openlinksw.com/schemas/virtrdf
SQL>PREDICATES_OF_IRI_PATH ('http://www.openlinksw.com/schemas/virtrdf', 1);
PRED_IRI_ID
VARCHAR
_____________________________________________________________________________

#i159
```

```
#i10
#i8
#i6
#i160
#i269
#i278
#i275

-- http://www.openlinksw.com/schemas/virtrdf#
SQL>PREDICATES_OF_IRI_PATH ('http://www.openlinksw.com/schemas/virtrdf#',1);
PRED_IRI_ID
VARCHAR
_______________________________________________________________________________

#i159
#i10
#i8
#i6
#i160
#i269
#i278
...

-- http://www.openlinksw.com/schemas/virtrdf#i
SQL>PREDICATES_OF_IRI_PATH ('http://www.openlinksw.com/schemas/virtrdf#i',1);
PRED_IRI_ID
VARCHAR
_______________________________________________________________________________

#i159
#i10
#i8

-- other
SQL>PREDICATES_OF_IRI_PATH ('no://such :)', 1);
0 Rows. -- 0 msec.
```

3. Show all predicates that start with:

```
-- http://www.openlinksw.com/
SQL>PREDICATES_OF_IRI_PATH ('http://www.openlinksw.com/');
PRED_IRI
VARCHAR
_______________________________________________________________________________

http://www.openlinksw.com/schemas/DAV#ownerUser
http://www.openlinksw.com/schemas/virtrdf#inheritFrom
http://www.openlinksw.com/schemas/virtrdf#isSpecialPredicate
http://www.openlinksw.com/schemas/virtrdf#item
http://www.openlinksw.com/schemas/virtrdf#loadAs
http://www.openlinksw.com/schemas/virtrdf#noInherit
http://www.openlinksw.com/schemas/virtrdf#qmGraphMap
http://www.openlinksw.com/schemas/virtrdf#qmMatchingFlags
http://www.openlinksw.com/schemas/virtrdf#qmObjectMap
http://www.openlinksw.com/schemas/virtrdf#qmPredicateMap
http://www.openlinksw.com/schemas/virtrdf#qmSubjectMap
...

-- http://www.openlinksw.com/schemas/virtrdf
SQL>PREDICATES_OF_IRI_PATH ('http://www.openlinksw.com/schemas/virtrdf');
PRED_IRI
VARCHAR
_______________________________________________________________________________

http://www.openlinksw.com/schemas/virtrdf#inheritFrom
http://www.openlinksw.com/schemas/virtrdf#isSpecialPredicate
http://www.openlinksw.com/schemas/virtrdf#item
http://www.openlinksw.com/schemas/virtrdf#loadAs
http://www.openlinksw.com/schemas/virtrdf#noInherit
http://www.openlinksw.com/schemas/virtrdf#qmGraphMap
http://www.openlinksw.com/schemas/virtrdf#qmMatchingFlags
http://www.openlinksw.com/schemas/virtrdf#qmObjectMap
http://www.openlinksw.com/schemas/virtrdf#qmPredicateMap
```

```
http://www.openlinksw.com/schemas/virtrdf#qmSubjectMap
http://www.openlinksw.com/schemas/virtrdf#qmTableName
...

-- http://www.openlinksw.com/schemas/virtrdf#
SQL>PREDICATES_OF_IRI_PATH ('http://www.openlinksw.com/schemas/virtrdf#');
PRED_IRI
VARCHAR
_______________________________________________________________________________

http://www.openlinksw.com/schemas/virtrdf#inheritFrom
http://www.openlinksw.com/schemas/virtrdf#isSpecialPredicate
http://www.openlinksw.com/schemas/virtrdf#item
http://www.openlinksw.com/schemas/virtrdf#loadAs
http://www.openlinksw.com/schemas/virtrdf#noInherit
http://www.openlinksw.com/schemas/virtrdf#qmGraphMap
http://www.openlinksw.com/schemas/virtrdf#qmMatchingFlags
http://www.openlinksw.com/schemas/virtrdf#qmObjectMap
http://www.openlinksw.com/schemas/virtrdf#qmPredicateMap
http://www.openlinksw.com/schemas/virtrdf#qmSubjectMap
http://www.openlinksw.com/schemas/virtrdf#qmTableName
http://www.openlinksw.com/schemas/virtrdf#qmf01blankOfShortTmpl
http://www.openlinksw.com/schemas/virtrdf#qmf01uriOfShortTmpl
http://www.openlinksw.com/schemas/virtrdf#qmfBoolOfShortTmpl
http://www.openlinksw.com/schemas/virtrdf#qmfBoolTmpl
...

-- http://www.openlinksw.com/schemas/virtrdf#i
SQL>PREDICATES_OF_IRI_PATH ('http://www.openlinksw.com/schemas/virtrdf#i');
PRED_IRI
VARCHAR
_______________________________________________________________________________

http://www.openlinksw.com/schemas/virtrdf#inheritFrom
http://www.openlinksw.com/schemas/virtrdf#isSpecialPredicate
http://www.openlinksw.com/schemas/virtrdf#item

3 Rows. -- 15 msec.

-- other
SQL>PREDICATES_OF_IRI_PATH ('no://such :)');
PRED_IRI
VARCHAR
_______________________________________________________________________________

0 Rows. -- 15 msec.
```

If you want to use the procedure's output inside SPARQL queries, it can be wrapped by a procedure view and it in turn can be used in an Linked Data View but it is redundant for most applications.

For typical "almost static" data, it is more practical to write a procedure that will store all found predicates in some dedicated "dictionary" graph and then use the graph as usual.

1.5.45. How can I use INSERT via CONSTRUCT Statements?

You can write an ordinary CONSTRUCT statement, ensure that it generates the triples intended to be added, and then replace the leading CONSTRUCT keyword with the INSERT INTO phrase.

Example:

1. Assume new triples need to be added for the equivalentClass:

```
CONSTRUCT
  {
    ?s <http://www.w3.org/2002/07/owl#equivalentClass> `iri (bif:replace(?o,'http://schema.rdfs.or
   }
FROM <http://www.openlinksw.com/schemas/rdfs>
WHERE
  {
    ?s <http://www.w3.org/2002/07/owl#equivalentClass> ?o
```

```
        };
```

2. Execute the CONSTRUCT query from the htp://cname/sparql SPARQL endpoint.
3. View the generated triples to ensure they are correct.
4. Replace CONSTRUCT with INSERT INTO:

```
SPARQL INSERT INTO <http://www.openlinksw.com/schemas/rdfs>
  {
    ?s <http://www.w3.org/2002/07/owl#equivalentClass> `iri (bif:replace(?o,'http://schema.rdfs.or
   }
FROM <http://www.openlinksw.com/schemas/rdfs>
WHERE
  {
    ?s <http://www.w3.org/2002/07/owl#equivalentClass> ?o
  } ;
```

1.5.46. How to clear graphs which are related to empty graphs?

The following example demonstrates how to remove graphs which are related to empty graphs:

```
PREFIX nrl:<http://www.semanticdesktop.org/ontologies/2007/08/15/nrl#>
SELECT ( bif:exec(bif:sprintf("SPARQL CLEAR GRAPH<%s>", str(?mg))))
WHERE
  {
    ?mg nrl:coreGraphMetadataFor ?g .
    FILTER(?g in ( <urn:nepomuk:local:8a9e692a> )) .
    FILTER ( !bif:exists((SELECT (1) WHERE { GRAPH ?g { ?s ?p ?o . } . })) ) .
  }
```

1.5.47. How can I use sub-queries to enable literal values based joins?

What?

Use of subqueries to enable literal values based joins.

Why?

Sophisticated access to literal values via subqueries provides powerful mechanism for enhancing sparql graph patterns via dynamic literal value generation.

How?

Use select list variables from subqueries to produce literal object values in sparql graph patterns.

Example

Assume in the following query, where should be used a sub-query to replace ?app:

```
SELECT DISTINCT ?r
WHERE
  {
    graph ?g
      {
        ?r nie:url ?url .
      }  .
      ?g nao:maintainedBy ?app .
      ?app nao:identifier "nepomukindexer" .
  }
```

If it is not sure that ?app is the only, for e.g., the triple ?app nao:identifier "nepomukindexer" can appear in more than one graph, then the query should be changed to:

```
SELECT DISTINCT ?r
WHERE
  {
    graph ?g
      {
        ?r nie:url ?url .
```

```
        }  .
        ?g nao:maintainedBy ?app .
        FILTER (?app = (SELECT ?a WHERE { ?a nao:identifier "nepomukindexer" }))
}
```

or even shorter:

```
SELECT DISTINCT ?r
WHERE
  {
   graph ?g
     {
       ?r nie:url ?url .
     }  .
   ?g nao:maintainedBy `(SELECT ?a WHERE { ?a nao:identifier "nepomukindexer" })` .
 }
```

1.5.48. How can I execute query with labels preference order?

The way to prefer one label to the other can be done as follows:

- Have a procedure which returns preference order:

```
create procedure lbl_order (in p any)
{
  declare r int;
  r := vector (
  'http://www.w3.org/2000/01/rdf-schema#label',
  'http://xmlns.com/foaf/0.1/name',
  'http://purl.org/dc/elements/1.1/title',
  'http://purl.org/dc/terms/title',
  'http://xmlns.com/foaf/0.1/nick',
  'http://usefulinc.com/ns/doap#name',
  'http://rdf.data-vocabulary.org/name',
  'http://www.w3.org/2002/12/cal/ical#summary',
  'http://aims.fao.org/aos/geopolitical.owl#nameListEN',
  'http://s.opencalais.com/1/pred/name',
  'http://www.crunchbase.com/source_description',
  'http://dbpedia.org/property/name',
  'http://www.geonames.org/ontology#name',
  'http://purl.org/ontology/bibo/shortTitle',
  'http://www.w3.org/1999/02/22-rdf-syntax-ns#value',
  'http://xmlns.com/foaf/0.1/accountName',
  'http://www.w3.org/2004/02/skos/core#prefLabel',
  'http://rdf.freebase.com/ns/type.object.name',
  'http://s.opencalais.com/1/pred/name',
  'http://www.w3.org/2008/05/skos#prefLabel',
  'http://www.w3.org/2002/12/cal/icaltzd#summary',
  'http://rdf.data-vocabulary.org/name',
  'http://rdf.freebase.com/ns/common.topic.alias',
  'http://opengraphprotocol.org/schema/title',
  'http://rdf.alchemyapi.com/rdf/v1/s/aapi-schema.rdf#Name',
  'http://poolparty.punkt.at/demozone/ont#title'
   );
  if (isiri_id (p))
    p := id_to_iri (p);
  r := position (p, r);
  if (r = 0)
    return 100;
  return r;
}
;
```

- Execute the following query:

```
SPARQL
DEFINE input:inference "facets"
SELECT ?o
WHERE
  {
    <http://uriburner.com/about/id/entity/http/www.linkedin.com/in/kidehen#CV_mfrTl4s6Jy> virtrdf:
       ?p ?o .
```

```
  }
ORDER BY (sql:lbl_order (?p));
```

- Or execute the following query using sql:

```
SELECT __ro2sq (O), lbl_order (P)
FROM RDF_QUAD table option (with ''facets'')
WHERE G = __i2id (?)
  AND S = __i2id (?)
  AND P = __i2id (''http://www.openlinksw.com/schemas/virtrdf#label'', 0)
ORDER BY 2
```

1.5.49. How can I get object datatype?

To get object datatype should be used the internal Virtuoso/PL function `DB.DBA.RDF_DATATYPE_OF_OBJ` , visible in SPARQL as sql:RDF_DATATYPE_OF_OBJ.

Suppose the following scenario:

```
# Explicit typecast (insert)
SQL> sparql insert into <test_datatype> { <a> <string> "string 1"^^xsd:string . };
callret-0
VARCHAR
_______________________________________________________________________________

Insert into <test_datatype>, 1 (or less) triples -- done

1 Rows. -- 94 msec.

#Not explicit typecast (insert)
SQL> sparql insert into <test_datatype> { <a> <string> "string 2". };
callret-0
VARCHAR
_______________________________________________________________________________

Insert into <test_datatype>, 1 (or less) triples -- done

1 Rows. -- 16 msec.

SQL> SPARQL
SELECT ?o (iri(sql:RDF_DATATYPE_OF_OBJ(?o, 'untyped!')))
FROM <test_datatype> { <a> <string> ?o} ;
o                      callret-1
VARCHAR                VARCHAR
_______________________________________________________________________________

string 1               http://www.w3.org/2001/XMLSchema#string
string 2               untyped!

2 Rows. -- 16 msec.
SQL>
```

1.5.50. How Can I Backup and Restore individual table(s) and individual index(s)?

See more details here .

1.5.51. What bif:contains free-text options can I use?

Virtuoso supports the following free-text options for bif:contains:

1. *OFFBAND*

   : See description for this free-text option in this section .

   - Note: it is useful only if data comers via an Linked Data View and the source relational table uses this trick;
2. *SCORE*

: This free-text option is calculated as described in this section :

```
SQL>SPARQL
SELECT ?s1 as ?c1, ?sc, ?rank
WHERE
  {
    {
      {
        SELECT ?s1, (?sc * 3e-1) as ?sc, ?o1,
          (sql:rnk_scale (<LONG::IRI_RANK> (?s1))) as ?rank
        WHERE
          {
            ?s1 ?s1textp ?o1 .
            ?o1 bif:contains  '"CNET"'  option (score ?sc)  .
          }
        ORDER BY DESC (?sc * 3e-1 + sql:rnk_scale (<LONG::IRI_RANK> (?s1)))
        LIMIT 20
        OFFSET 0
      }
    }
  };

c1                                                sc      rank
_____________________________________________________________________
http://www.mixx.com/stories/45003360/justi...   3           5.881291583872891e-14
http://www.mixx.com/stories/45099313/bing_...   2.7     5.881291583872891e-14
http://dbpedia.org/resource/CBS_Interactive       1.5     5.881291583872891e-14
http://dbpedia.org/resource/CBS_Interactive       1.5     5.881291583872891e-14

4 Rows. -- 1 msec.
```

3. *SCORE_LIMIT*

: This free-text option works as it is in plain SQL free-text. See more details :

```
SQL> SPARQL
SELECT ?s ?sc
WHERE
  {
    ?s ?p ?o .
    ?o bif:contains "tree" OPTION (score ?sc , score_limit 20)
  };

s                                                                   sc
VARCHAR                                                             INTEGER
_____________________________________________________________________________

http://www.openlinksw.com/virtrdf-data-formats#default              24
http://www.openlinksw.com/virtrdf-data-formats#default              24
http://www.openlinksw.com/virtrdf-data-formats#sql-longvarbinary    21
http://www.openlinksw.com/virtrdf-data-formats#sql-varchar-dt       20
http://www.openlinksw.com/virtrdf-data-formats#sql-nvarchar-dt      20
http://www.openlinksw.com/virtrdf-data-formats#sql-varchar-lang     20
http://www.openlinksw.com/virtrdf-data-formats#sql-nvarchar-lang    20

7 Rows. -- 2 msec.
```

1.5.52. What SPARQL Endpoint Protection Methods can I use?

Virtuoso supports the following SPARQL Endpoint protection methods:

1. Secure SPARQL Endpoint Guide using WebID Protocol
2. Secure SPARQL Endpoint Guide via OAuth
3. Secure SPARQL Endpoint Guide via SQL Accounts -- usage path digest authentication

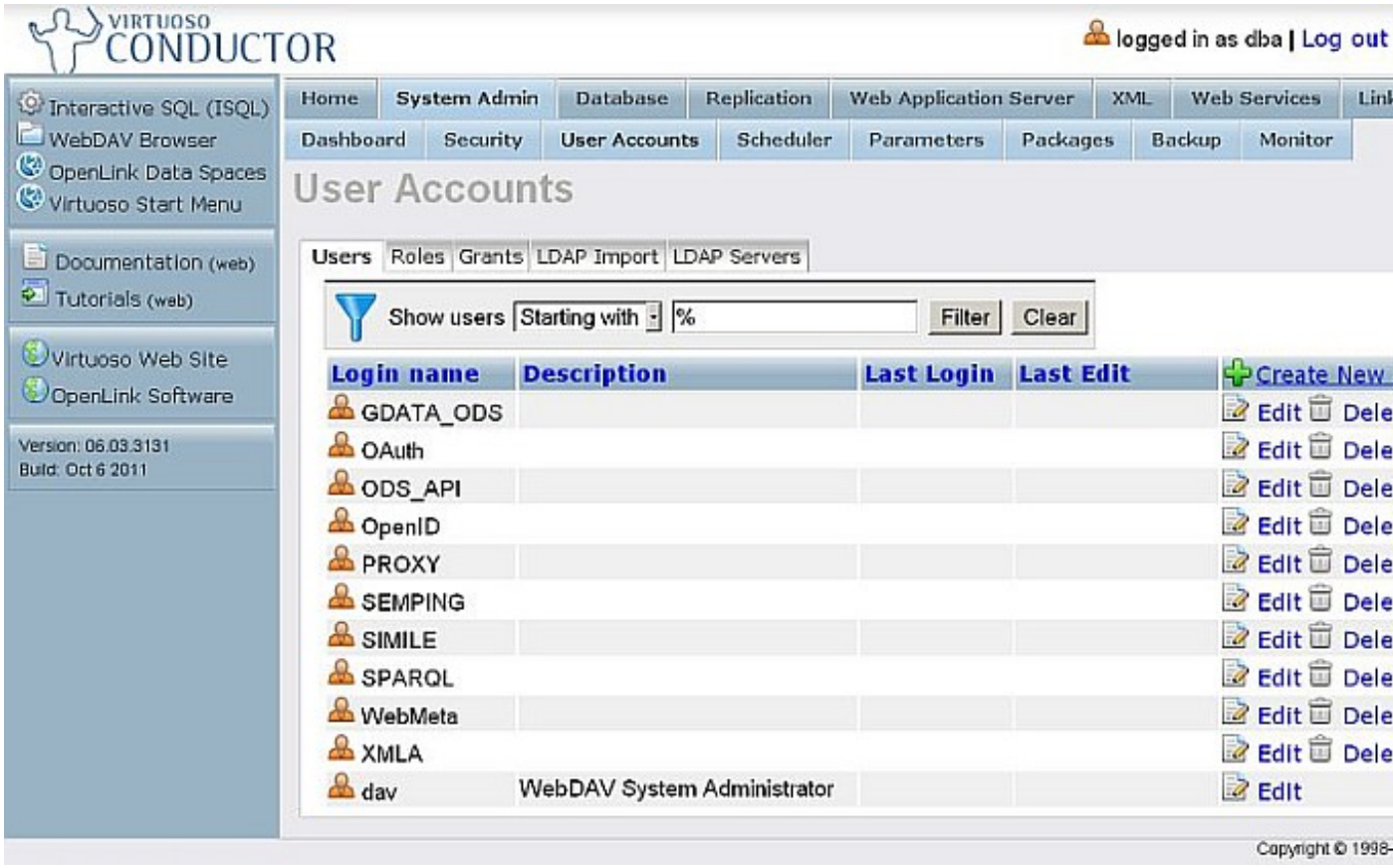
1.5.53. How do I assign SPARQL role to SQL user?

This section a sample scenario how to assign SPARQL (for ex. SPARQL_SELECT) role to Virtuoso SQL user:

1. Go to http://cname/conductor .

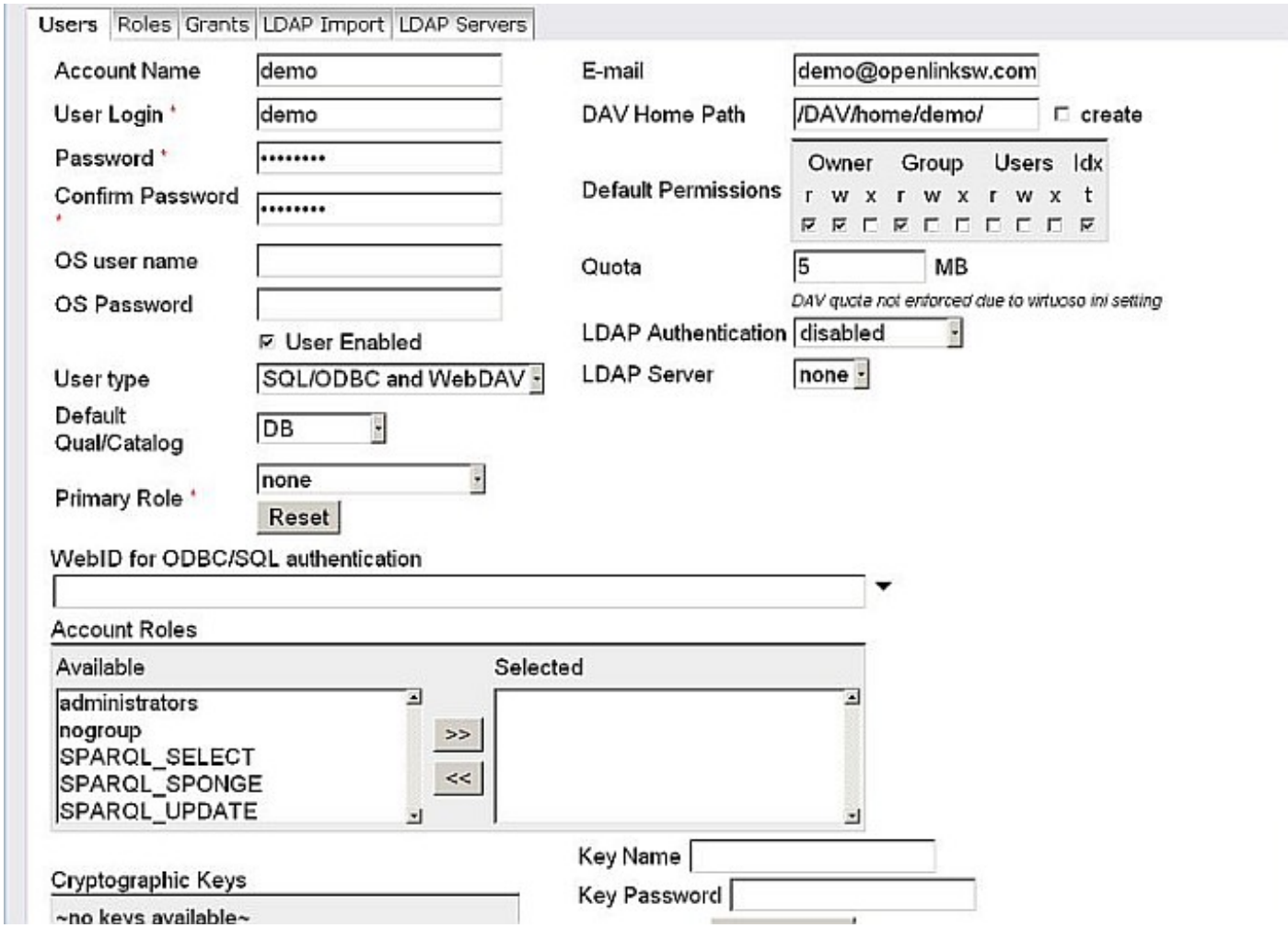
2. Enter dba credentials.
3. Go to System Admin -> User Accounts:

Figure 1.7. Assign SPARQL Role to SQL User



4. Click "Edit" for a given user from the very last right column:

Figure 1.8. Assign SPARQL Role to SQL User

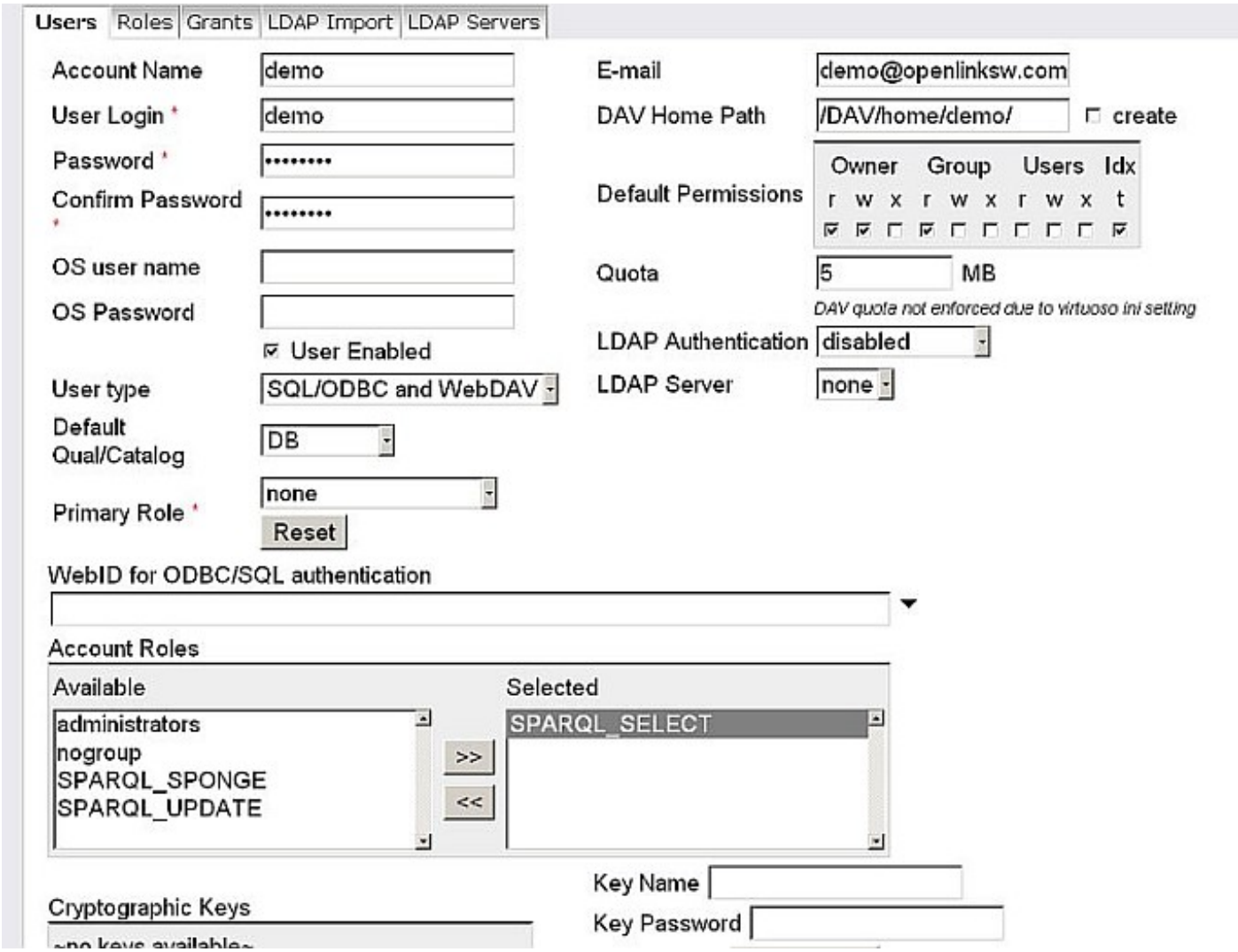


5. From "Accounts Roles" drop-down list select a SPARQL Role, for ex.

   SPARQL_SELECT

   and click the ">>" button:

   Figure 1.9. Assign SPARQL Role to SQL User



6. Click "Save".

1.5.54. How Do I Gecode Data?

What?

Automatic geocoding of data in quad store.

Why?

Enable exploitation of SPARQL-GEO for geospatial oriented queries.

How?

To gecode data one should call the `rdf_geo_fill` API from Conductor or iSQL:

```
SQL> rdf_geo_fill ();

Done. -- 282 msec.
```

SPARQL-GEO sample queries

- All Educational Institutions within 10km of Oxford, UK :

```
SELECT DISTINCT ?thing AS ?uri ?thingLabel AS ?name ?date AS ?established ?lat ?long WHERE
  {
    <http://dbpedia.org/resource/Oxford> geo:geometry ?sourcegeo .
    ?resource geo:geometry ?matchgeo .
```

```
    ?resource geo:lat ?lat .
    ?resource geo:long ?long .
    FILTER ( bif:st_intersects ( ?matchgeo, ?sourcegeo, 5 ) ) .
    ?thing ?somelink ?resource .
    ?thing <http://dbpedia.org/property/established> ?date .
    ?thing rdfs:label ?thingLabel .
    FILTER ( lang ( ?thingLabel ) = "en" )
  }
```

- Things within close proximity of London:

```
SELECT DISTINCT ?resource ?label ?location WHERE
  {
    <http://dbpedia.org/resource/London> geo:geometry ?sourcegeo .
    ?resource geo:geometry ?location ; rdfs:label ?label .
    FILTER ( bif:st_intersects ( ?location, ?sourcegeo, 20 ) ) .
    FILTER ( lang ( ?label ) = "en" )
  }
```

See Also:

Tip

- `rdf_geo_fill`
- Creating Geometries From RDF Data
- Programmatic Manipulation of Geometries in RDF
- Using Geometries With Existing Databases
- OpenLink Billion Triple Demo queries

1.5.55. How Can I Delete a Specific Triple Across Graphs?

Suppose the following situation:

- There is a specific triple somewhere in a massive dataset in a Virtuoso DBMS.
- There are too many possible named graphs associated with the pattern so

  SPARQL DELETE

  (which requires

  FROM

  i.e., Named Graph scoping) isn't adequate

What is the solution?

Using SQL you can execute the following:

```
SQL> SPARQL
DELETE FROM rdf_quad
WHERE s = iri_to_id ('http://linkeddata.uriburner.com/about/id/entity/http/twitter.com/kidehen')
  AND o = iri_to_id ('http://purl.org/ontology/bibo/Document')
  AND p = iri_to_id ('http://www.w3.org/1999/02/22-rdf-syntax-ns#type');
```

1.5.56. How Do I Use NOT EXISTS in SPARQL Query?

Virtuoso supports "NOT EXISTS" SPARQL 1.1 feature. For example:

```
SQL> SPARQL
SELECT COUNT(*)
WHERE
  {
    ?s ?p "Novosibirsk"  FILTER NOT EXISTS { ?s ?p "&#1053;&#1086;&#1074;&#1086;&#1089;&#1080;&#1073;&#10
  }

callret-0
```

```
INTEGER
313
No. of rows in result: 1
```

This query is equivalent to the following query:

```
SQL> SPARQL
SELECT COUNT(*)
WHERE
  {
    { ?s ?p "Novosibirsk" } MINUS { ?s ?p "&#1053;&#1086;&#1074;&#1086;&#1089;&#1080;&#1073;&#1080;&#1088

callret-0
INTEGER
 313
No. of rows in result: 1
```

1.5.57. How Do I Use MINUS in SPARQL Query?

Virtuoso supports "MINUS" SPARQL 1.1 feature. For example:

```
SQL> SPARQL
SELECT COUNT(*)
WHERE
  {
    { ?s ?p "Novosibirsk" } MINUS { ?s ?p "&#1053;&#1086;&#1074;&#1086;&#1089;&#1080;&#1073;&#1080;&#1088

callret-0
INTEGER
 313
No. of rows in result: 1
```

This query is equivalent to the following query:

```
SQL> SPARQL
SELECT COUNT(*)
WHERE
  {
    ?s ?p "Novosibirsk"  FILTER NOT EXISTS { ?s ?p "&#1053;&#1086;&#1074;&#1086;&#1089;&#1080;&#1073;&#10
  }

callret-0
INTEGER
313
No. of rows in result: 1
```

1.5.58. How Do I Use Transitive SPARQL Query Options and Exploit Inference Rules?

This section contains subClassOf oriented subsumption examples and examples for Transitive Options e.g., Scope the reasoning to layers in the hierarchy.

Transitive query options examples usage

- Transitivity with sameAs and graph
- Transitive Closure via Graph Path Expressions: TBox
- Transitive Closure via Graph Path Expressions: ABox
- Inference Rule using transitive properties from SKOS vocabulary

subClassOf examples usage

- Processing Large Hierarchies in SPARQL
- Example for TBox Subsumption
- Example for Receptors
- Inference Rule using transitive properties from SKOS vocabulary

1.5.59. What is the difference between the functions SAMPLE, GROUP_CONCAT and GROUP_DIGEST?

This section demonstrates the differences between the functions `SAMPLE` , `GROUP_CONCAT` and `GROUP_DIGEST` .

Assume the following query that should get all ?assets as a list with delimiters:

```
SPARQL SELECT ?view ?path (sql:GROUP_CONCAT (?asset, ' ')) as ?asset_list
FROM <http://mygraph.com>
WHERE
 {
   ?view <viewPath> ?path ;
     <viewContent> ?asset ;
     <viewType>  'phyview'.
 }
;
```

This method is not universal, because conversion to strings will eliminate the difference between strings and IRIs and there should be some delimiter that never appears in values of ?asset. In addition, the query may fail with "row too long" error if values of ?asset are lengthy and/or numerous enough. It is also possible the query not work completely free from duplicates if more than one list is desired. E.g.:

```
SPARQL
SELECT ?view (sql:GROUP_CONCAT (?path, ' ')) as ?path_list
  (sql:GROUP_CONCAT (?asset, ' ')) as ?asset_list
FROM <http://mygraph.com>
WHERE
  {
    ?view <viewPath> ?path ;
      <viewContent> ?asset ;
      <viewType> 'phyview' .
  }
```

will not contain duplicates in lists only if either ?path or ?asset is unique for every found ?view; but if it's so unique then there's no need in the corresponding sql:GROUP_CONCAT().

If there are many values per property but it's enough to return any single value and ignore the rest then use sql:`SAMPLE ()` function instead of sql:`GROUP_CONCAT ()` .

If there are many values per property and it's better to show more than one value but "row too long" error happens, then the sql:`GROUP_DIGEST ()` function can be used:

```
SPARQL
SELECT ?view (sql:GROUP_DIGEST (?path, ' ', 1000, 1)) as ?path_list
  (sql:GROUP_DIGEST (?asset, ' ', 1000, 1)) as ?asset_list
FROM <http://mygraph.com>
WHERE
  {
    ?view <viewPath> ?path ;
      <viewContent> ?asset ;
      <viewType> 'phyview' .
  }
```

1.5.60. How Do I use CONSTRUCT with objects which are value of aggregate function?

The following section demonstrates usage example of CONSTRUCT Query with object which is a value of the aggregate function COUNT:

1. Assume following SPARQL SELECT Query:

```
SELECT DISTINCT ?category COUNT(?category) AS ?count
WHERE
  {
    ?s<http://purl.org/dc/terms/subject> ?category
    FILTER(?s =<http://dbpedia.org/resource/Higgs_boson> || ?s =<http://dbpedia.org/resource/Gluon
  }
```

2. To present it as CONSTRUCT Query, one should put "SELECT DISTINCT" into the CONSTRUCT's WHERE clause as a subquery. So for example it could be:

```
CONSTRUCT
  {
    ?category a<http://www.w3.org/2004/02/skos/core#Concept> .
    ?category<http://www.turnguard.com/virtuoso/aggregates#count>  ?count
  }
WHERE
  {
    {
      SELECT DISTINCT ?category COUNT(?category) AS ?count
      WHERE
        {
          ?s <http://purl.org/dc/terms/subject> ?category
          FILTER(?s =<http://dbpedia.org/resource/Higgs_boson> || ?s =<http://dbpedia.org/resource
        }
    }
  }
```

1.5.61. How Do I Clean Up Errant Data using SPARQL Update Language?

What?

Cleaning up errant data using SPARQL Update Language.

Why?

All data endeavors involve varying degrees of prospective and retrospective error correction.

How?

Given a triple in a Virtuoso Quad store that contains an errant URI e.g. one that accidentally contains spaces, here is a SPARQL example that showcases how to delete said triple using a built-in function:

```
DELETE FROM <your_graph>
  {
    `iri("http://isbsg.clearbluewater.com.au/wsf/datasets/378/Capers Jones_107_12122011081257")`
     a <http://isbsg.org/ontology/data/Dataset>
  };
```

1.5.62. How to Use SPARQL to add missing isDefinedBy relations to an Ontology?

What?

Using SPARQL to add missing isDefinedBy relations to an Ontology.

Why?

isDefinedBy relations make Ontologies (TBox) more navigable using follow-your-nose pattern. This also makes ABox instance data more discoverable.

How?

Use SPARQL to generate relations that associate Classes and Properties the Ontology that describes them.

Example Using the Ontology for vCards

In this example we will use:

- Ontology Document URL: http://www.w3.org/2006/vcard/ns ;
- Ontology URI: http://www.w3.org/2006/vcard/ns# ;
- A local Named Graph IRI to host SPARQL Update (SPARUL) the new relations.

```
## Uncomment line below if using Virtuoso and executing SPARQL via iSQL
## or via an ODBC, JDBC, ADO.NET connection

## SPARQL

## Uncomment line (a SPARQL pragma) below if using Virtuoso and there
## isn't a local Named Graph holding triples retrieved from the Ontology URL

## DEFINE get:soft "add"

INSERT INTO <urn:data:qos:vcard>
  { ?s rdfs:isDefinedBy <http://www.w3.org/2006/vcard/ns#> }
FROM <http://www.w3.org/2006/vcard/ns>
WHERE
  {
    ?s a ?o
  }
```

Example Using the Recorded Media Ontology

In this example we will use:

- Ontology Document URL: http://www.w3.org/ns/ma-ont ;
- Ontology URI: http://www.w3.org/ns/ma-ont# ;
- A local Named Graph IRI to host SPARQL Update (SPARUL) the new relations.

```
SPARQL
INSERT INTO <urn:data:qos:ma-ont>
  { ?s rdfs:isDefinedBy <http://www.w3.org/ns/ma-ont#> }
FROM <http://www.w3.org/ns/ma-ont>
WHERE
  {
    ?s a ?o
  }
```

1.5.63. How Can I execute load of sql dump from jdbc?

LOAD" is not a Virtuoso SQL command, it's an ISQL one. The ISQL checks the command to be executed, whether it's of special "isql"-ish syntax or not, it executes commands it understand and send the rest to the server, unchanged.

There's no way to mimic ISQL's behavior right on the server, without ISQL. However some files can be loaded via the `DB.DBA.VAD_LOAD_SQL_FILE` function.

1.5.64. How Can I Use MODIFY to update triples?

The following queries are equivalent:

```
MODIFY <http://test.com/>
DELETE {?s ?p ?o}
INSERT {?s_new ?p ?o}
FROM <http://test.com/>
WHERE
{
  {
    SELECT iri(bif:replace(str(?s),"http://test.com/link","http://test.com/extra/link" ) )
      AS ?s_new ?s ?p ?o
    WHERE
      {
        ?s ?p ?o FILTER (regex (str(?s), "http://test.com/link"))
      }
  }
}

MODIFY <http://test.com/>
DELETE {?s ?p ?o}
INSERT {`iri(?s_new)` ?p ?o}
FROM <http://test.com/>
WHERE
  {
```

```
    {
      SELECT bif:replace(str(?s),"http://test.com/link","http://test.com/extra/link" )
        AS ?s_new ?s ?p ?o
      WHERE
        {
          ?s ?p ?o FILTER (regex (str(?s), "http://test.com/link"))
        }
    }
  }

MODIFY <http://test.com/>
DELETE {?s ?p ?o}
INSERT
  { `iri(bif:replace(str(?s),"http://test.com/link","http://test.com/extra/link" ))`  ?p ?o }
FROM <http://test.com/>
WHERE
  {
    {
      ?s ?p ?o FILTER (regex (str(?s), "http://test.com/link"))
    }
  }
}
```

1.5.65. How Can I execute INSERT/DELETE (SPARUL) statements against a WebID? protected SPARQL endpoint?

View our sample scenario .

1.5.66. How can I make HTTP Logging and Recording in Virtuoso?

HTTP Logging

Virtuoso can keep http log with all the requests that are made to the HTTP endpoint. Here are the steps:

1. Edit your virtuoso.ini and add the following setting:

   ```
   [HTTPServer]
   HTTPLogFile                = logs/http15022012.log
   ```
2. Restart Virtuoso.
3. Virtuoso will now maintain a http log in the logs subdirectory, with one line per request as in:

   ```
   180.76.5.87 - - [15/Feb/2012:21:50:44 +0100] "GET /data/Wilton_power_stations.json HTTP/1.0" 200 8
     "Mozilla/5.0 (compatible; Baiduspider/2.0; +http://www.baidu.com/search/spider.html)"
   180.76.5.87 - - [15/Feb/2012:21:50:45 +0100] "POST /sparql HTTP/1.0" 200 3012 "" ""
   ```
4. The first request after midnight will open a new logfile, to make sure the logfile does not keep growing. Old logfiles can be gzipped or removed by hand to conserve disk space.

The HTTP log files that Virtuoso produces can be processed by programs like Webalizer or AWstats to accurately measure site usage.

HTTP Logging Format

Virtuoso supports HTTP Logging format string like Apache Module mod_log_config . That string can be set in the "HTTPLogFormat" INI file param which works in conjunction with the "HTTPLogFile" INI file param. For example:

```
...
[HTTPServer]
...
HTTPLogFormat = %h %u %t "%r" %s %b "%{Referer}i" "%{User-agent}i" "%{NetId}e"
```

In this example we have %{User-Agent}i which means to log the HTTP header for user-agent. In similar way can log other input headers, "e" modifier is for environment variable NetId in this case.

Note that not all escapes from Apache Module mod_log_config are supported.

HTTP Recording

Virtuoso can also record the complete HTTP request for both GET and POST requests, including all incoming headers, POST parameters etc. This is a very useful tool for debugging, but it will cost performance and disk space, so it should not be left on for long periods of time. Each request will be written to a separate file.

Note : Some filesystem types like ext2 and earlier versions of ext3 on Linux cannot handle huge amounts of files in a single directory without slowing down the whole system.

Example of a GET request

```
GET /sparql?query=DESCRIBE%20%3CnodeID%3A%2F%2Fb15481%3E&output=text%2Fcxml     HTTP/1.1
Host: example.com
Connection: Keep-alive
Accept: */*
From: googlebot(at)googlebot.com
User-Agent: Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
Accept-Encoding: gzip,deflate
```

Example of a POST request

```
POST /ods_services/Http/usersGetInfo HTTP/1.1
User-Agent: Opera/9.80 (Macintosh; Intel Mac OS X 10.5.8; U; en) Presto/2.9.168 Version/11.51
Host: example.com
Accept: text/html, application/xml;q=0.9, application/xhtml+xml, image/png, image/webp, image/jpeg, image
  image/x-xbitmap, */*;q=0.1
Accept-Language: en,en-US;q=0.9,ja;q=0.8,fr;q=0.7,de;q=0.6,es;q=0.5,it;q=0.4,pt;q=0.3,pt-
  PT;q=0.2,nl;q=0.1,sv;q=0.1,nb;q=0.1,da;q=0.1,fi;q=0.1,ru;q=0.1,pl;q=0.1,zh-CN;q=0.1,zh-TW;q=0.1,ko;q=0.
Accept-Encoding: gzip, deflate
Referer: https://example.com/ods/
Cookie: interface=js; oatSecurityCookie=0123456878794576; sid=b3fae40reb78bc4bababab3cb2a70fb111
Connection: Keep-Alive
Content-Length: 77
Content-Type: application/x-www-form-urlencoded
Authorization: Basic bnQacPPuhhxs
Content-Transfer-Encoding: binary
```

Example Enabling recording

1. Go to the directory where the database is started from:

   ```
   mkdir sys_http_recording
   chmod 777 sys_http_recording
   ```
2. Edit virtuoso.ini and set:

   ```
   [HTTPServer]
   EnableRequestTrap          = 1
   ```
3. Next connect with isql to your database and run the following commands:

   ```
   registry_set ('__save_http_history_on_disk', '1');
   registry_set ('__save_http_history', '/');
   ```
4. Finally restart Virtuoso.
5. As result at this point every HTTP GET and POST request will be logged with all the parameters, headers and settings.

Example Disabling recording

1. To temporary disable the recordings, edit virtuoso.ini and set:

   ```
   [HTTPServer]
   EnableRequestTrap          = 0
   ```
2. Next you should remove the two registry items:

   ```
   registry_remove ('__save_http_history_on_disk');
   registry_remove ('__save_http_history');
   ```
3. Finally restart Virtuoso.

1.5.67. Quad Store Data Loading via Virtuoso's In-built Content Crawler?

This section covers the use of Virtuoso's in-built content crawler as a mechanism for scheduled of one-off data loading operations for its native quad store.

Why is this important?

Transforming external data sources into Linked Data "on the fly" (e.g., via the 'Sponger') is sufficient for many use cases, but there are times when the volume or sheer nature of a data source makes batch-loading necessary. For example, Freebase offers RDF representations of its data, but it doesn't publish RDF dumps; even if it did, such dumps would usually be outdated by the time they were loaded. Thus, a scheduled crawl of that resource collection offers a viable alternative.

How to Set Up the Content Crawler for Linked Data generation and import

The Virtuoso Conductor can be used to set up various Content Crawler Jobs:

- Setting up a Content Crawler Job to Import Linked Data into the Virtuoso Quad Store
- Setting up a Content Crawler Job to Retrieve Sitemaps (when the source includes RDFa)
- Setting up a Content Crawler Job to Retrieve Semantic Sitemaps (a variation of the standard sitemap)
- Setting up a Content Crawler Job to Retrieve Content from Specific Directories
- Setting up a Content Crawler Job to Retrieve Content from ATOM feed
- Setting up a Content Crawler Job to Retrieve Content from SPARQL endpoint

1.5.68. What is the ShortenLongURIs Virtuoso configuration parameter?

The ShortenLongURIs parameter is a Virtuoso configuration setting to shorten extremely long URIs in datasets when loading with the RDF Bulk Loader.

Some RDF datasets have long URIs that exceed the Virtuoso internal maximum size of 1900 bytes and thus need to be shortened to avoid errors like:

```
"SR133: Can not set NULL to not nullable column 'DB.DBA.RDF_QUAD.S' (or .O, or .P)"
```

which has been seen loading some of the DBpedia 3.7 datasets, for example.

The Virtuoso ShortenLongURIs parameter needs to be set in the [SPARQL] section of the Virtuoso configuration file (virtuoso.ini by default) and restart the Virtuoso Server.

```
[SPARQL]
.
.
.
ShortenLongURIs = 1
.
.
.
```

Note : This parameter is only in the Virtuoso 06.03.3131+ commercial builds, at the time of writing it is not included in the open source 6.1.4 archives but will be in the next 6.1.5 release. A patch to enable this feature is however available from the Virtuso patches page on source forge , which can be applied to a 6.1.4 archive from source forge and the Virtuoso server binary rebuilt.

1.5.69. How Can I send SOAP requests to Virtuoso SPARQL Endpoint?

This section presents sample scenario how to execute SPARQL query in SOAP request over Virtuoso SPARQL Endpoint.

1. Assume the following sample SOAP request containing simple SPARQL query:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.
   <soapenv:Body>
      <query-request xmlns="http://www.w3.org/2005/09/sparql-protocol-types/#">
         <query xmlns="">SELECT DISTINCT ?z FROM virtrdf: {?x ?y ?z .} LIMIT 10</query>
      </query-request>
   </soapenv:Body>
```

```
</soapenv:Envelope>
```

2. Save locally the content from above for ex. to file with name "soap.xml".
3. To pass the SOAP request to a Virtuoso SPARQL Endpoint, execute the following curl command:

```
$ curl -d@soap.xml -H "Content-Type:text/xml" -H "SOAPAction: ''" http://example.com/sparql
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <query-result xmlns="http://www.w3.org/2005/09/sparql-protocol-types/#">
      <sparql xmlns="http://www.w3.org/2005/sparql-results#" xmlns:xsi="http://www.w3.org/2001/XML
        <head>
         <variable name="z"/>
        </head>
        <results distinct="false" ordered="true">
          <result>
            <binding name="z">
             <uri>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</uri>
            </binding>
          </result>
          <result>
            <binding name="z">
             <uri>http://www.openlinksw.com/schemas/virtrdf#QuadStorage</uri>
            </binding>
          </result>
          <result>
            <binding name="z">
             <uri>http://www.openlinksw.com/schemas/virtrdf#array-of-QuadMapFormat</uri>
            </binding>
          </result>
          <result>
            <binding name="z">
             <uri>http://www.openlinksw.com/schemas/virtrdf#QuadMap</uri>
            </binding>
          </result>
          <result>
            <binding name="z">
             <uri>http://www.openlinksw.com/schemas/virtrdf#QuadMapValue</uri>
            </binding>
          </result>
          <result>
            <binding name="z">
             <uri>http://www.openlinksw.com/schemas/virtrdf#array-of-QuadMapColumn</uri>
            </binding>
          </result>
          <result>
            <binding name="z">
             <uri>http://www.openlinksw.com/schemas/virtrdf#QuadMapColumn</uri>
            </binding>
          </result>
          <result>
            <binding name="z">
             <uri>http://www.openlinksw.com/schemas/virtrdf#array-of-QuadMapATable</uri>
            </binding>
          </result>
          <result>
            <binding name="z">
             <uri>http://www.openlinksw.com/schemas/virtrdf#QuadMapATable</uri>
            </binding>
          </result>
          <result>
            <binding name="z">
             <uri>http://www.openlinksw.com/schemas/virtrdf#QuadMapFText</uri>
            </binding>
          </result>
        </results>
      </sparql>
    </query-result>
  </soapenv:Body>
</soapenv:Envelope>
```

1.5.70. How Can I Delete Triples containing blank nodes?

Please see details and sample scenario here .

1.5.71. How Can I Get a full explain plan for a simple SPARQL query?

Please see details and sample scenario here .

1.5.72. How Can I Use Expressions inside CONSTRUCT, INSERT and DELETE {...} Templates?

Please examples here .

1.5.73. How to optimize bif:dateadd in SPARQL query using selective index-friendly filter?

What?

Index-friendly filter for Date range (bif:dateadd) optimization within SPARQL query.

Why?

Achieve fast results and better performance.

How?

Assume the following SPARQL query:

```
SELECT ?wiki,
       ?dbp,
       bif:datediff('second',  xsd:DateTime(?extracted), now()) AS ?secondsAgo
 FROM <http://nl.dbpedia.org>
WHERE
  {
    ?wiki foaf:primaryTopic ?dbp .
    ?dbp dcterms:modified ?extracted .
    FILTER ( bif:datediff('minute', now(), xsd:DateTime(?extracted)) <= 10 )
  }
ORDER BY DESC(?extracted)
LIMIT 30
```

Let's take a look at the calculation of:

```
FILTER ( bif:datediff('minute', now(),  xsd:DateTime(?extracted)) <= 10 ) .
```

For each "is modified" triple we:

1. Convert ?extracted to xsd:dateTime;
2. Calculate datediff;
3. Make a comparison and know whether we hit or miss 10 minutes interval

Written so, this will lead to a potentially long loop, because even if the optimizer will realize that the filter is selective, it can't discover why is it so selective.

Now let's change the filter to:

```
FILTER ( ?extracted > bif:dateadd('minute', -10, now())) .
```

`now()` can be calculated once at the very beginning of the query execution because it does not depend on any rows in a given table. Then `bif:dateadd` has all arguments known and thus the whole `bif:dateadd('minute', -10, now())` can be calculated only once and produce some value. Therefor `FILTER ( ?extracted > some_known_value )` can be represented as a single search step: look at index and get triples with known P, known G, O greater than the given one and any S. That's pretty fast and predictable step, good for both optimizer and the runtime.

We can rephrase the query to filter index-friendly:

```
SELECT ?wiki,
       ?dbp,
        bif:datediff('second',  xsd:DateTime(?extracted) ,
        now()) AS ?secondsAgo
 FROM <http://nl.dbpedia.org>
WHERE
  {
    ?wiki foaf:primaryTopic ?dbp .
    ?dbp dcterms:modified ?extracted .
     FILTER ( ?extracted > bif:dateadd('minute', -10, now()))
  }
ORDER BY DESC (?extracted)
LIMIT 30
```

In this case the presence or the absence of the order by does not matter too much, because the query is way more straightforward: selective index-friendly filter first, and the selection could be ordered naturally via hot index used by the filter.

Note also that if you know the datatype of an object literal, there's no need to write a cast like xsd:dateTime --- it can make an expression index-unfriendly even if it will always return the argument unchanged on your specific data.

1. View the SPARQL Query Definition via SPARQL Protocol URL
2. View the SPARQL Query Results via SPARQL Protocol URL

1.5.74. How can I Determine the data usage across a Virtuoso instance?

What?

How to determine the data usage across a Virtuoso instance.

Why?

There are times when a variety of DBMS activities (e.g., automated content crawling) can lead to increase in database size, without the instance operator being fully aware of such activities and their consequences.

How?

Run the following statement:

```
        SELECT TOP 20 * FROM sys_index_space_stats ORDER BY iss_pages DESC;

Query result:
ISS_KEY_TABLE ISS_KEY_NAME ISS_KEY_ID            ISS_NROWS       ISS_ROW_BYTES ISS_BLOB_PAGES ISS_ROW_PAG
 WS.WS.SYS_DAV_RES         WS.WS.SYS_DAV_RE..    492   630627      437635794     501049         57521
 DB.DBA.RDF_QUAD           DB.DBA.RDF_QUAD       487   25134153  321502272     0                    49779
 DB.DBA.RDF_QUAD           RDF_QUAD_POGS         489   14749814  307163163     0                    47191
 WS.WS.SYS_DAV_RES....     DB.DBA.RDF_OBJ        501   1139934   160445830     10748          21478
 WS.WS.SYS_DAV_RES         WS.WS.SYS_DAV_RE..    729   432       307124        28840          97
 DB.DBA.RDF_IRI            DB.DBA.RDF_IRI        485   4378779   138542078     0                    22479
 DB.DBA.RDF_IRI            DB_DBA_RDF_IRI_UN..   486   4259557   139496730     0                    20187
......
```

1.5.75. How to discover the capabilities of a SPARQL endpoint to enhancing SPARQL-FED usage from Virtuoso instances?

What?

How to discover the capabilities of a SPARQL endpoint en route to enhancing SPARQL-FED usage from Virtuoso instances.

Why?

There are features supported in Virtuoso SPARQL that aren't supported by other SPARQL engines. There are also a lot of Virtuoso instances behind bubbles in the LOD cloud. Net effect, a Virtuoso instance operator is faced with varied behavior when attempting to use SPARQL-FED functionality.

How?

Run this command to load metadata (or lack thereof) from an external SPARQL endpoint:

```
SPARQL LOAD SERVICE <{sparql-endpoint-iri> DATA
```

Example

1. Run:

```
Query result:
ANY
 Load service <http://lod.openlinksw.com/sparql> data -- done.
 Trying to query <http://lod.openlinksw.com/sparql> as SPARQL web service endpoint, POST mode...
```

2. Check the retrieved data:

```
SPARQL
SELECT *
  FROM virtrdf:
 WHERE
     {
       <http://lod.openlinksw.com/sparql> ?p ?o
     }
 LIMIT 10

Query result:
p                                                                o
ANY                                                              ANY
http://www.openlinksw.com/schemas/virtrdf#isEndpointOfService      http://lod.openlinksw.com/sparq
http://www.openlinksw.com/schemas/virtrdf#dialect                000037ff
```

1.5.76. How to split a urlencoded ";-" separated list of urls in a SPARQL query?

1. Assume the following string: "http://example.org/test1;http://example.org/test2".
2. In order to split the given string into two values i.e. http://example.org/test1 and http://example.org/test2 , one should use `split_and_decode()` which returns an array. Thus the `aref()` also needs to be used for loading the elements.
3. Example:

```
SELECT bif:aref (bif:split_and_decode('http%3A%2F%2Fexample.org%2Ftest1%3Bhttp%3A%2F%2Fexample.org
{ ?S ?P ?O }
LIMIT 1
```

- View the SPARQL Query Definition via SPARQL Protocol URL
- View the SPARQL Query Results via SPARQL Protocol URL

1.5.77. How to Update Large SPARQL Data avoiding due to database checkpoint abortion?

Assume while performing large SPARQL update, for example of triples around 80 millions in total, one gets the following error which aborts the update:

```
"Transaction aborted due to a database checkpoint or database-wide
atomic operation. Please retry transaction" .
```

The error means that the SPARQL update has possibly encountered a database checkpoint during the load. Thus one should check the CheckpointInterval in the INI file and possible increase its value:

```
;
;  Server parameters
;
[Parameters]
...
CheckpointInterval       = 60
...
```

Note: The RDF Bulk loader can be used for loading large datasets, and it will handle the checkpoint matter automatically.

1.5.78. How to Manage SSL Protocols and Ciphers used with Virtuoso?

What?

As of Virtuoso 7.2, SSL protocol and cipher support is now configurable for connections from all HTTP, ODBC, JDBC, ADO.NET, and OLE-DB clients.

Why?

Default binding to OpenSSL can expose Virtuoso instances to version- and cipher-specific SSL vulnerabilities (e.g., recent Poodle exploit). Being able to scope Virtuoso's use of SSL to one or more specific versions provides instance administrators better protection against a moving target.

How?

Basic SSL Protocol Configuration

Basic configuration is through the `SSL_Protocols` values in the `[Parameters]` and `[HTTP]` sections of the Virtuoso INI file. These are comma+space-separated (", ") value lists. Including a protocol name explicitly enables it; preceding the protocol name with an exclamation point ("`!` ") explicitly disables it.

Basic SSL Protocol Configuration

Table 1.1. Supported SSL Protocols and INI keyword values List

SSL/TLS Version	Value for INI file	Notes
SSLv2	-	Permanently disabled.
SSLv3	`SSLv3`	Disabled by default. To our knowledge, only required by IE6/Windows XP clients.
TLSv1	`TLSv1`	Enabled by default.
TLSv1.1	`TLSv1.1`	Enabled by default, supported if available in local `openssl` library.
TLSv1.2	`TLSv1.2`	Enabled by default, supported if available in local `openssl` library.

Advanced SSL Cipher List Configuration

The `SSL_Cipher_List` values in the `[Parameters]` and `[HTTP]` stanzas of the Virtuoso INI file may also be adjusted, to disable particular ciphers when there are security reports about some new attack that breaks them. These are colon-separated ("`:` ") value lists.

Including a protocol name or groupname explicitly enables it; preceding the protocol name with an exclamation point ("`!` ") explicitly disables it. You can review the ciphers supported by your local `OpenSSL` library with the command:

```
openssl ciphers -v ALL
```

For instance, we recommend explicitly forbidding anonymous cipher suites (i.e., ones that do not use certificates, and are therefore susceptible to man-in-the-middle attacks) using `!aNULL` .

We also recommend including `@STRENGTH` at the end of the list, so that `OpenSSL` will prioritize the enabled ciphers by key length, regardless of the list order.

Recommended Settings

The sample settings below provide a reasonable tradeoff of security versus flexibility. As shown, we have enabled `SSLv3` on the HTTPS ports for IE6 users, but left this disabled on the SQL data port.

```
[Parameters]
SSL_Protocols   = TLSv1, TLSv1.1, TLSv1.2
SSL_Cipher_List = HIGH:!aNULL:!eNULL:!RC4:!DES:!MD5:!PSK:!SRP:!KRB5:!SSLv2:!EXP:!MEDIUM:!LOW:!DES-CBC-SHA

[HTTP]
SSL_Protocols   = SSLv3, TLSv1, TLSv1.1, TLSv1.2
SSL_Cipher_List = HIGH:!aNULL:!eNULL:!RC4:!DES:!MD5:!PSK:!SRP:!KRB5:!SSLv2:!EXP:!MEDIUM:!LOW:!DES-CBC-SHA
```

Chapter 2. Installation Guide

Abstract

This chapter applies exclusively to the various commercial releases of Virtuoso. If you are working with the open source version, please refer to the instructions on the web site where you obtained it.

This chapter describes how to install OpenLink Virtuoso VDBMS. It contains both a quick start install guide and full walk through install guides for major platforms.

Virtuoso has been designed from the ground-up to be easy to implement.

The installation generally runs without problems, run the setup program or script and follow the on screen instruction prompts. You will be asked some questions, if in any doubt please follow the instructions in this guide in order to install and configure your Virtuoso components correctly.

Once installed, each instance of Virtuoso is controlled by an INI (initialization) file. This file links the Virtuoso Server process with an actual database file or device. For example, the demo instance of Virtuoso on all platforms is controlled by the demo.ini file that is found in the demo directory along with the demo.db and related files. There is a one-to-one relationship between server processes and database files.

The INI file is a text file consisting of keys and key values. You will set several of these values during installation. At any time thereafter, their values can be changed by editing the INI file with a standard text editor.

The details of the INI file sections, keys, and the meanings of the key values are described in the Configuring Server Startup Files section.

2.1. Virtuoso on All platforms Commmon Specifics

2.1.1. Installation Requirements

A typical installation will require a minimum of 400Mb of hard disk space to install the code, samples, documentation and sample database. The database will need additional space for data inserted, backups, logs and reports, web pages, etc.

The size of the database .db file will not reduce when data is removed. The spare space will however be reclaimed for later use.

2.1.2. Operational Requirements

The Virtuoso database requires a minimum of 64Mb of system memory for each instance to operate in. Each connection will take between 70kb and 130kb of memory.

The memory usage is affected by the following parameters in the Virtuoso configuration file .

ServerThreads

ServerThreadSize

MainThreadSize

FutureThreadSize

NumberOfBuffers

See the following Parameter section for more details.

2.1.3. Operating System Support

Virtuoso runs on the following operating systems:

Windows

Linux

Unix: AIX, HP-UX, Solaris, etc.

Mac OS X

2.1.4. Limits

The below table lists the most important parameters and limits for a database or a Virtuoso server instance. For space consumption of individual data types, see Space Consumption .

Table 2.1. Limits

Limit Description	Value
Identifier length	100
User name and password length	100
SQLstmt and proc text	100K+
varchar length	Constrained by max row length when stored into non-LOB column, 10MB inside procedure code.
Numeric precision	Max 40 digits
LOB column length	2GB
row length	4086
Columns per row	200
Database size	32TB for data, 32TB for temp data
Page size	8K
File size	64 bit file offsets on supporting OS's
Files per database	unlimited
Table size	Up to database size
Tables per database	64K keys, a table takes 1 key for primary key, 1 key per index and 1 key for each obsolete primary key layout resulting from adding or dropping columns.
Indexes per table	Unlimited, subject to global cap on keys.
Row in sorting/distinct temp	4078b, as in tables
Locks per transaction	Depends on memory, 16b per row lock, 60b per page with at least one row lock.
Changes per transaction	Depends on memory, ini parameter allows cap on rollback before image.
Virtual memory>	Depends on OS, 64 bit pointers on 64 bit platforms
Threads	Depends on OS, ini files specifies cap for pool for SQL client and web server worker thread pools.
Max connections	Depends on OS fdset size, at least 2048, license can set a limit
Memory per connection	22K plus up to 100 cached SQL statement compilations

2.2. Virtuoso for Windows

2.2.1. Before You Start

To ensure a smooth installation, please review the following checklist before you start the setup program.

Have you downloaded the software?

If you have not already done so, please download your copy of OpenLink Virtuoso from the OpenLink web site. The URL is http://www.openlinksw.com . When you visit the site, select "Download & Evaluation".

1. Select product "OpenLink Virtuoso: Personal Edition "
2. From "Product Category" choose "Universal Server"
3. From "Product" choose "Virtuoso Universal Server"
4. Select "Database Version"
5. From "server OS" select "Windows" and click "Next"
6. Select a release and click "Next"
7. You will be prompt to login in not already.
8. Click "Next"
9. Download the desired component. For example, click the HTTP link for "Multi-Threaded Universal Server (Commercial Edition) for OpenLink Virtuoso 4.x"

Do you have the License file?

The server will need a license file, and this is automatically delivered by email as part of the download process. It is therefore important to supply a valid email address. The installer will prompt for the license file location.

Reinstalling Virtuoso?

If you are reinstalling Virtuoso, you must first shutdown any additional database servers you have configured on non default ports. The databases on port 1111 and 1112 will automatically be stopped during the install process. On Windows NT/2000/XP, go to the services applet in the Control Panel. There you can stop services that appear with names of the form "OpenLink Virtuoso DBMS Server [instance name]".

On Windows NT/2000/XP you can also shut them down with the following commands in the Virtuoso\bin directory:

```
virtuoso-odbc-t.exe +service stop
virtuoso-odbc-t.exe +instance myinstance +service stop
virtuoso-odbc-t.exe +instance anotherinstance +service stop
```

Note:

There is an alternative executable available for Virtuoso called: virtuoso-odbc-f.exe, which you may have installed. This version is designed for versions of Windows that offer no native threading support such as Windows 98, and offers no advantages over virtuoso-odbc-t.exe for Windows NT/2000/XP.

You can list the services with their status using:

```
virtuoso +service list
```

Allocation of TCP/IP ports

The Default database have the following port allocations:

Table 2.2. Port Usage

Port	Used by
1111	Default Virtuoso Server port
8889	Default Virtuoso Visual Server Administration Interface HTTP port

By default, the Virtuoso DBMS listens on port 1111. This port number is not selectable at install time. If the port is currently in use by another server then after the installation completes you will need to change the port number in the Local Virtuoso ODBC DSN that is created during the installation process, and also in the following file:

```
C:\Program Files\OpenLink Software\Virtuoso 5.0\bin\virtuoso.ini
```

To check if a port is in use on NT, you can use the following command, and review the output:

```
netstat -an | more
```

Do you already have an ODBC Data Source Name (DSN) that works?

The Virtual Database aspects of the Virtuoso for ODBC assumes that you already have an ODBC driver on your machine from a database vendor, OpenLink Software, or from another 3rd party vendor. It is a useful sanity-check to ensure that you have an ODBC System Data Source Name (DSN) on your machine, and that it successfully connects to your database, and that it retrieve data. Any ODBC compliant tool can be used to test that your ODBC Data Source Name (DSN) works, such as the sample application we provide - C++ Demo32, and that it is sufficiently robust to operate with Virtuoso. Virtuoso is ODBC Driver Independent and certified to work with any drivers that are ODBC level 1 or higher.

2.2.2. Getting To Know Your Virtuoso Components

Virtuoso is made up of several components, these components fall into two distinct categories: Client Components and Server Components.

Client Components Package

These are the components typically used to communicate with a local or remote Virtuoso server, this includes:

Virtuoso Drivers for ODBC

Virtuoso Drivers for JDBC

ISQL Utility

ISQLO Utility

Documentation

Samples

Server Components

These are the components that service Virtuoso clients, these include:

Virtuoso Virtual Database Servers for: ODBC based interfaces - the files "virtuoso-odbc-f" (for Virtuoso Lite) and "virtuoso-odbc-t" (For Virtuoso Enterprise Edition).

Virtuoso HTTP Server.

Virtuoso System Manager to configure the database through the HTTP interface.

2.2.3. Installation Steps

Prepare to Install

If you are upgrading an existing Virtuoso installation and want to preserve your existing data and configuration files:

1. Shut down your Virtuoso instance as normal.
2. Right-click on the existing `Virtuoso.app` (exact name may vary), and select `Show Package Contents`.
3. Drill down to `Contents` -< `virtuoso` -< `database`.
4. Check the size of the `virtuoso.trx` file, found in the `database` folder alongside the `virtuoso.db` and `virtuoso.ini` files.

    - If zero bytes, proceed to step 6.
    - If larger than zero bytes, start the instance from the command line with the extra argument, " +checkpoint-only ", and recheck `.trx` file size -- expecting zero bytes.
    - If larger `.trx` files persist, contact Technical Support .

5. Make a backup of your existing Virtuoso Database and Configuration files (defaults are `virtuoso.db` and `virtuoso.ini`) prior to performing the steps that follow. *Note* : Virtuoso's online backup feature , if enabled, should have already generated a backup-set.
6. Use the Add/Remove Programs utility to uninstall any previously installed Virtuoso components. Data and configuration files will be left behind by this process.

Download the Virtuoso Installer Program

1. Download the Virtuoso 7.2 installer archive (wavpz2zz.msi) from the download site .

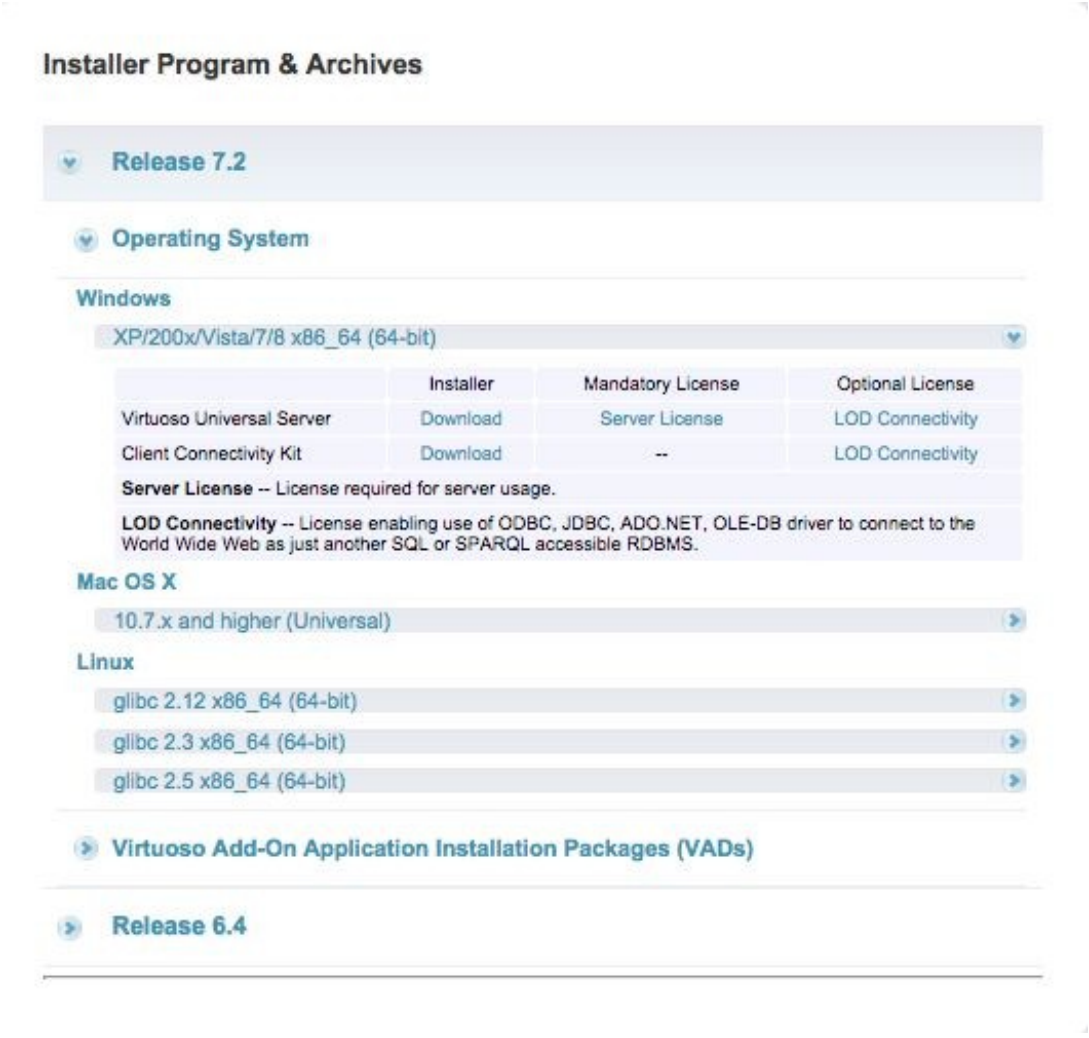
   Note

   : Virtuoso 7.x is 64-bit only, and requires Vista or Windows Server 2003, or later.

Obtain a License File

1. If you haven't already obtained a suitable Virtuoso 7.x license file, return to the download site , and click the link for *Free Eval License* (15 day duration, requires only your email address) or *Free Pilot License* (30 day duration, requires some additional personal and business information).

   Figure 2.1. Installing the Virtuoso Universal Server on Windows -- Obtain a License File



2. Fill out the relevant form, and download the license file when presented.

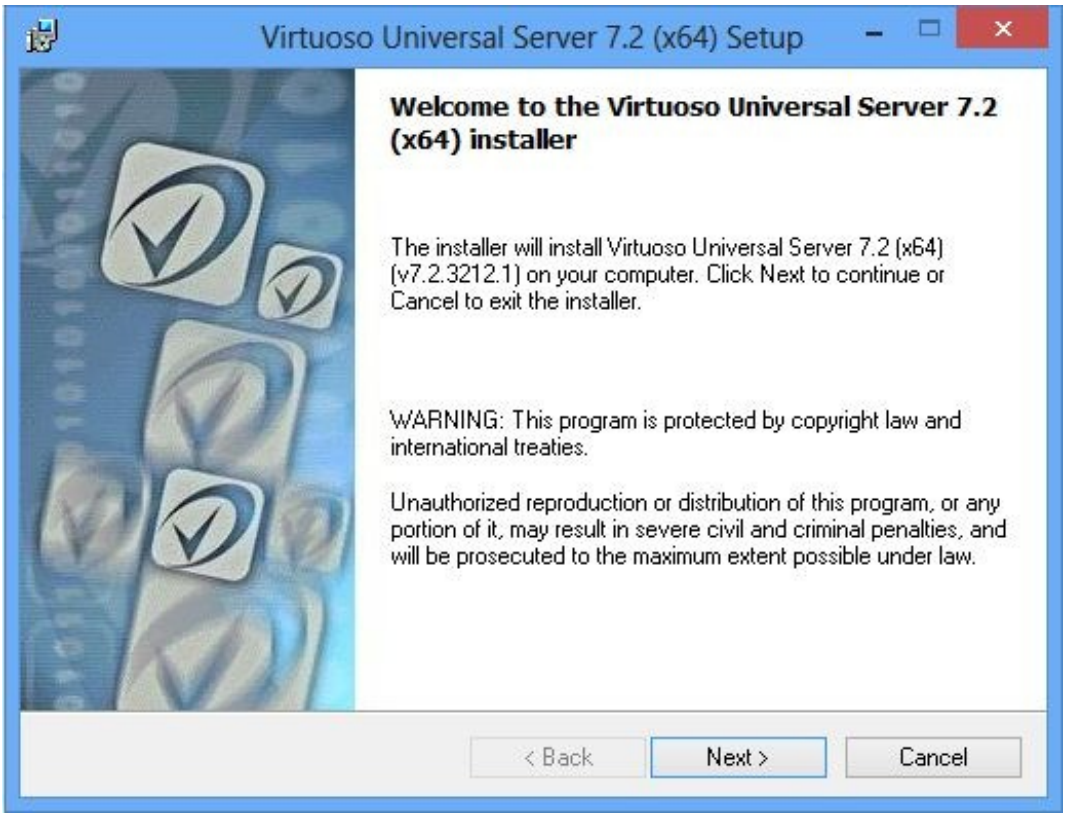
   Note

   : the file must be named `virtuoso.lic` for use. Some browsers may rename the file during download; you can simply rename the file on your machine.

Run the Virtuoso Installer

1. Double-click the installer archive, `wavpz2zz.msi` , to start the process with the Installer Program Introduction.

   Figure 2.2. Installing the Virtuoso Universal Server on Windows -- Run the Virtuoso Installer

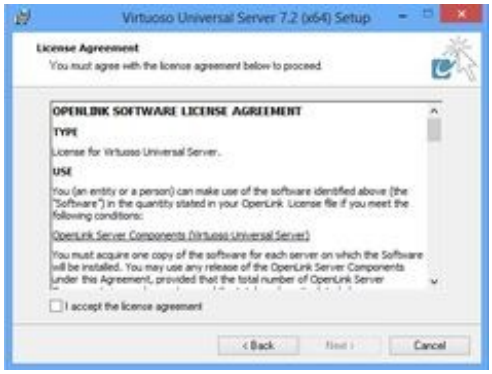
   

2. Click

   Next

   , and you'll be presented with the License Agreement:

   Figure 2.3. Installing the Virtuoso Universal Server on Windows -- License Agreement

3. Tick the box for

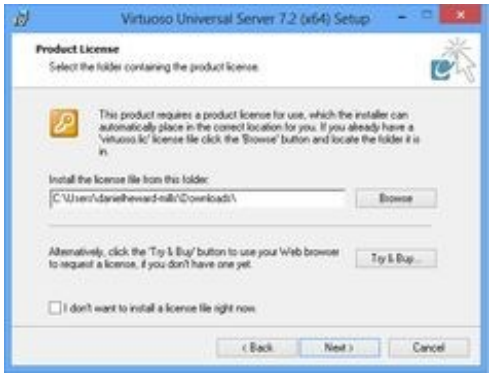
   I accept

   , and click

   Next

. You'll be asked to locate your license file:

Figure 2.4. Installing the Virtuoso Universal Server on Windows -- License Agreement Accept



4. Use the

   Browse

   button to locate the license file you obtained earlier, or click the

   Try and Buy

   button to obtain a new one now, or tick the box for

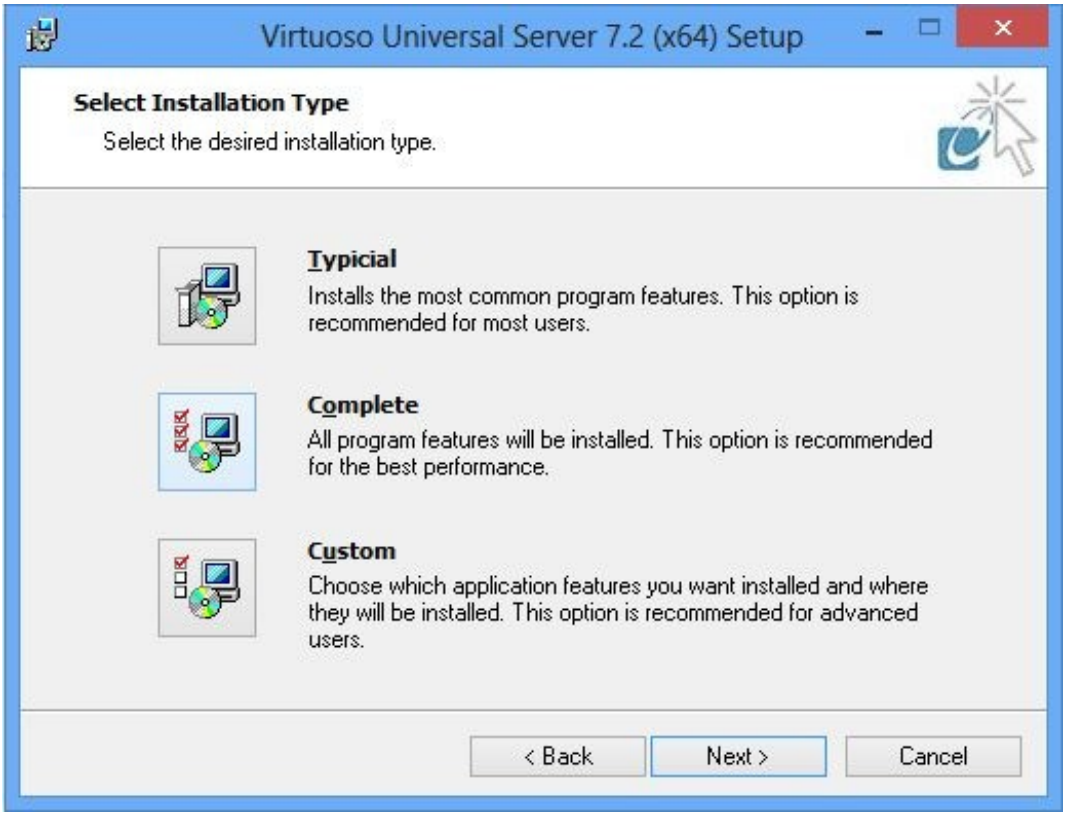
   I don't want to install a license file now

   . Then, click

   Next

   . You'll be prompted to select an Installation Type:

Figure 2.5. Installing the Virtuoso Universal Server on Windows -- Choose Installation Type



5. Select the desired option:

   - *Typical*

     -- installs to the default filesystem location, may not include all components
   - *Complete*

     -- installs to the default filesystem location, includes all components
   - *Custom*

     -- to install to any filesystem location, and select which components will be installed

6. Click

   Next

   , and if you chose

   Typical

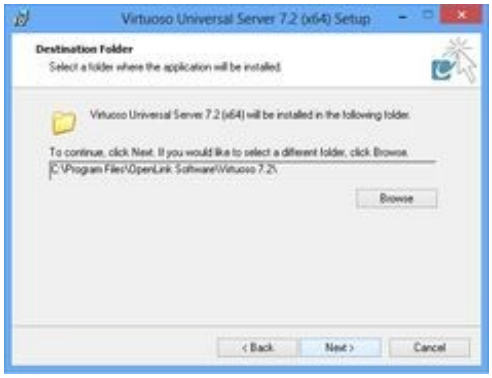
   or

   Complete

   , skip this and the next step; if you chose

   Custom, you'll be prompted to choose a destination folder:

Figure 2.6. Installing the Virtuoso Universal Server on Windows -- Choose Destination Folder



7. Click

   Next

   , and if you chose

   Custom

   , you'll be prompted for Feature Selection:

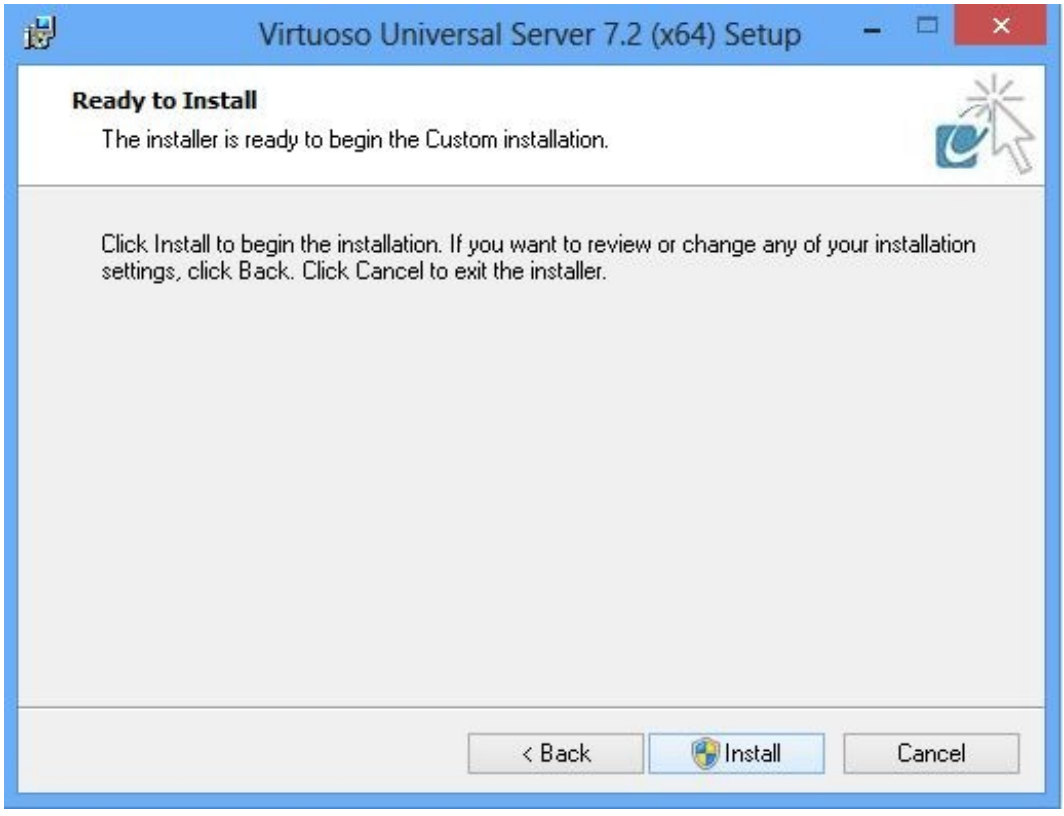
   Figure 2.7. Installing the Virtuoso Universal Server on Windows -- Feature Selection

8. Click

   Next

   , and the installer will indicate it's ready to begin:

   Figure 2.8. Installing the Virtuoso Universal Server on Windows -- Read to install



9. Click install, and wait for notification of completion.

Start the Virtuoso Service Manager, and Locate it in the Task Bar

1. Launch the Virtuoso Service Manager from your

   Start

   menu.

   Figure 2.9. Installing the Virtuoso Universal Server on Windows -- Launch the Virtuoso Service Manager.

2. Look for an icon to appear in the Task Bar at the bottom right of your screen.

   Figure 2.10. Installing the Virtuoso Universal Server on Windows -- Look for an icon

Start and Administer your Virtuoso Instance

1. Right-click on the Virtuoso Service Manager icon in the Task Bar, and click on

   OpenLink Virtuoso Server - Start

   :

   Figure 2.11. Installing the Virtuoso Universal Server on Windows -- Start your Virtuoso Instance

2. Open the

   Virtuoso Conductor

   , Virtuoso's HTTP-based administration tool, by loading this URL in your Web Browser: http://example.com/conductor .

   Figure 2.12. Installing the Virtuoso Universal Server on Windows -- Virtuoso Conductor

3. Use the Conductor to configure your Virtuoso instance for future use. For example, it is

   strongly

   advised that you change the default passwords for the

   dba

   and

   dav

   super-user accounts (drill down to

   System Admin

-<

User Accounts

).

Optional - Restore your old Data and Configuration Files

If you preserved existing data and configuration files at the start of this process, now is the time to bring them back into play.

1. If you used Virtuoso's online backup feature, the documentation will walk you through the restoration process.
2. If you simply copied the INI and DB files elsewhere, and if they were Virtuoso 7.x files, you can just stop the current Virtuoso instance, and replace the newly created INI and DB with the old ones. For best results, we do advise that you compare the content of the old INI with the new, and add any new settings from the new to the old, adjust as may be appropriate to your deployment.
3. If you simply copied the INI and DB files elsewhere, and if they were Virtuoso 6.x or older files, there are some extra steps that must be taken.

2.2.4. Post Installation

Post-Installation Sanity Check

A quick way to check that the database is running, is to point a browser to the http port. The following example URLs will show the System Manager for the default, and the demo Virtuoso databases:

```
http://example.com
http://example.com:8890
```

Troubleshooting DB Startup Failures

Install failure

Check the .log file in your Virtuoso installation db location, for ex:

```
C:\Program Files\OpenLink Software\Virtuoso 7.2\db\virtuoso.log
```

which is for the database and log files etc. in order to review the installed files binaries and databases (defaults) for issues if install fails.

.lck existance

Check .log in case of .lck issue

Delete the .lck file and repeat the steps from above.

.trx incompatiblity issue

Check .log in case of .trx incompatiblity issue

oplmgr issue

See Troubleshooting License Problems

2.2.5. Starting Your Virtuoso Server

Due to subtle differences between the Windows 95 family (including Windows 98 and Windows ME) and the Windows NT family (including Windows 2000, Windows XP, Vista, Windows 7, Windows 2000 Server, Windows Server 2003, and Windows Server 2008), specific instructions for starting and stopping your Virtuoso server are supplied for each Windows family.

Windows NT family

You can start your Virtuoso server in one of two ways, automatically or manually, the default mode configured by your installer is Automatic.

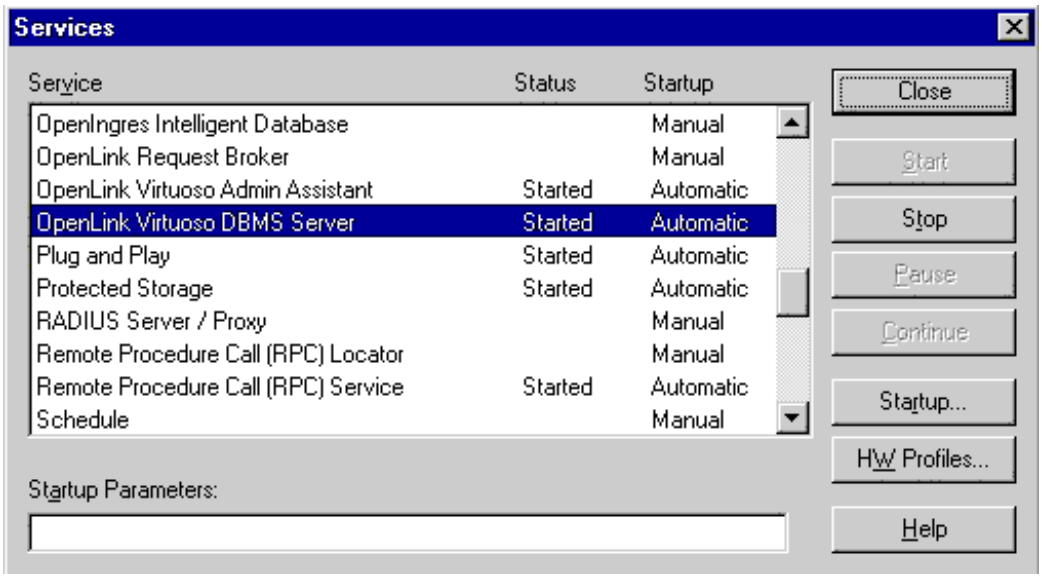
Automatic Startup

Your Virtuoso servers (default and demonstration databases) are configured at installation time as "Automatic Startup" services. Thus, rebooting your machine after installation is all that is required in order for your Virtuoso servers to be ready to start receiving connections from client applications. The service is actually started during the installation, however some files may be locked, so a reboot is required to complete the install.

Manual Startup

You can manually start your Virtuoso server in one of two ways, either via the "services" control panel applet or from within a DOS shell. When doing this via the "Services" control panel Applet you need to simply open up your control panel and click on the "services" applet. This will present you with a screen similar to the one below:

Figure 2.13. Services Applet



Click the "Start" or "Stop" button in order to start or stop a highlighted Virtuoso Server.

To start your Virtuoso servers manually using DOS command prompts, start a DOS shell and then enter the following command:

```
virtuoso-odbc-f +service start
```

Windows 95 family

You can start your Virtuoso server under Windows 95/98 in one of two ways, automatically or manually, the default mode configured by your installer is Automatic.

Automatic Startup

Your Virtuoso servers (default and demonstration databases) are configured at installation time as "Automatic Startup" services. Thus, rebooting your machine after installation is all that is required in order for your Virtuoso servers to be ready to start receiving connections from client applications.

If Virtuoso does not start automatically on Windows 95/98 after installation you will need to add the following line to the system's "autoexec.bat" file (assuming you have installed Virtuoso using the installers default installation folder):

```
C:\Program Files\OpenLink\Virtuoso\bin\virtuoso-odbc-f +service start
```

Manual Startup

You can start your Virtuoso servers manually by entering the following commands at the DOS command line prompt:

```
virtuoso-odbc-f +service start
virtuoso-odbc-f +instance demo +service start
```

Note:
If you encounter a "Program Not Found" error this is because your Virtuoso installation's "bin" sub-directory is not part of your PATH environment variable.

2.2.6. Creating and Deleting Virtuoso Services

At sometime you may need to have multiple Virtuoso server processes running on your Windows servers or simply need to add or delete existing Virtuoso servers that have been registered with the "Services" control panel applet. You do this by executing the following commands:

Adding new Virtuoso Server service: virtuoso-odbc-f +service create -I <your chosen service name>

Deleting an existing Virtuoso service: virtuoso-odbc-f +service delete -I <service that you are deleting>

Starting a new Virtuoso service manually: virtuoso-odbc-f +service start -I <service name>

Showing a list of existing Virtuoso services: virtuoso-odbc-f +service list

Note:
This functionality is only available on Windows NT/2000/XP.

Ensure that the services applet from the control panel is closed while issuing the above commands to prevent locking.

2.2.7. Configuring Virtuoso Client Components

The Virtuoso client components set consists of the following:

Virtuoso Driver for ODBC
Virtuoso Driver for JDBC
Native Virtuoso Interactive SQL Query (ISQL)
ODBC based Interactive SQL Query Utility (ISQLO)
The sections that follow address the configuration and utilization of each one of these client components separately.

Virtuoso Driver for ODBC

The Virtuoso Driver for ODBC conforms to both the ODBC 1.x,2.x,and 3.x versions of the ODBC specification, it implements Core, Level 1, Level 2, and Extensions functionality. This driver enables you to communicate with local or remote Virtuoso servers across any combination of platforms supported by Virtuoso.

At installation time two ODBC data source names (DSN's) are created with default values by the Virtuoso installer, the first DSN named "Local Virtuoso" provides a link to a local default Virtuoso database server instance, while the other named "Local Virtuoso Demo" provides a link to a local Virtuoso server for the Virtuoso demonstration database.

See Also:
the Data Access Interfaces chapter for more detailed information regarding the Virtuoso ODBC Driver setup

Virtuoso Drivers for JDBC

The Virtuoso Drivers for JDBC are available in "jar" file formats for the JDBC 1.x, JDBC 2.x and JDBC 3.x JDBC specifications. These are Type 4 Drivers implying that utilization is simply a case of adding the relevant "jar" file to your CLASSPATH and then providing an appropriate JDBC URL format in order to establish a JDBC session with a local or remote Virtuoso server. It is important to note that when you make a JDBC connection to a Virtuoso Server, you do also have access to Native and External Virtuoso tables. Thus, you actually have a type 4 JDBC Driver for any number of different database types that have been linked into Virtuoso.

See Also:
The JDBC Driver section in the Access Interfaces chapter.

Native Virtuoso Interactive SQL Query (ISQL)

To assist you with the use of Virtuoso an interactive SQL interpreter utility called "ISQL" is bundled with all Virtuoso installations. This utility allows you to enter and execute SQL statements from a command line prompt against a local or remote Virtuoso server.

The command line format for executing this utility is:

```
isql [<hostname>][:<port#>]
```

The steps that follow guide you through the process of using ISQL to verify your Virtuoso installation:

1. Open up a DOS shell
2. Move into the "bin" sub-directory of your Virtuoso installation
3. Type the following command: isql

   This will connect you to the default Virtuoso server at port "1111", if you want to connect to your demonstration Virtuoso database (which listens at port 1112 by default) then enter: isql 1112
4. At the ISQL command line prompt type in the following command:

   ```
   tables ;
   ```

   This will produce a list of tables in the current Virtuoso database, you can also enter other SQL statements e.g.:

   ```
   select * from Customers
   ```

   This will produce a listing of records in the "Customers" table.
5. If you are not able to perform these task that it implies that your Virtuoso database server is not running or that it has not been installed properly.

ODBC based Interactive SQL Query Utility (ISQLO)

In addition to a Native ISQL utility, your Virtuoso installation includes an ODBC based version of "ISQL" called "ISQLO", this enable you to connect to ODBC DSN's rather than Virtuoso only. To use this program perform the following steps:

1. Open up a DOS shell
2. Move into the "bin" sub-directory of your Virtuoso installation
3. Type the following command :

   ```
   isqlo <enter a valid ODBC Data Source Name>
   ```

   Note:
   If your DSN contains spaces you will need to enclose it within double quotes when passing it as a parameter to ISQLO
4. Enter any valid SQL at the ISQLO command line prompt.
5. You can also use this utility from within the Virtuoso Conductor

2.2.8. Default passwords

See the following Quick Start chapter for very important information about changing the default passwords .

2.3. Installing the Virtuoso Universal Server on Unix-like OS (AIX, HP-UX, Linux, Solaris, etc.)

2.3.1. Installation

Once the required tar file has been downloaded from the web or ftp site and placed in a designated installation directory the

installation process is ready to commence.

- Choose or create a suitable user account to own your Virtuoso installation.
- Un-tar the file to obtain the install script and archive file using a command like this, where XX specifies the OS identifier (e.g., l3 = Linux glibc23 x86, l9 = Linux glibc25 x86_64, sv = Solaris 2.10 x86_64, etc.):

```
tar -xvf XXvpz2zz.tar
```

- If a license file exists, place this in the same directory as the installation files, and it will automatically be applied during installation. If upgrading an existing Virtuoso instance, be sure to take a backup of your database file and shut down the existing instance before proceeding.
- Run the install script using the following command:

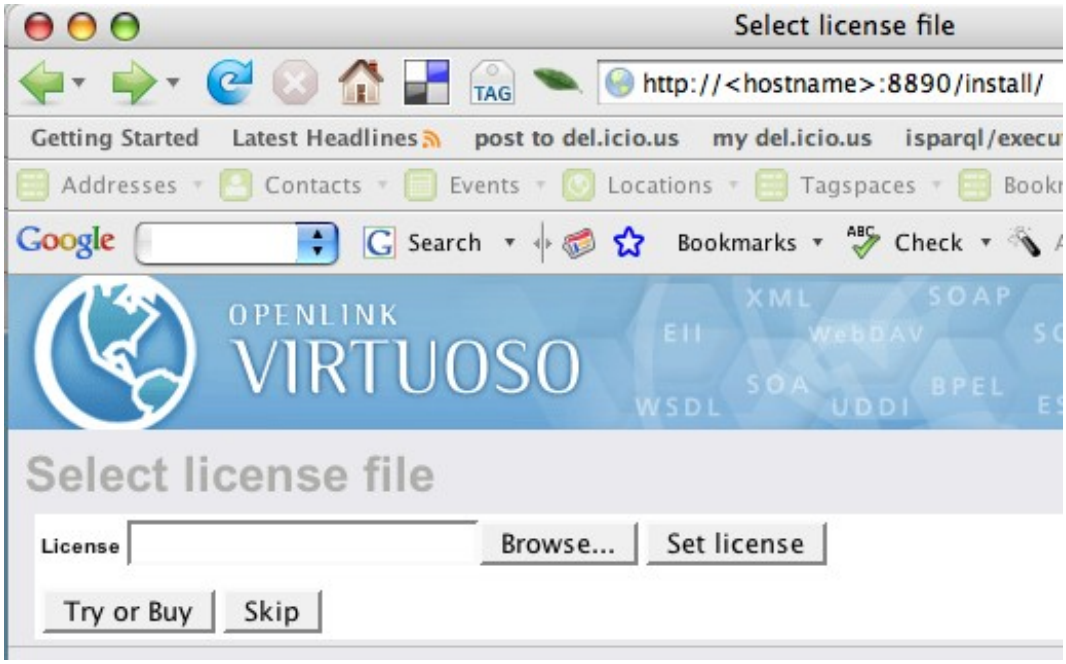
```
$ sh ./install.sh

- Extracting Virtuoso Universal Server vX.Y

- Creating default environment settings

- Creating default database settings
Installing new virtuoso.ini in $VIRTUOSO_HOME/database
Installing new php.ini in $VIRTUOSO_HOME/database
Creating symlink to $VIRTUOSO_HOME/bin/virtuoso-iodbc-t

- Registering ODBC drivers

- Registering .NET provider for Mono

- Installing VAD packages in database (this can take some time)

- Checking where license should be stored

- Starting OpenLink License Manager

- Checking for initial Virtuoso license

- Starting Virtuoso server instance

- Finalizing installation

This concludes the first part of the installation.

Please start a browser manually and open the following URL to finalize
the installation process:

     http://<hostname>:8890/install/

Installation completed
$
```

- Proceed to the Configuration section below.

2.3.2. Configuration

- Once the URL above has been loaded into a browser, you can select a license file for use with the installation; if a license file is already in place, you can "Skip" this task with the obvious button:

Figure 2.14. License file

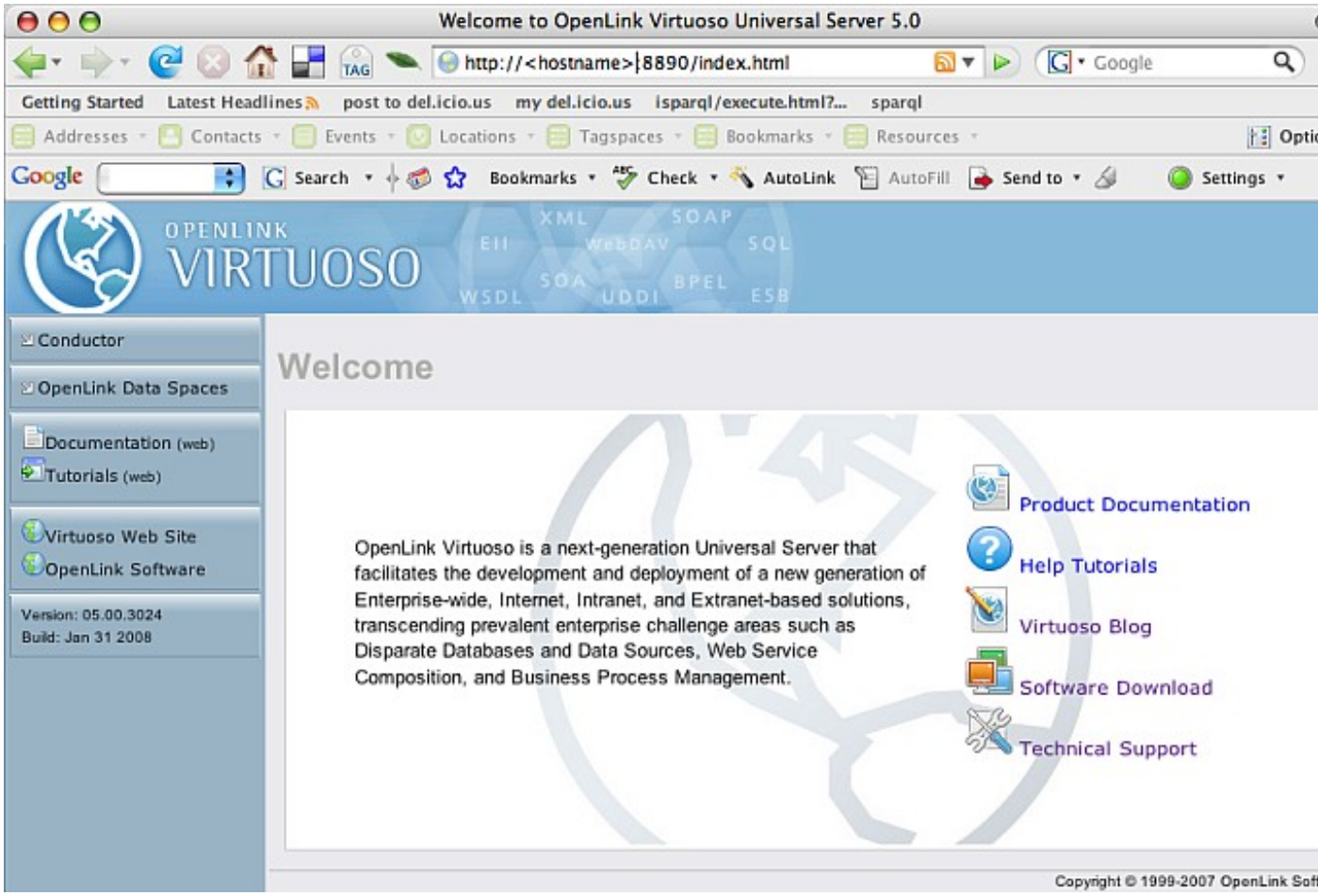


- It is strongly advised that the standard "dba" and "dav" user account password be changed using the post installation page provided:

Figure 2.15. DAV/DBA authentication Setting page

- The post installation page is now complete and the Virtuoso Welcome page is presented:

Figure 2.16. Post installation page



- The installation is now ready for use.

2.3.3. Demo Database

Installation steps .

2.4. Virtuoso for Mac OS X

2.4.1. Before You Install

Before installing the Virtuoso software for Mac OS X you must ensure that you have the correct package from the OpenLink Web Site.

2.4.2. Virtuoso Drag and Drop Installer for Mac OS X

Note : Virtuoso 7 and its Drag-and-Drop Installer for Mac OS X require Lion (10.7.x), Mountain Lion (10.8.x), Mavericks (10.9.x), or later. Virtuoso 6 remains available for deployment on Leopard (10.5.x) or Snow Leopard (10.6.x).

1. Download and double-click on the Virtuoso installer disk image,

   mwvp2zzz.dmg

   , to start the installation.

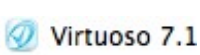
   Figure 2.17. Mac OS X installer: Start installation

2. Drag the Virtuoso application icon to your Applications folder (recommended) or any preferred location on your Mac.

Figure 2.18. Mac OS X installer: Drag Virtuoso Application

3. Double-click the Virtuoso application (e.g., Virtuoso 7.1.app) to launch the Virtuoso Application Manager Menu Extra, which will display as a V symbol in the menu bar.

Figure 2.19. Mac OS X installer: Drag Virtuoso Application



4. Select

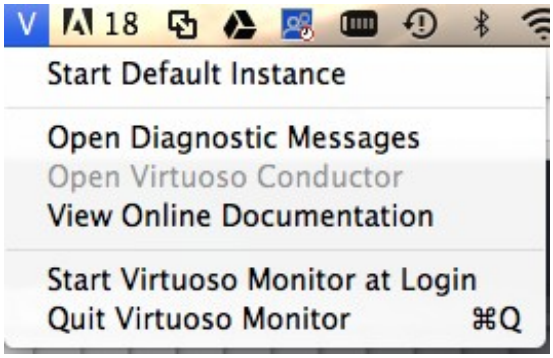
   Start Default Instance

   from the

   V

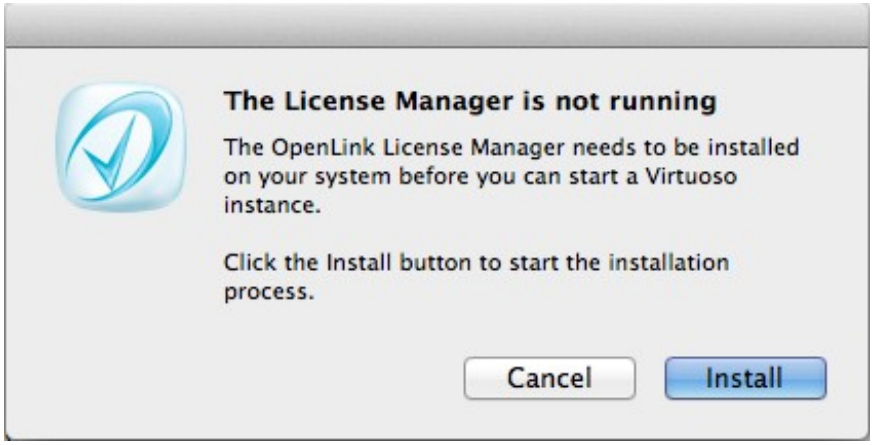
   menu.

Figure 2.20. Mac OS X installer: Drag Virtuoso Application



5. If this is a first time installation of OpenLink components, a dialog will be prompt you to install the OpenLink License Manager.

Figure 2.21. Mac OS X installer: Drag Virtuoso Application

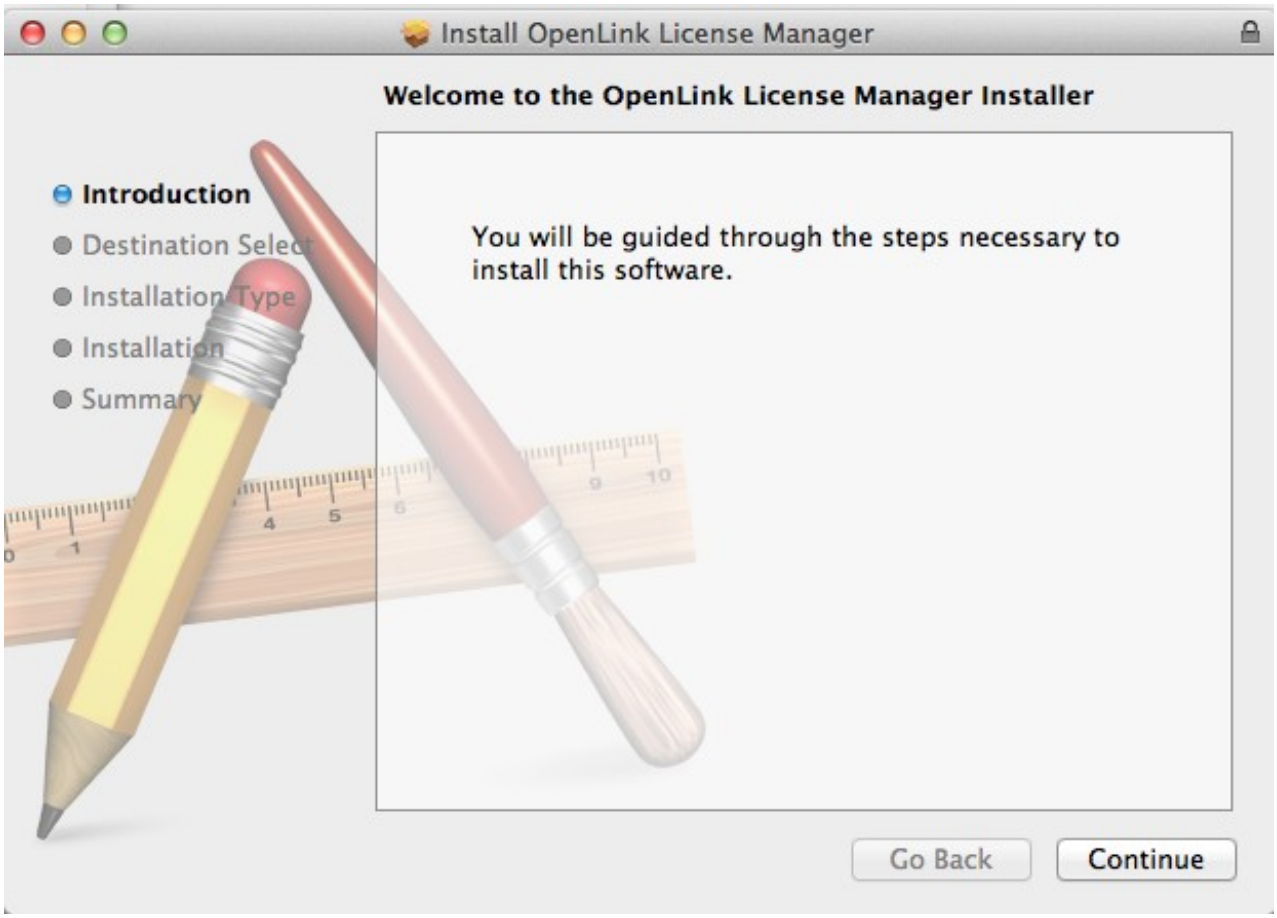


a. Click the

Continue

button to start the installation.

Figure 2.22. Mac OS X installer: Drag Virtuoso Application



b. Select the disk the installation should be performed on.

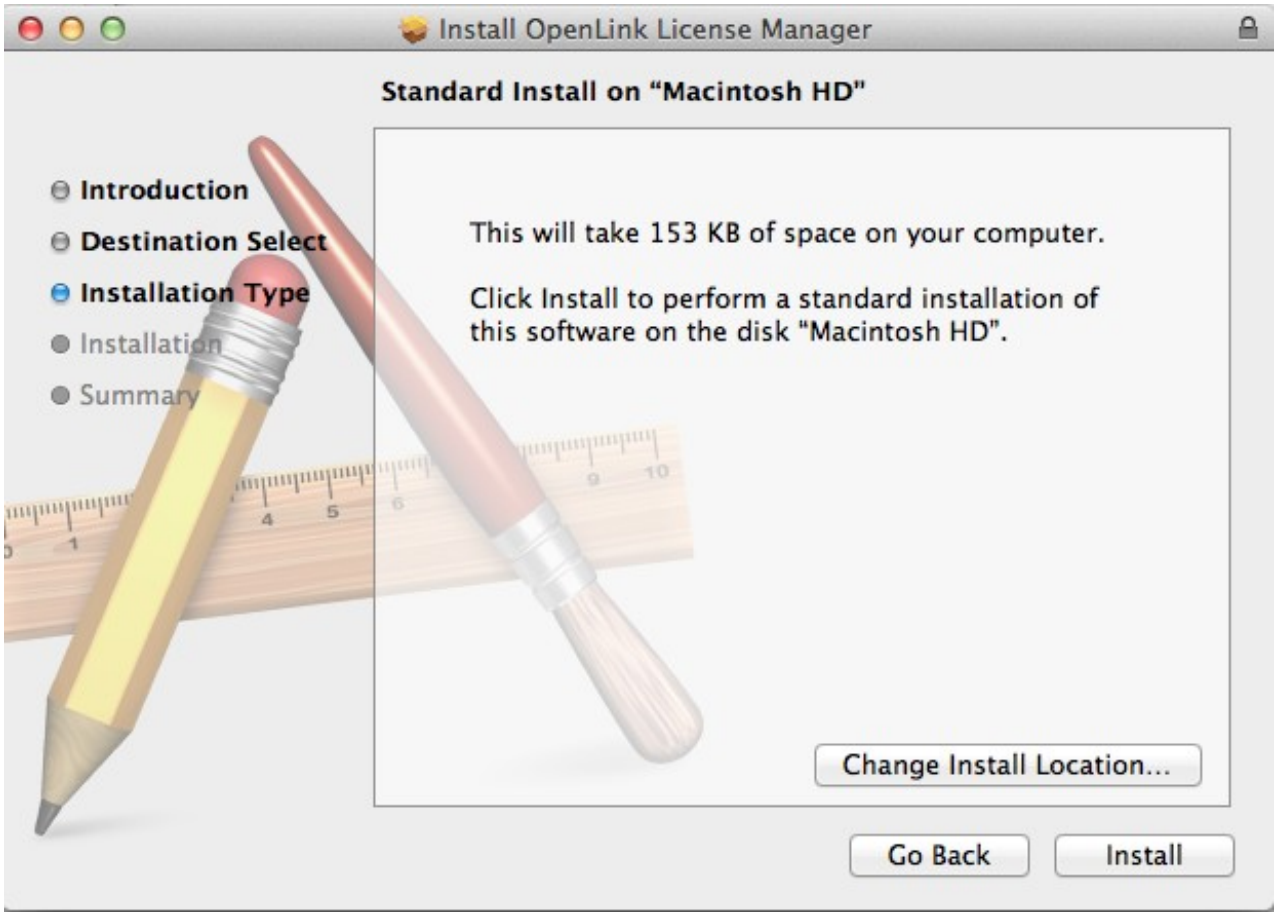
Figure 2.23. Mac OS X installer: Drag Virtuoso Application

c. Click

Install

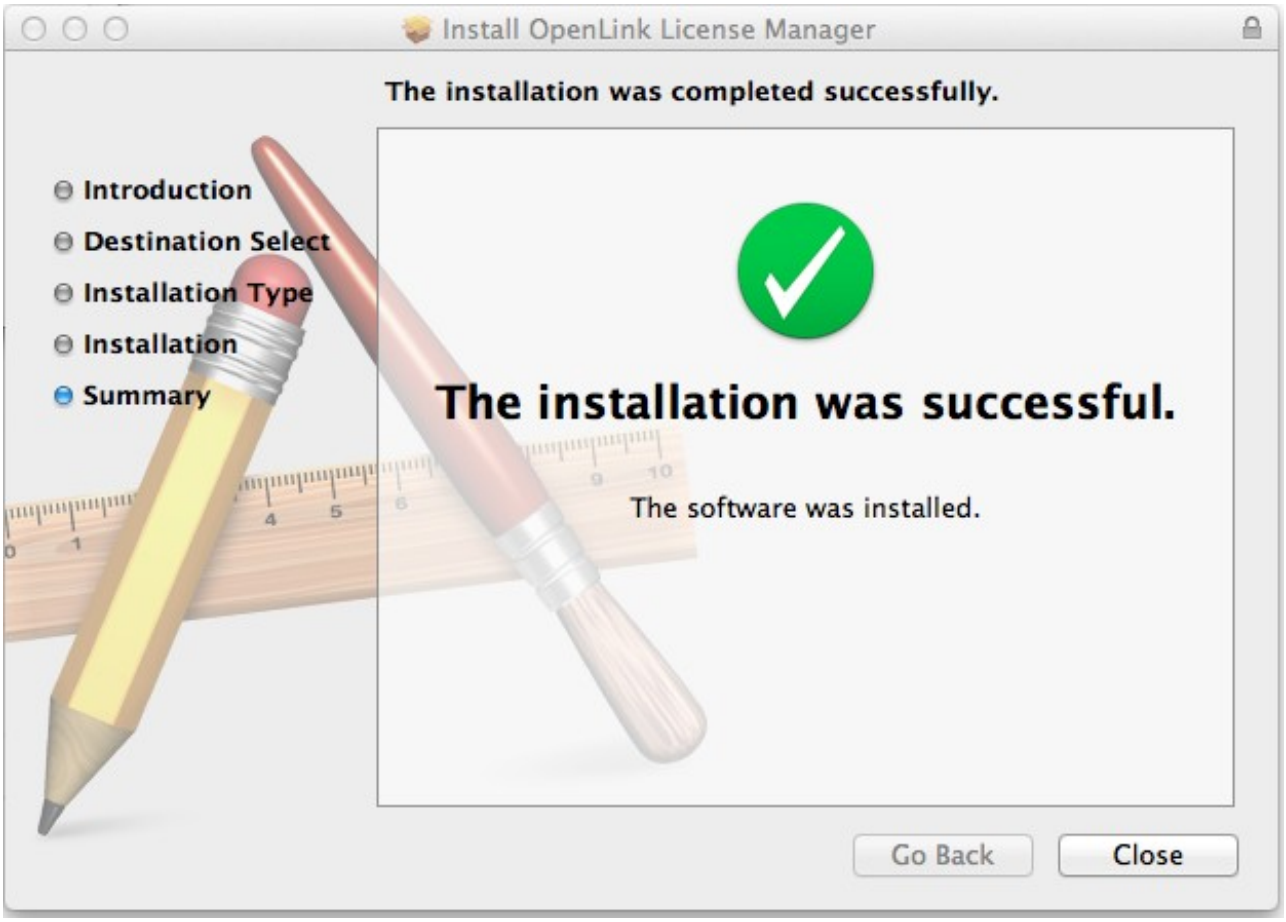
to commence the installation of the License Manager components.

Figure 2.24. Mac OS X installer: Drag Virtuoso Application



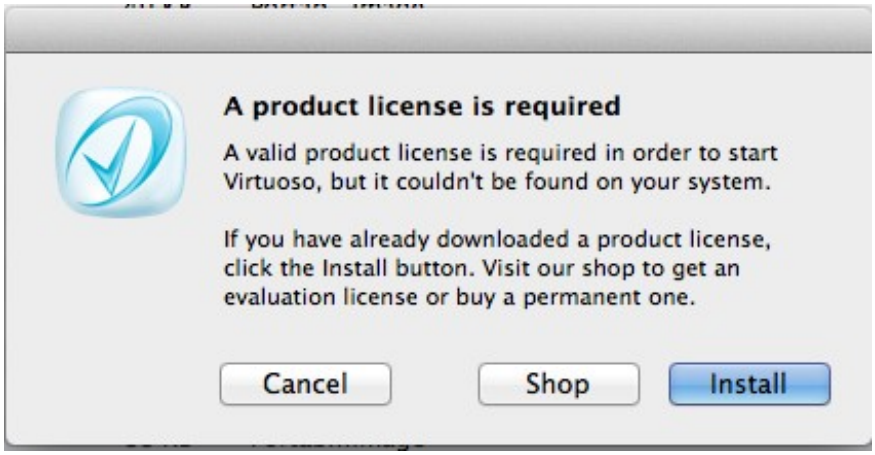
d. The following dialog will be displayed to confirm successful installation of the License Manager, which will be automatically started and ready for use upon installation.

Figure 2.25. Mac OS X installer: Drag Virtuoso Application



6. If a license file is not already in place, you will be prompted to locate an available license file or to purchase one for installation. Click the Install button to install the license file of your choice.

Figure 2.26. Mac OS X installer: Drag Virtuoso Application



   a. Navigate to the location of the license file to be installed, and click the

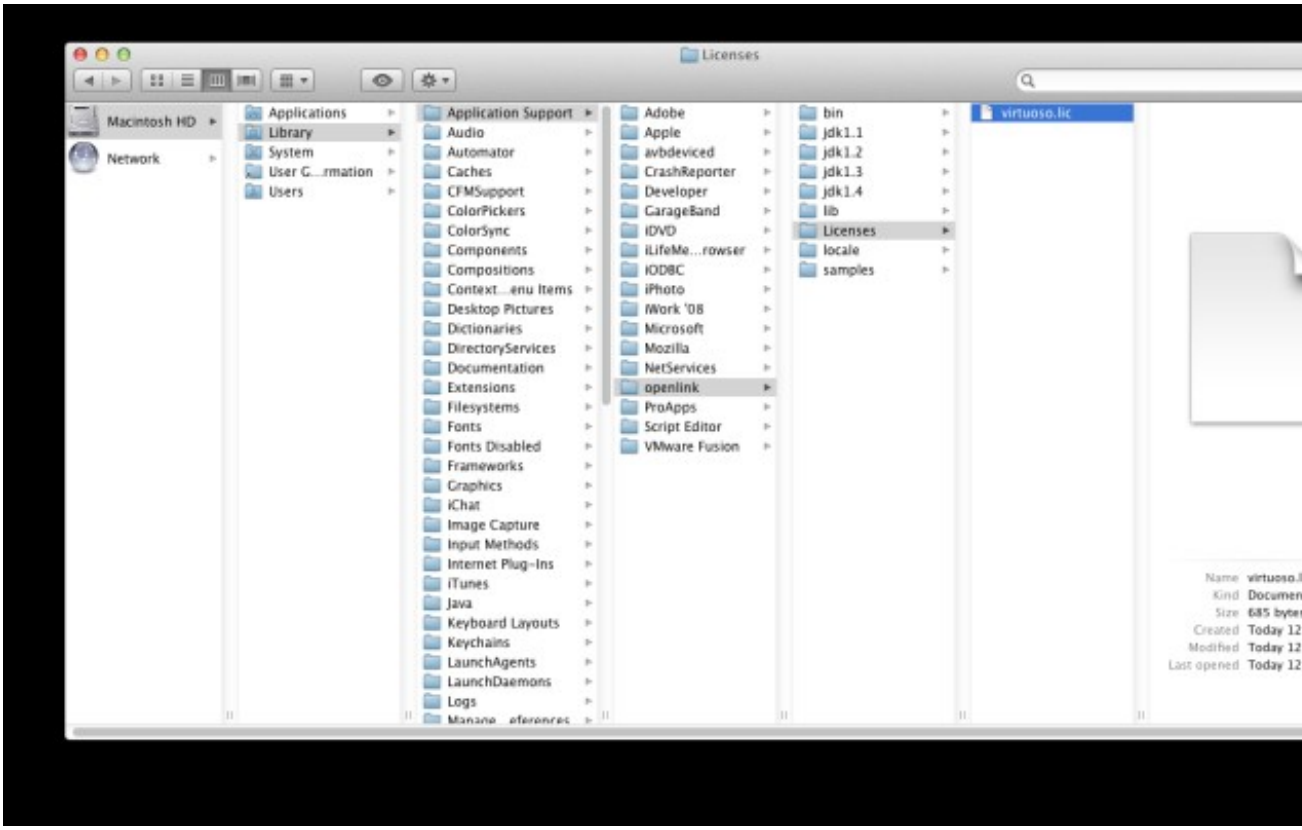
   Open

   button. The installer will move it into the

   /Library/Application Support/openlink/Licenses/

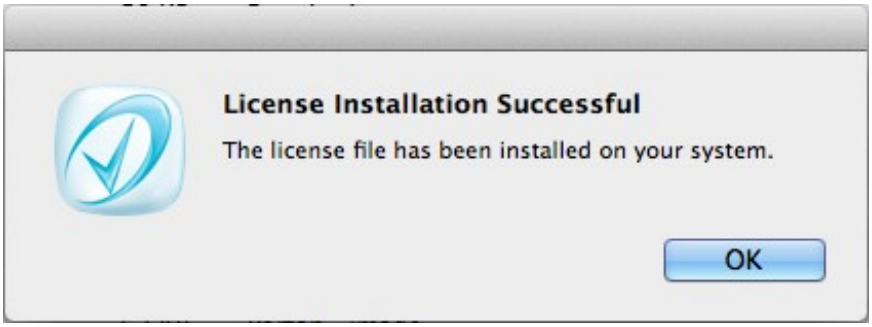
   folder.

Figure 2.27. Mac OS X installer: Drag Virtuoso Application



b. A dialog confirming the successful installation of the license file will be displayed.

Figure 2.28. Mac OS X installer: Drag Virtuoso Application

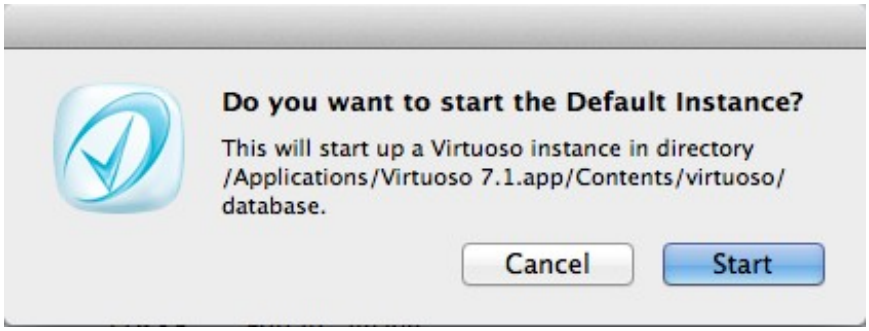


7. The Default database instance can now be started by clicking the

   Start

   button.

Figure 2.29. Mac OS X installer: Drag Virtuoso Application



8. The Virtuoso server is now ready for use, and the browser-based Virtuoso Conductor administration interface may be accessed by selecting

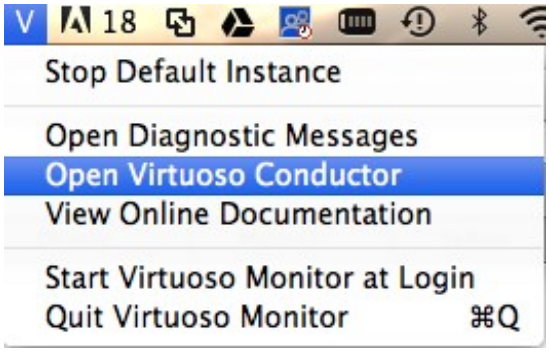
   Open Virtuoso Conductor

   from the

   V

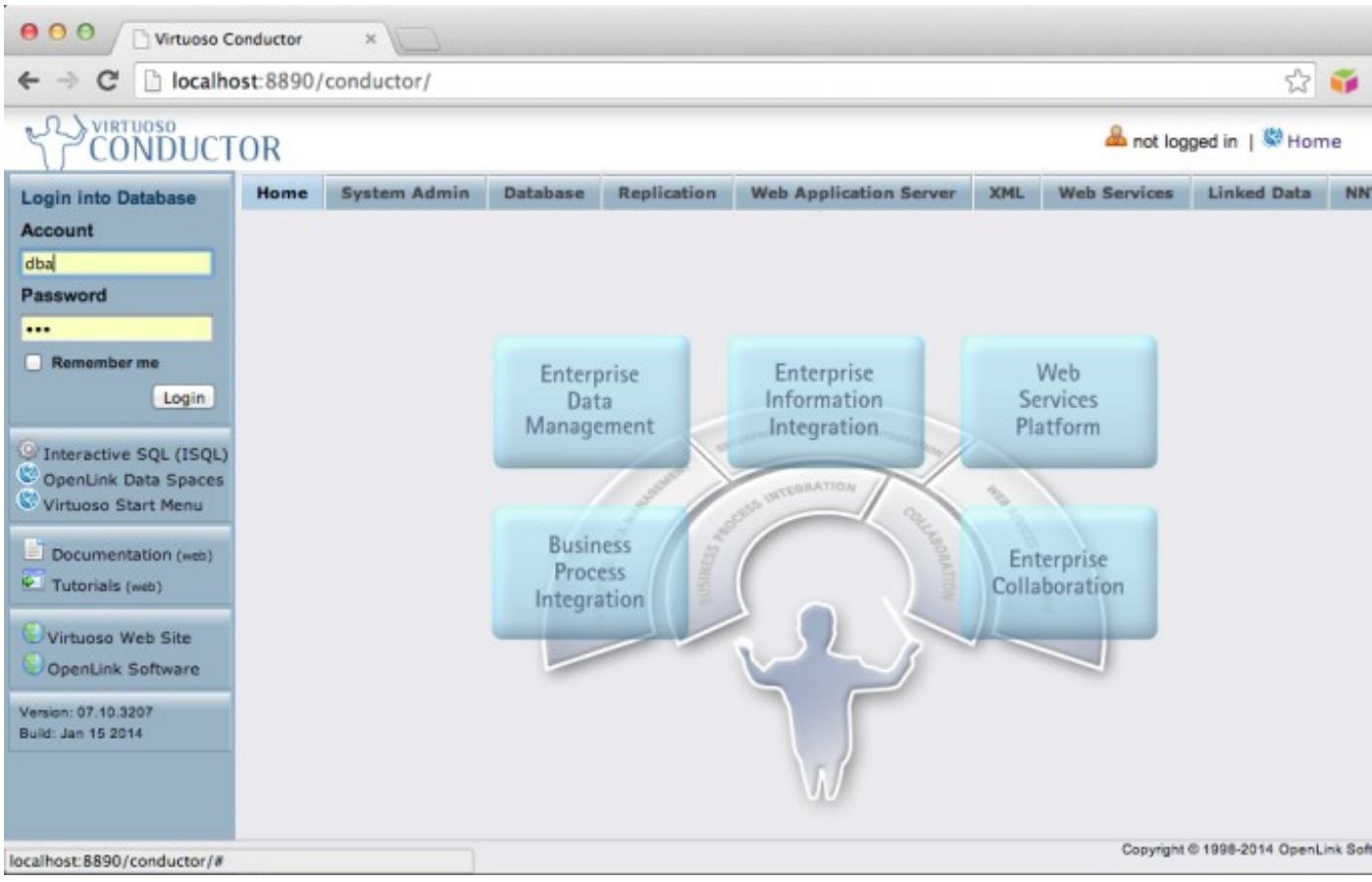
   menu.

Figure 2.30. Mac OS X installer: Drag Virtuoso Application



9. The Virtuoso Conductor may also be accessed by manually loading the URL <http://example.com/conductor> (or <http://full-hostname.example.com:8890/conductor> from a remote host) into any web browser.

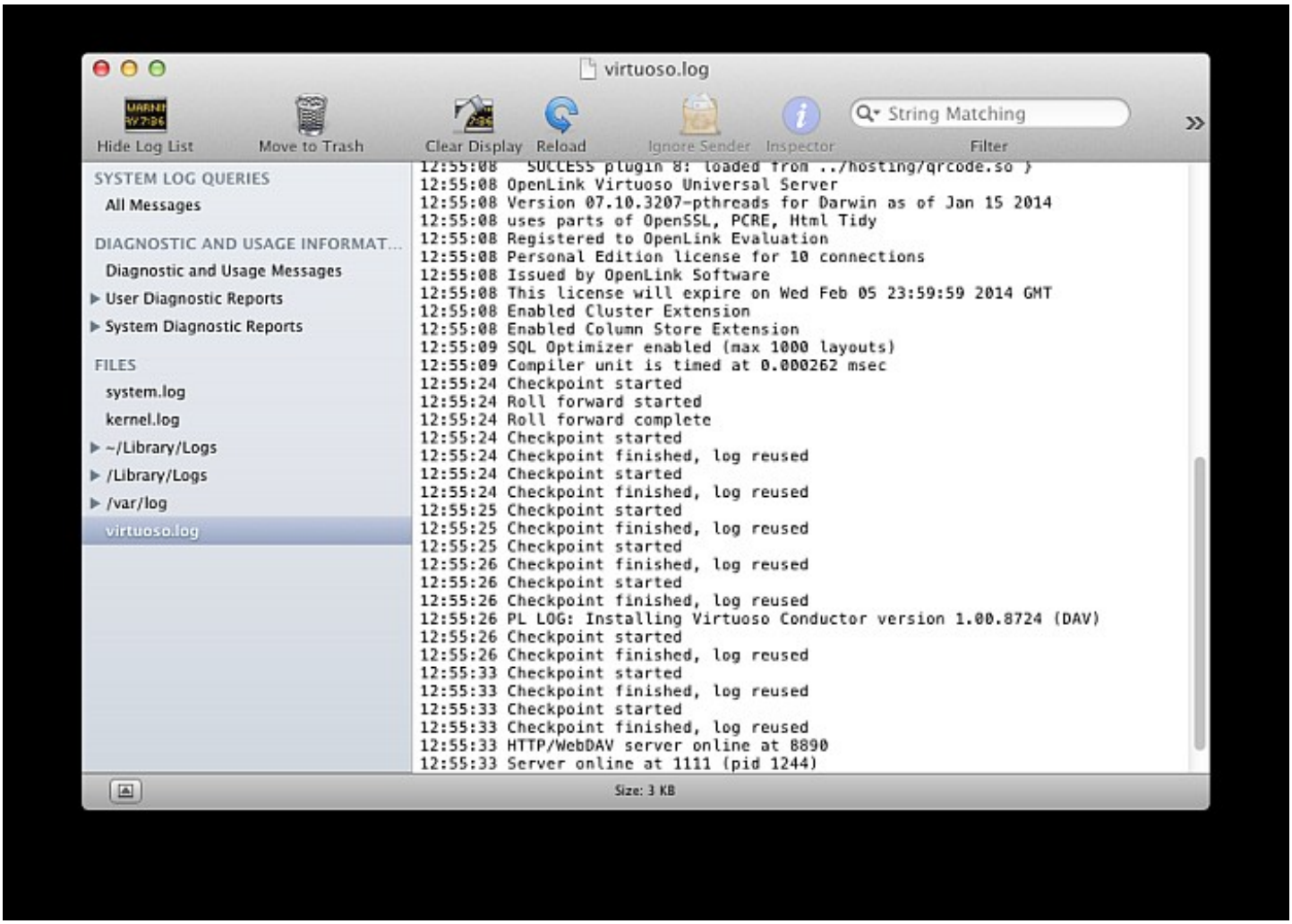
Figure 2.31. Mac OS X installer: Drag Virtuoso Application



Troubleshooting

Many issues may be quickly diagnosed and resolved by reviewing Virtuoso's `.log` file, typically located alongside the active `.db` file.

Figure 2.32. Mac OS X installer: Troubleshooting



The default instance log file may be opened in Mac OS X's Console.app by:

- selecting

  Open Diagnostic Messages

  from the

  V

  menu
- right-clicking on the

  Virtuoso 7.1.app

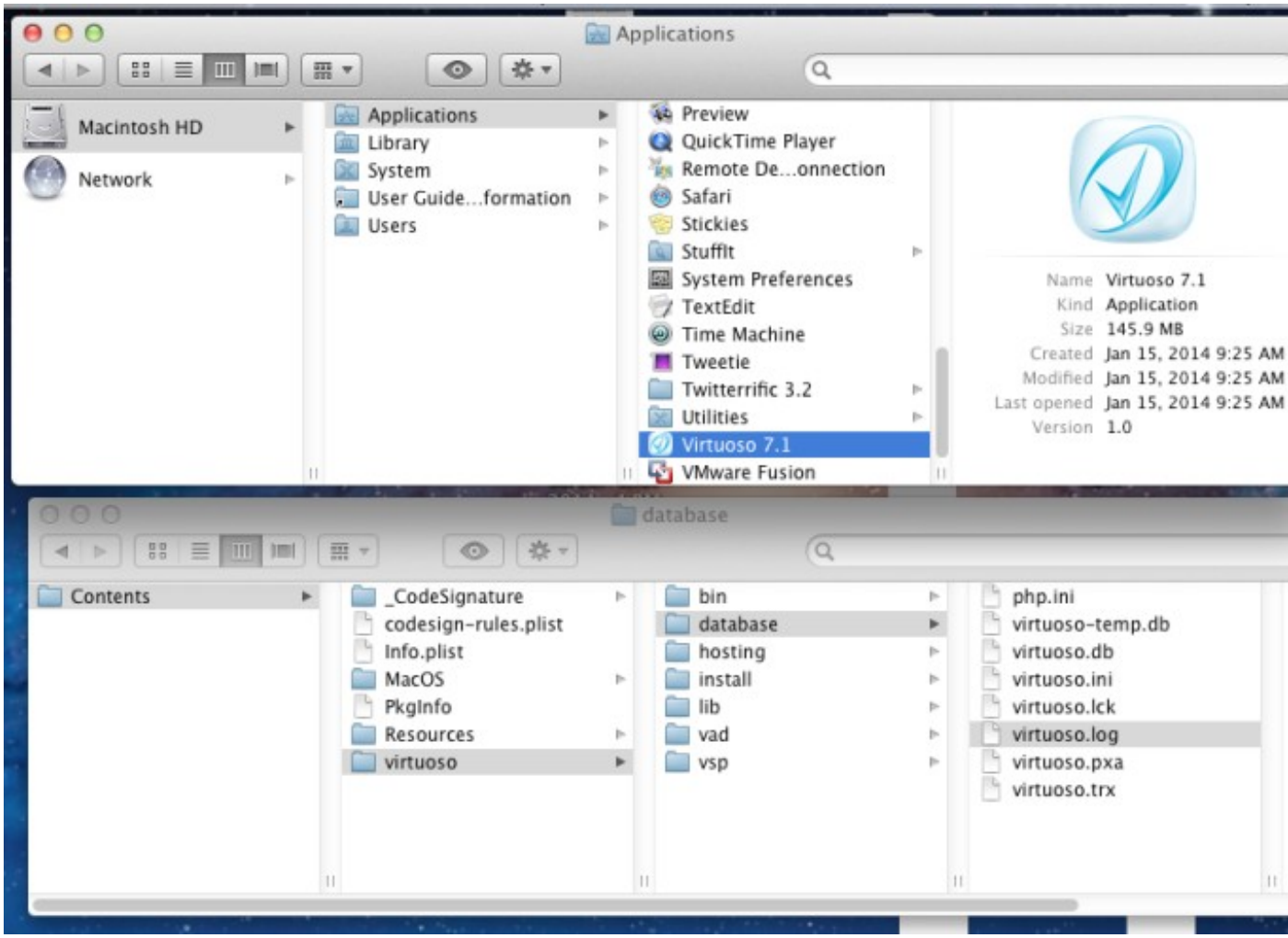
  icon, select

  View Package Contents

  , and then drilling down to

Contents -> virtuoso -> database -> virtuoso.log

Figure 2.33. Mac OS X installer: Troubleshooting



- executing the following command in

  Terminal.app

  :

```
open /Applications/Virtuoso\ 7.1.app/Contents/virtuoso/database/virtuoso.log
```

Missing license file

Make sure that the license file, always named `virtuoso.lic` is in the correct location, as described above -- `/Library/Application Support/openlink/Licenses/.`

Restarting the License Manager may be necessary. Rebooting should always result in a running License Manager, but you can also use the *Terminal.app* :

```
cd /Library/Application\ Support/openlink/bin
./oplmgr -fd /Library/Application\ Support/openlink/Licenses
```

Expired license file

Acquire a non-expired license file, by purchase or conversation with Technical Support or your Account Manager, and replace the expired file with the non-expired, as documented .

Cannot contact the License Manager

Rebooting should always result in a running License Manager. You can use *Terminal.app* to check whether it's running:

```
ps -ef | grep oplmgr | grep -v grep
```

If no process is listed, these two commands should get it running:

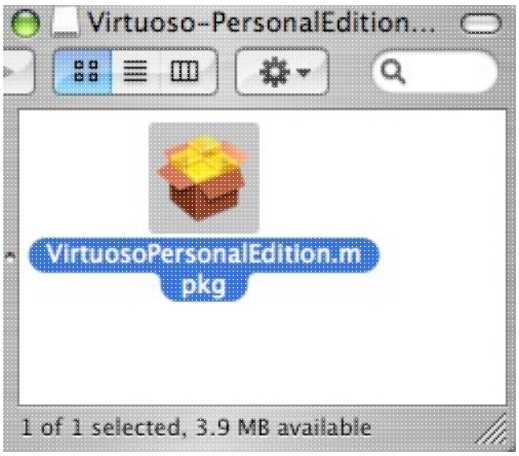
```
cd /Library/Application\ Support/openlink/bin
./oplmgr -fd ../Licenses
```

If the process is listed, there are many possible causes, including firewall or other networking issues. Please allow our Support Team to assist you in diagnosis and resolution.

2.4.3. Installing Virtuoso 6 or earlier

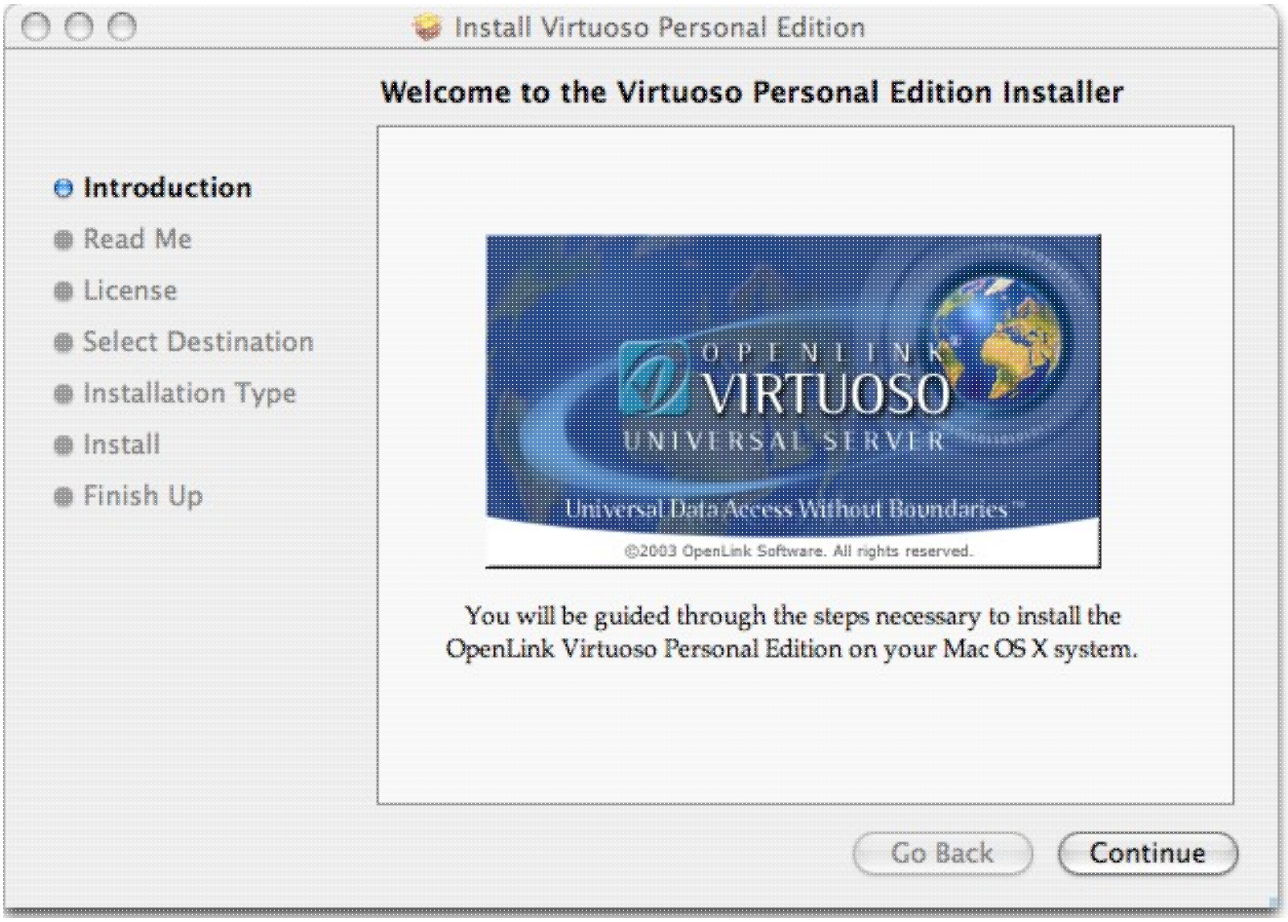
Open the Virtuoso Personal Edition Disk Image file ntvpz2zz.dmg by either double clicking on the DMG file in the Finder windows or using the "open" command from a Terminal window, to mount the Disk Image containing the VirtuosoPersonalEdition.mpkg installer.

Figure 2.34. Mounting the Disk Image



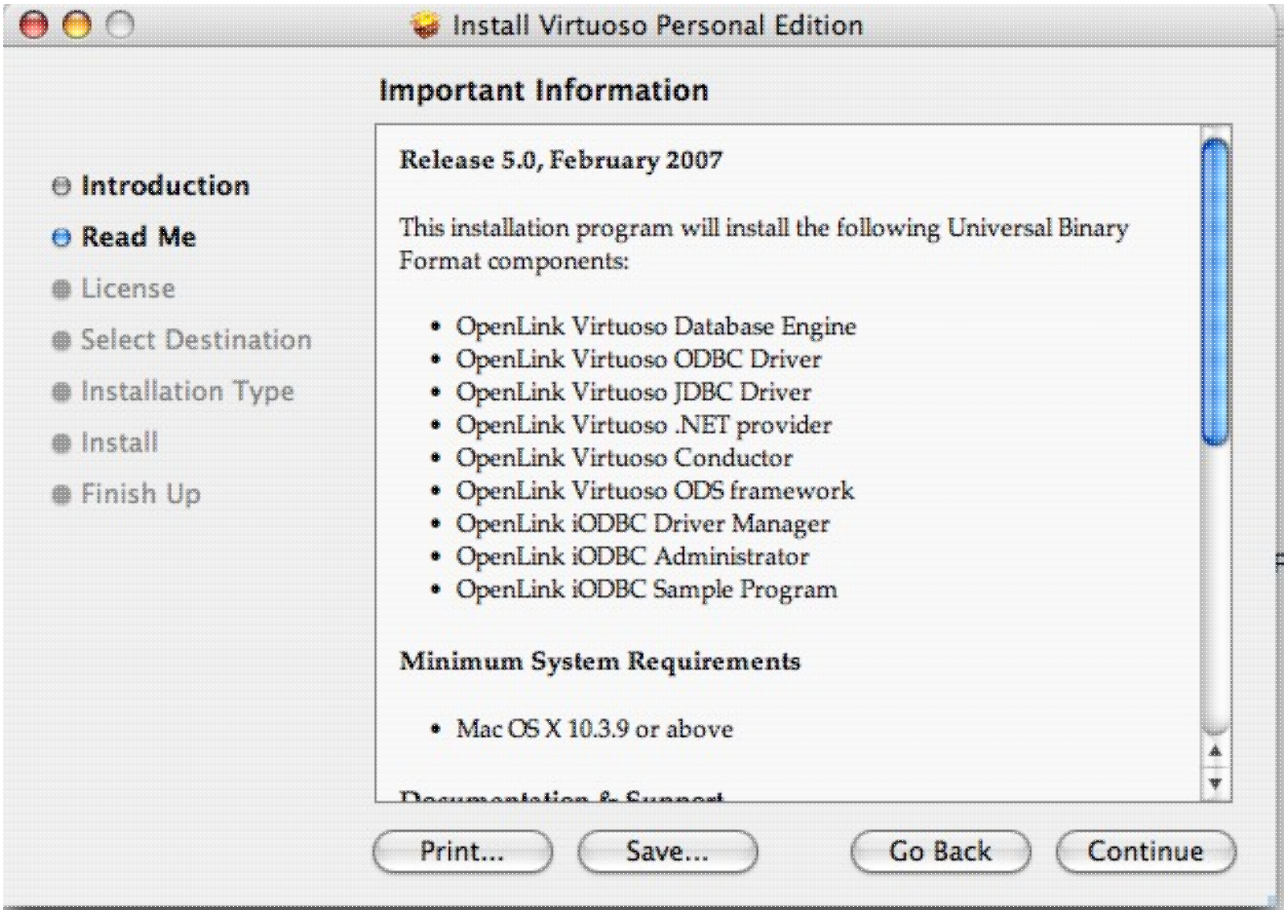
The installer will load the Introduction page.

Figure 2.35. Installer Introduction Page



Review the Readme page for any important information concerning the product installation.

Figure 2.36. Important Information

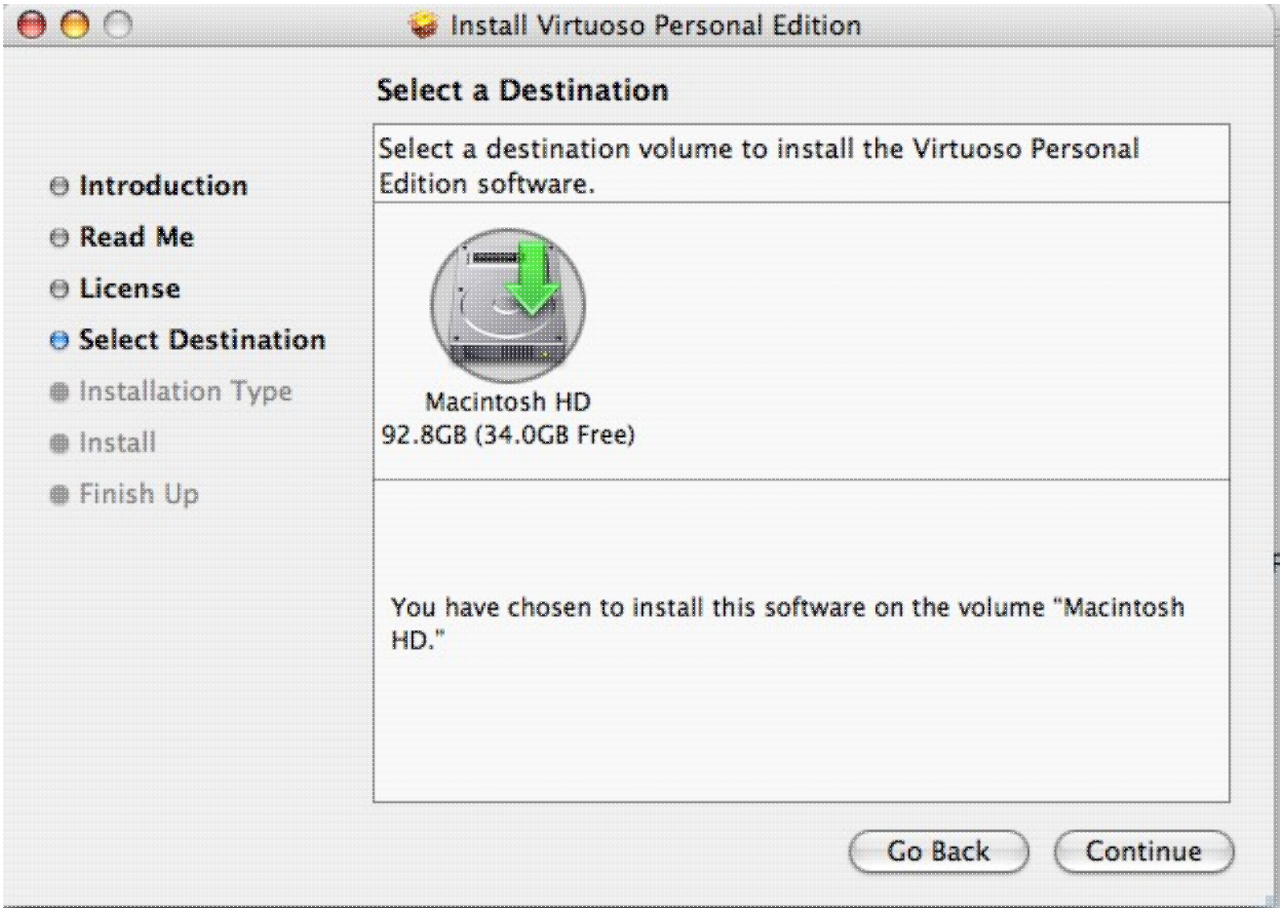


Read the license agreement and "agree" to the terms.

Figure 2.37. Software License Agreement

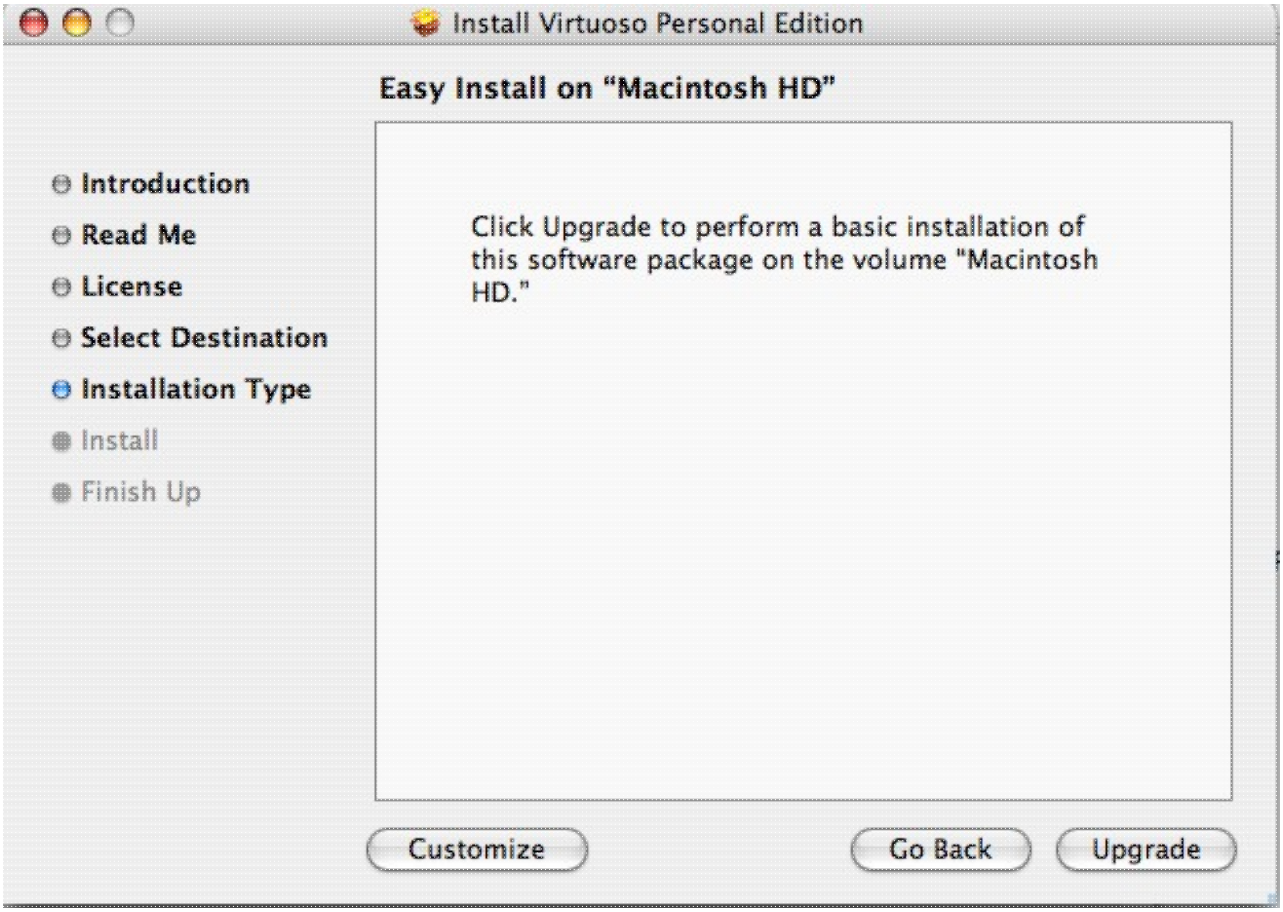
Select the Destination volume the software is to be installed on.

Figure 2.38. Select Destination



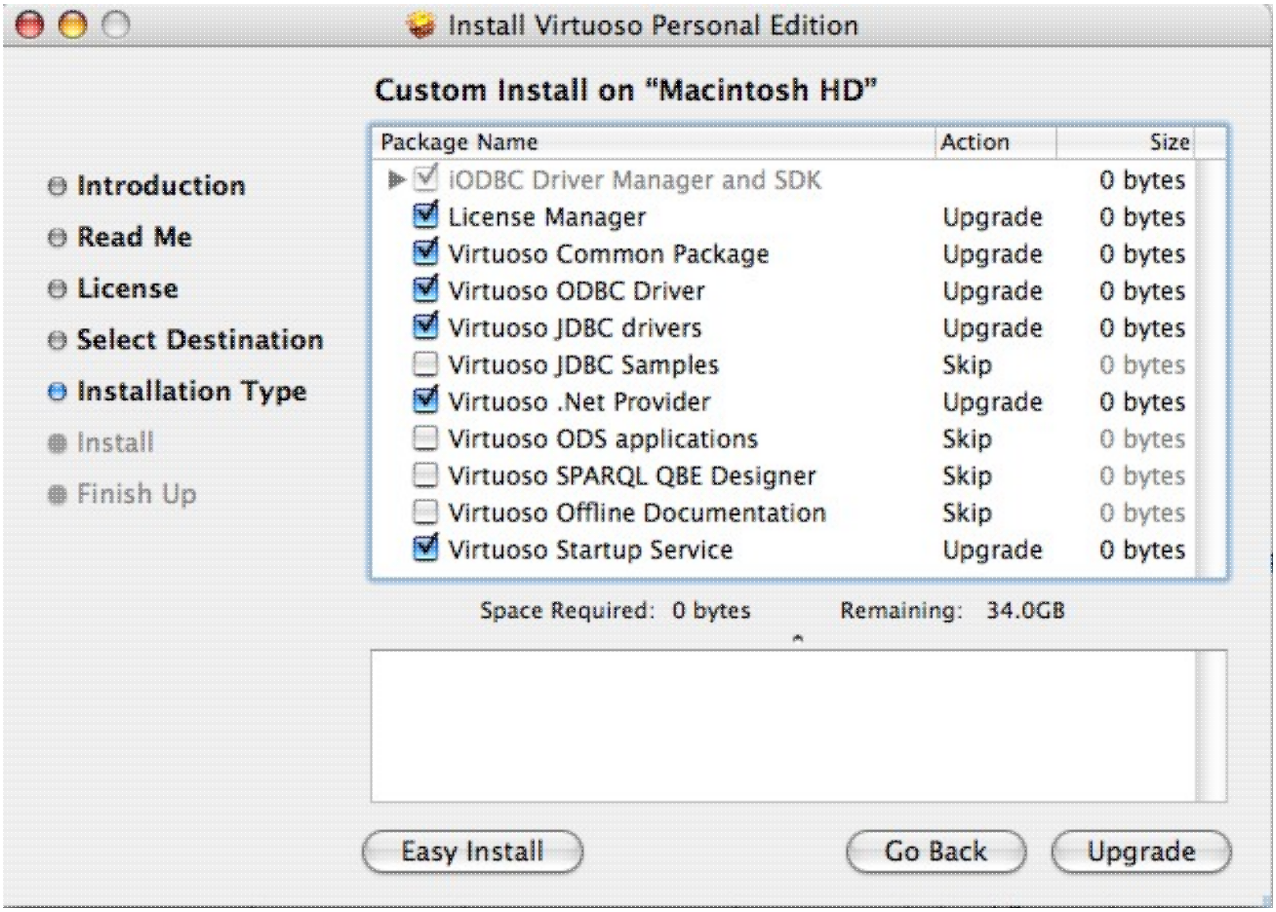
Choose to perform and Easy(default) installation or click on the "customize" button to perform a Custom installation.

Figure 2.39. Selecting the Type of Installation



If the "Custom" option was selected in the previous step then select the packages to be installed.

Figure 2.40. Customizing the Installation

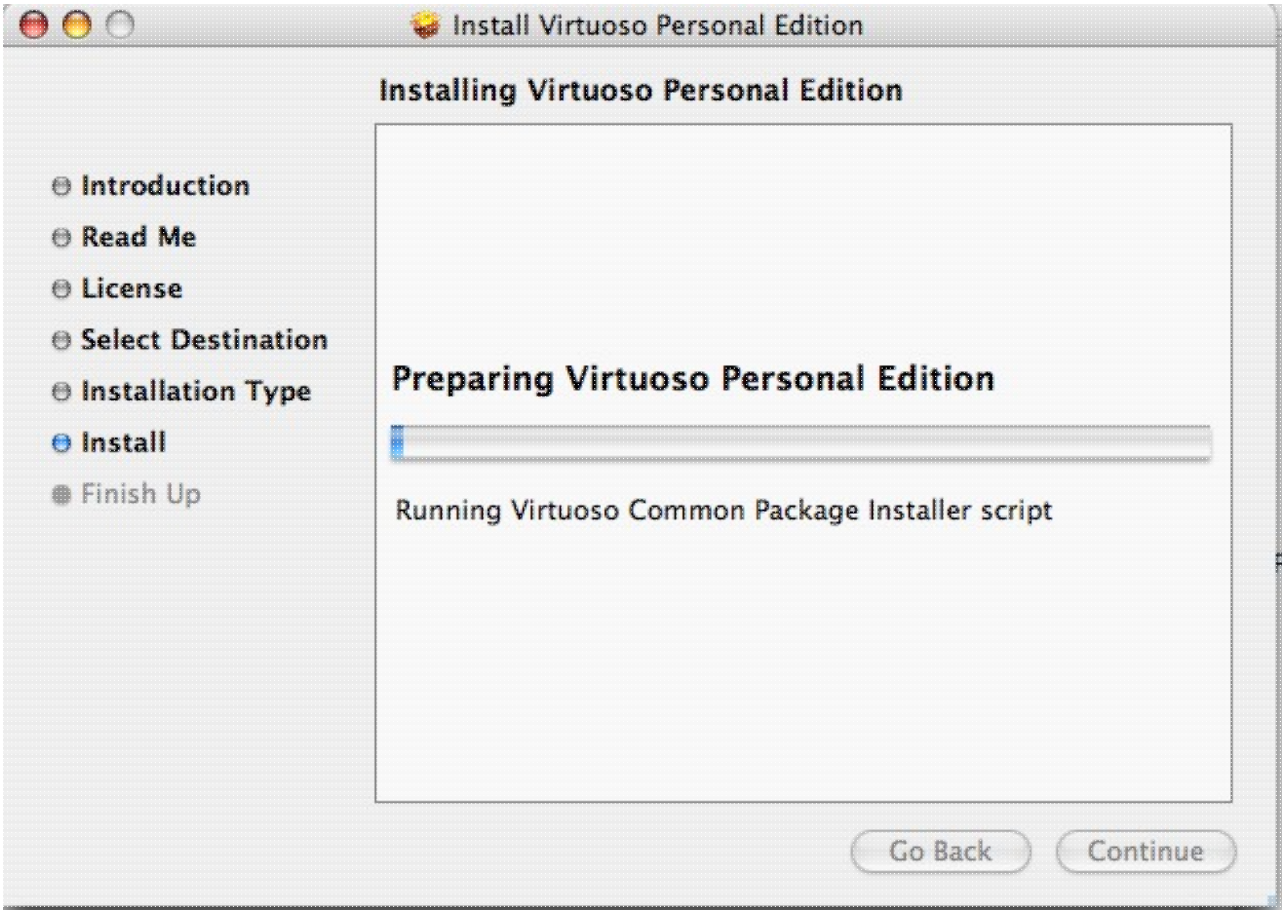


The software must be install by a user with Administrative privileges on the machine.

Figure 2.41. Identifying yourself as Administrator

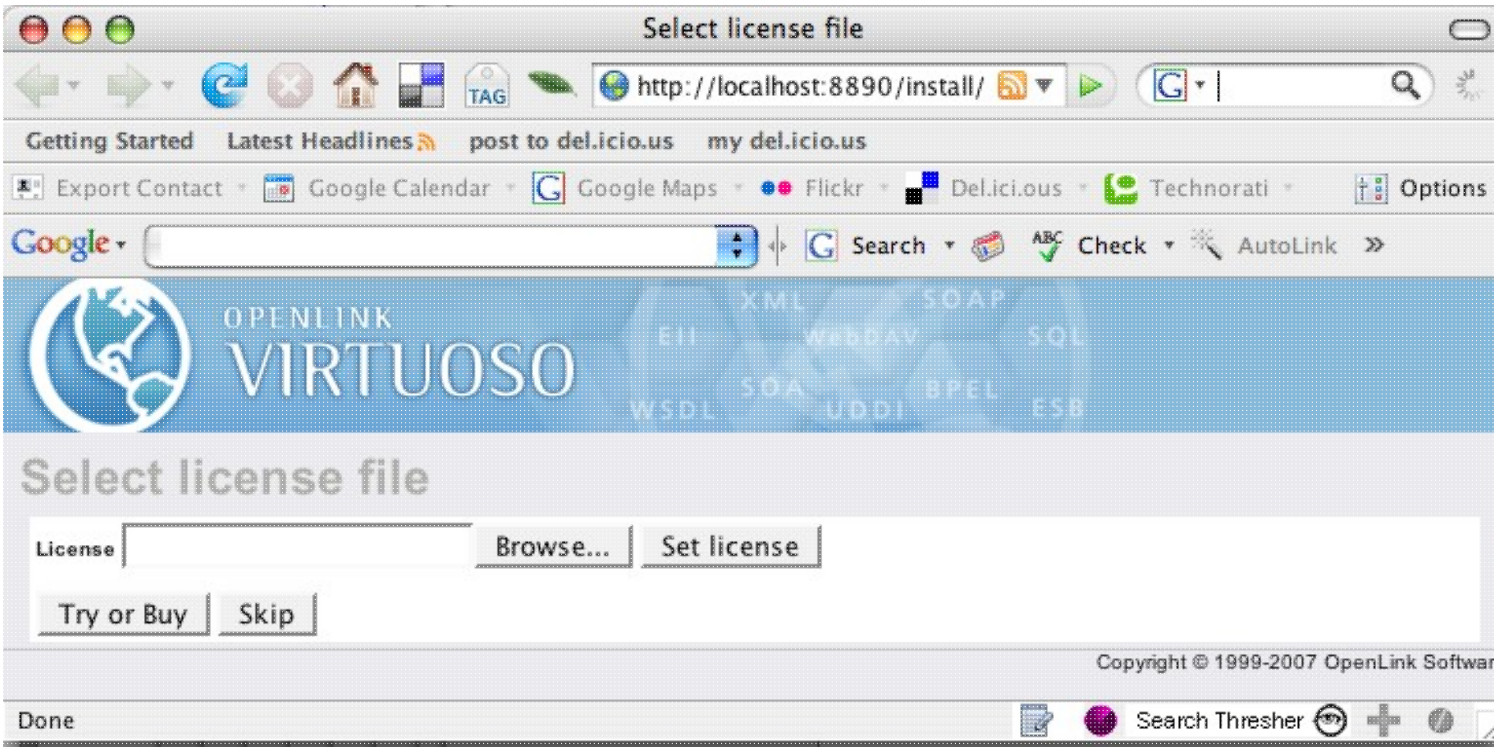
The software components will be installed the status of which is indicated by the progress bar.

Figure 2.42. Preparing ...



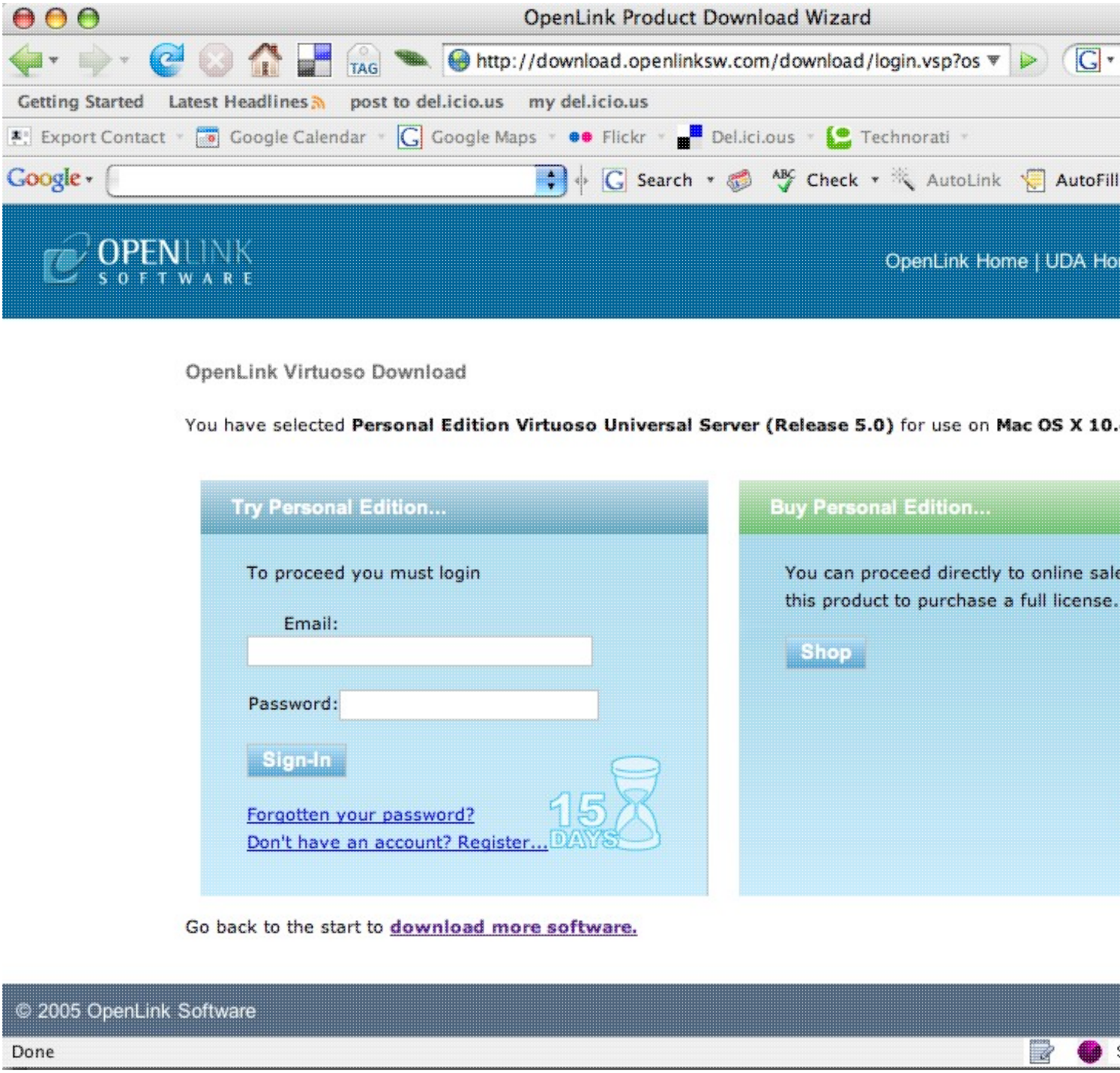
The Virtuoso Server is started at the end of this process. A Web page is loaded to enable you to obtain a trial or full license from the OpenLink online shop:

Figure 2.43. Admin Assistant



Choose to obtain a Trial license for an evaluation or enter the shop site to purchase a full license.

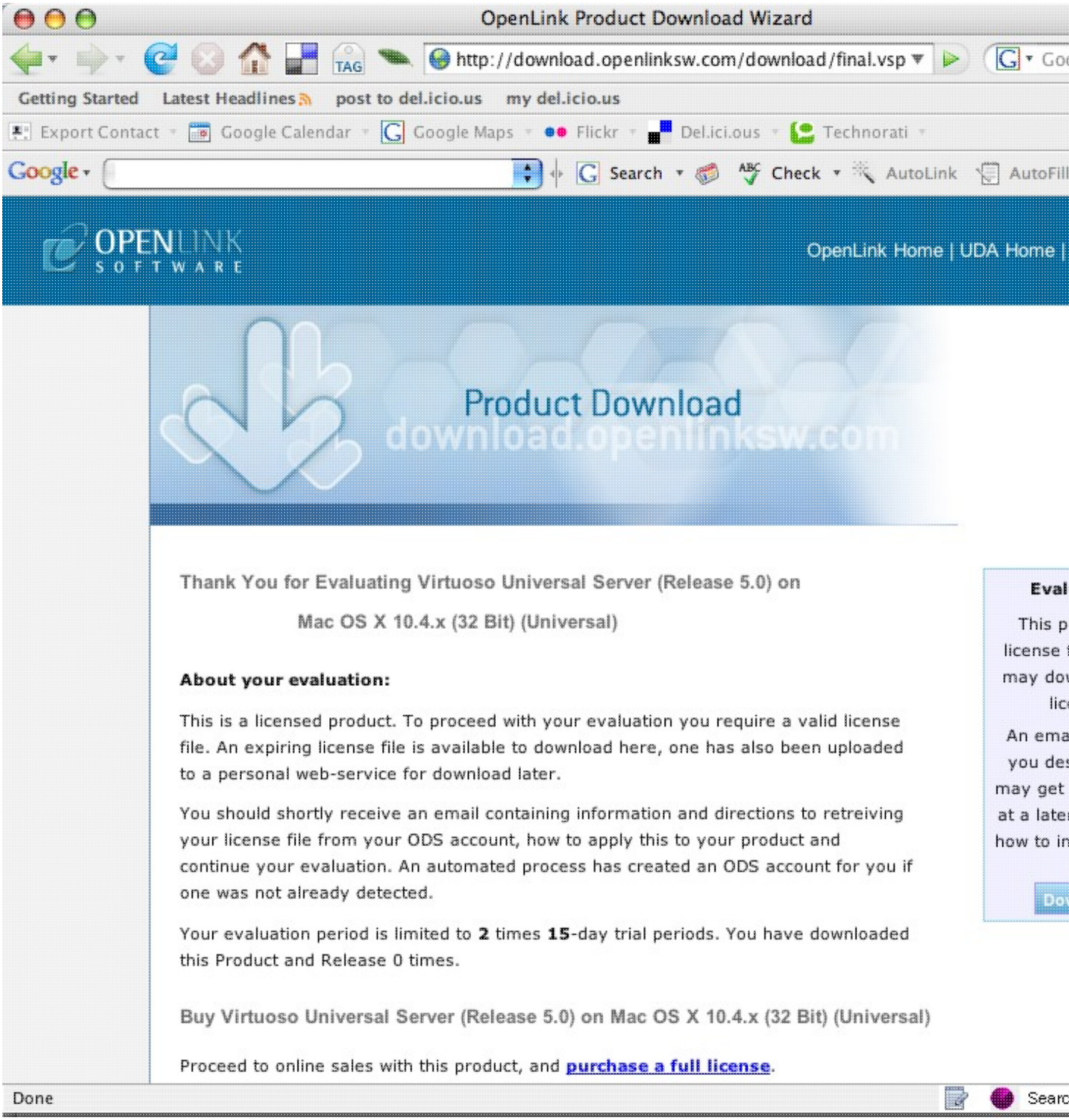
Figure 2.44. The Trial license page



Click on the "Download License" button on the right to obtain the license file immediately. An e-mail will also be sent to your registered email address. This email contains the details of the Virtuoso license and the location of an OpenLink Data Spaces account where you can obtain an additional demo license at a later date.

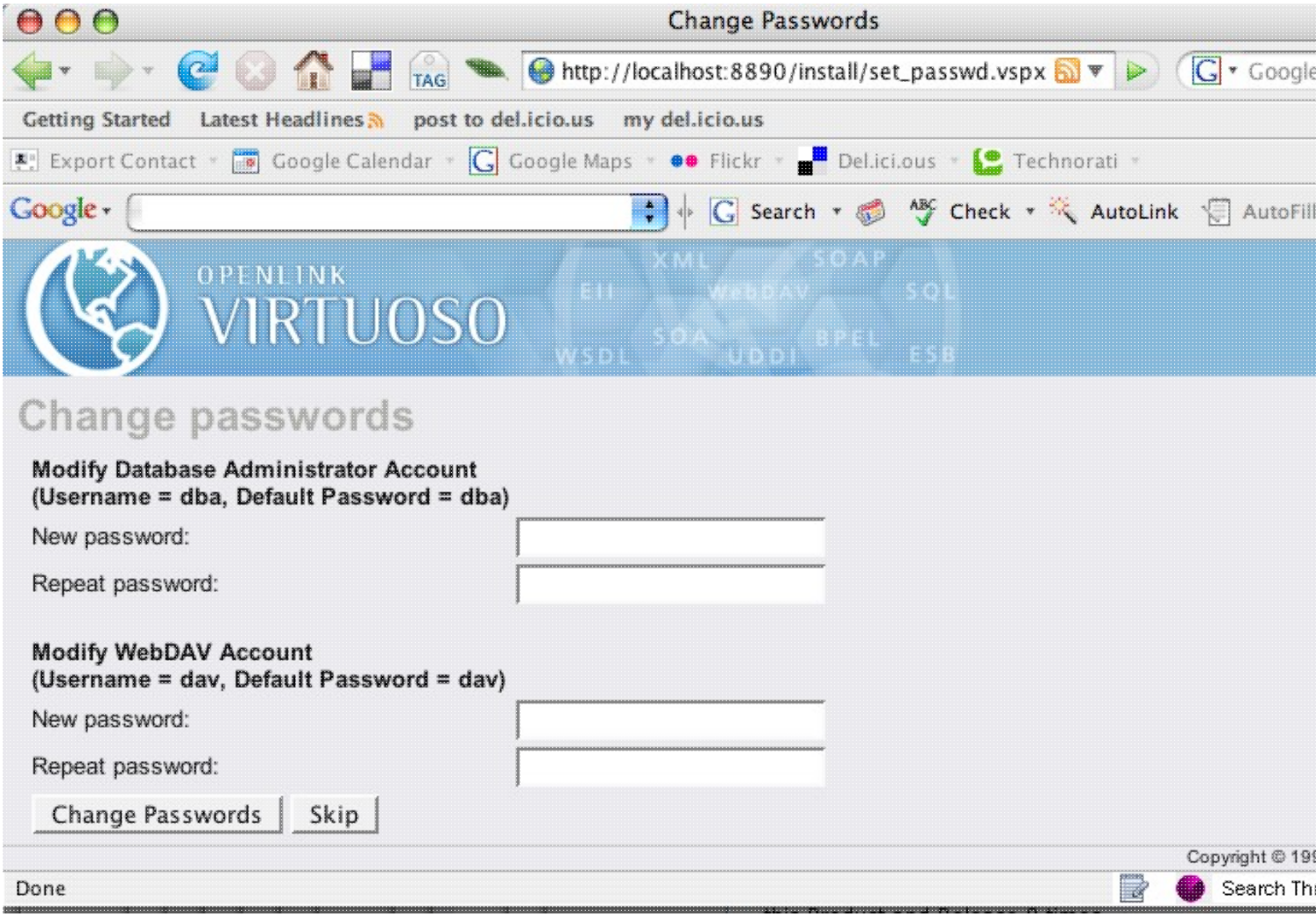
Once a license file is available on your Mac, use the "Browse" button to locate it. Then, use the "Set License" button to initialize the Virtuoso server with it:

Figure 2.45. Download license



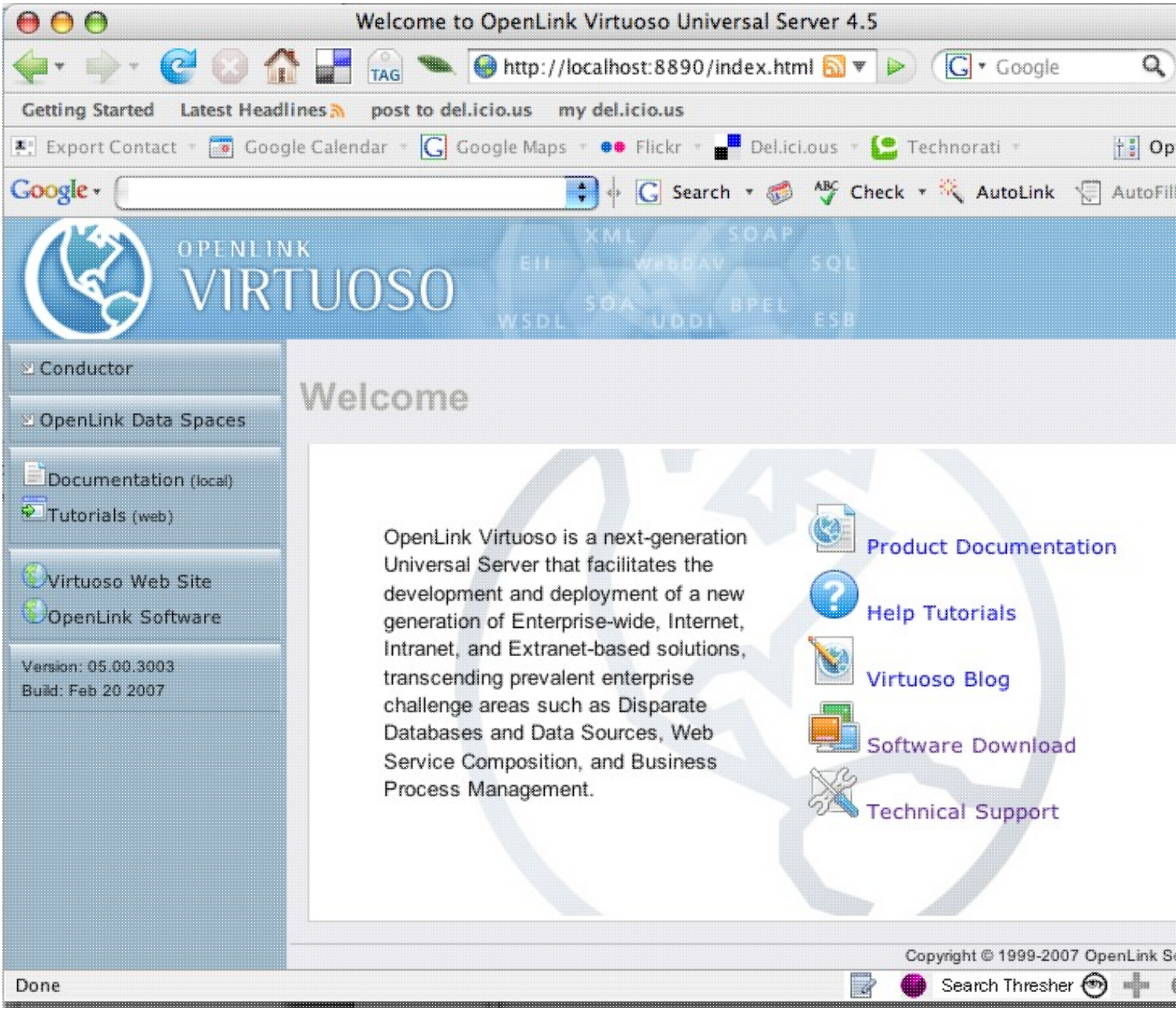
Change the "dba" and "dav" user passwords from their default of "dba" and "dav" to a value of your choice:

Figure 2.46. Change passwords



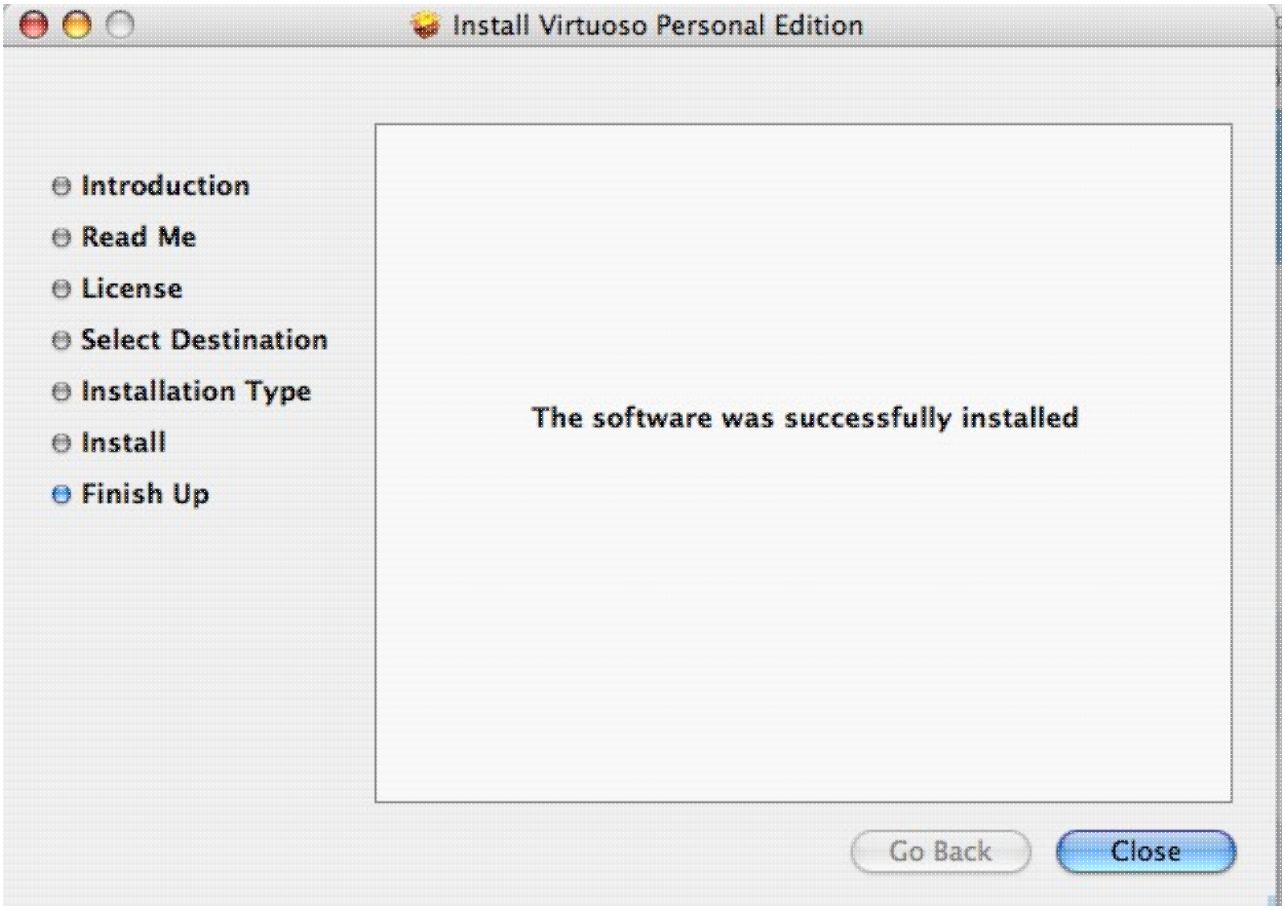
The Virtuoso Server Welcome page will now be displayed. The server is online. This Web based page can be used for general configuration and usage of the Virtuoso Server:

Figure 2.47. Welcome



The Installation is now complete.

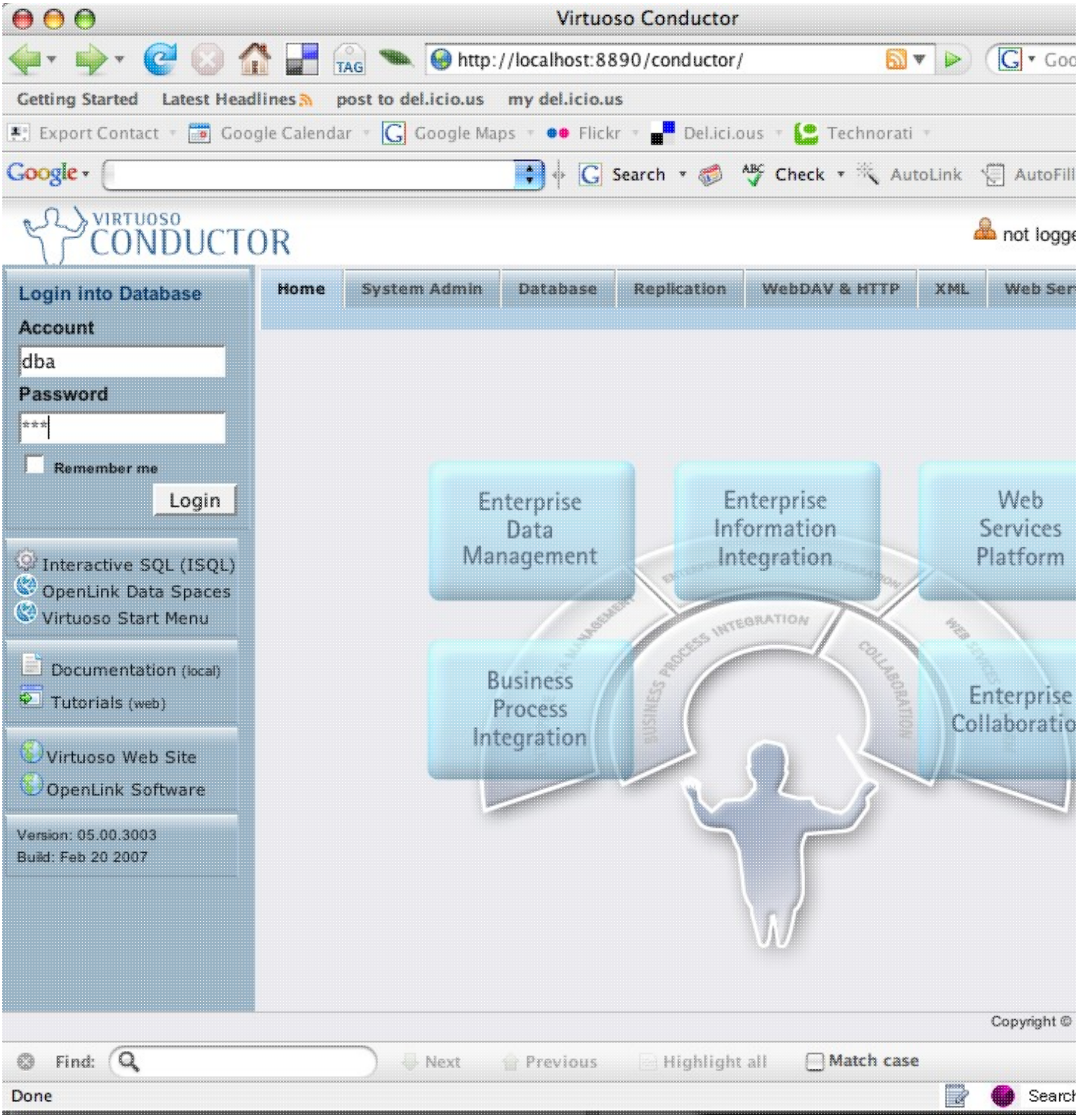
Figure 2.48. Installation complete



2.4.4. Configuration

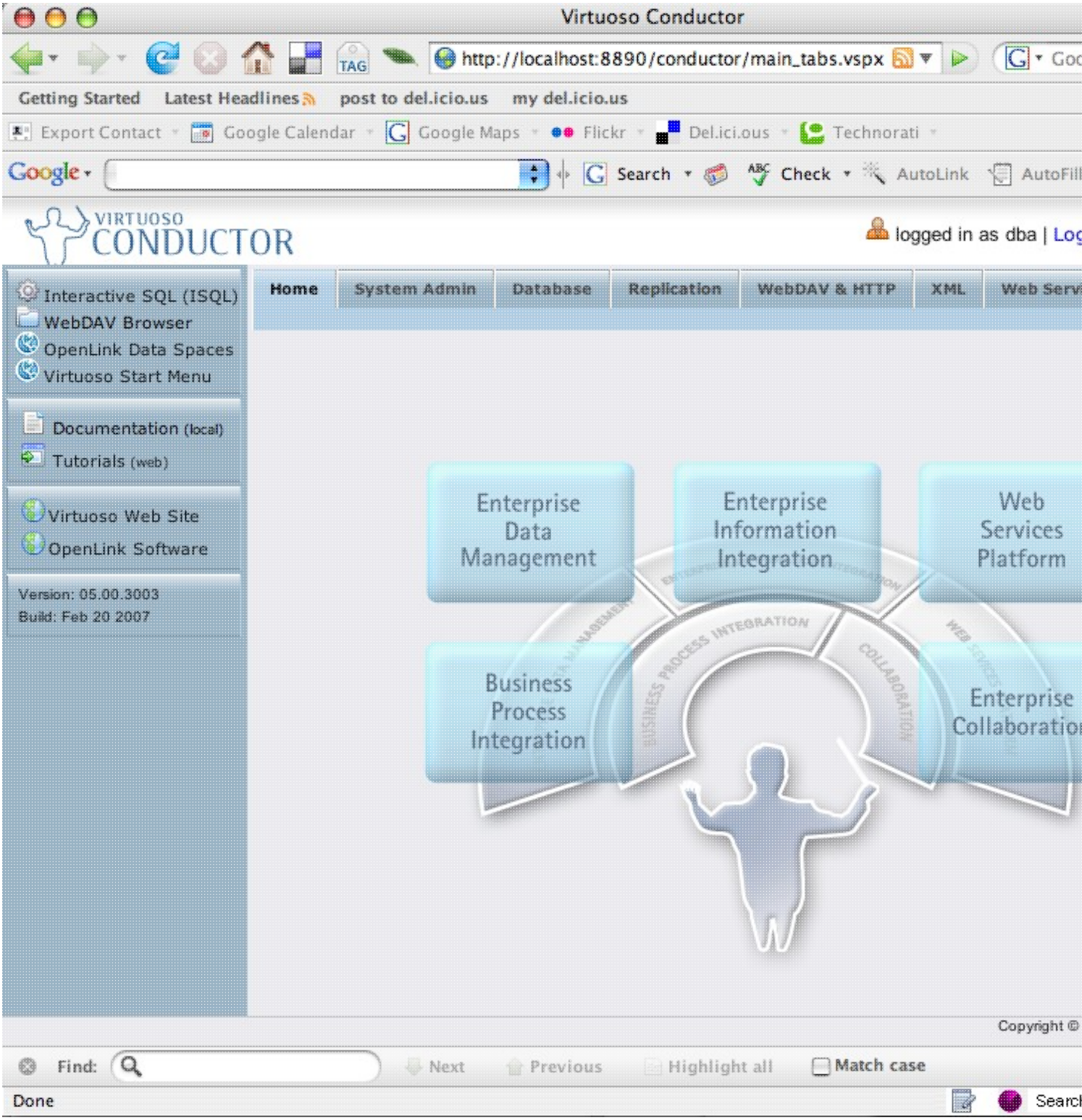
Click on the "Conductor" link in the left frame of the Welcome page to load the Virtuoso Conductor which is the Administration interface for the installation. Enter the "dba" username and password to login.

Figure 2.49. Configuration



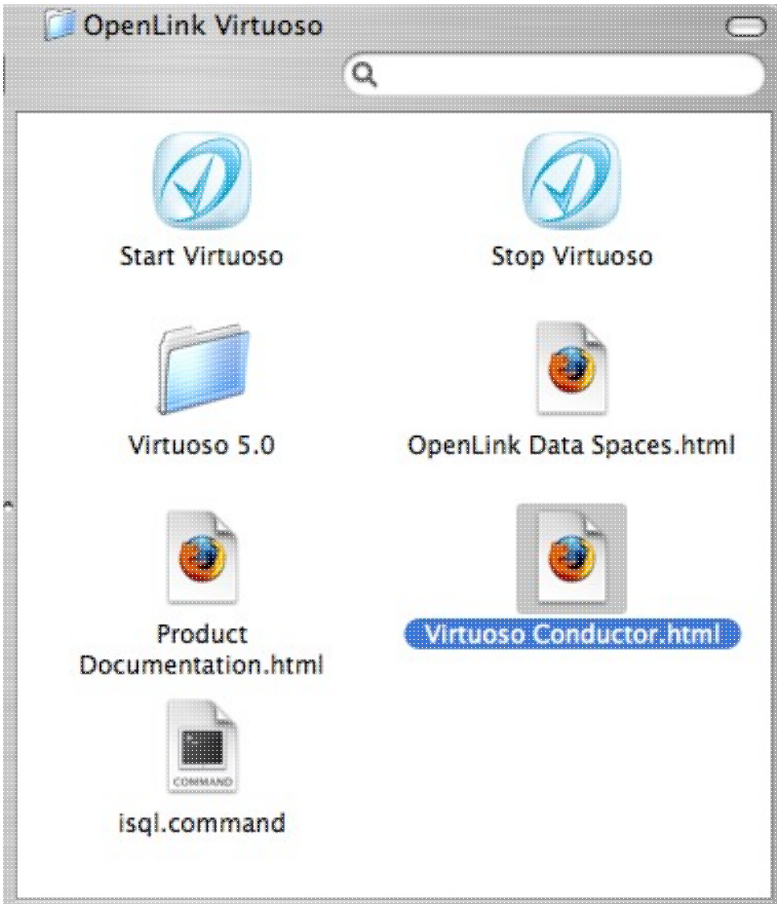
Once logged into the conductor click on any of the "Tabs" along the top to Administer the various features of the Virtuoso Universal Server installation.

Figure 2.50. Various features using Conductor Tabs



Apple scripts for starting and stopping your Virtuoso Universal Server installation are located in the "/Application/OpenLink Virtuoso" folder of the machine.

Figure 2.51. Location of apple scripts



2.4.5. Post Installation

Post-Installation Sanity Check

A quick way to check that the database is running, is to point a browser to the http port. The following example URLs will show the System Manager for the default, and the demo Virtuoso databases:

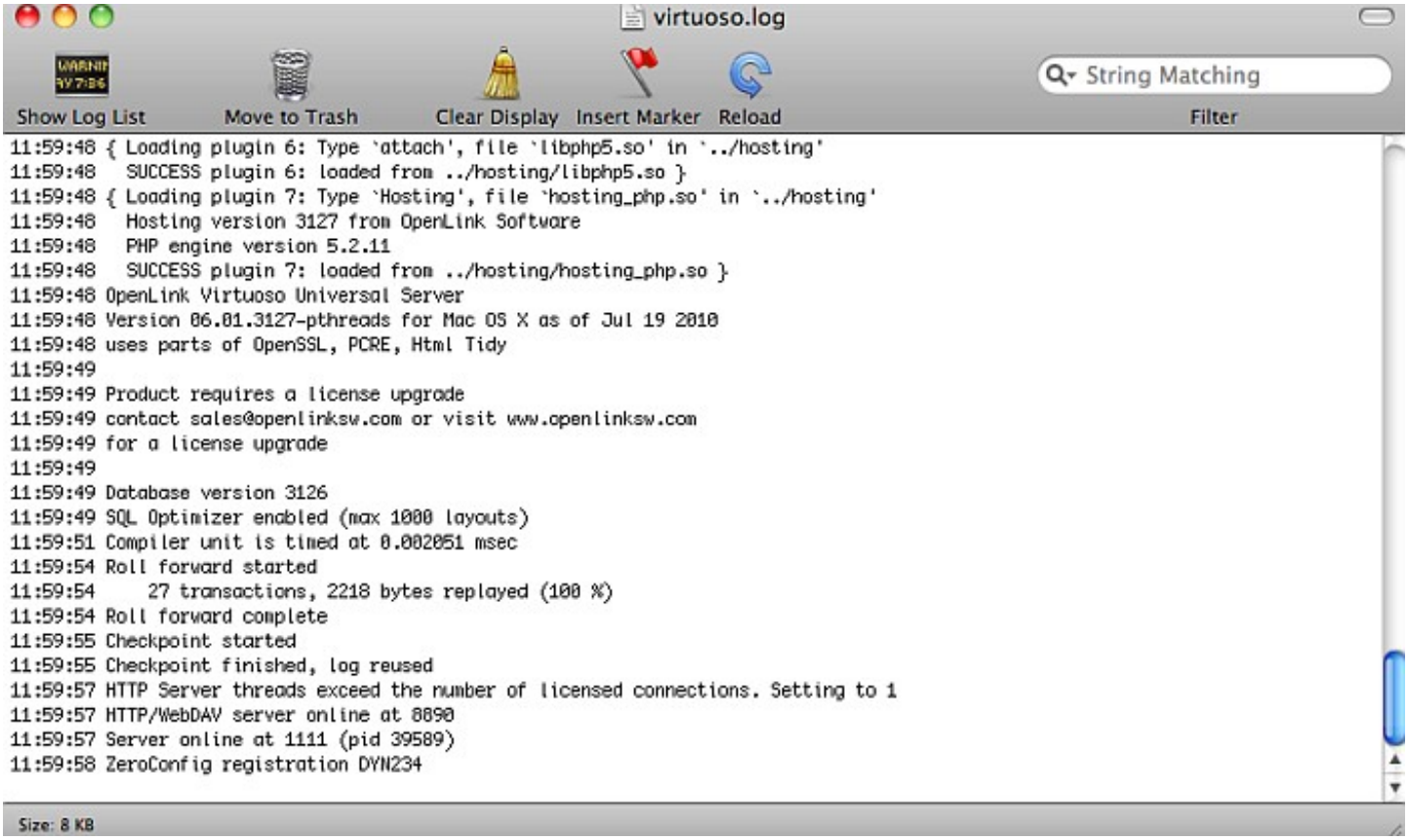
```
http://example.com
http://example.com
http://a_virtuoso_server.org:8890
```

Troubleshooting DB Startup Failures

Install failure

Check the .log file in oplmgr (License Manager) location: `/Library/Application Support/openlink/bin/` , which is for the database and log files etc. in order to review the installed files binaries and databases (defaults) for issues if install fails.

Figure 2.52. The .log file



.lck existance

Check .log in case of .lck issue

Delete the .lck file and repeat the steps from above.

.trx incompatiblity issue

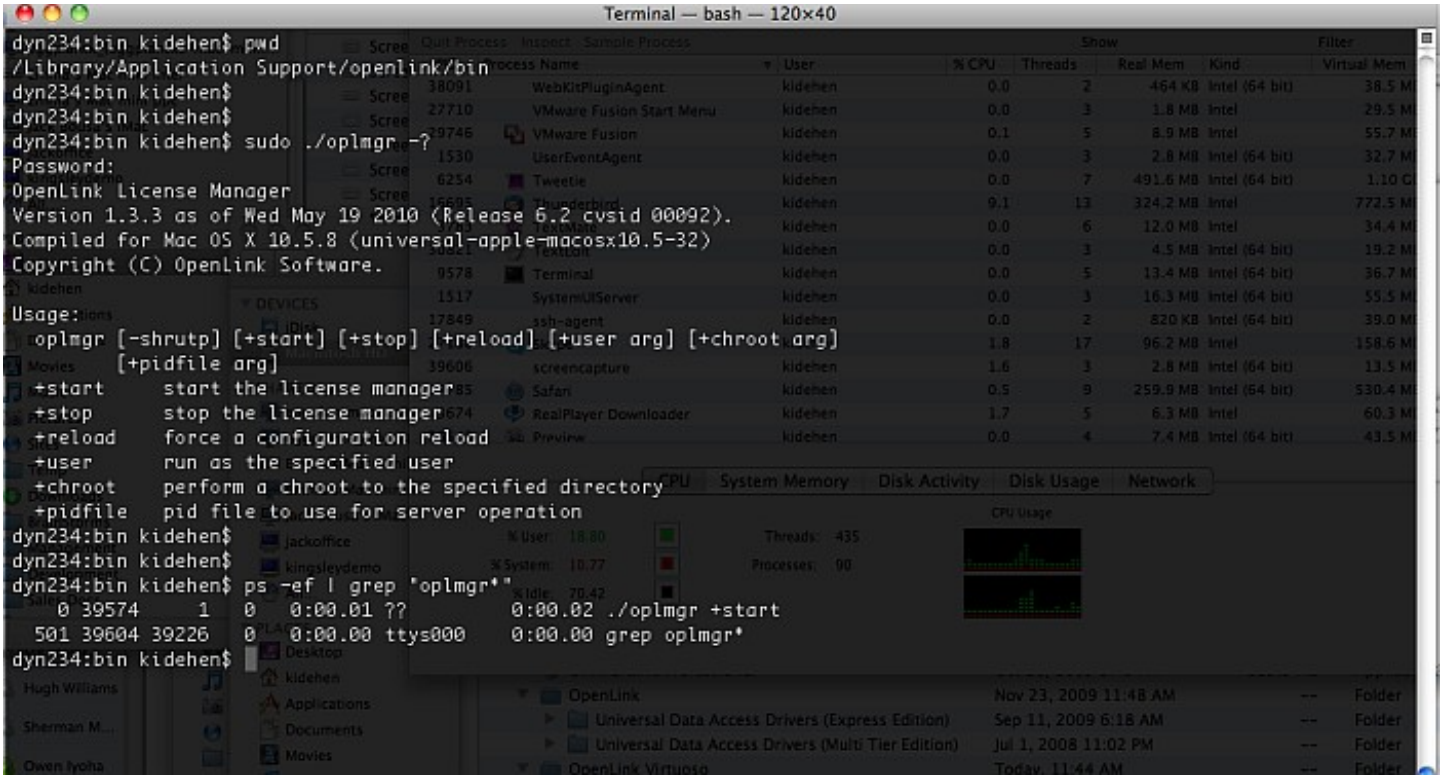
Check .log in case of .trx incompatiblity issue

oplmgr issue

To stop, start, or reload (which updates licenses) use the following commands:

```
/Library/Application Support/openlink/bin/oplmgr +start | +stop | +reload
```

Figure 2.53. The .log file



See Also:
OpenLink License Management

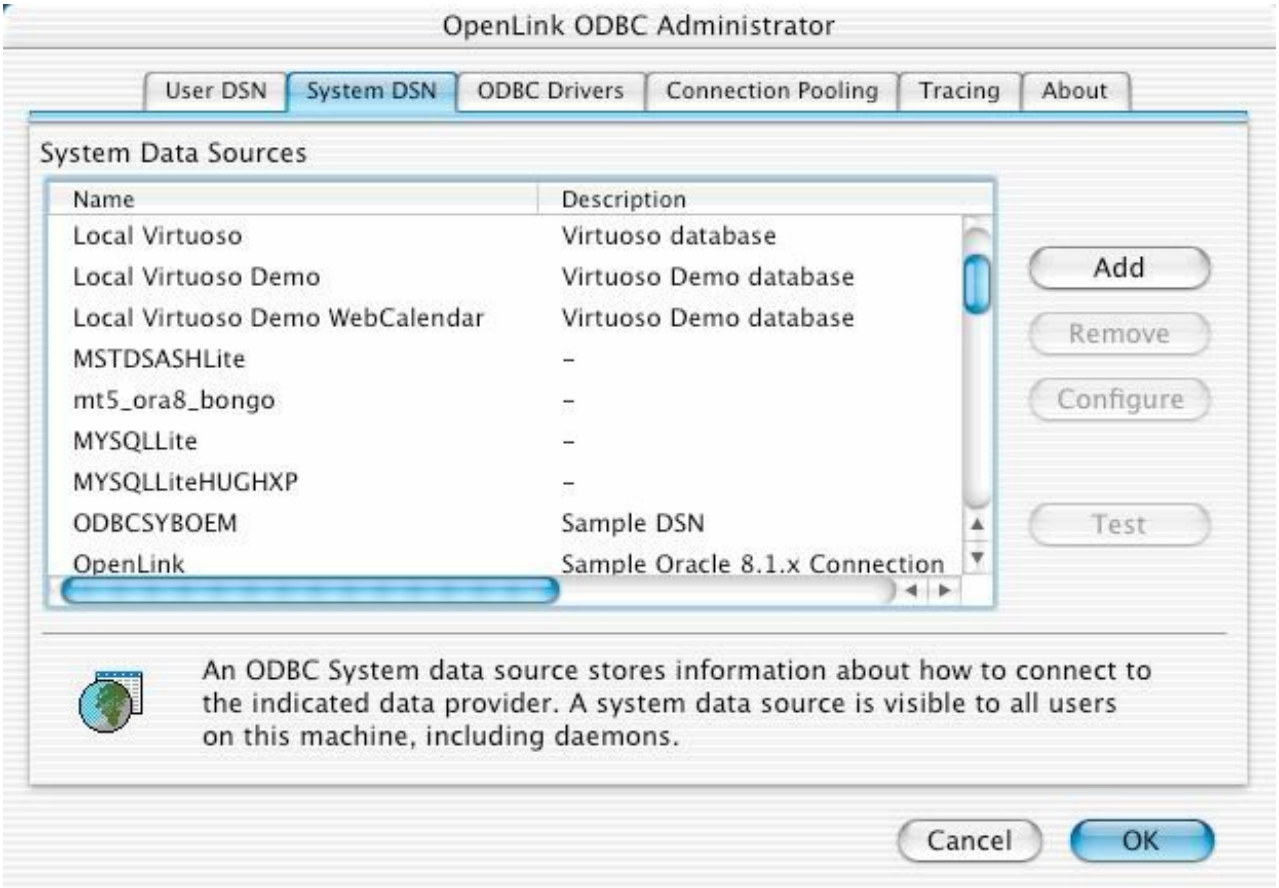
2.4.6. Configuring an ODBC Data Source

Launch the iODBC Administrator, assuming that you have the iODBC Administrator installed.

On some Mac OS X systems installed on an HFS partition, the icons may not be displayed correctly or at all, due to a Mac OS X bug. In such cases a simple reboot of your system should correct this.

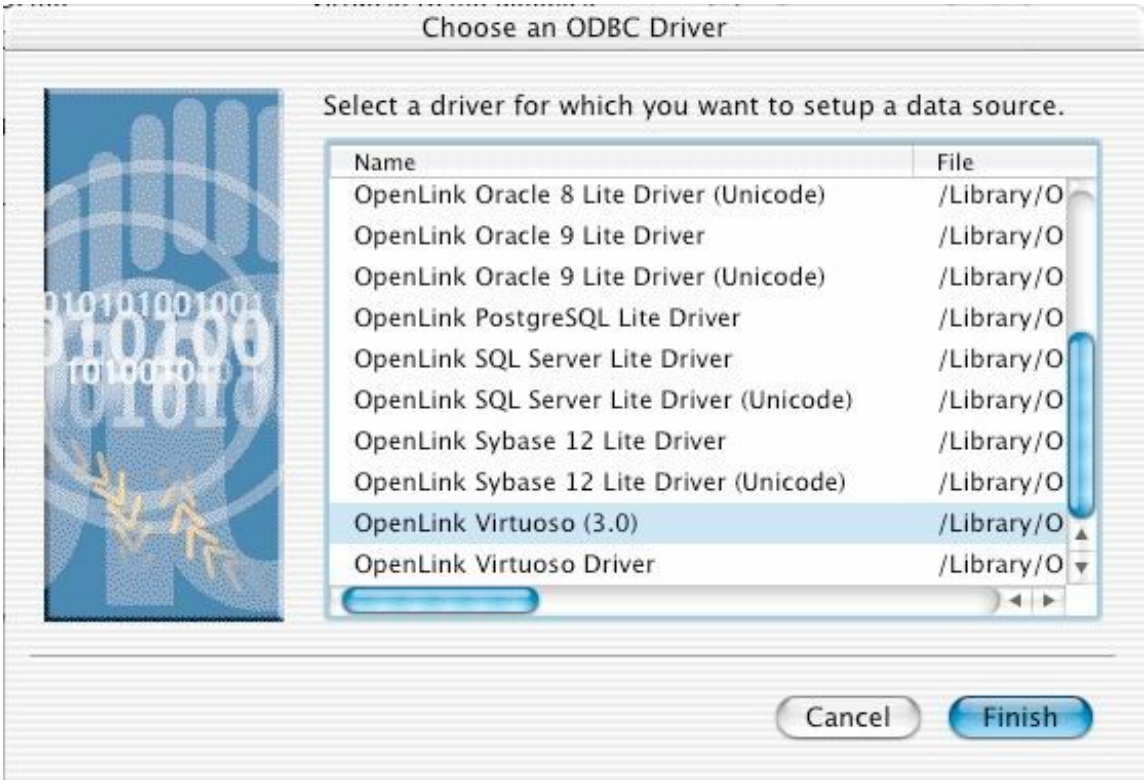
Once the iODBC Administrator window has appeared you can press the *Add* button on the either the *User DSN* or the *System DSN* panel to add a new Data Source entry.

Figure 2.54. ODBC Administrator



Choose the driver that you wish to create a Data Source for, in this case the Virtuoso Driver (3.0).

Figure 2.55. ODBC Administrator - Choose Virtuoso Driver

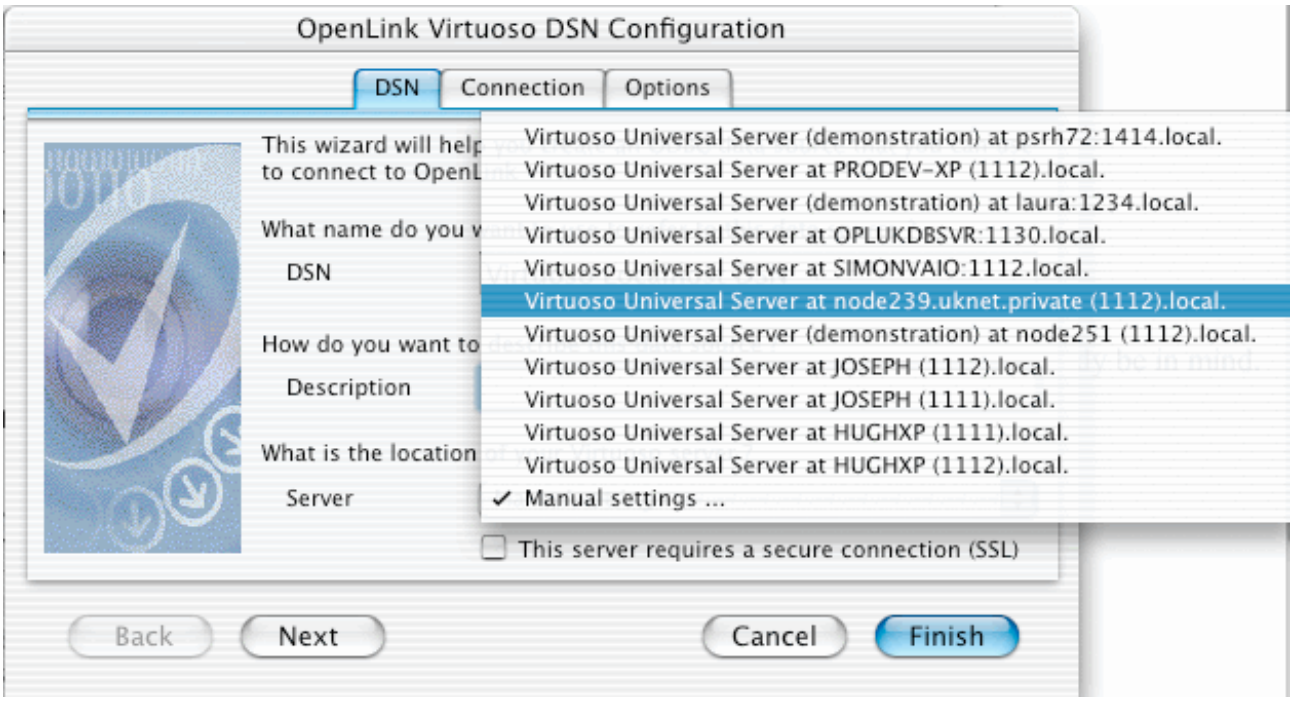


Provide all required details in the fields provided by the setup panel.

Figure 2.56. ODBC Administrator - Configure Virtuoso DSN

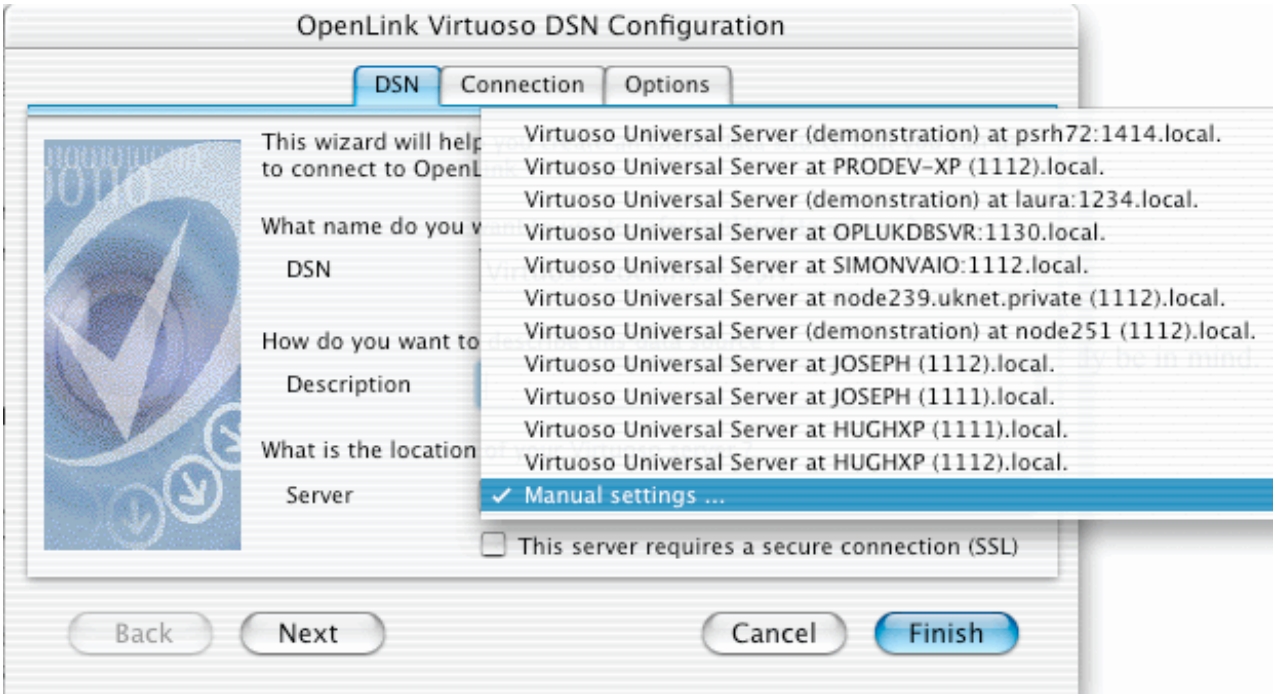
The server can be picked from the list of Zero Configuration Datasources that is discovered.

Figure 2.57. Pick Zero Config



Alternatively a manual selection can be made for the server.

Figure 2.58. Pick manual mode



In this case enter the hostname (or IP) and port number.

Figure 2.59. Manual Server Entry

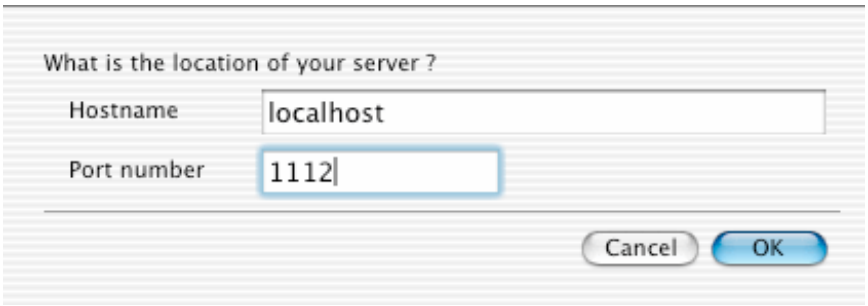


Figure 2.60. ODBC Administrator - Configure Virtuoso DSN

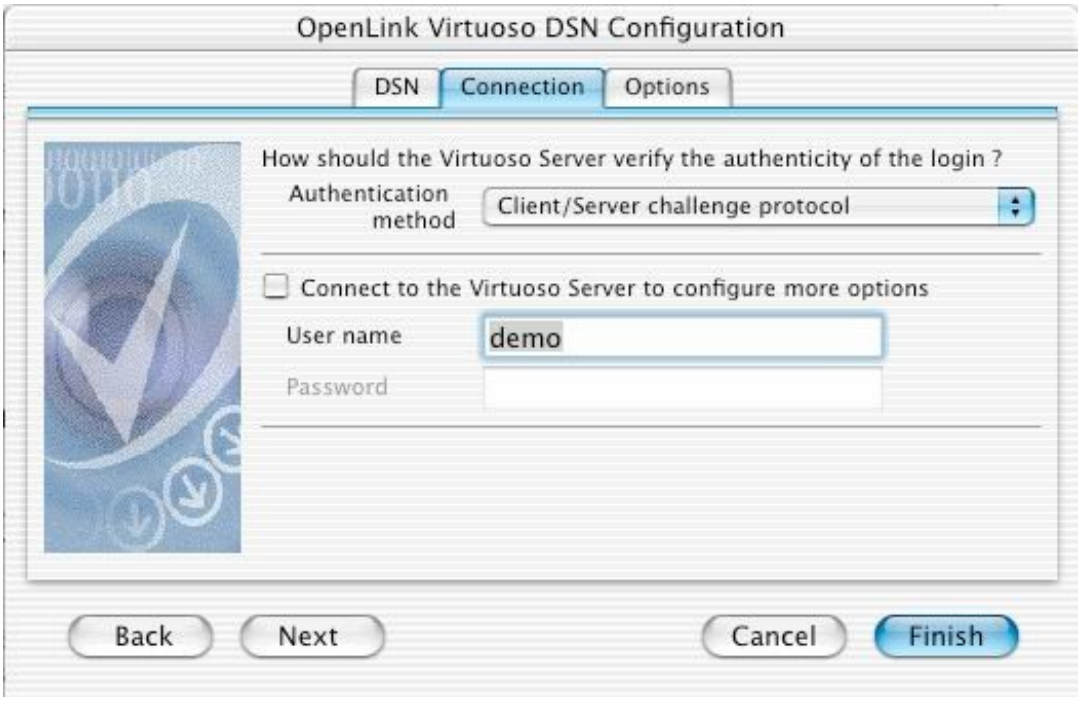
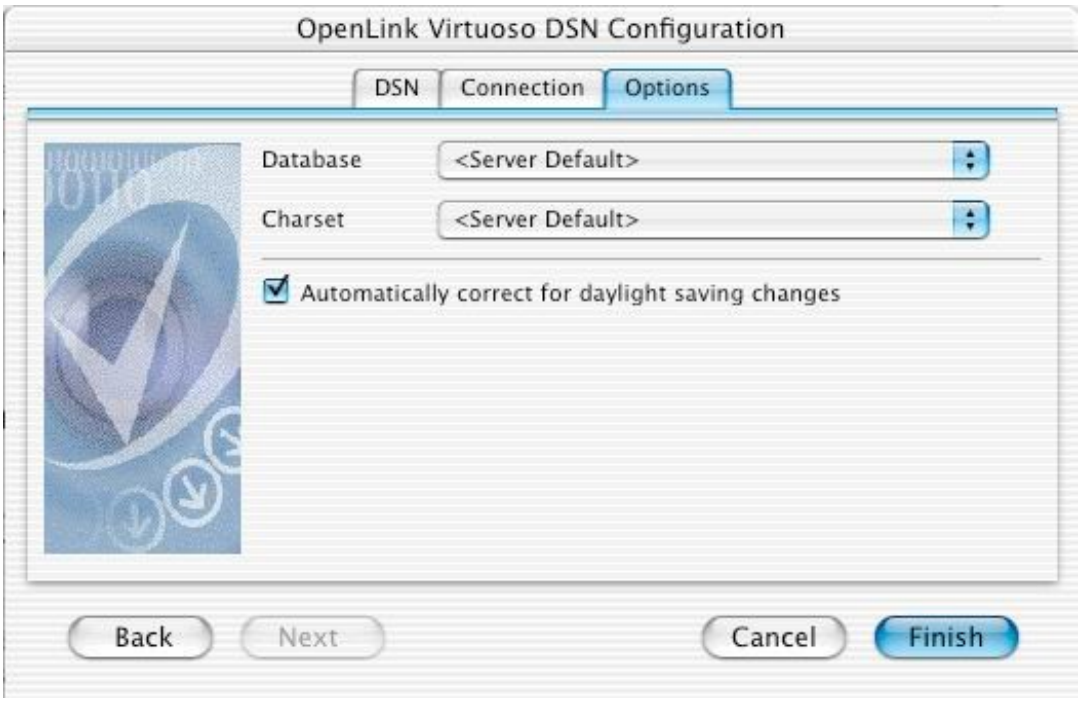


Figure 2.61. ODBC Administrator - Configure Virtuoso DSN



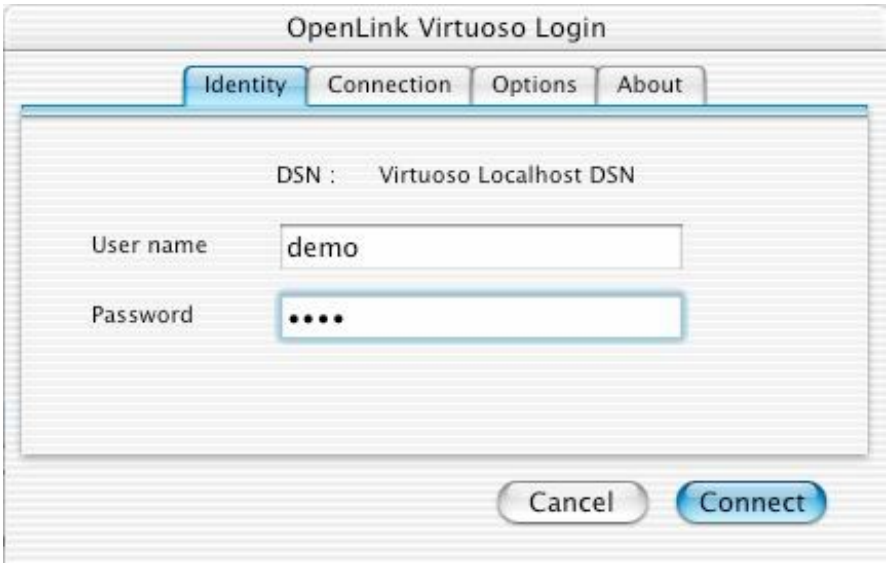
Press the *Finish* button to save the Data Source information.

2.4.7. Testing an ODBC Data Source

Select the Data Source from the DSN list, and press the *Test* button.

You will be prompted for a username and password to establish a connection with the Data Source.

Figure 2.62. Login - Identity Tab



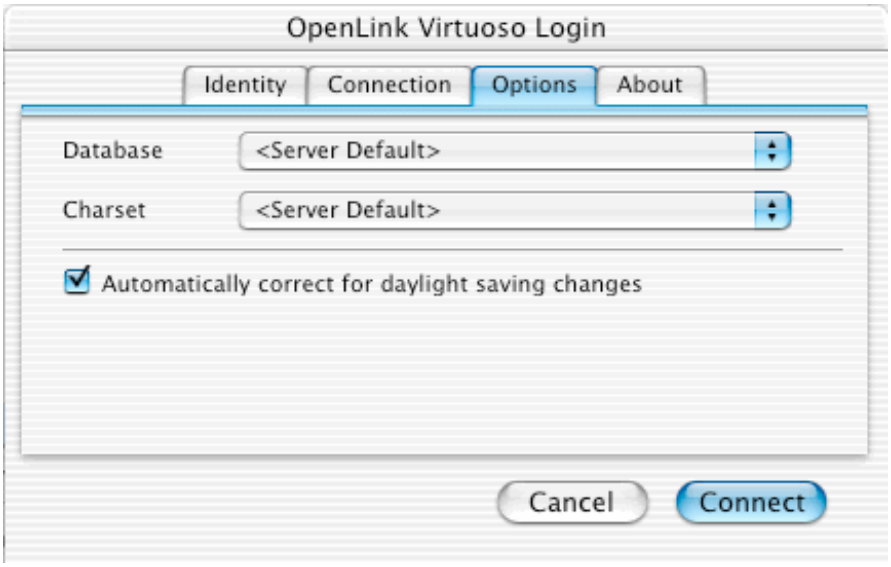
The login dialogue also allows you to alter the configuration if required.

Under the Connection Tab are the fields for the remote server, and the authentication method. Typically the defaults are used.

Figure 2.63. Login - Connection Tab

The Options page has Database, Charset and daylight savings configuration. Typically the defaults are used.

Figure 2.64. Login - Options Tab

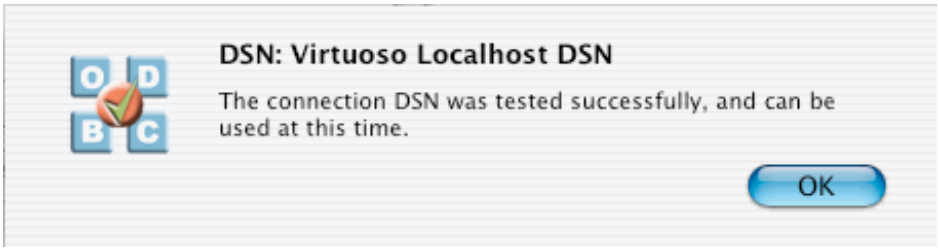


The About page shows the software details.

Figure 2.65. Login - About

The result of the test is shown in a popup window.

Figure 2.66. Test Result



2.4.8. Default passwords

See the following Quick Start chapter for very important information about changing the default passwords .

2.4.9. Demo Database

Installation steps .

2.5. OpenLink License Management

2.5.1. License Files

License Activation

Mac OS X

1. Launch `Terminal.app (/Applications/Utilities/)`
2. Execute the command `cd "/Library/Application Support/openlink/bin/"`
3. Execute the command `oplmgr +stop`
4. Retrieve your new Virtuoso license file from an email or ODS Briefcase .
5. Replace the old license file with the new license file. (You may delete the old file, or safely retain it in this location by changing the `.lic` file extension to `.lic-old` .)
6. Execute the command `oplmgr +start`

Windows

1. Launch the

   Services

Control Panel (may be in the

Administrative Tools

sub-folder).
2. Locate and select the

   OpenLink License Manager service

   .
3. Click the

   Stop

   icon.
4. Retrieve your new Virtuoso license file from an email or ODS Briefcase .
5. Replace the old license file with the new license file. (You may delete the old file, or safely retain it in this location by changing the `.lic` file extension to `.lic-old` .)
6. Go back to the

   Services

   Control Panel.
7. Locate and select the

   OpenLink License Manager

   service.
8. Click the

   Start

   icon.

Unix-like OS (AIX, HP-UX, Linux, Solaris, etc.)

1. Open a Unix terminal.
2. `cd` into the root of your Virtuoso installation.
3. Use one of the following commands to set Virtuoso-related environment variables. (Note that they do, and must, begin with dot-space-dot-slash.)

   - `. ./virtuoso-enterprise.sh` - bash, bsh, ksh, and related shells
   - `. ./virtuoso-enterprise.csh` - csh, tcsh, and related shells
4. Execute the command: oplmgr +stop
5. Retrieve your new Virtuoso license file from an email or ODS Briefcase . Use binary-mode ftp to transfer the new license to the Unix box, if required.
6. Place the new license in the bin sub-directory of the Virtuoso installation.
7. Set and export an `OPL_LICENSE_DIR` environment variable that passes the path to the directory that contains your Virtuoso license file, e.g

   - `export OPL_LICENSE_DIR="/opt/virtuoso/bin/"`
   - `OPL_LICENSE_DIR="/opt/virtuoso/bin/" ; export OPL_LICENSE_DIR`
8. Execute the command: `oplmgr +start`

Retrieve OpenLink Licenses from Your ODS Briefcase

OpenLink's registered users receive an automatic OpenLink Data Spaces (ODS) account on the My OpenLink home page. This account allows registered users to experience and explore OpenLink's Virtuoso Universal Cilent applications.It also provides a convenient storage mechanism for OpenLink commercial and evaluation license keys.

This page shows you how to access licenses that reside in your ODS Briefcase . A second 15 Day evaluation license is

automatically copied to the Briefcase, when you download our software and receive your first 15 Day key. You may access the Briefcase by clicking on the link that appears in the email that contains the first license. Alternatively, you may identify the ODS url by logging into the OpenLink site.

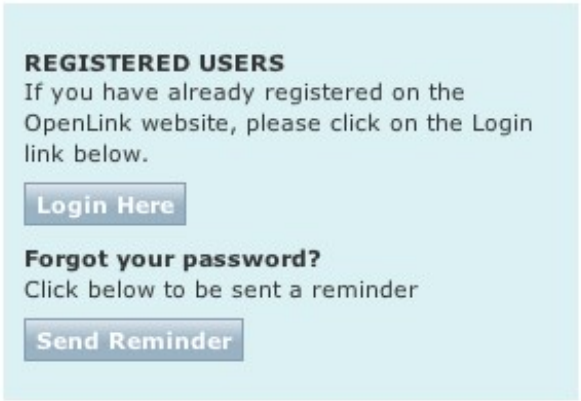
The following instructions will allow you to access your ODS Briefcase via the OpenLink Web site.

1. Click the Login link that appears on the OpenLink Web page.

Figure 2.67. ODS Briefcase

2. Click the "Login Here" button that appears in the Registered Users box.

Figure 2.68. ODS Briefcase

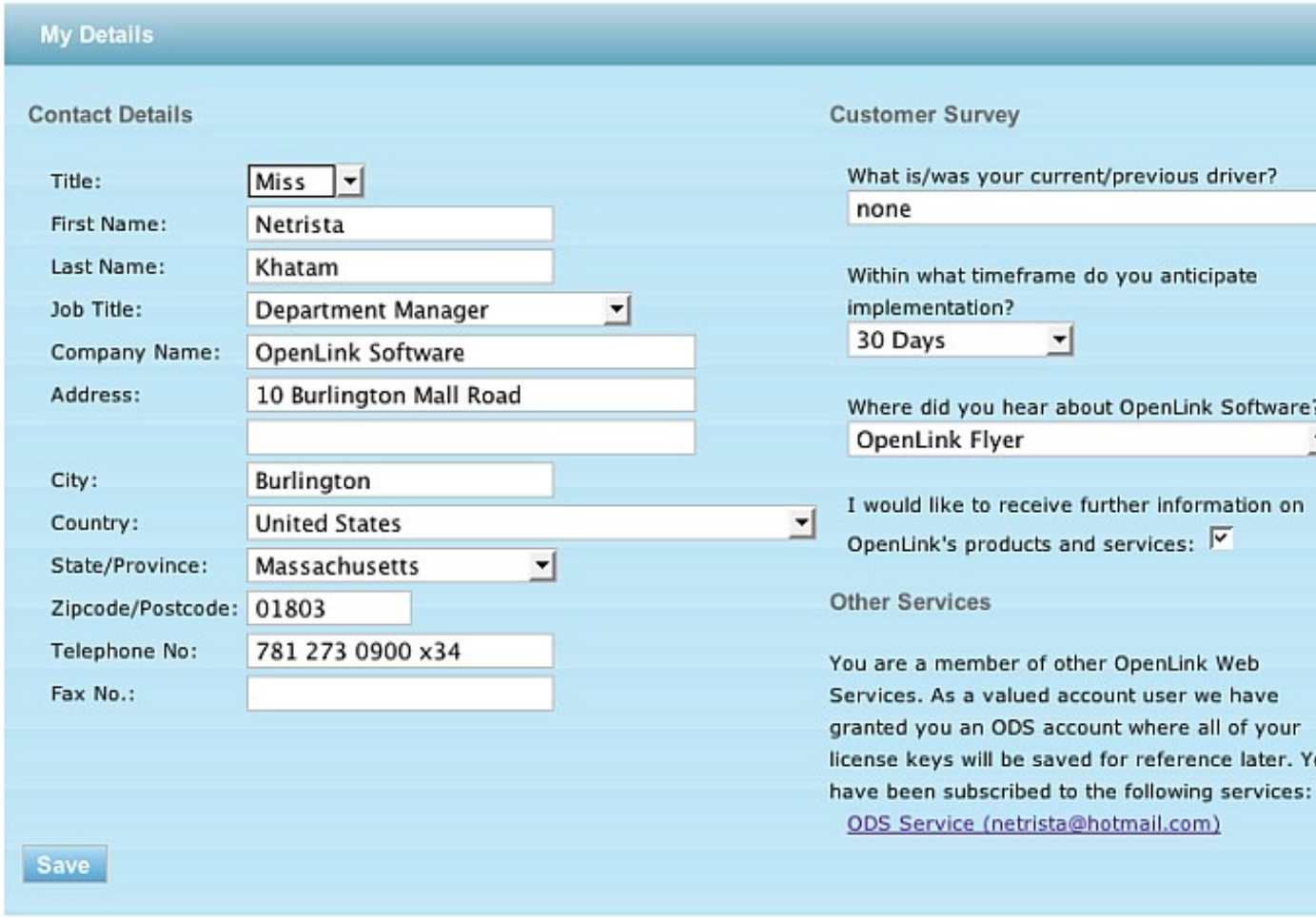


3. Click the "Check my profile" link that appears in the Welcome box.

Figure 2.69. ODS Briefcase

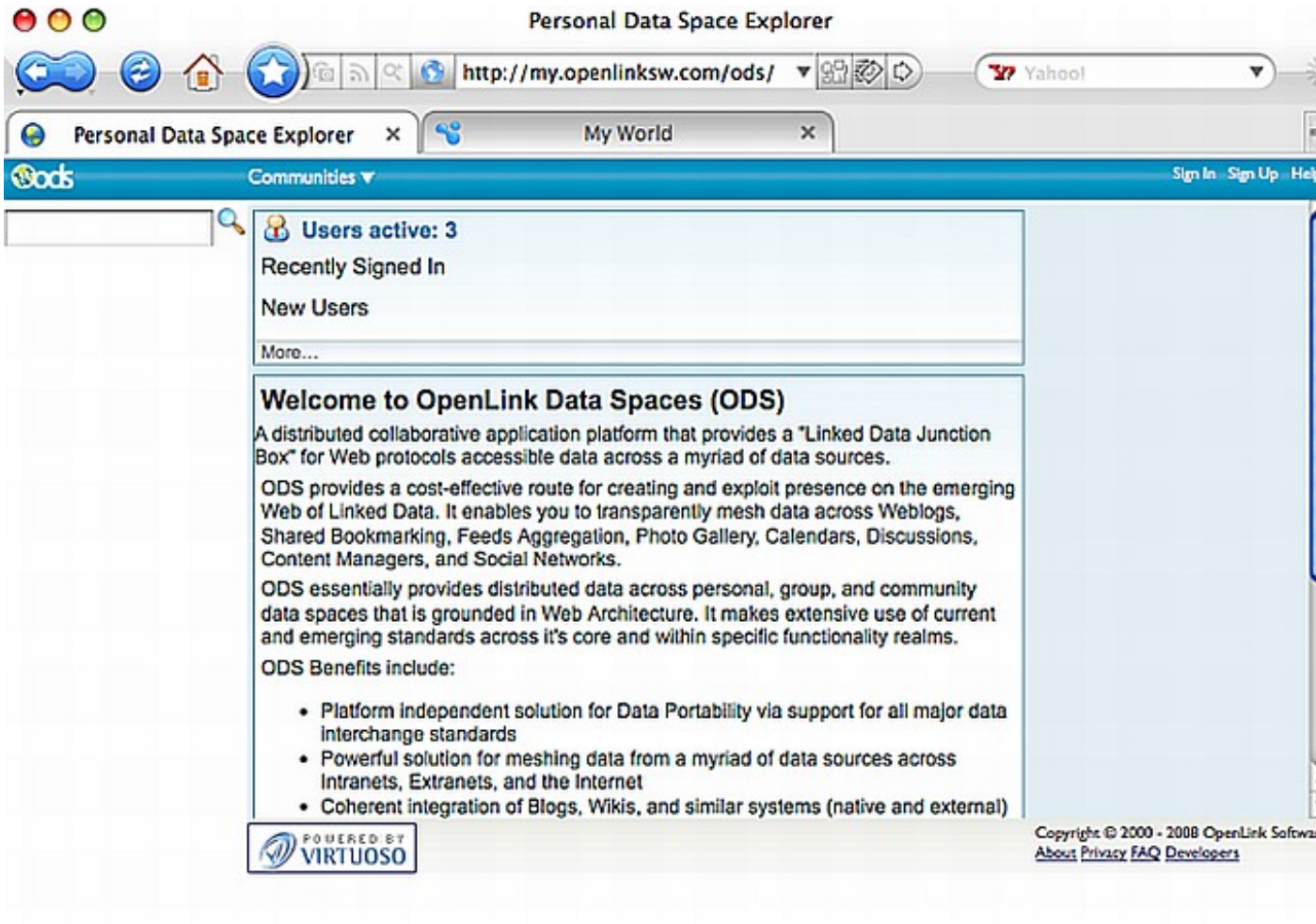
4. Click the ODS Service URL that appears in the lower right hand corner of the My Details dialog.

Figure 2.70. ODS Briefcase



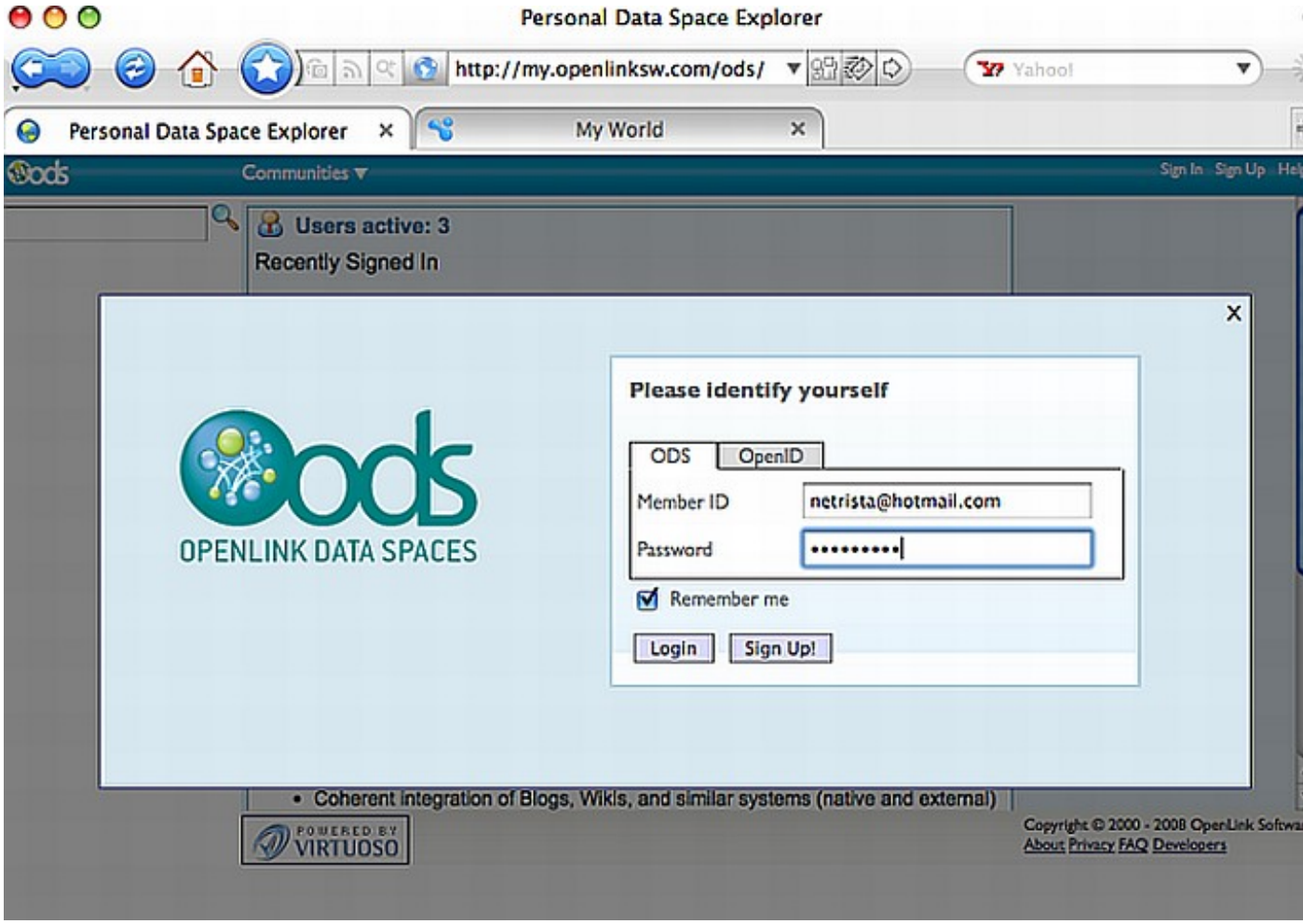
5. Allow the My OpenLink Web page to load in a Web browser.

Figure 2.71. ODS Briefcase



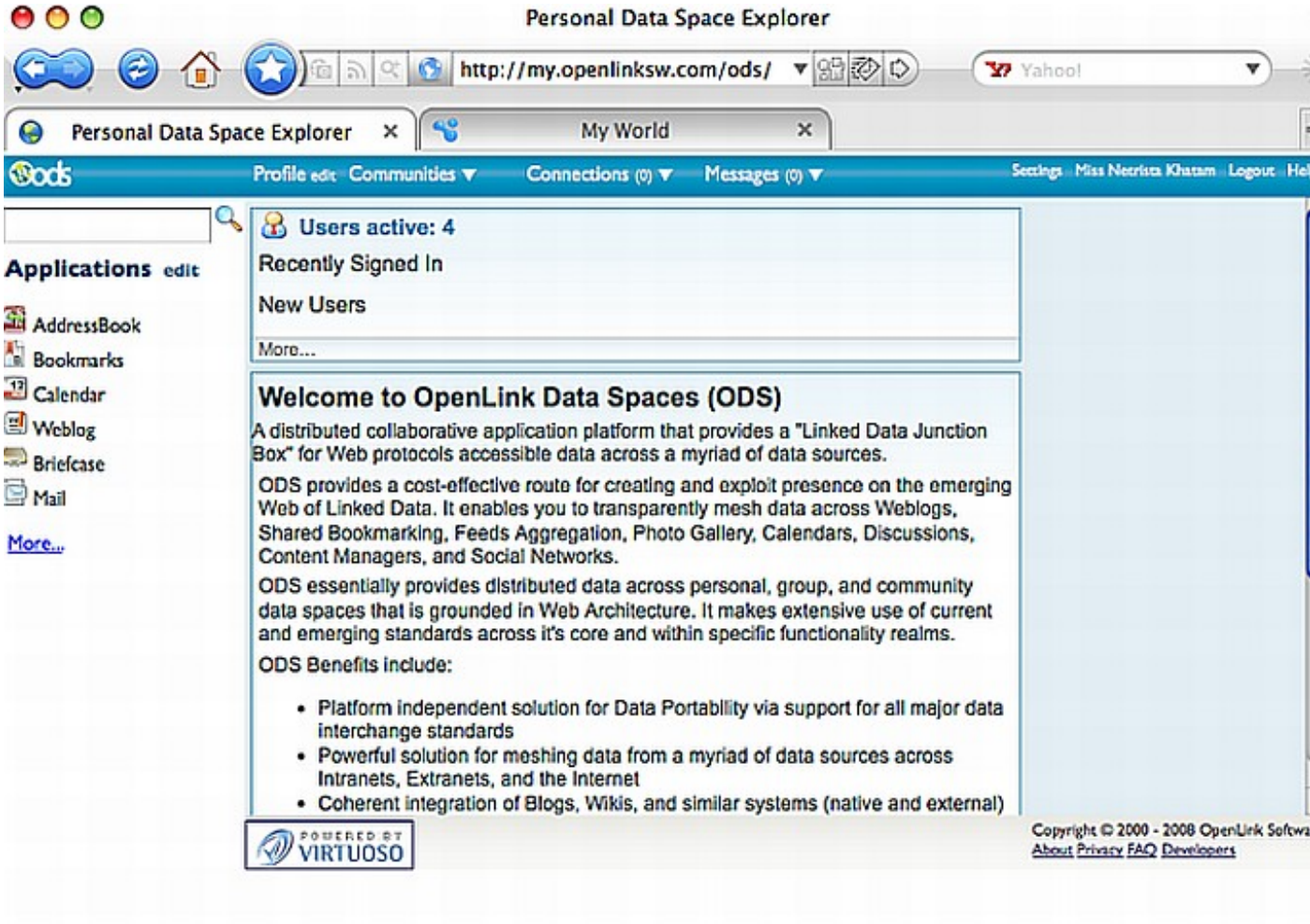
6. Login to your OpenLink Data Space using your OpenLink login.

Figure 2.72. ODS Briefcase



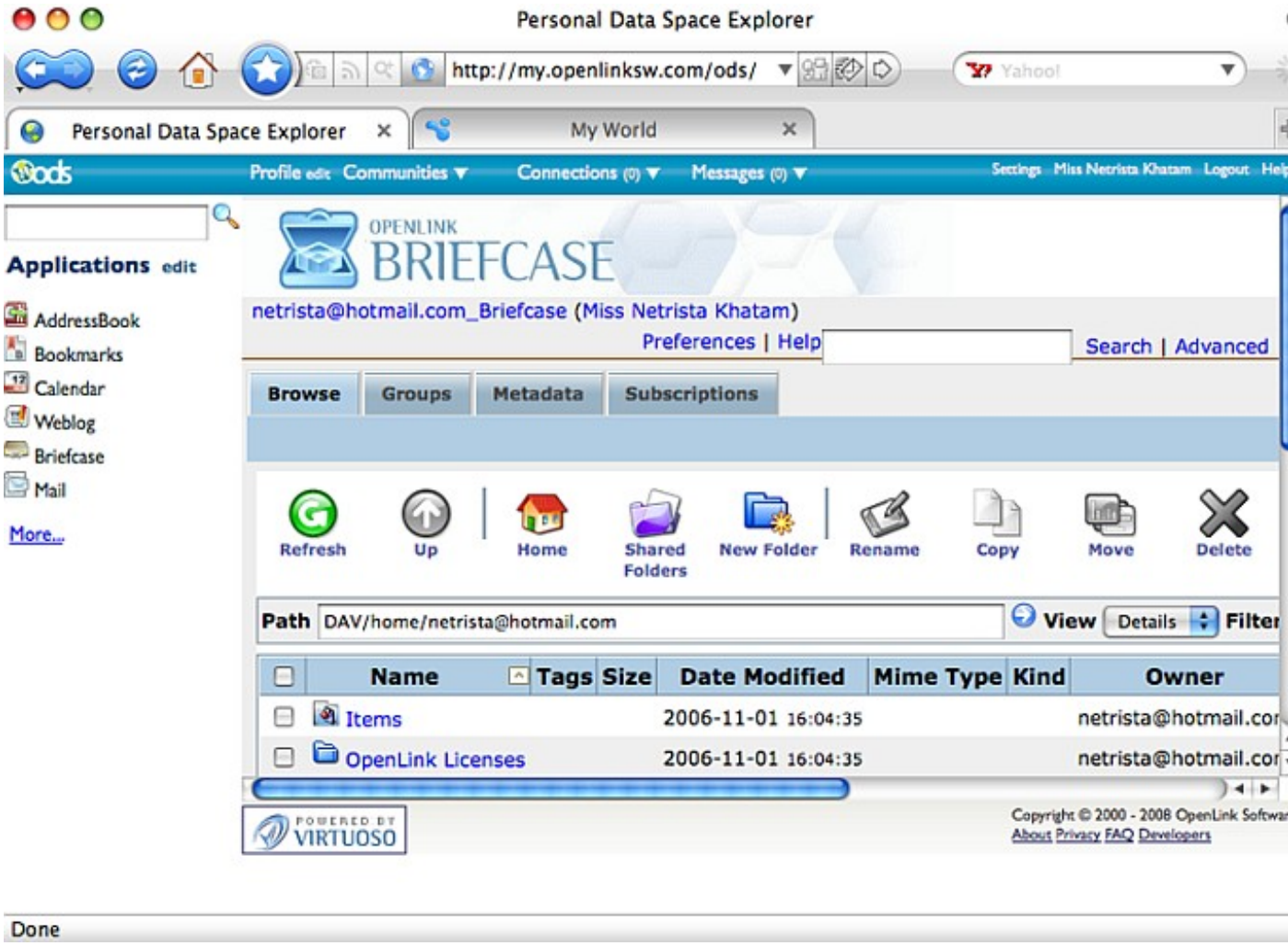
7. Click the Briefcase link that appears in the ODS menu bar at the top of the page.

Figure 2.73. ODS Briefcase



8. Your license will appear as a Resource Item under the Briefcase Summary.

Figure 2.74. ODS Briefcase



Click the license link to download your license. Provide your OpenLink username and password if prompted for a DAV login.

License Preservation

Users may choose to preserve the existing license file when applying a new license file. This facilitates a rollback to the original file, if the replacement file is problematic.

Mac OS X

- Use StuffIt or a similar utility to store the file in a compressed archive.
- Rename the file, e.g., from `virtuoso.lic` to `virtuoso.lic-old`

Windows

Use WinZip or a similar utility to copy your license file to a .zip archive.

Unix-like OS (AIX, HP-UX, Linux, Solaris, etc.)

Use tar or gzip to copy your license file into an archive. For example: `tar cvzf virtuoso-lic.tgz virtuoso.lic`

License Removal

Users may choose to remove expired or deprecated license files altogether to ensure the uptake of a new license file.

Mac OS X

1. Shutdown your OpenLink License Manager
2. Click the Search icon.

3. Type your license name into the search box. For example: virtuoso.lic
4. Drag the license from the search results onto the trash icon.
5. Expand the Finder menu.
6. Click Empty Trash.

Windows

1. Launch the

   Services

   Control Panel (may be in the

   Administrative Tools

   sub-folder).
2. Locate and select the

   OpenLink Virtuoso

   service.
3. Click the

   Stop

   icon.
4. Locate and select the

   OpenLink License Manager

   service.
5. Click the

   Stop

   icon.
6. Launch the Windows Search utility, from the Start menu.
7. Search for all files with the name `virtuoso.lic`..
8. Drag all found licenses to the Recycle Bin.
9. Empty the Recycle Bin

Unix-like OS (AIX, HP-UX, Linux, Solaris, etc.)

1. Login to the Unix or Linux operating system.
2. Use the find command to locate relevant license files, and check their permissions and ownership.

```
bash-2.03$ find / -name virtuoso.lic -exec ls -l {} \;
-rw-r--r--   1 virtuoso other        645 Mar 11 08:06 /usr/virtuoso/bin/virtuoso.lic
bash-2.03$
```

3. Use the rm command to remove the license file(s).

```
bash-2.03$ rm /usr/virtuoso/bin/virtuoso.lic
```

4. If the file permissions and ownership require, you can use sudo -

```
bash-2.03$ sudo rm /usr/virtuoso/bin/virtuoso.lic
Password:
bash-2.03$ ls /usr/virtuoso/bin/virtuoso.lic
bash-2.03$
```

   - or su to the appropriate user or group -

```
bash-2.03$ su virtuoso
Password:
```

```
bash-2.03$ rm /usr/virtuoso/bin/virtuoso.lic
bash-2.03$ ls /usr/virtuoso/bin/virtuoso.lic
bash-2.03$
```

2.5.2. License Manager

Monitor License Consumption

OpenLink's License Manager technology provides a sophisticated and easy to use License statistics and monitoring facility. This page introduces you to that facility and provides alternative techniques for versions of the software that do not contain it.

1. Launch the Virtuoso Web interface.
2. Click the

   Conductor

   link.
3. Login with your Virtuoso dba account.
4. Click the

   System Admin

   tab.
5. Locate the

   License

   section to review the terms of your Virtuoso license.
6. Locate the

   Clients

   section to review open connections that consume Virtuoso license points.

Start the License Manager Process

Mac OS X

1. Launch `Terminal.app (/Applications/Utilities/)`
2. Execute the command `cd "/Library/Application Support/openlink/bin/"`
3. Execute the command `oplmgr +start`

Windows

1. Launch the

   Services

   Control Panel (may be in the

   Administrative Tools

   sub-folder).
2. Locate and select the

   OpenLink Virtuoso

   service.
3. Click the

   Start

icon.

Unix-like OS (AIX, HP-UX, Linux, Solaris, etc.)

1. Open a Unix terminal.
2. `cd` into the root of your Virtuoso installation.
3. Use one of the following commands to set Virtuoso-related environment variables. (Note that they do, and must, begin with dot-space-dot-slash.)
   - `. ./virtuoso-enterprise.sh` > - bash, bsh, ksh, and related shells
   - `. ./virtuoso-enterprise.csh` - csh, tcsh, and related shells
4. Execute the command: `oplmgr +start`

Note : OpenLink recommends that you create an `/etc/init.d/` script that sets the environment and runs `oplmgr +start` at boot time.

Stop the License Manager Process

Mac OS X

1. Launch `Terminal.app (/Applications/Utilities/)`
2. Execute the command `cd "/Library/Application Support/openlink/bin/"`
3. Execute the command `oplmgr +stop`

Windows

1. Launch the

   Services

   Control Panel (may be in the

   Administrative Tools

   sub-folder).
2. Locate and select the

   OpenLink License Manager service

   .
3. Click the

   Stop

   icon.

Unix-like OS (AIX, HP-UX, Linux, Solaris, etc.)

1. Open a Unix terminal.
2. `cd` into the root of your Virtuoso installation.
3. Use one of the following commands to set Virtuoso-related environment variables. (Note that they do, and must, begin with dot-space-dot-slash.)
   - `. ./virtuoso-enterprise.sh` - bash, bsh, ksh, and related shells
   - `. ./virtuoso-enterprise.csh` - csh, tcsh, and related shells
4. Execute the command: oplmgr +stop

Reinitialize the License Manager Process

Mac OS X

1. Launch `Terminal.app (/Applications/Utilities/)`
2. Execute the command `cd "/Library/Application Support/openlink/bin/"`
3. Execute the command `oplmgr +reload`

Windows

1. Launch the

   Services

   Control Panel (may be in the

   Administrative Tools

   sub-folder).
2. Locate and select the

   OpenLink License Manager service

   .
3. Click the

   Restart

   icon.

Unix-like OS (AIX, HP-UX, Linux, Solaris, etc.)

1. Open a Unix terminal.
2. `cd` into the root of your Virtuoso installation.
3. Use one of the following commands to set Virtuoso-related environment variables. (Note that they do, and must, begin with dot-space-dot-slash.)

   - `. ./virtuoso-enterprise.sh` - bash, bsh, ksh, and related shells
   - `. ./virtuoso-enterprise.csh` - csh, tcsh, and related shells
4. Execute the command: oplmgr +reload

Additional Usage Notes

The license manager takes several command-line arguments, as revealed by its "usage" output (produced by any unrecognized argument) --

```
bash$ oplmgr --help
OpenLink License Manager
Version 1.2.2 as of Thu Feb 15 2007 (Release 6.0 cvsid 00084).
Compiled for Linux 2.4.20-46.9.legacysmp (i686-generic-linux-glibc23-32)
Copyright (C) OpenLink Software.

Usage:
oplmgr [-shrutp] [+start] [+stop] [+reload] [+user arg] [+chroot arg]
[+pidfile arg]
+start     start the license manager
+stop      stop the license manager
+reload    force a configuration reload
+user      run as the specified user
+chroot    perform a chroot to the specified directory
+pidfile   pid file to use for server operation
```

2.5.3. Debugging License Problems

Virtuoso license problems are easy to debug. The following information provides a conceptual framework and troubleshooting tips suitable for all licensing problems.

Identifying License Problems

License problems occur when users attempt to apply them to products or when they hit a ceiling associated with concurrent usage. Most license errors are explicit and include text that states that a license is invalid, expired, or exceeded. Problems associated with performance or stability problems that do not produce license related errors are not likely to be caused by licenses. There is no scenario in which use of an evaluation key would contribute to limited functionality. OpenLink's products are fully functional regardless of the license key applied. The license key simply dictates the length of time for which a product may be used.

Source of License Problems

Licenses problems appear under the following circumstances:

- A license file is ftp'd in ASCII.
- The license covers a different OpenLink product.
- The license covers a different OpenLink release.
- A machine has more physical Processor cores than allotted for by the license.
- The license can not be found.
- The wrong license is being found.
- You did not perform the appropriate steps to register the license.
- The License Manager process needs to be restarted.
- The license is inadequate for your level of concurrent activity

Troubleshooting License Problems

Use the following tips to resolve your licensing tips or supply OpenLink Technical Services with your findings:

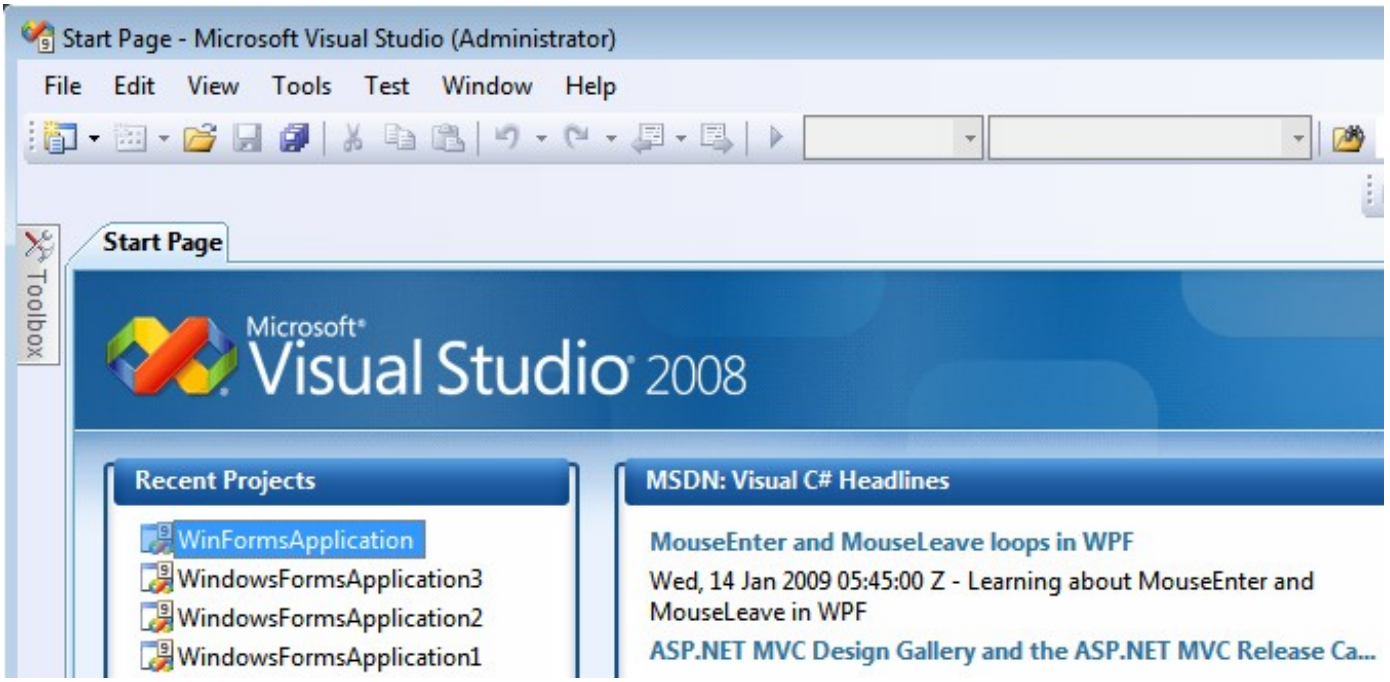
- FTP all licenses (virtuoso.lic) in binary mode.
- Review the terms of the email that contained the license.
- Ensure that the license that you have received is for Virtuoso Universal Server and not a data access provider or driver.
- Compare the Virtuoso version on your machine with the version covered by the license key.
- Identify the number of physical Processor cores on the box that hosts the license and ensure those cores are covered by the license key.
- Use OpenLink's licensing guidelines to ensure that the license is placed in the appropriate location on the target machine.
- Use the host machine's search utilities to ensure that expired or deprecated license files are removed from the host machine and deleted from the trash.
- Use OpenLink's licensing guidelines to ensure that you have performed the license application procedure using the correct steps in the appropriate order.
- Use the host monitoring utilities to ensure that applicable processes have been restarted as detailed in OpenLink's licensing guidelines.

2.6. Virtuoso ADO.Net Data Grid Form Application

This section details the steps required to create a simple Visual Studio 2008 Windows Form application with associated DataGridView control for displaying data in selected tables from a Virtuoso database.

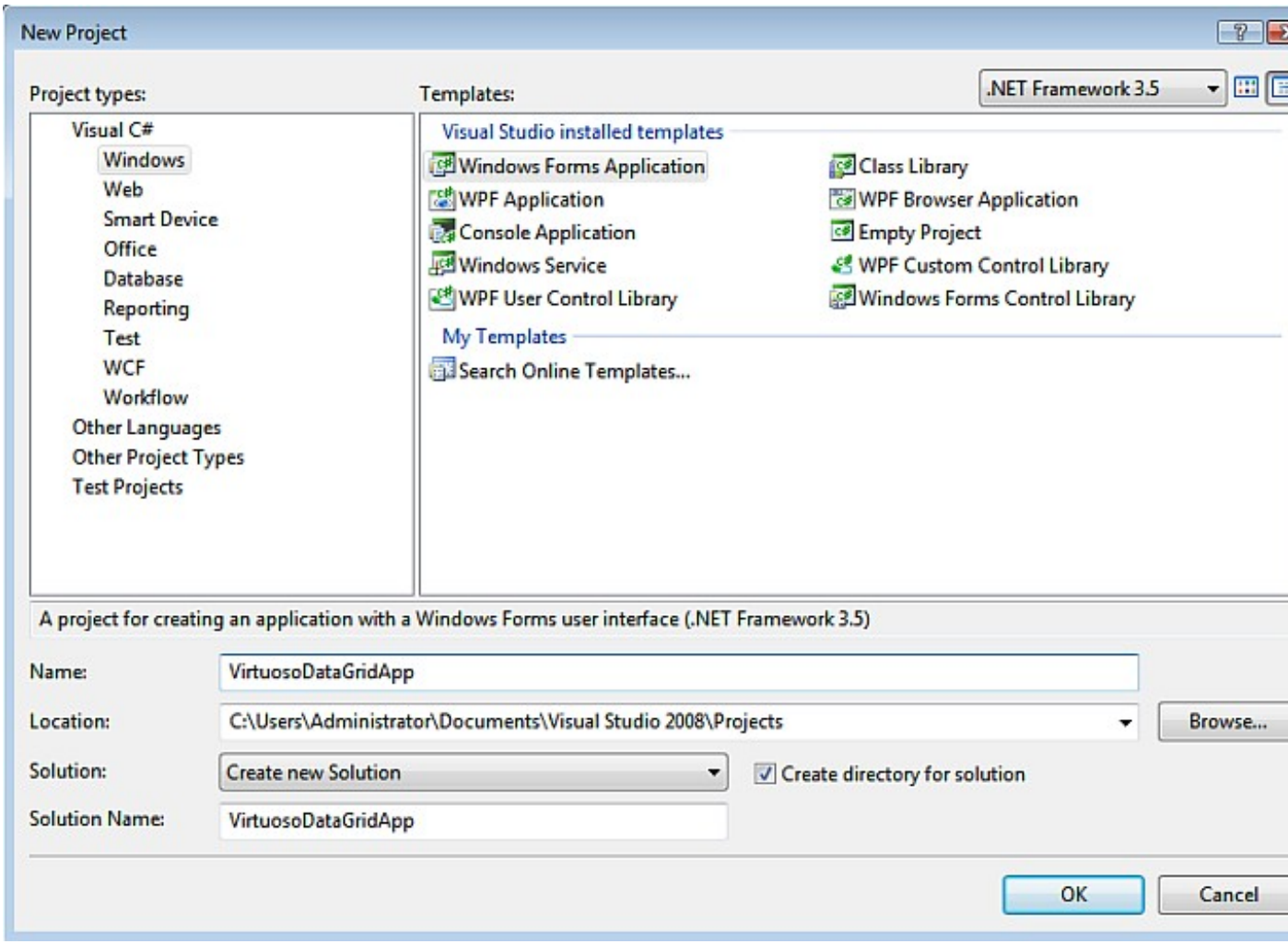
1. Launch the Visual Studio 2008 SP1 IDE.

   Figure 2.75. Visual Studio



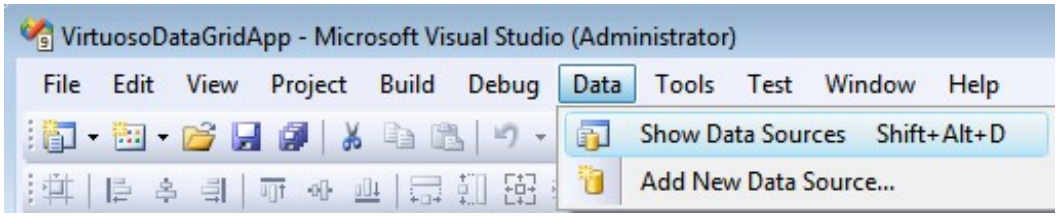
2. Create a Web Application project by going to the File menu in Visual Studio and choosing New Project.
3. When the New Project window appears, choose either Visual Basic or Visual C# as the programming language.
4. Within the language category, click on Windows and select Windows Form Application from the right-hand panel.
5. Choose a name for the project, for example VirtuosoDataGridApp, and click OK.

Figure 2.76. name for the project



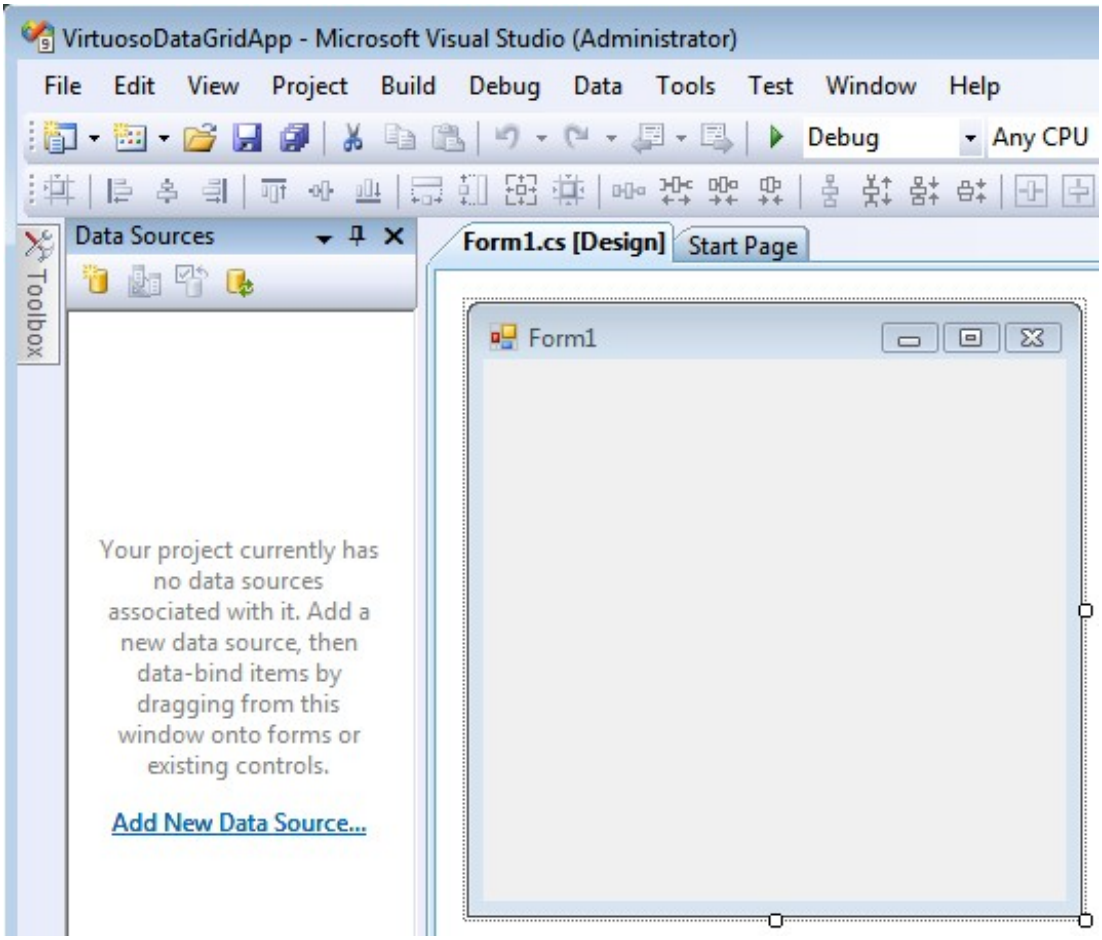
6. From the Data, menu select the Show Data Sources menu item to display a list of available Data sources.

Figure 2.77. Show Data Sources



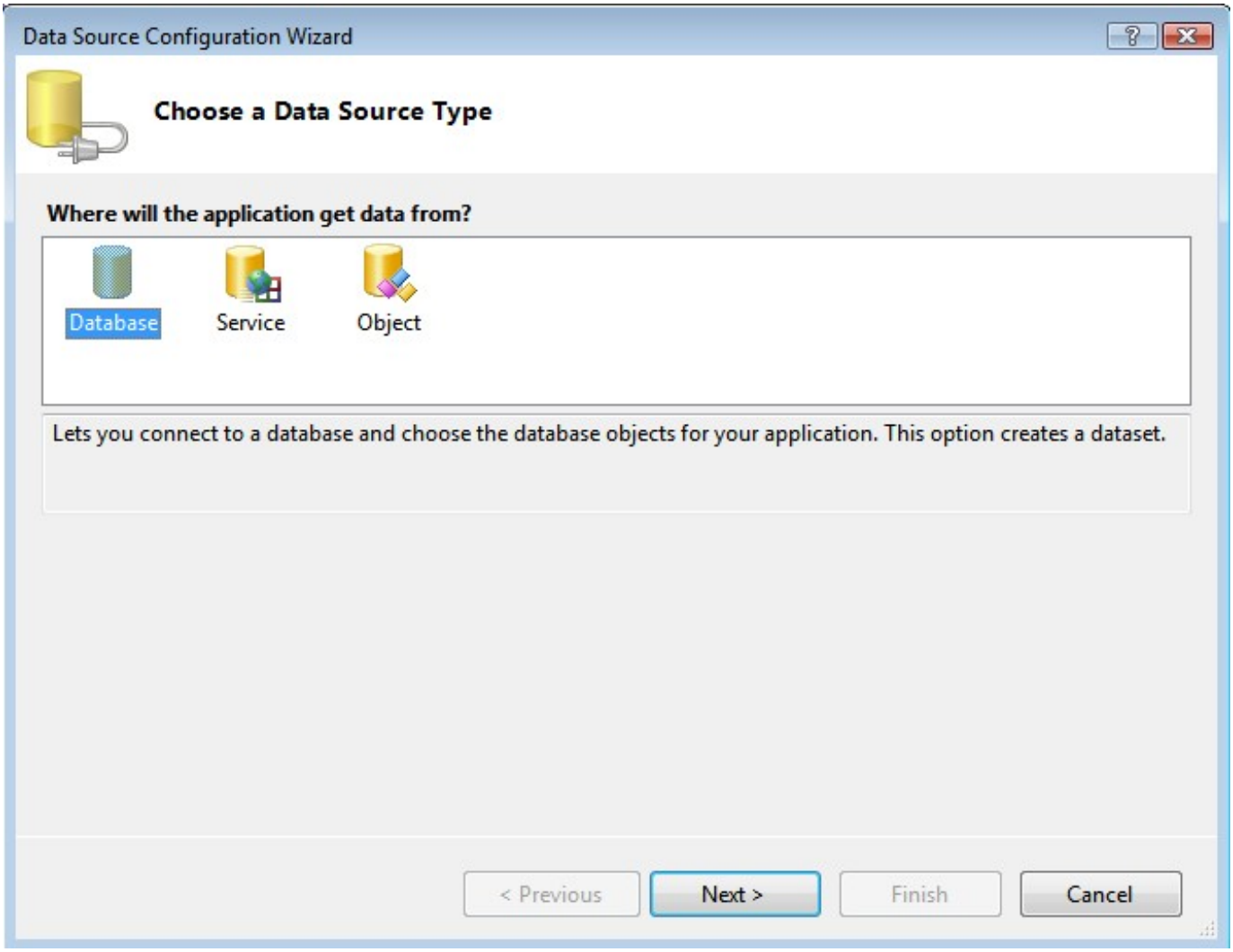
7. Click on the Add New Data Source link to add a new data source.

Figure 2.78. Add New Data Source



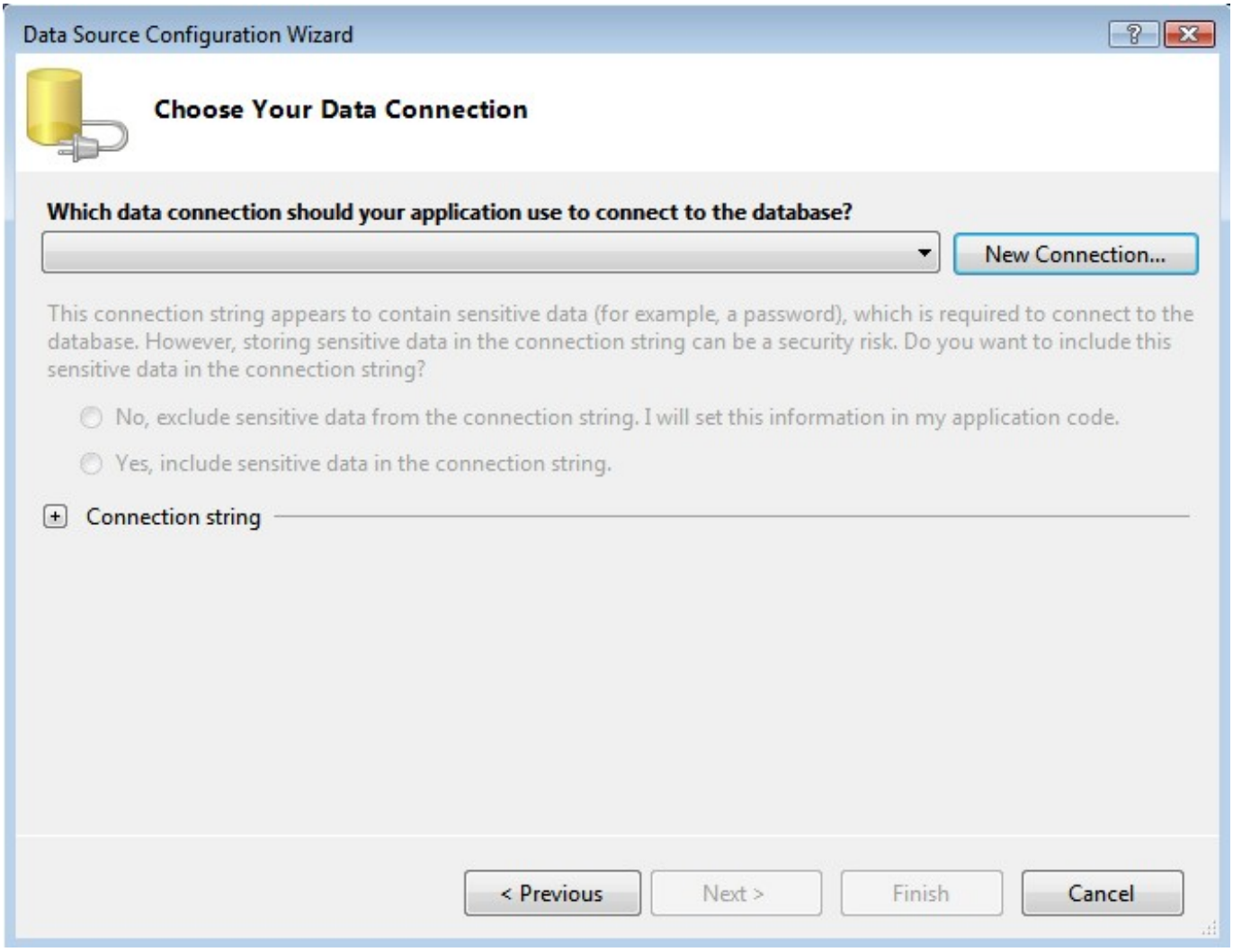
8. In the Data Source Configuration Wizard dialog Choose Data Source Type page, select the Database data source type and click Next.

Figure 2.79. Data Source Configuration Wizard



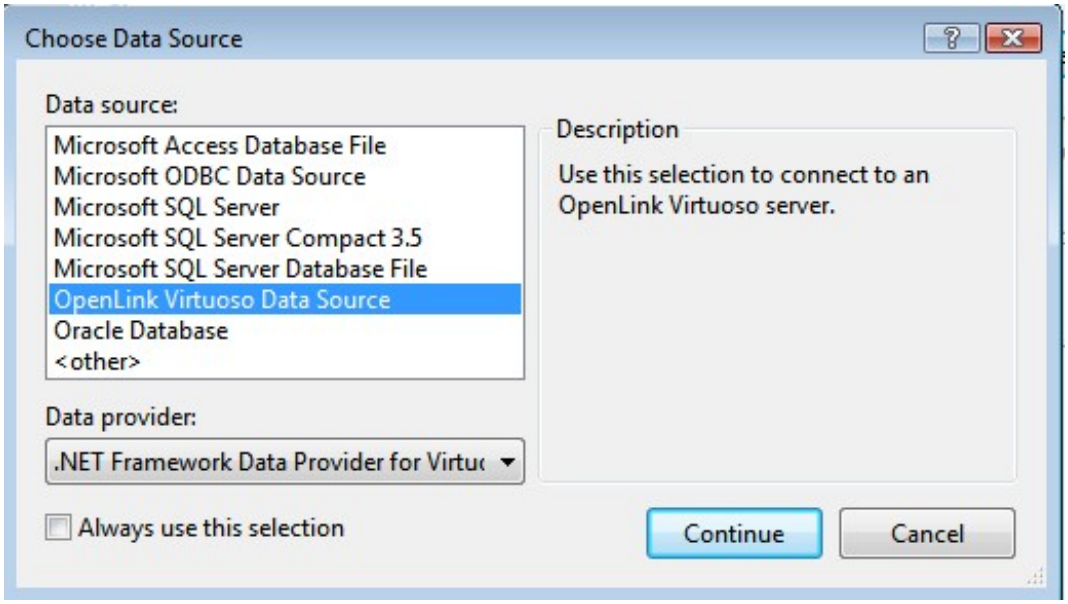
9. In the Data Source Configuration Wizard dialog Choose your Data Connection page, select the New Connection button.

Figure 2.80. Data Connection page



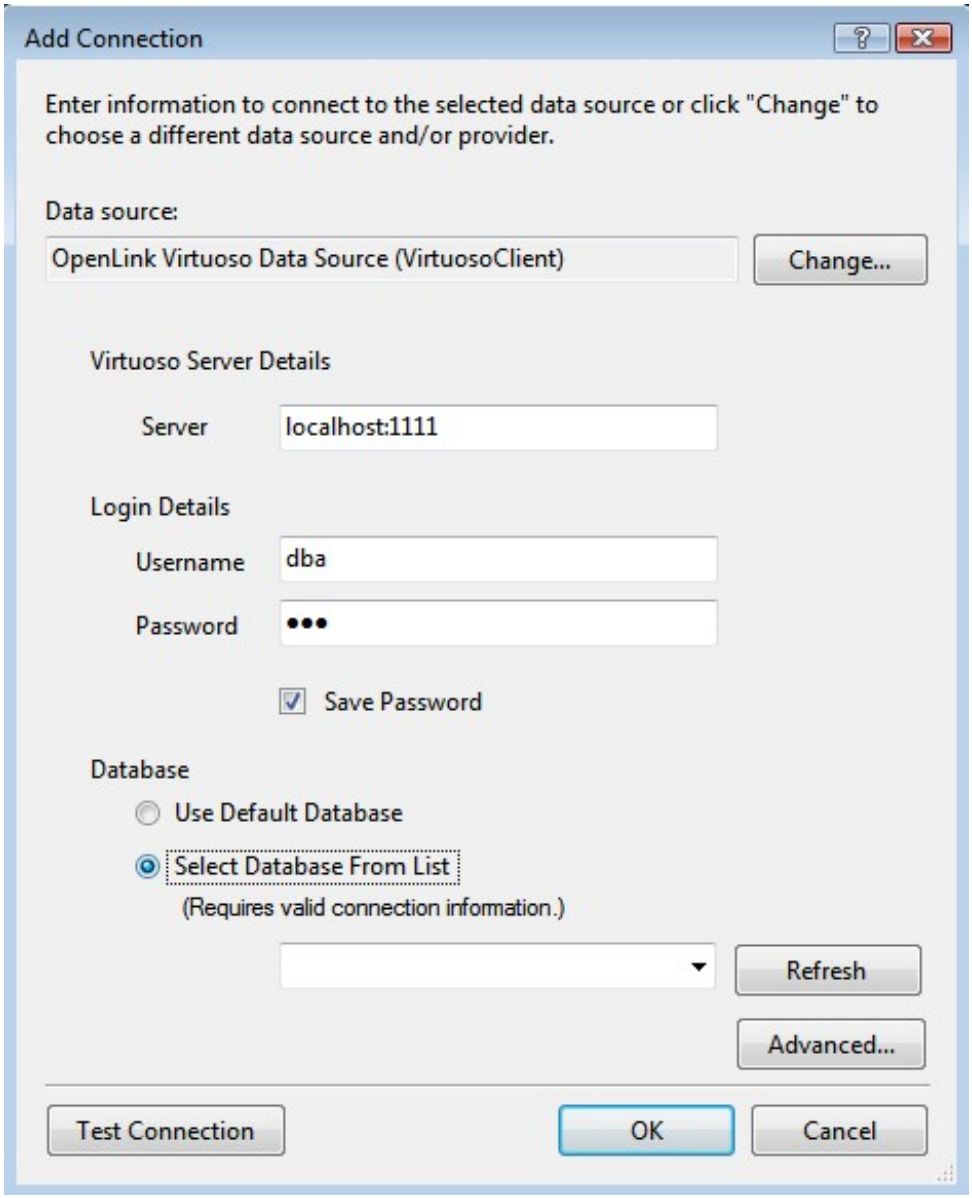
10. In the Choose Data Source dialog, select OpenLink Virtuoso Data Source from the list and click Continue.

Figure 2.81. OpenLink Virtuoso Data Source



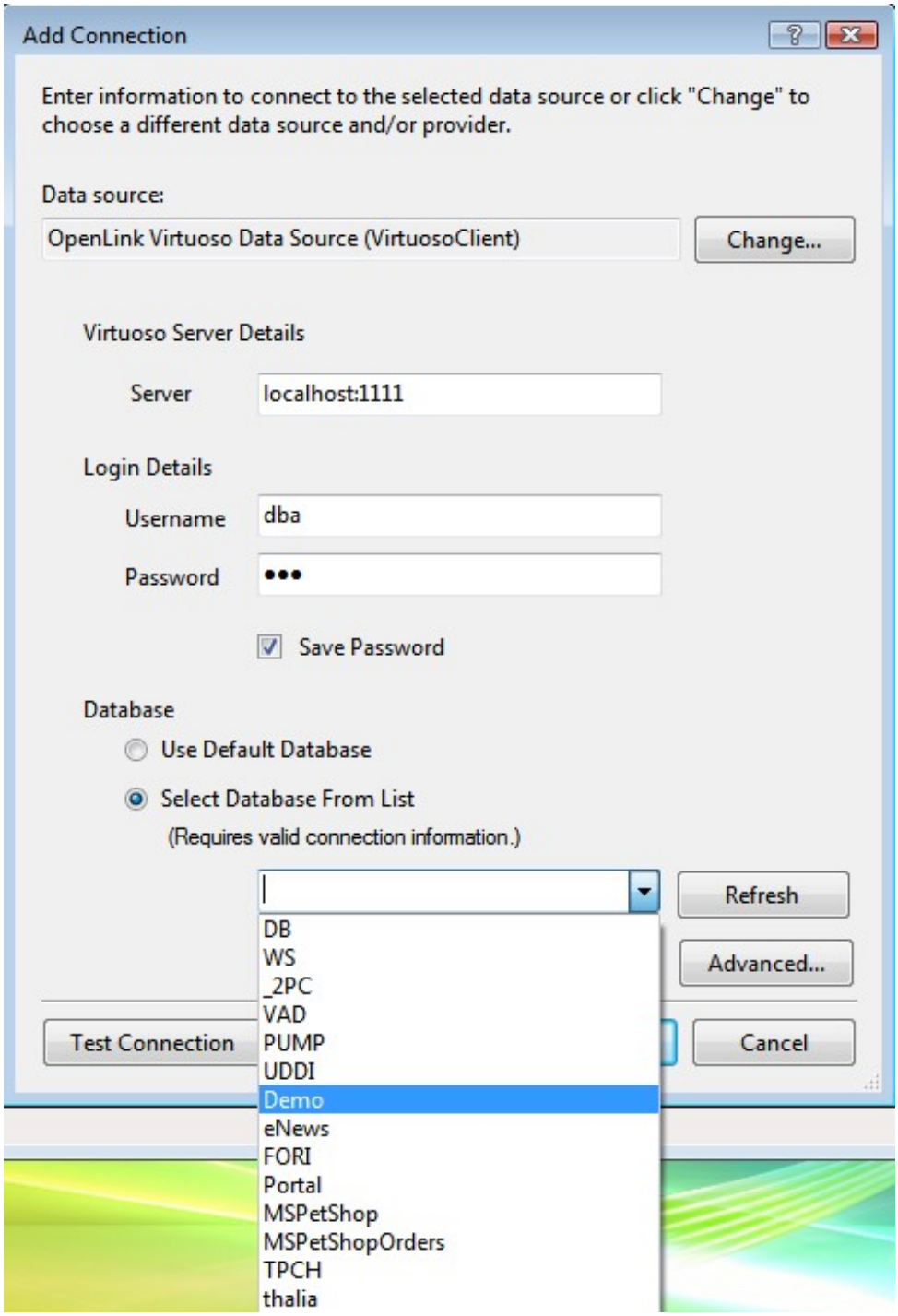
11. In the Add Connection dialog, specify the hostname, portno, username, and password for the target Virtuoso Server and check the Save Password check box.

Figure 2.82. Add Connection dialog



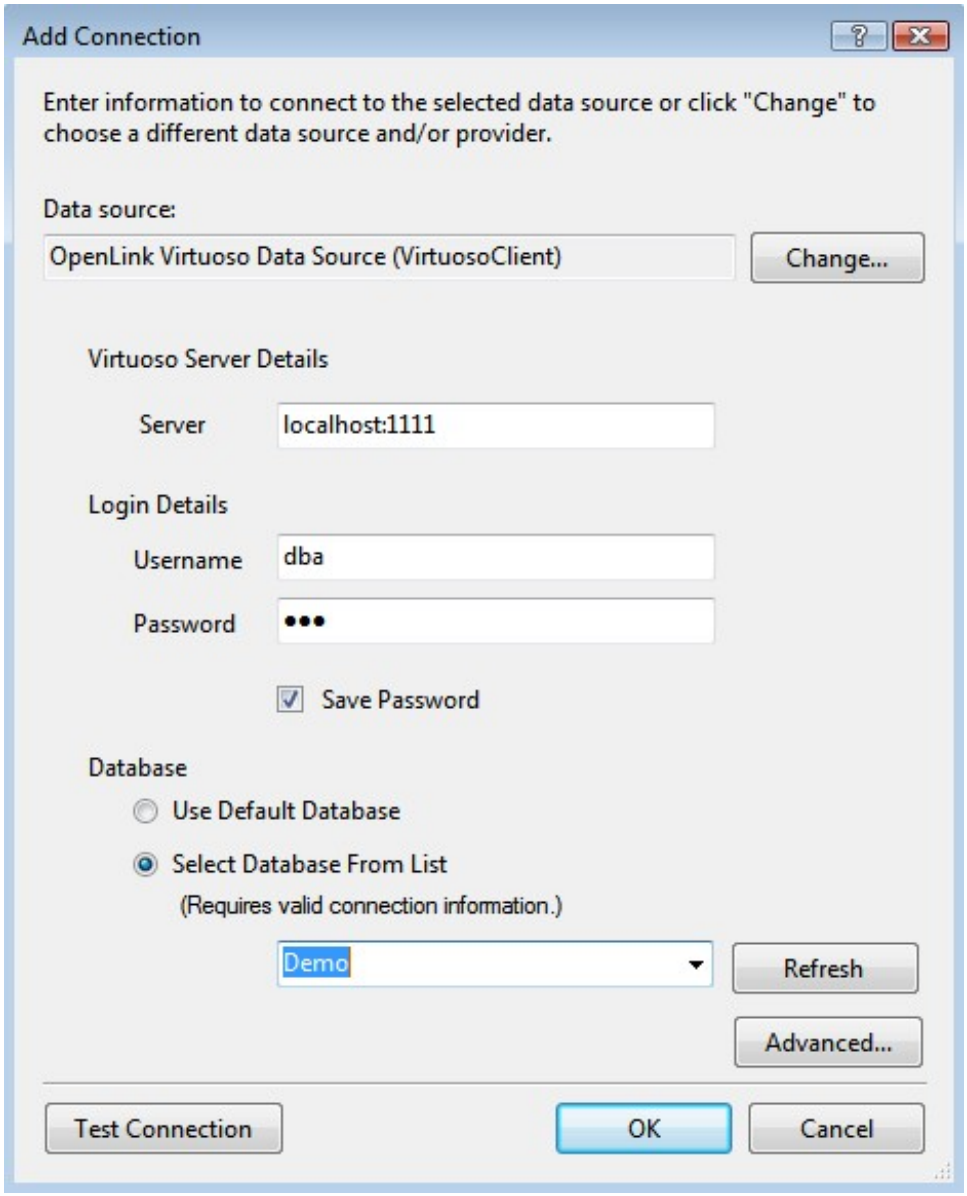
12. Select the Select Database From List radio button and choose School from the drop down list of available databases.

Figure 2.83. Select Database From List



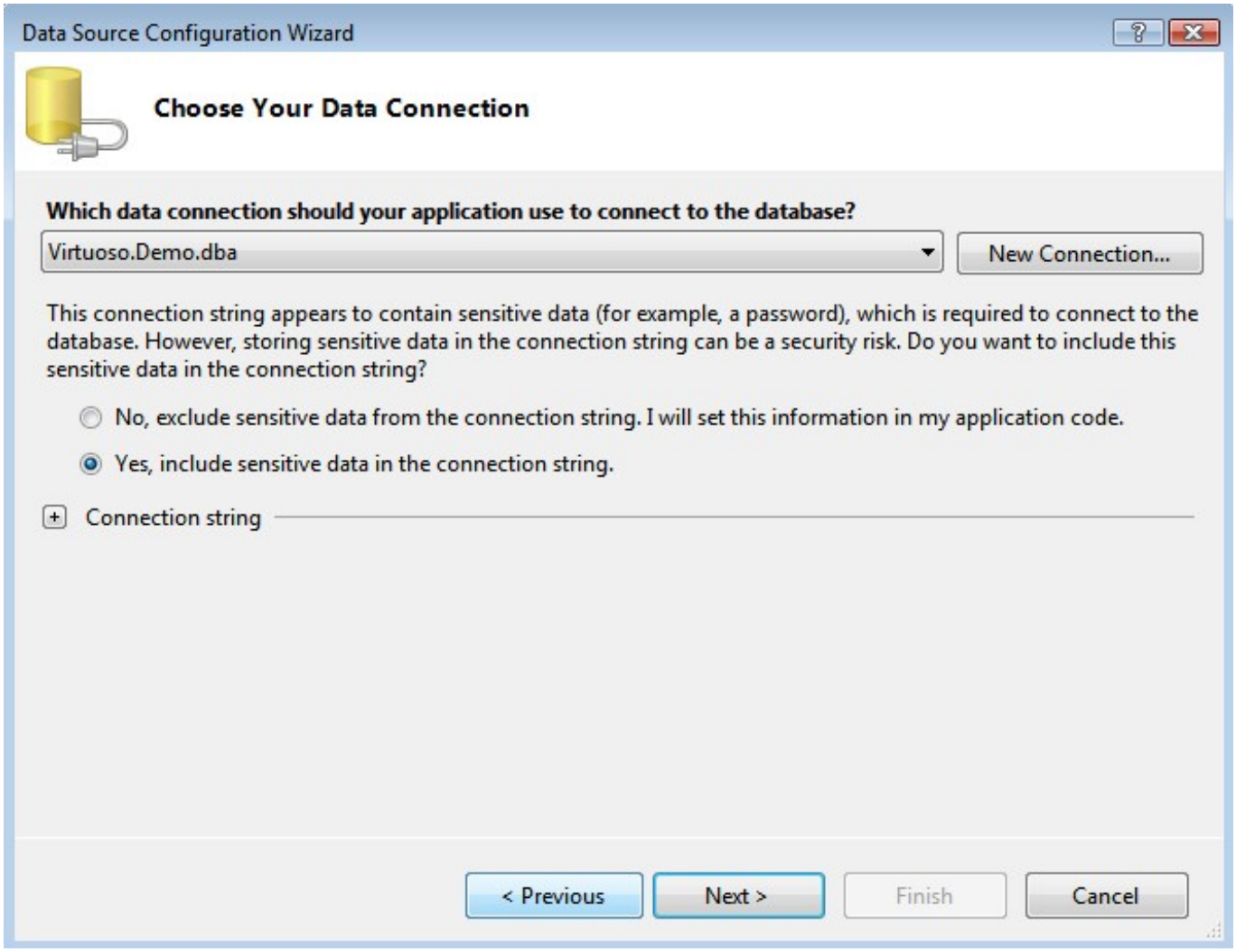
13. Click the Test Connection button to verify the connection is successful and then click OK to add the connection.

Figure 2.84. Test Connection



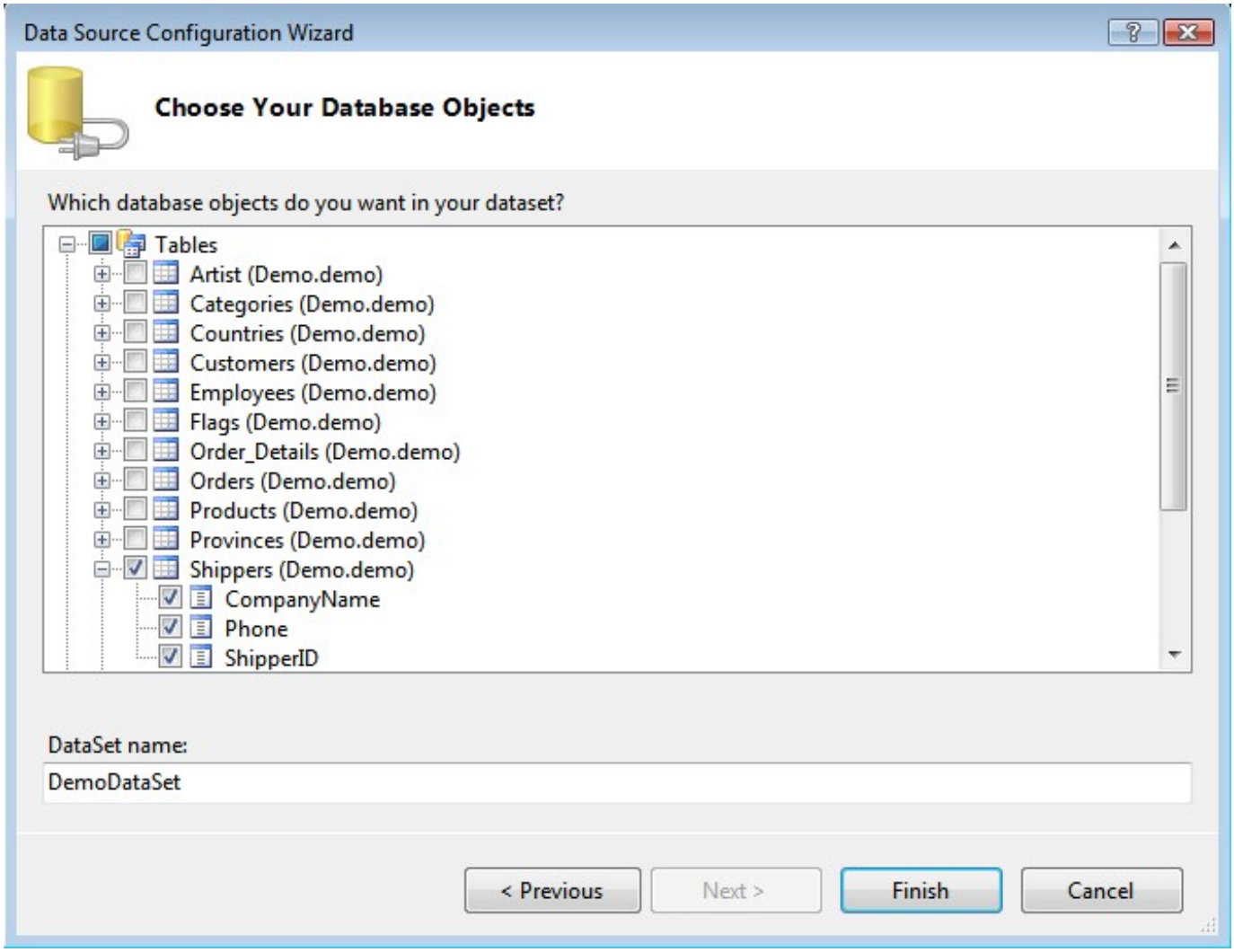
14. Select the Yes, include sensitive data in the connect string radio button and click Next to continue.

Figure 2.85. connect string



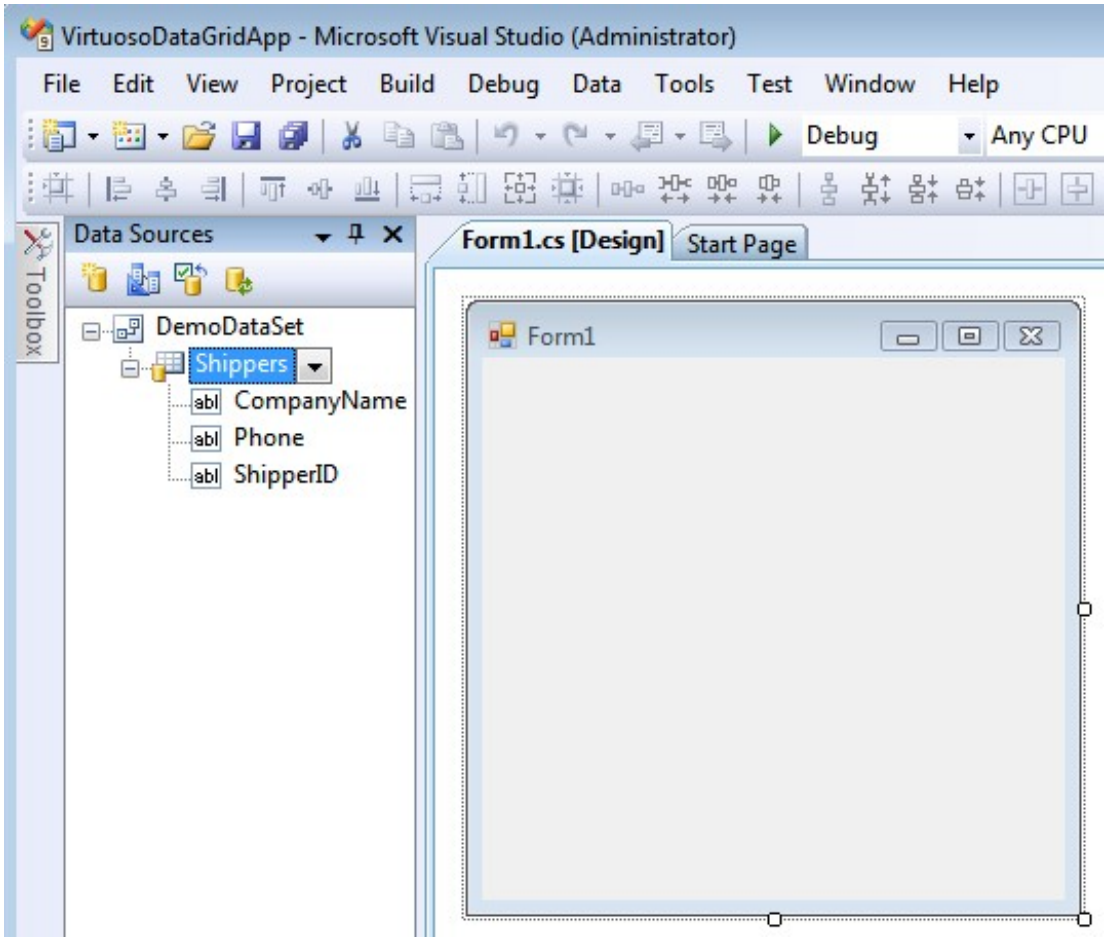
15. In the Choose your Database Objects dialog select the table and columns to be queried, the shippers table is used in this example.

Figure 2.86. Choose your Database Objects



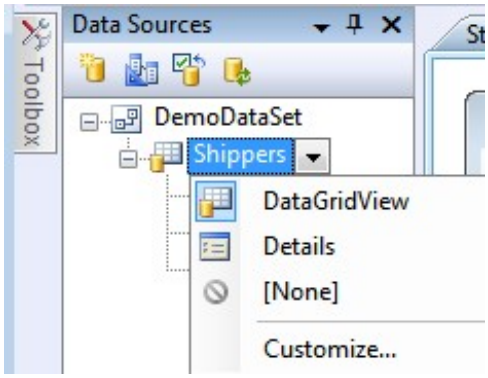
16. A DataSet for the shippers database object is created

Figure 2.87. shippers database



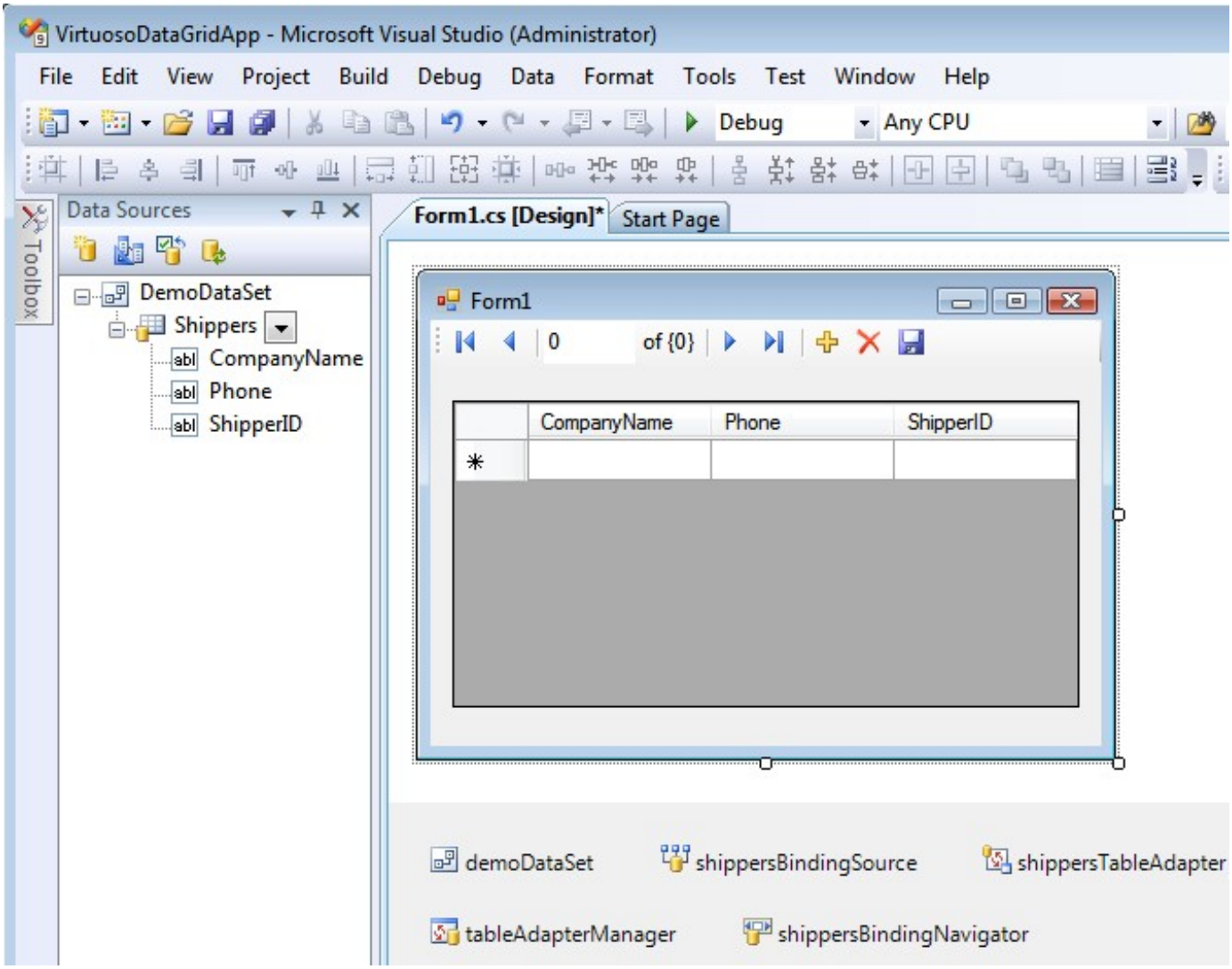
17. From the drop down list box next to the Shippers table ensure the DataGridView item is selected

Figure 2.88. DataGridView



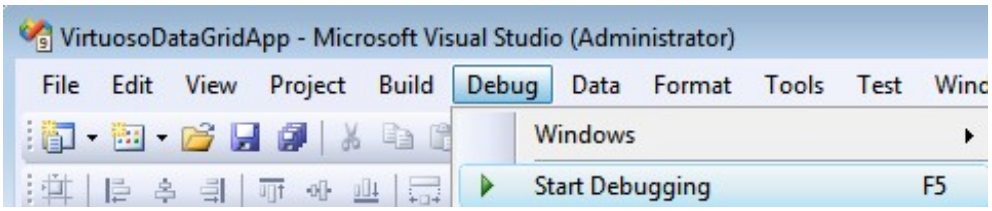
18. Drag the Shippers DataSet item onto the Form to create a scrollable and editable association of the Shippers table object with the Data Grid View automatically.

Figure 2.89. Drag



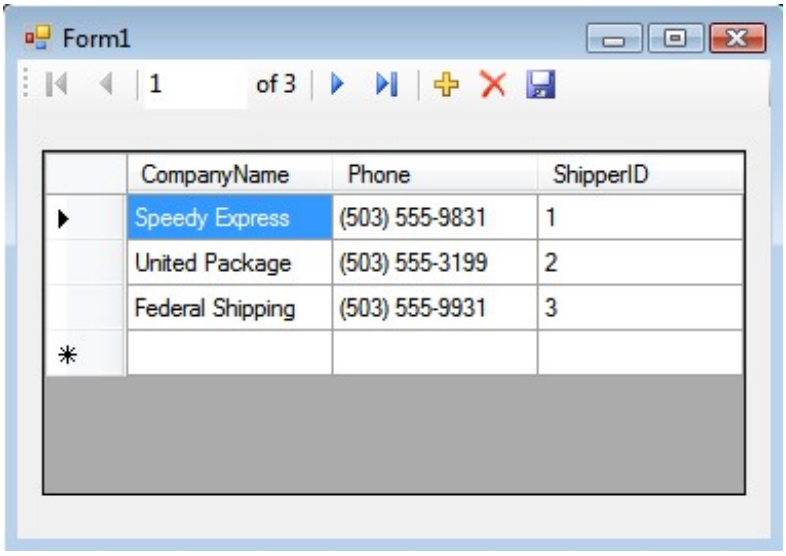
19. From the Debug Menu select the Start Debugging menu item to run the application.

Figure 2.90. Debug



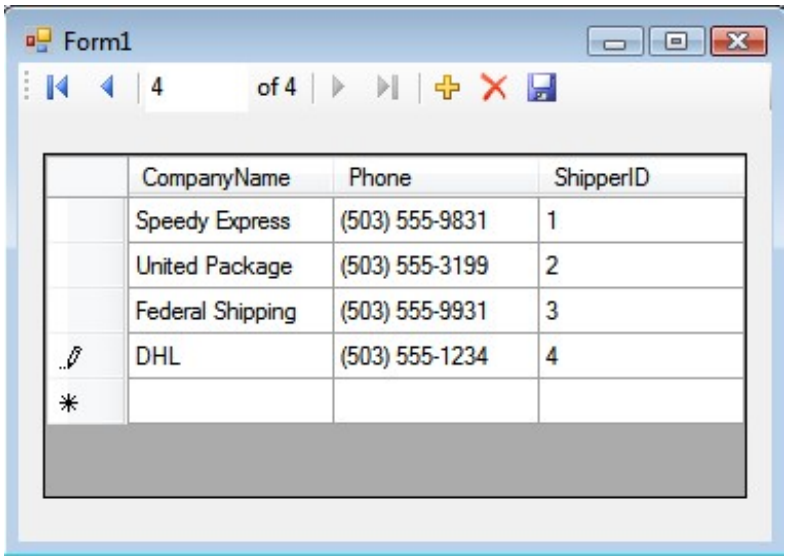
20. The data in the Shippers table will be displayed in the DataGrid application created.

Figure 2.91. DataGrid



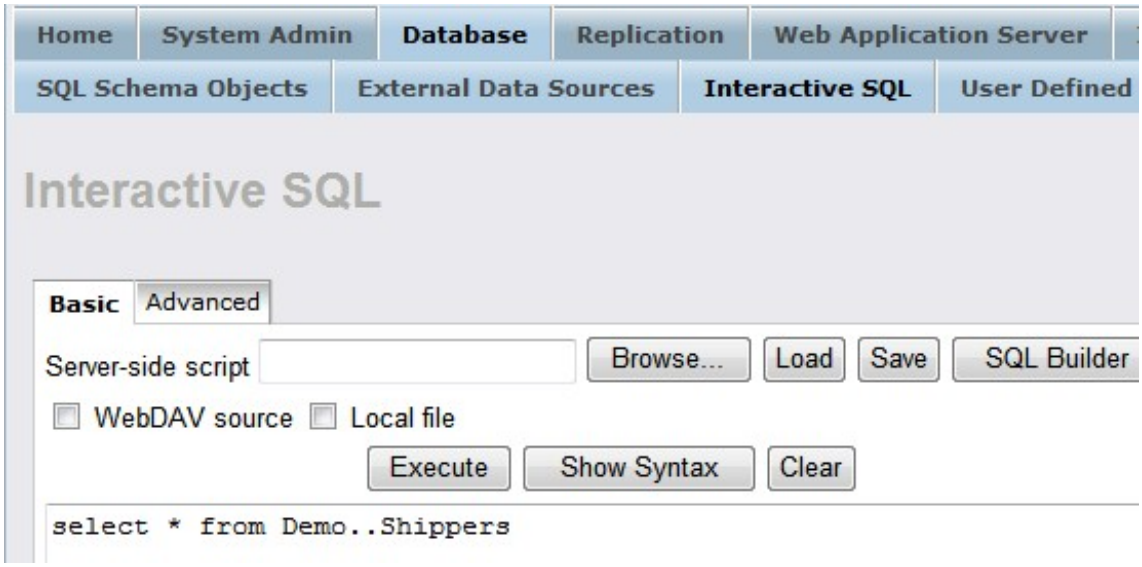
21. A new row can be inserted (updated or deleted) as indicated for the new DHL record inserted below and the save button clicked to save the change to the database.

Figure 2.92. new row



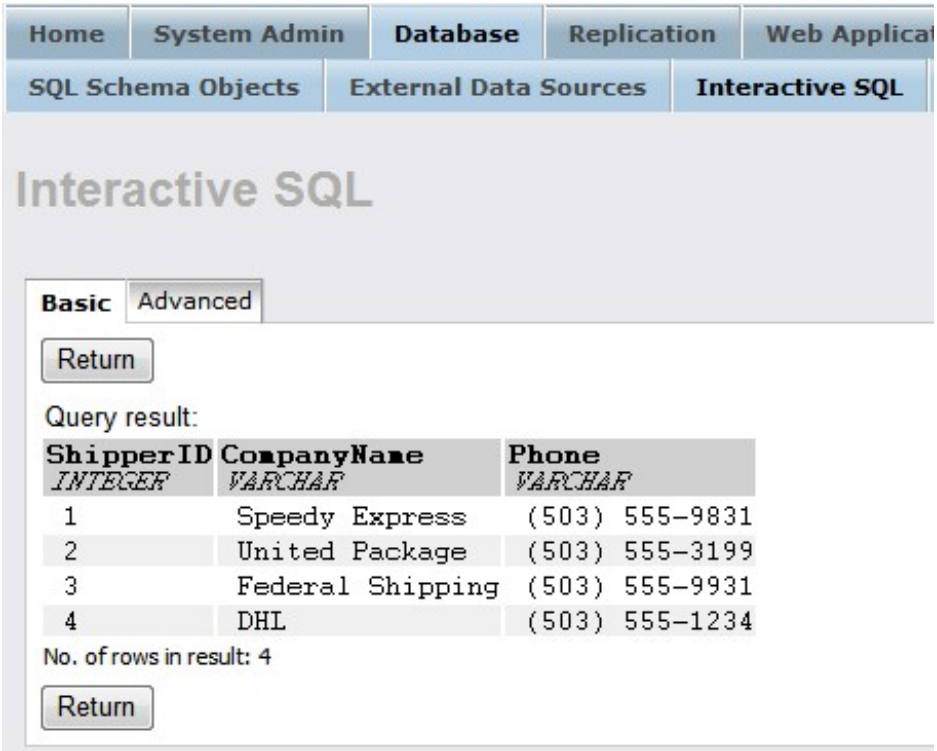
22. The Virtuoso Interactive SQL tab of the Conductor can be used to run the query select * from Demo..Shippers.

Figure 2.93. Interactive SQL tab



23. To verify the change has been successfully made in the database.

Figure 2.94. verify



The task is now complete.

2.7. Using Visual Studio 2008 to Build an Entity Frameworks based Windows Form Application

Virtuoso Entity Framework School DB Windows Form Application. This section demonstrates how Visual Studio 2008 can be used to generate mapping files for an Entity Data Model (EDM), based on a 1:1 mapping to tables in the School database. This will be done by creating a Windows Forms application in Visual Studio, create queries that access data in the School model, bind the query results to display controls to show the results of the queries, and then make updates to objects and persist the changes to the database.

2.7.1. Pre-requisites

In order to create an Entity Framework Application in your own environment you will need:

1. Microsoft Visual Studio 2008 SP1, the ADO.NET Entity Framework runtime and associated tools are included in Visual Studio 2008 SP1.
2. A running Virtuoso Universal Server instance.

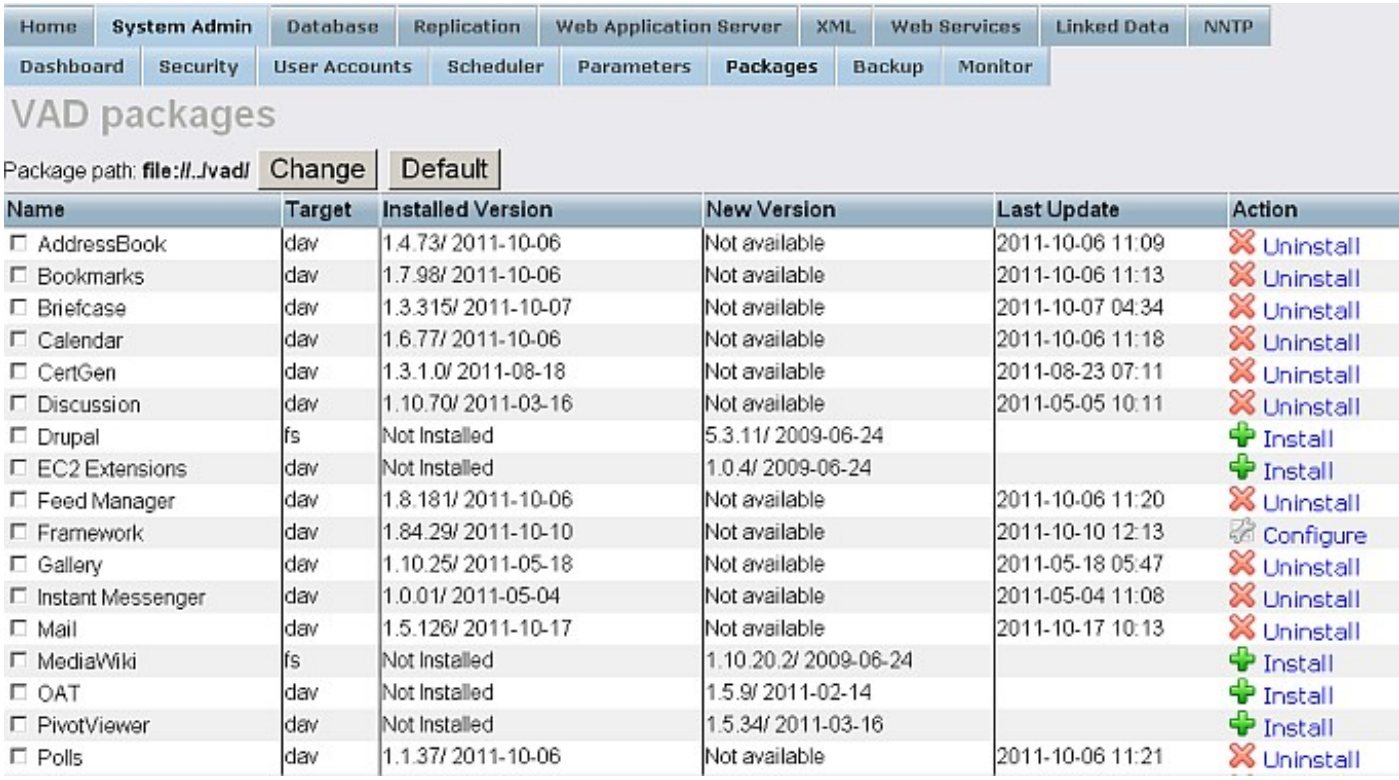
2.7.2. Create the School database and schema

1. Download the Schools Database VAD (schools_db_dav.vad) package.
2. Navigate to the

   System Admin -> Packages

   tab of the Virtuoso Conductor.

   Figure 2.95. Conductor Packages



3. Scroll down to the

   Install Package

   section of the tab, use the

   Upload Package

   option

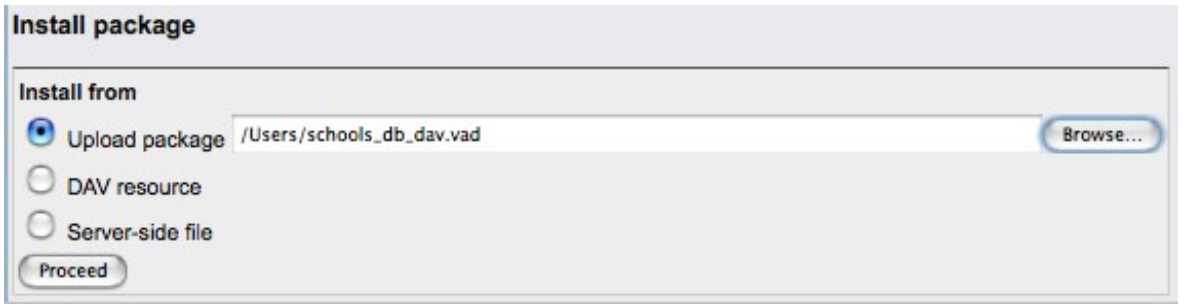
   Browse

   button to locate the schools_db_dav.vad package and click

   proceed

   .

Figure 2.96. Install package

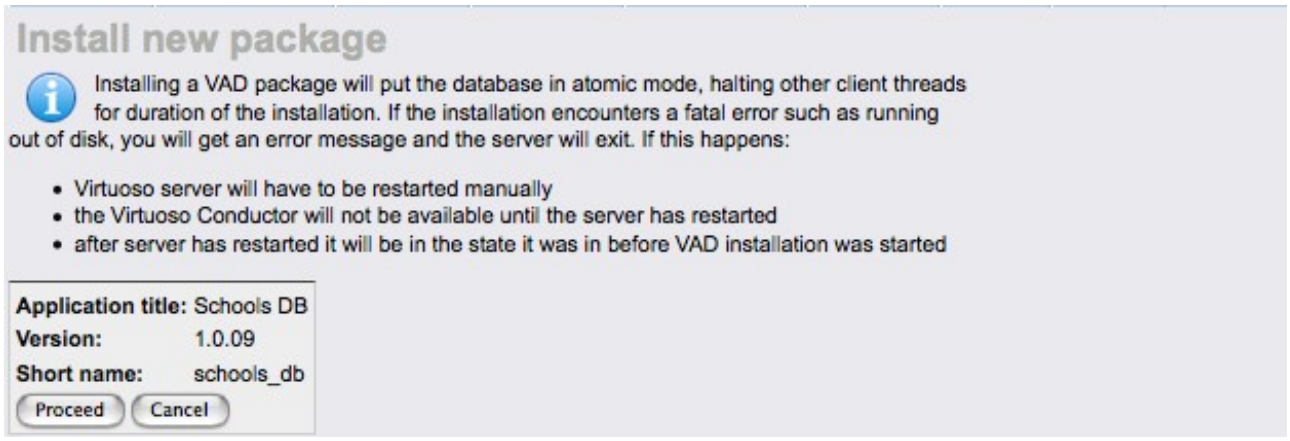


4. Click the

   Proceed

   button to begin the installation process.

   Figure 2.97. Install package.



5. Once complete return to the

   Packages

   tab and scroll down to confirm the *schools_db* package is listed as installed.

   Figure 2.98. Install package

Create the Course Manager application using Visual Studio

1. Launch the Visual Studio 2008 SP1 IDE.

   Figure 2.99. Visual Studio 2008 SP1 IDE

2. On the

   File menu

   click

   New Project

   .
3. Choose either

   Visual Basic

   or

   Visual C#

   in the

   Project Types

   pane.
4. Select

   Windows Forms Application

   in the

   Templates

   pane.
5. Enter

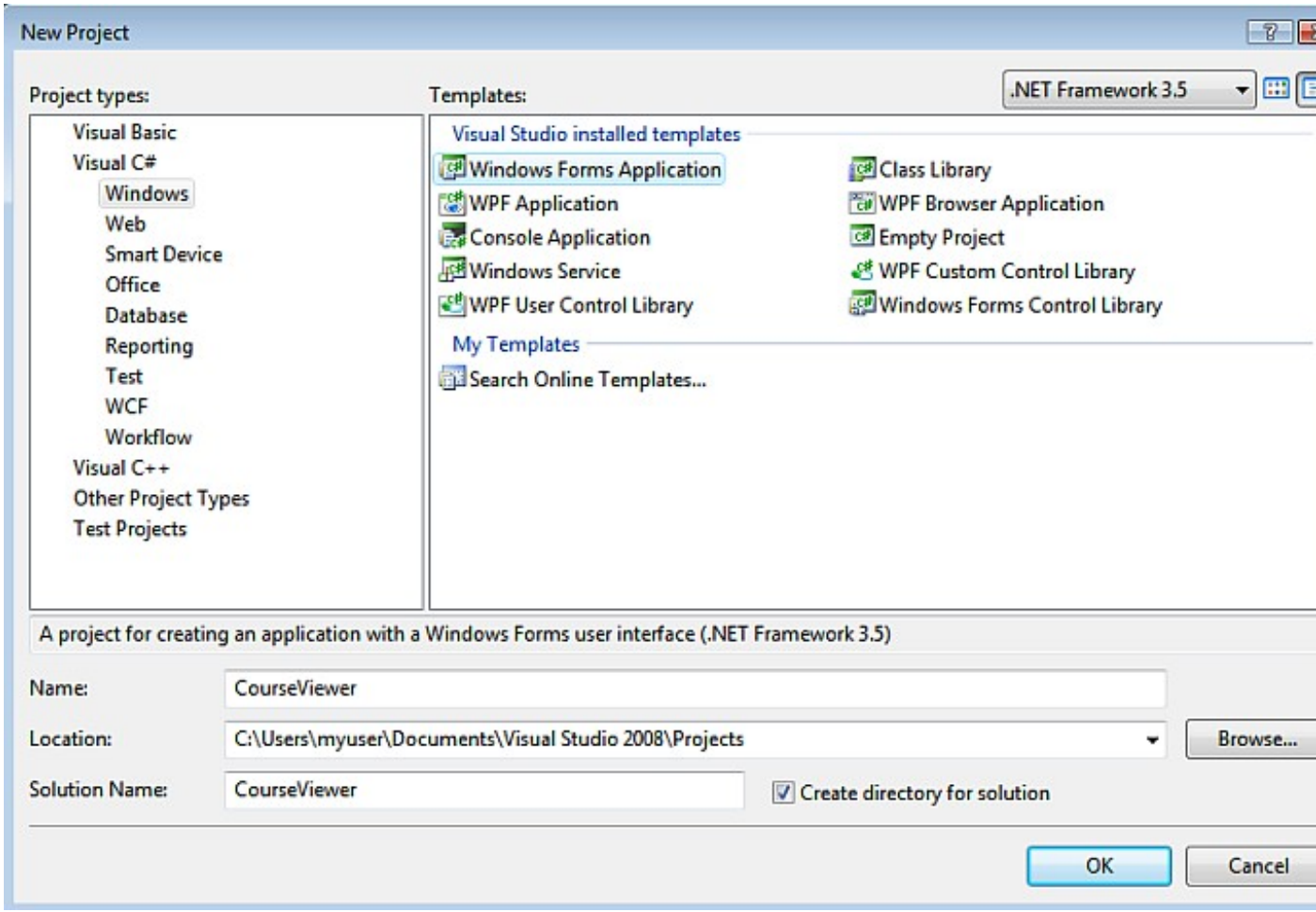
   CourseManager

   for the project name, and then click

   OK

   .

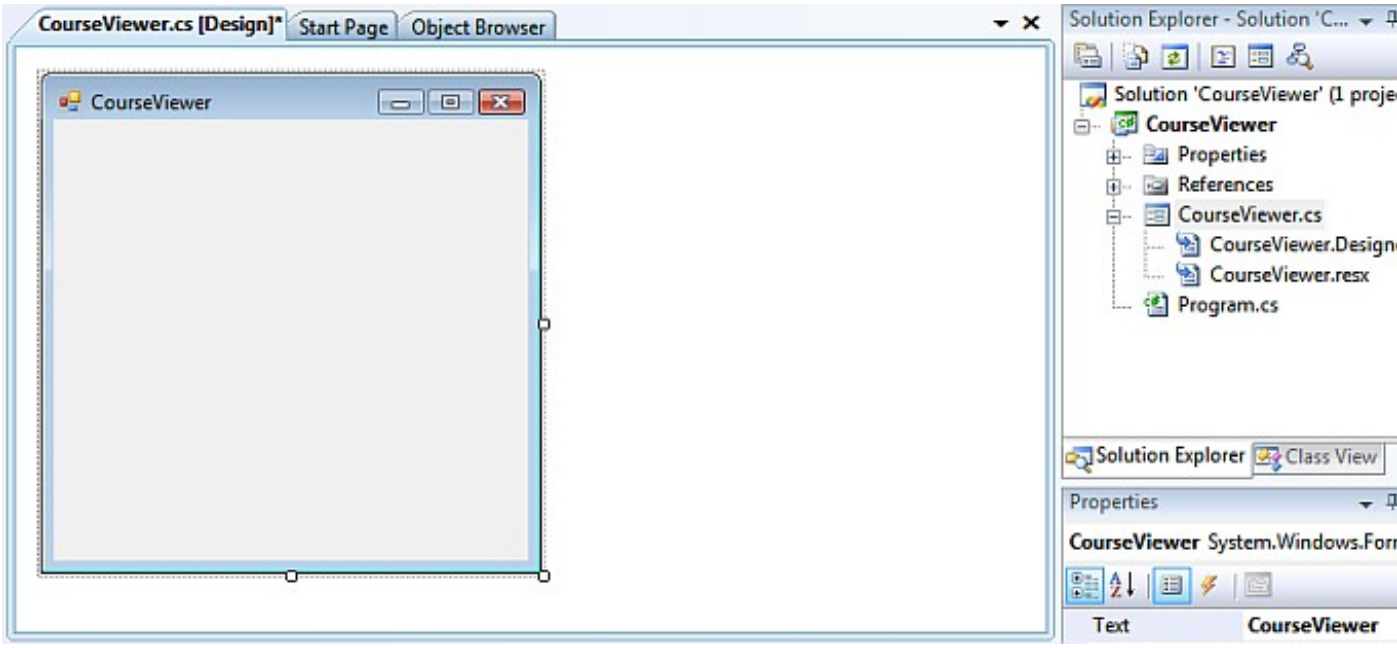
Figure 2.100. Create project



Create the Course Viewer form

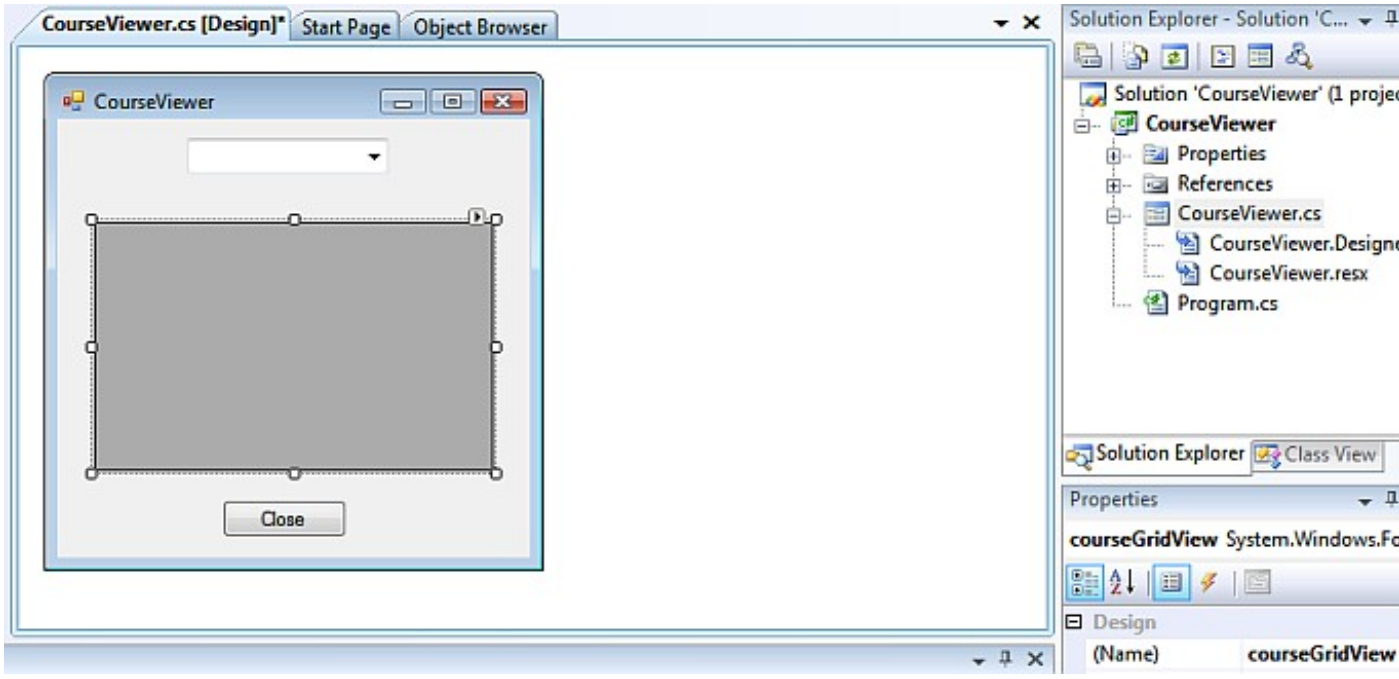
1. In the *CourseManager* project, select the default form (Form1).
2. In the *File Properties* pane, change the File Name to *CourseViewer.vb* or *CourseViewer.cs* .
3. In *Solution Explorer* , double-click *CourseViewer.vb* or *CourseViewer.cs* to open the form.

Figure 2.101. CourseViewer.vb



4. In the *Toolbox* , expand *Common Controls* , drag the *ComboBox* control to the form, and change the name of the control to *departmentList* .
5. In the *Toolbox* , drag the *Button* control to the form, change the *Name* of the control to *closeForm* , and change the *Text value* to *Close* .
6. In the *Toolbox* , expand *Data* , drag the *DataGridView* control to the form, and change the *Name* of the control to *courseGridView* .

Figure 2.102. DataGridView



7. Double-click the *closeForm* button control. This opens the code page for the form and creates the closeForm_Click event handler method.
8. In the *closeForm_Click* event handler method, type the following code that closes the form:

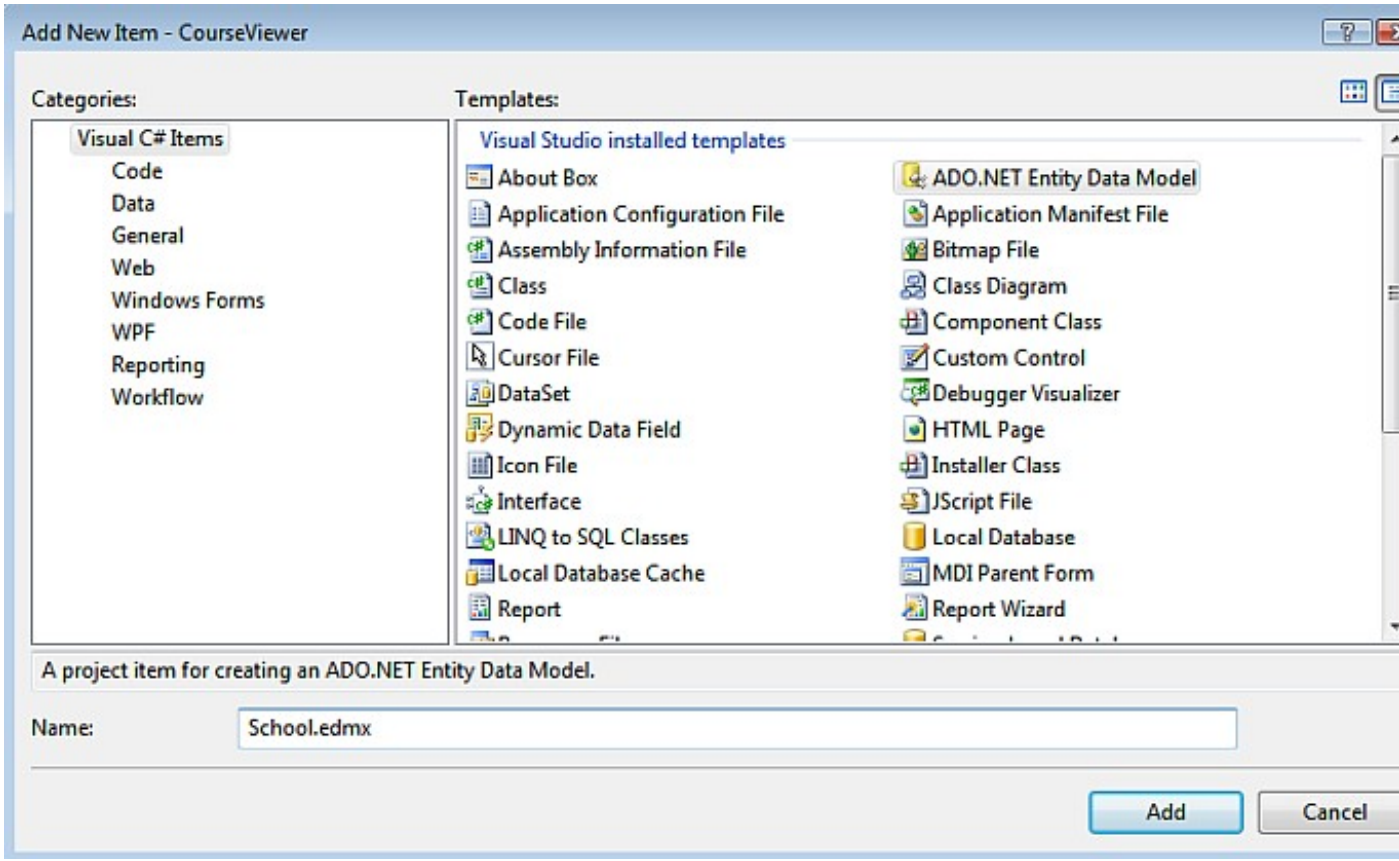
```
Visual Basic

' Close the form.
Me.Close()

C#

// Close the form.
this.Close();
```

2.7.3. Generating the School Entity Data Mode

Add the ADO.NET Entity Data Model item template

1. Select the CourseManager project in Solution Explorer, right-click, point to Add, and then click New Item.
2. Select ADO.NET Entity Data Model in the Templates pane.
3. Type School.edmx for the model name and click Add. The opening page of the Entity Data Model Wizard is displayed.

Figure 2.103. School.edmx



Generate the EDM

1. Select

   Generate from database

   in the

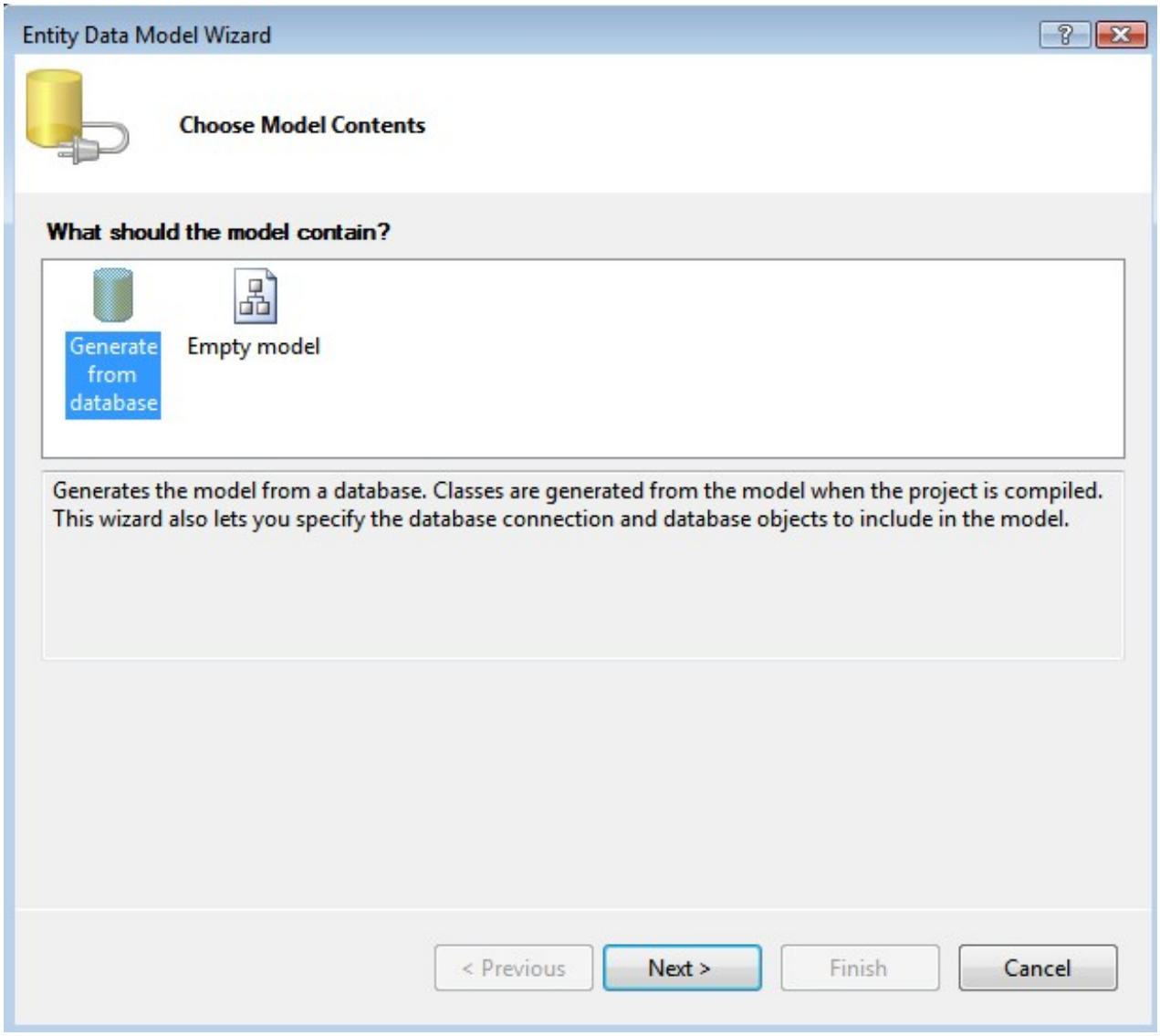
   Choose Model Contents dialog box

   . Then click

   Next

   .

   Figure 2.104. Model Contents

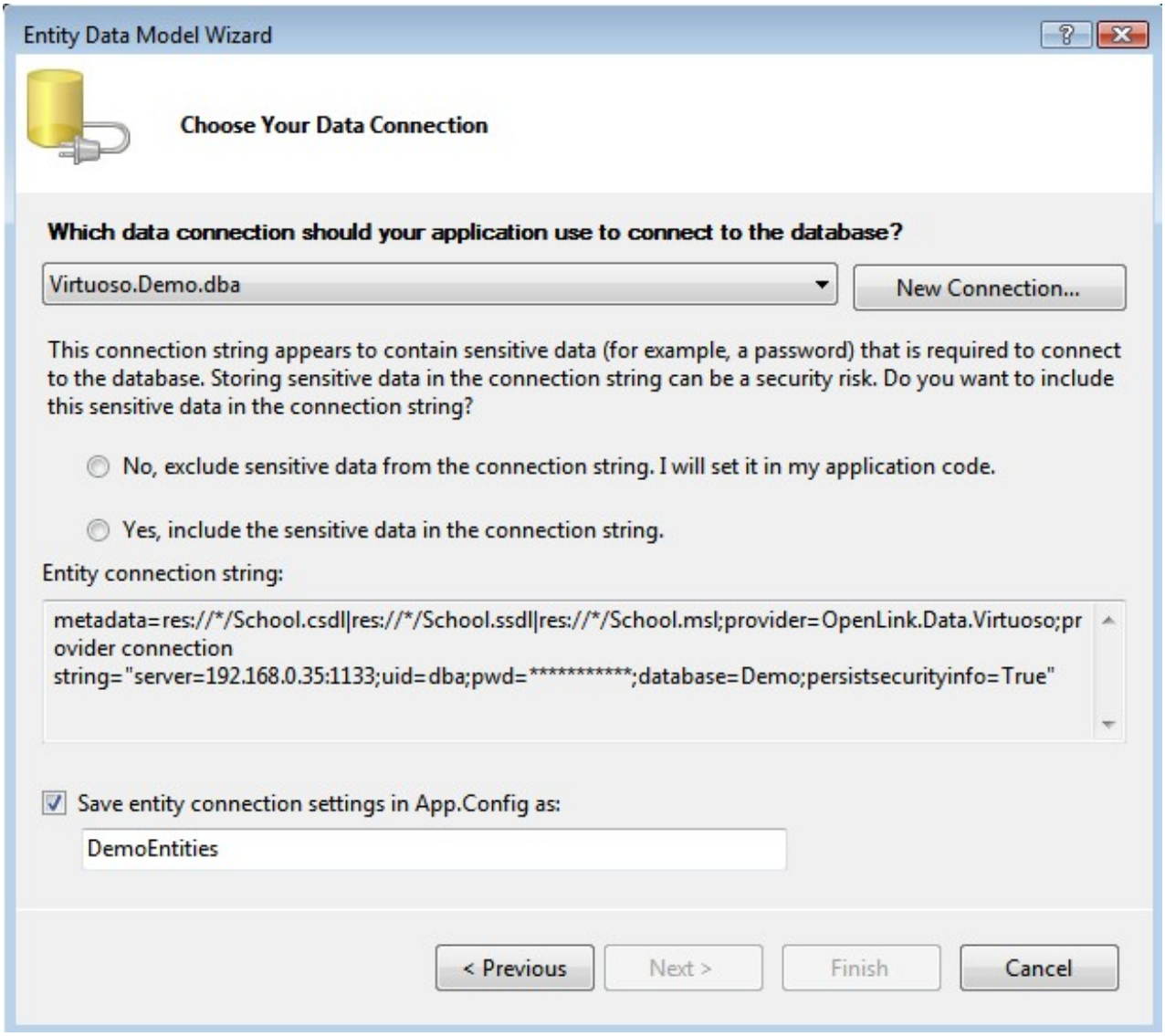


2. Click the

   New Connection

   button.

Figure 2.105. New Connection



3. Choose the OpenLink

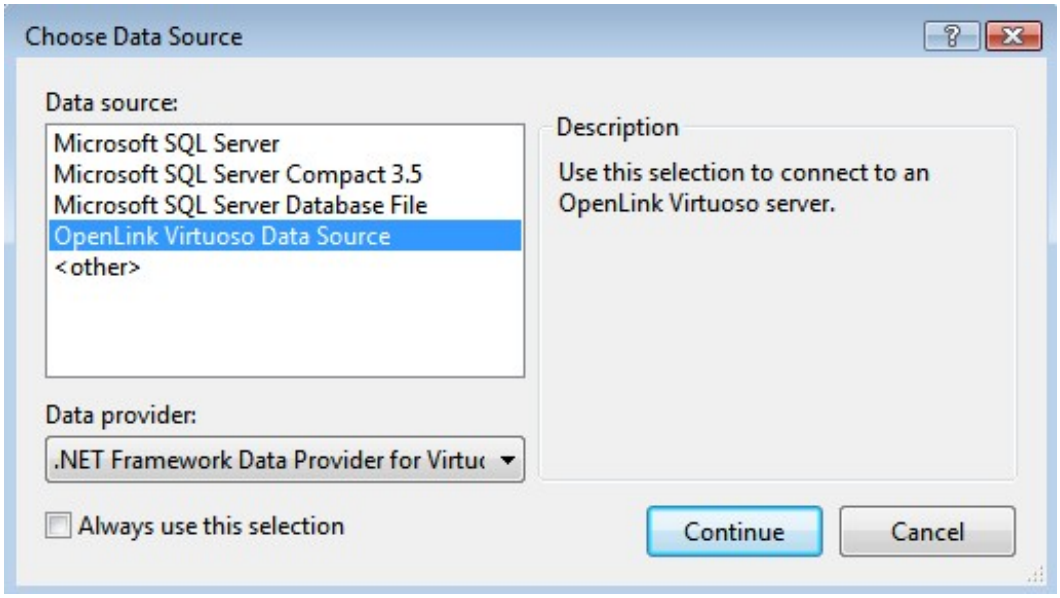
   Virtuoso Data Source

   and click

   Continue

   .

   Figure 2.106. Virtuoso Data Source



4. In the

   Connection Properties

   dialog specify the

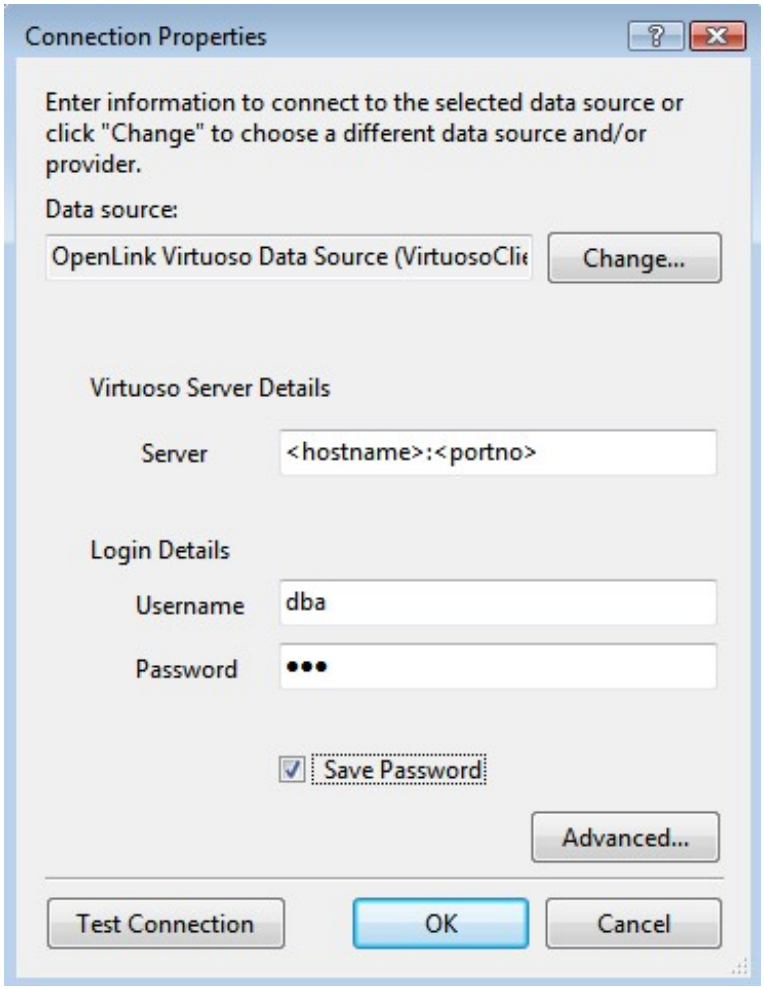
   hostname, portno, username and password

   for the target Virtuoso Server and click the

   Advanced

   button.

   Figure 2.107. Connection Properties



5. In the

   Advanced Properties

   dialog set the

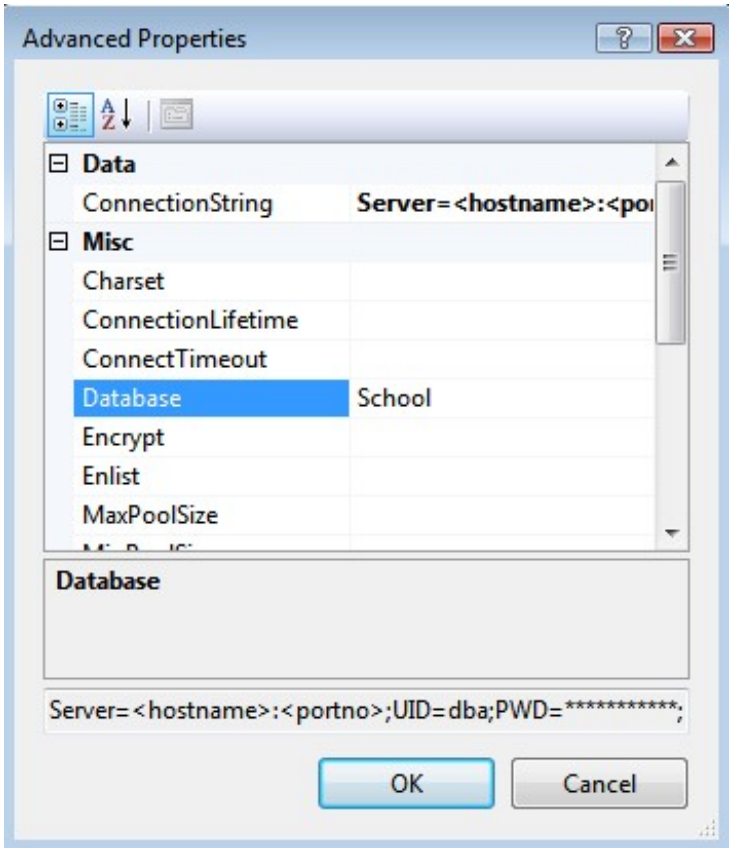
   Database parameter

   to

   School

   and click OK.

Figure 2.108. Advanced Properties

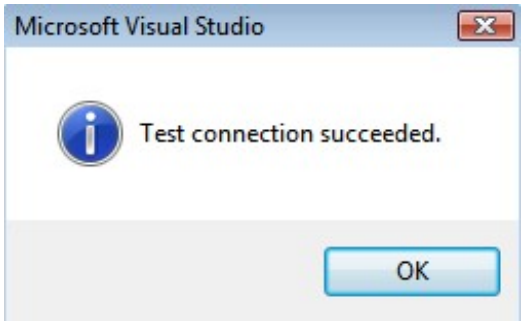


6. Press the

   Test Connection

   dialog to verify the database is accessible.

   Figure 2.109. Test Connection



7. Set the

   entity connect string name

   to

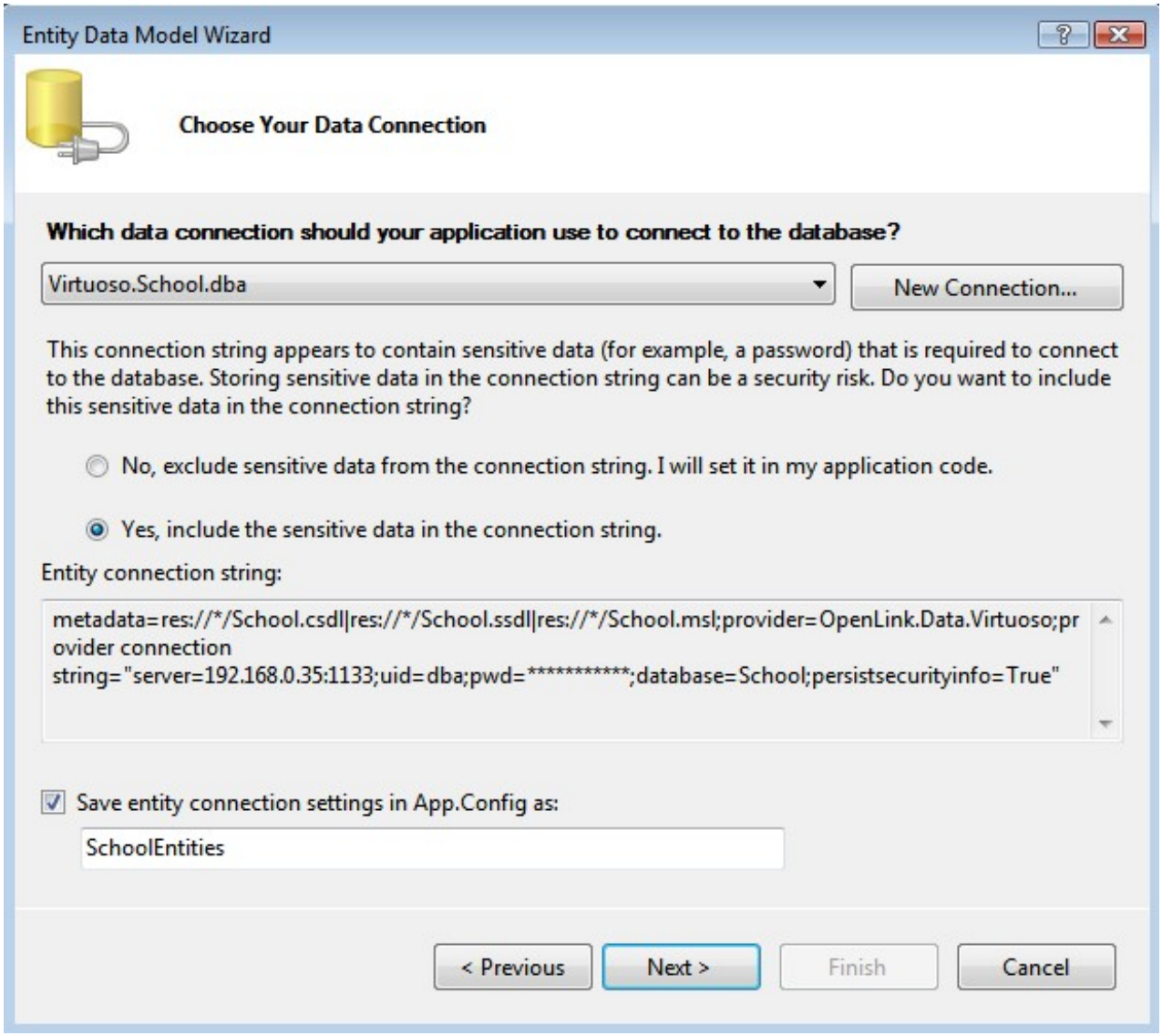
   SchoolEntities

   and click

   Next

   .

   Figure 2.110. entity connect string name



8. In the

   Choose your Database Objects

   page select the

   Tables

   check box to select all table in the

   School database

   for addition to the EDM, set the

   Model Namespace

   to

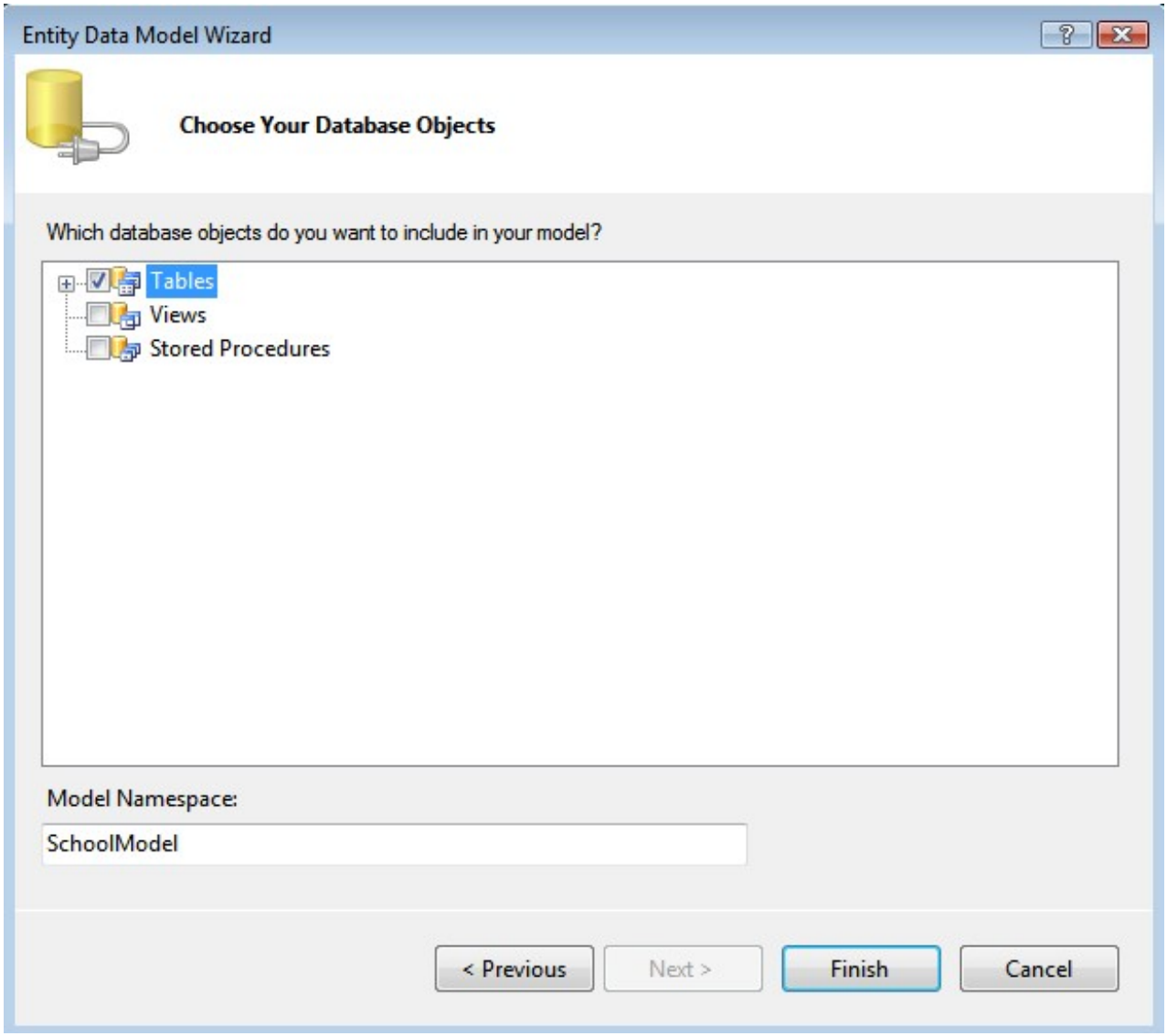
   SchoolModel

   and click

   Finish

   .

Figure 2.111. Database Objects



View the EDM in the ADO.NET Entity Data Model Designer

1. In the

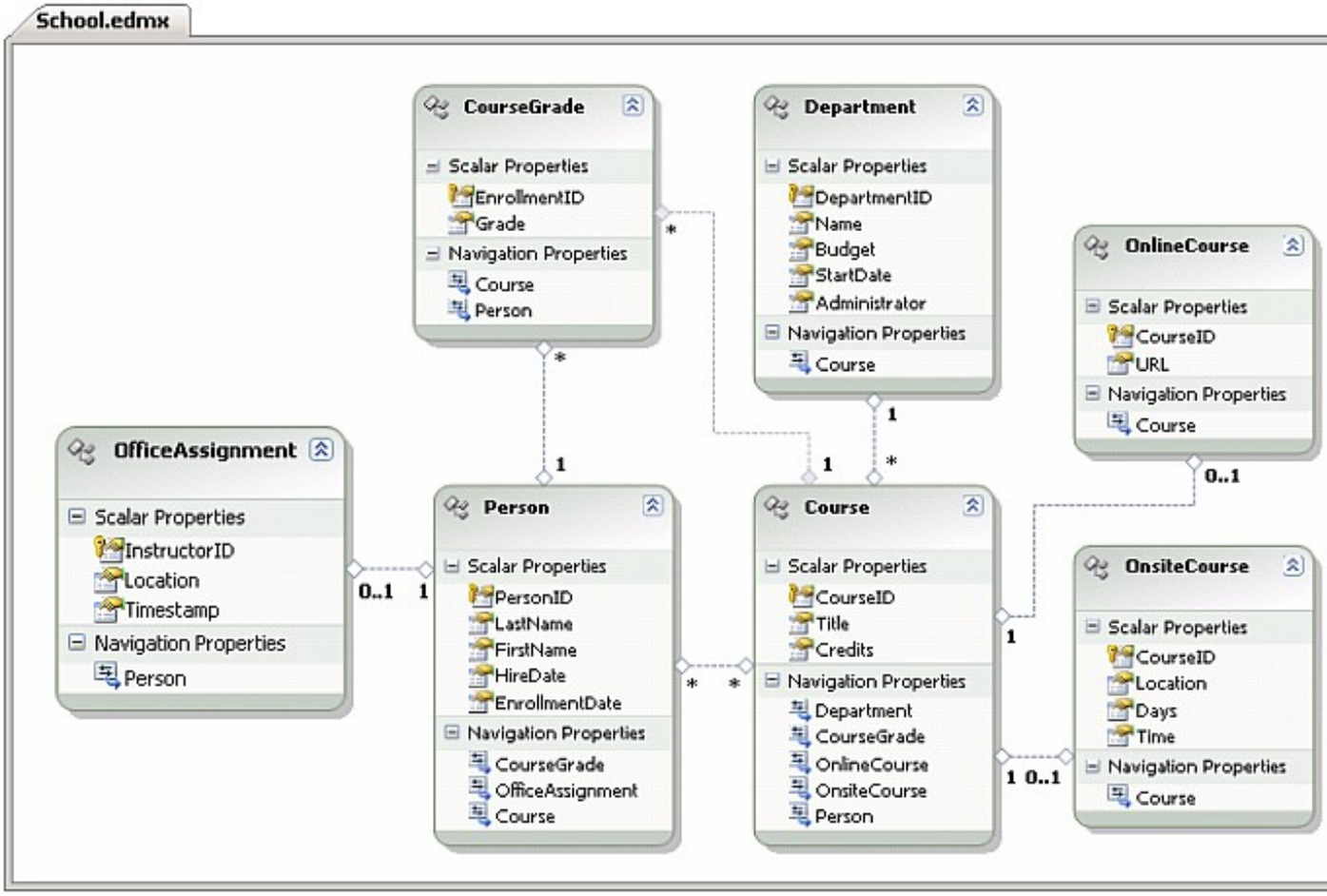
   Solution Explorer

   , double-click the

   School.edmx

   file. This displays the School model in the ADO.NET Entity Data Model Designer window.

   Figure 2.112. Solution Explorer



2. From the

   View

   menu, select

   Other Windows

   , and then click

   Model Browser

   to display the

   Entity Model Browser

   window.
3. Expand the

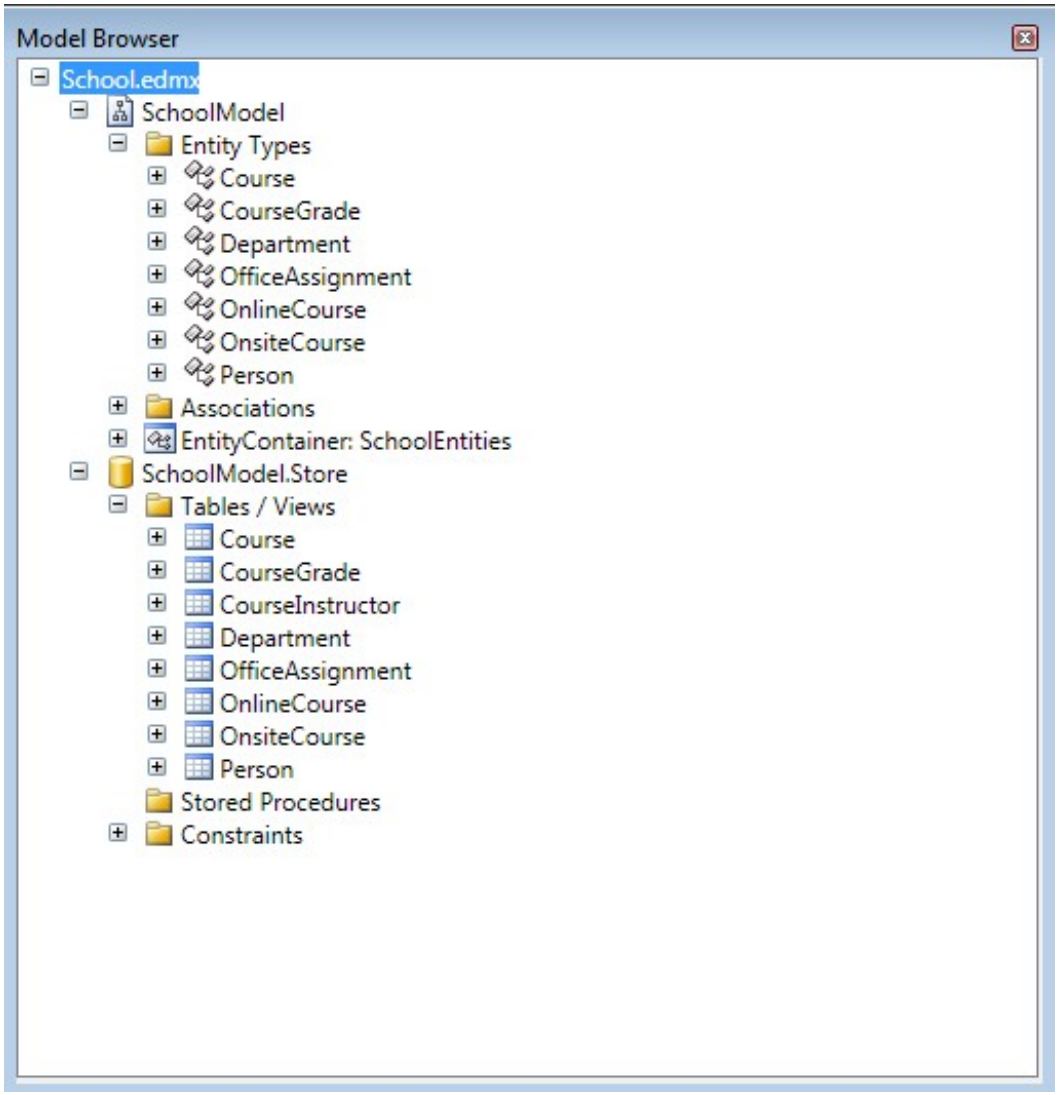
   SchoolModel

   and

   SchoolModel.Store

   nodes to view conceptual and store definitions, respectively.

   Figure 2.113. SchoolModel



4. From the

   View menu

   , select

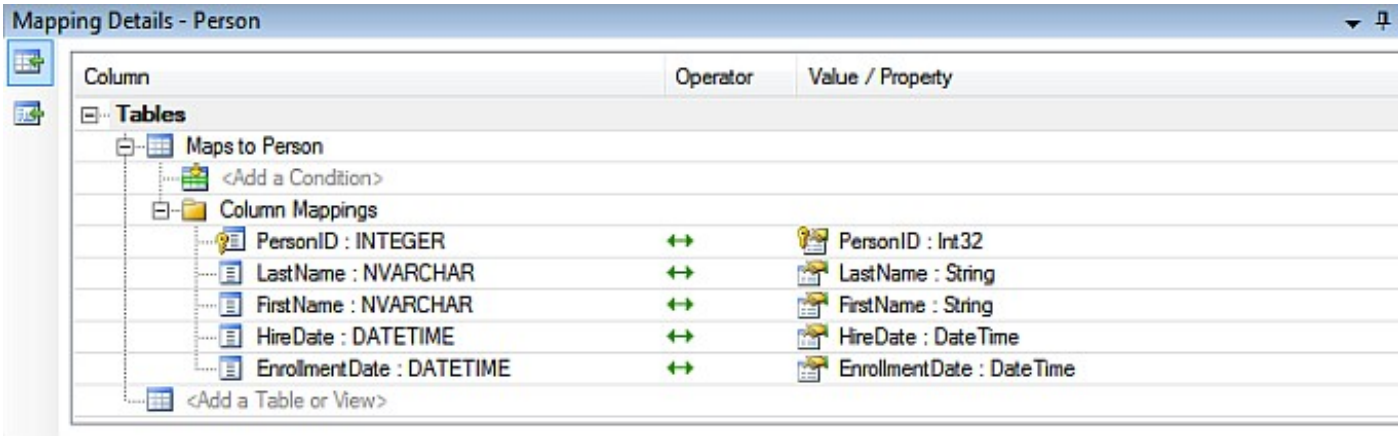
   Other Windows

   , click

   Mapping Details

   , and then click an entity (Person for example) or association in the Entity Designer.
5. This displays the Entity Mapping Details window with information about the object-relational mapping for the selected object.

Figure 2.114. View menu



2.7.4. Querying Entities and Associations

This section creates strongly-typed queries against the CLR objects that represent entities and associations in the School model, and bind display controls to the object collections returned from these queries.

Query the departments in the School database

1. At the beginning of the code file for the

   CourseViewer

   form, add the following

   using

   (C#) or

   Imports

   (Visual Basic) statements to reference the model created from the School database and the entity namespace.

   ```
   Visual Basic

   Imports System.Data.Objects
   Imports System.Data.Objects.DataClasses

   C#

   using System.Data.Objects;
   using System.Data.Objects.DataClasses;
   ```

2. At the top of the partial class definition for the

   CourseViewer

   form, add the following code that creates an

   ObjectContext

   instance.

   ```
   Visual Basic

   ' Create an ObjectContext instance based on SchoolEntity.
   Private schoolContext As SchoolEntities

   C#

   // Create an ObjectContext instance based on SchoolEntity.
   ```

```
private SchoolEntities schoolContext;
```

3. In the

   CourseViewer

   form designer, double-click the

   CourseViewer

   form. This opens the code page for the form and creates the

   courseViewer _Load

   event handler method.
4. In the

   courseViewer _Load

   event handler method, copy and paste the following code that defines the

   DataGridView

   , executes a query that returns a collection of departments (ordered by

   Name

   ), and binds the collection of

   Department

   objects to the departmentList control.

```
Visual Basic

' Initialize the ObjectContext.
schoolContext = New SchoolEntities()

' Define a query that returns all Department objects and related
' Course objects, ordered by name.
Dim departmentQuery As ObjectQuery(Of Department) = _
    schoolContext.Department.Include("Course").OrderBy("it.Name")

Try
    ' Bind the ComboBox control to the query, which is
    ' executed during data binding.
    Me.departmentList.DisplayMember = "Name"
    Me.departmentList.DataSource = departmentQuery
Catch ex As Exception
    MessageBox.Show(ex.Message)
End Try

C#

// Initialize the ObjectContext.
schoolContext = new SchoolEntities();

// Define a query that returns all Department objects and related
// Course objects, ordered by name.
ObjectQuery<Department> departmentQuery =
    schoolContext.Department.Include("Course").OrderBy("it.Name");

try
{
    // Bind the ComboBox control to the query, which is
    // executed during data binding.
    this.departmentList.DisplayMember = "Name";
    this.departmentList.DataSource = departmentQuery;
```

```
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

Display courses for the selected department

1. In the

   CourseViewer

   form designer, double-click the

   departmentList

   control. This creates the

   departmentList_SelectedIndexChanged

   event handler method.
2. Paste the following code that loads the courses that are related to the selected department.

```
Visual Basic

Try
    ' Get the object for the selected department.
    Dim department As Department = _
        CType(Me.departmentList.SelectedItem, Department)

    ' Bind the grid view to the collection of Course objects
    ' that are related to the selected Department object.
    courseGridView.DataSource = department.Course

    ' Hide the columns that are bound to the navigation properties on Course.
    courseGridView.Columns("Department").Visible = False
    courseGridView.Columns("CourseGrade").Visible = False
    courseGridView.Columns("OnlineCourse").Visible = False
    courseGridView.Columns("OnsiteCourse").Visible = False
    courseGridView.Columns("Person").Visible = False

    courseGridView.AutoResizeColumns(DataGridViewAutoSizeColumnsMode.AllCells)
Catch ex As Exception
    MessageBox.Show(ex.Message)
End Try

C#

try
{
    // Get the object for the selected department.
    Department department =
        (Department)this.departmentList.SelectedItem;

    // Bind the grid view to the collection of Course objects
    // that are related to the selected Department object.
    courseGridView.DataSource = department.Course;

    // Hide the columns that are bound to the navigation properties on Course.
    courseGridView.Columns["Department"].Visible = false;
    courseGridView.Columns["CourseGrade"].Visible = false;
    courseGridView.Columns["OnlineCourse"].Visible = false;
    courseGridView.Columns["OnsiteCourse"].Visible = false;
    courseGridView.Columns["Person"].Visible = false;

    courseGridView.AutoResizeColumns(DataGridViewAutoSizeColumnsMode.AllCells);
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
```

```
}
```

2.7.5. Inserting and Updating Data

In this section the changes made to Course objects bound are saved to the DataGridView control to the database and also run the completed Course Manager application.

Save changes made to objects

1. In the

   Toolbox

   , expand

   Common Controls

   , drag the

   Button

   control to the

   CourseViewer

   form designer, change the

   name

   of the control to

   saveChanges

   , and change the

   Text

   value to

   Update

   .
2. In the

   CourseViewer

   form designer, double-click the

   saveChanges

   control. This creates the

   saveChanges_Click

   event handler method.
3. Paste the following code that saves object changes to the database.

```
Visual Basic

Try
    ' Save object changes to the database, display a message,
    ' and refresh the form.
```

```
        schoolContext.SaveChanges()
        MessageBox.Show("Changes saved to the database.")
        Me.Refresh()
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    End Try

    C#

    try
    {
        // Save object changes to the database, display a message,
        // and refresh the form.
        schoolContext.SaveChanges();
        MessageBox.Show("Changes saved to the database.");
        this.Refresh();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
```

4. In the

   closeForm_Click

   event handler method, type the following code. This code disposes of the object context before the form is closed.

```
    Visual Basic

       ' Dispose the object context.
        schoolContext.Dispose()

    C#

        // Dispose the object context.
        schoolContext.Dispose();
```

Build and run the Class Scheduling application

1. From the

   Debug

   menu, select

   Start Debugging

   or

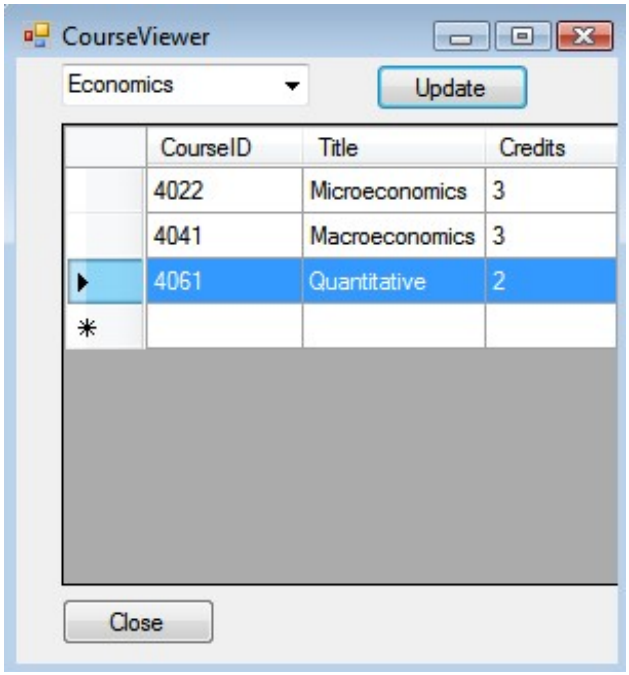
   Start Without Debugging

   , to build and run the application.
2. When the form loads, select a department from the

   ComboBox

   control to display the courses that belong to that department.

   Figure 2.115. ComboBox



3. In the

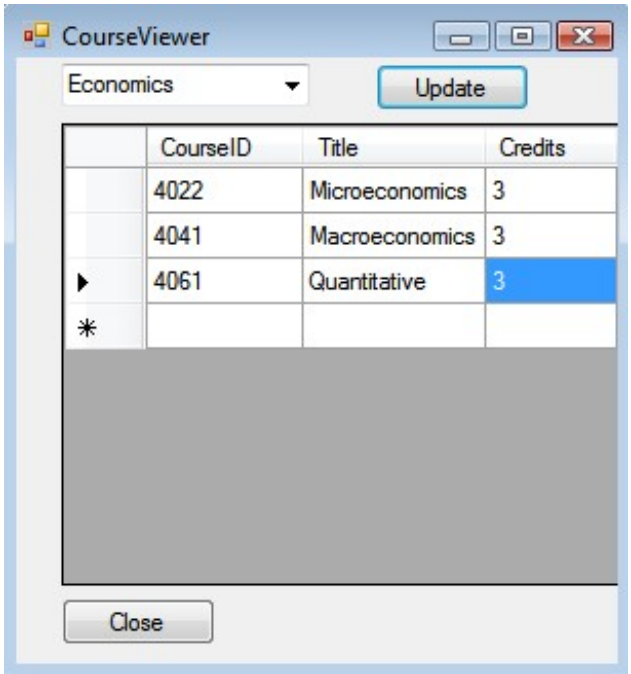
   DataGridView

   , update course information or add a new course and then click

   Update

   to save changes to the database and displays a message box that declares the number of saved changes.

   Figure 2.116. DataGridView



The process is now complete

2.8. Using Visual Studio 2008 to Build an ADO.NET Data Services based Application

Using Microsoft ADO.Net Data Services with Virtuoso

2.8.1. Introduction

The goal of Microsoft ADO.NET Data Services is to enable applications to expose data as a data service that can be consumed by web clients within corporate networks and across the internet. A data service is reachable via regular HTTP requests, using standard HTTP verbs such as GET, POST, PUT and DELETE to perform CRUD operations against the service. The payload format used by the service is controllable by the application, but all options are simple, open formats such as JSON and Atom/APP.

The use of web-friendly technologies make ADO.NET Data Services ideal as a data back-end for AJAX-style applications, Rich Interactive Applications and other applications that need to operate against data that is stored across the web.

2.8.2. Getting Started: Creating Data Services

Pre-requisites

In order to create a data service using ADO.NET Data Services in your own environment you will need Microsoft Visual Studio 2008 SP1. The ADO.NET Entity Framework runtime and associated tools are included in Visual Studio 2008 SP1.

Selecting a Data Source

The ADO.NET Data Service server framework is comprised of two halves. The top-half is the runtime itself; this part is "fixed", and it implements URI translation, the Atom/JSON wire formats, the interaction protocol, etc. This is what makes an ADO.NET Data Service look like an ADO.NET Data Service. The bottom half is the data-access layer and is pluggable. Communication between layers happens in terms of the IQueryable interface plus a set of conventions to map CLR graphs into the URI/payload patterns of ADO.NET Data Services.

The first step in creating an ADO.NET Data Service is to determine the data source that is to be exposed as a set of REST-based endpoints (ie. select or create a data access layer). For relational data stored in Microsoft SQL Server or other 3rd Party databases, ADO.NET Data Services currently enables easily exposing a conceptual model created using the ADO.NET Entity Framework (EF). For all other data sources (XML section, web service, application logic layer, etc) or to use additional database access technologies (ex. LINQ to SQL), a mechanism is provided which enables any data source, as per the plug-in model described above, to be exposed as an ADO.NET Data Service.

To create a data service which exposes a relational database through an Entity Framework conceptual model see "Creating a Data Service using the ADO.NET Entity Framework". To create a data service which exposes another data source see "Creating a Data Service from any Data Source".

2.8.3. Creating a Data Service using the ADO.NET Entity Framework

ADO.NET Data Services are a specialized form of Windows Communication Foundation services, and thus can be hosted in various environments. The below example will create an ADO.NET Data Service which is hosted inside an ASP.NET site. In order to create a data service, you must first create a web project; you will then need to establish a connection with the database that will be exposed by the service, and then create the data service itself within the web application. Below is a step-by-step description of this process.

The following steps can be used for creating a Data Service using the Virtuoso ADO.Net Provider for accessing the sample Northwind Demo database:

1. Launch the Visual Studio 2008 SP1 IDE.

   Figure 2.117. Visual Studio 2008 SP1 IDE.

2. Create a

   Web Application

   project by going to the

   File

   menu in Visual Studio and choosing

   New Project

   .
3. When the New Project window appears, choose either

   Visual Basic

   or

   Visual C#

   as the programming language.
4. Within the language category click on

   Web

   , and select

   ASP.NET Web Application

   from the right-hand panel.
5. Choose a name for the project, for example

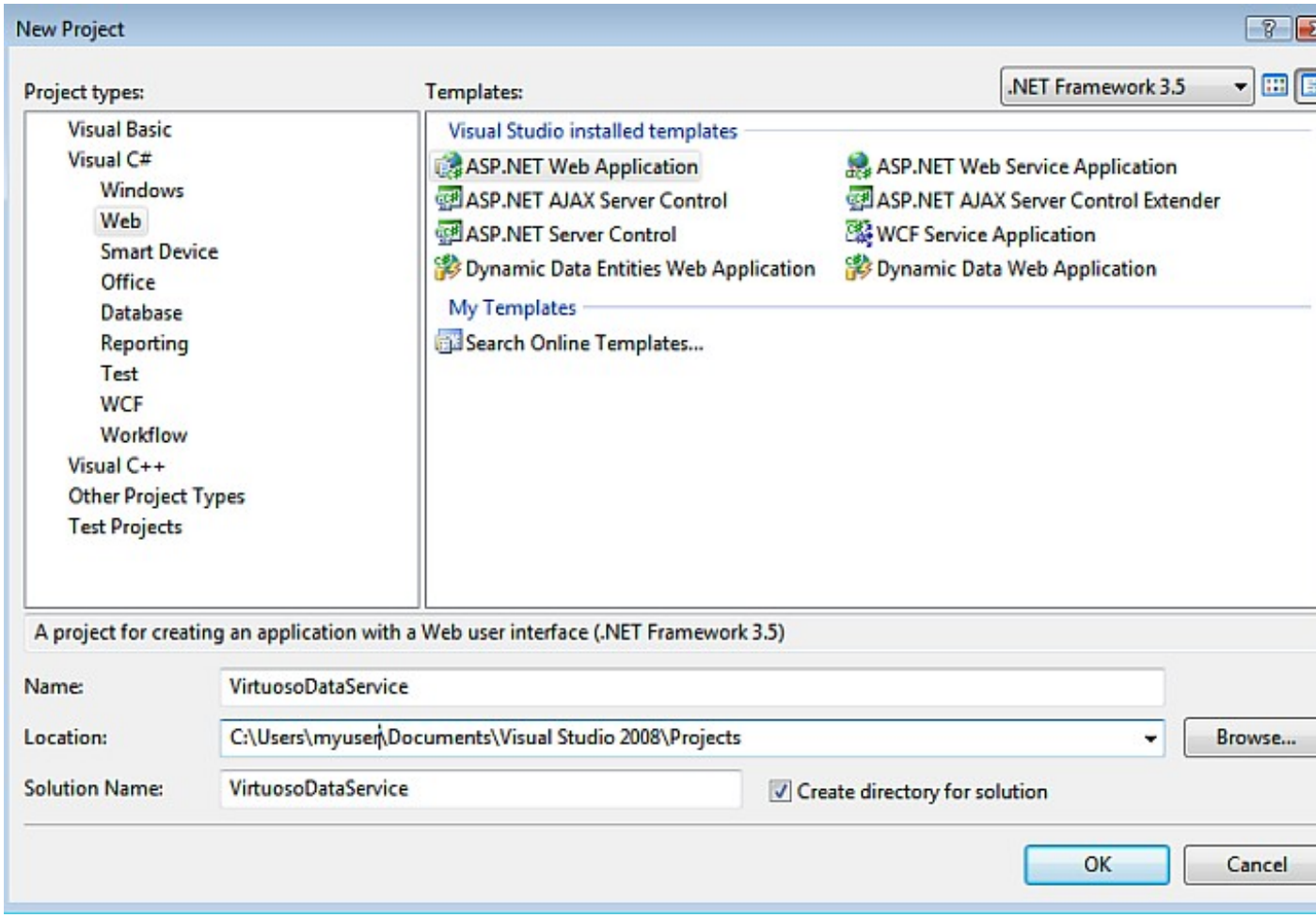
   VirtuosoDataService

   , and click

   OK

   .

   Figure 2.118. Name project

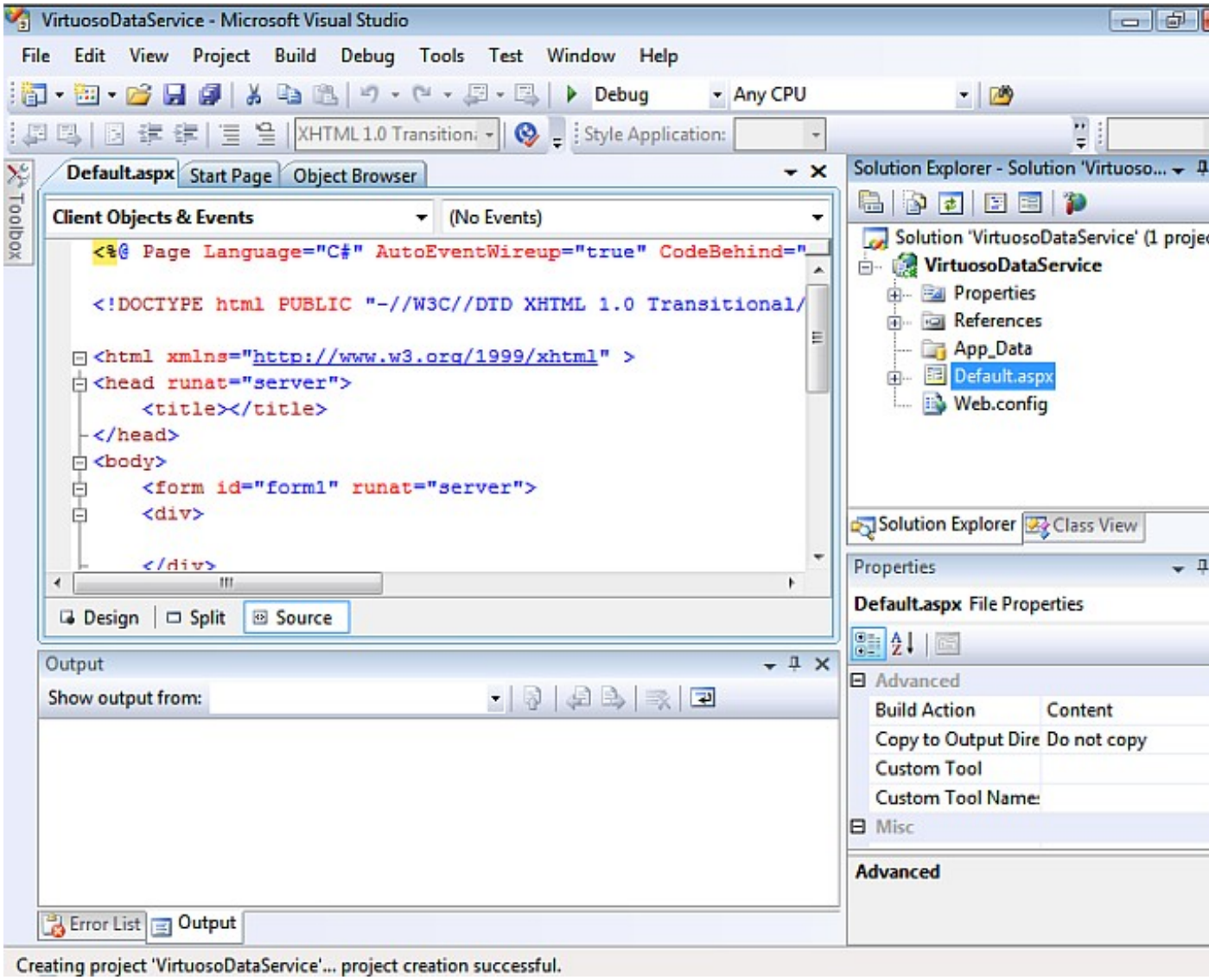


6. This will create a new project called

   VirtuosoDataService

   .

Figure 2.119. Create project



7. Right click on the

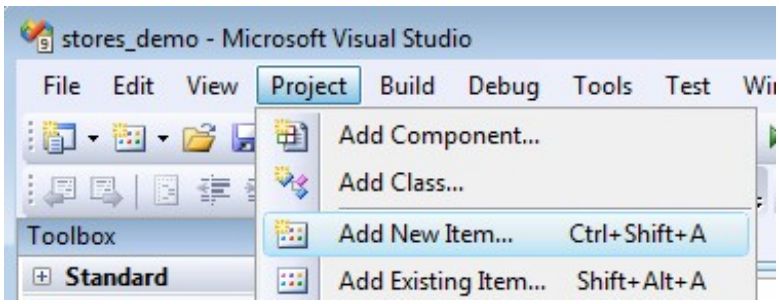
   VirtuosoDataService

   project name of the Solution Explorer pane, then select the

   Add -> New Item

   menu options.

   Figure 2.120. New Item



8. The

   Add

New Item dialog will appear, choose the

ADO.NET Entity Data Model

template, give it the name

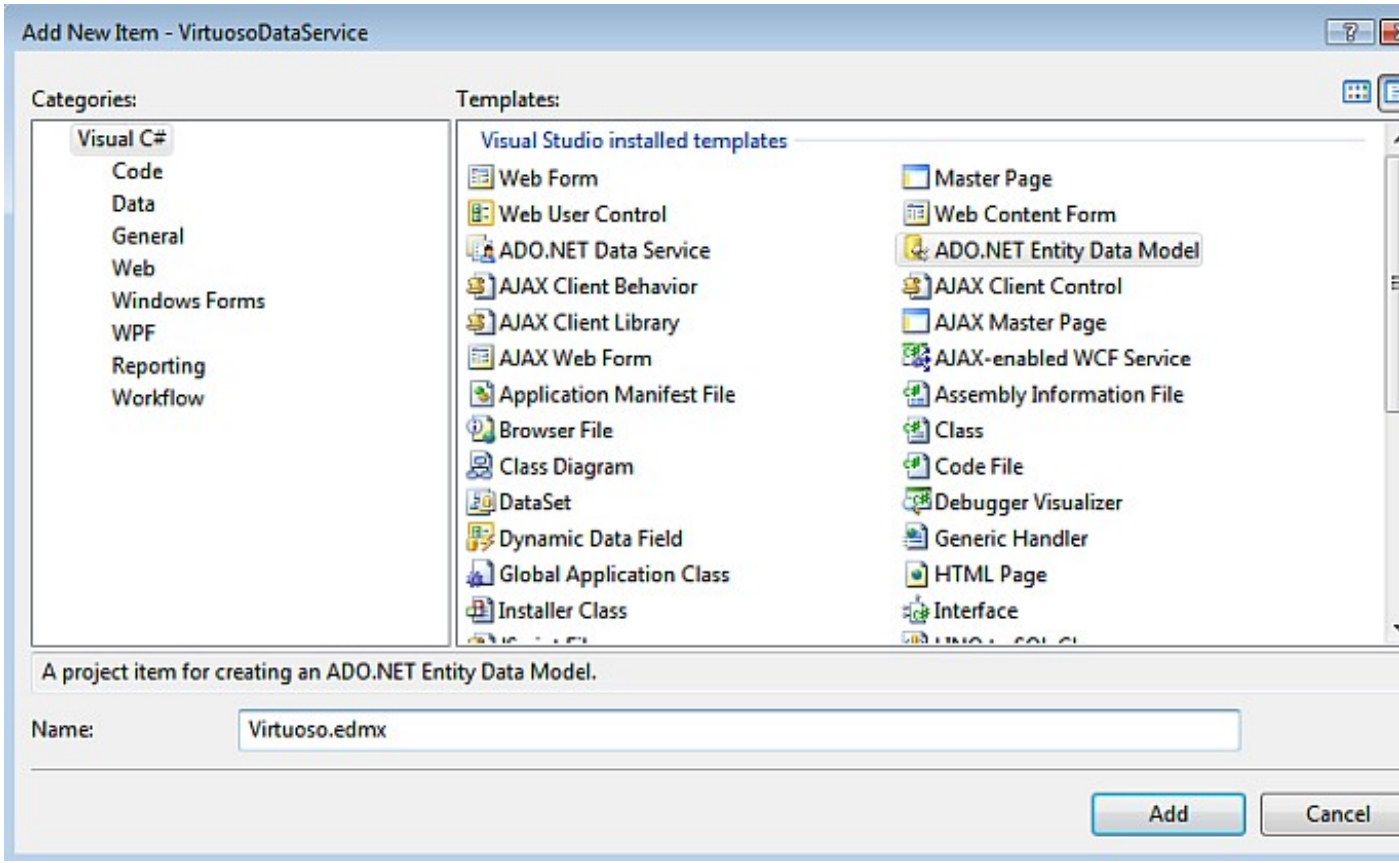
Virtuoso.edmx

and click

Add

to start the creation of the ADO.Net Entity Data Model.

Figure 2.121. Entity Model



9. In the

Entity Data Model Wizard

dialog

Choose Model Contents

page select the

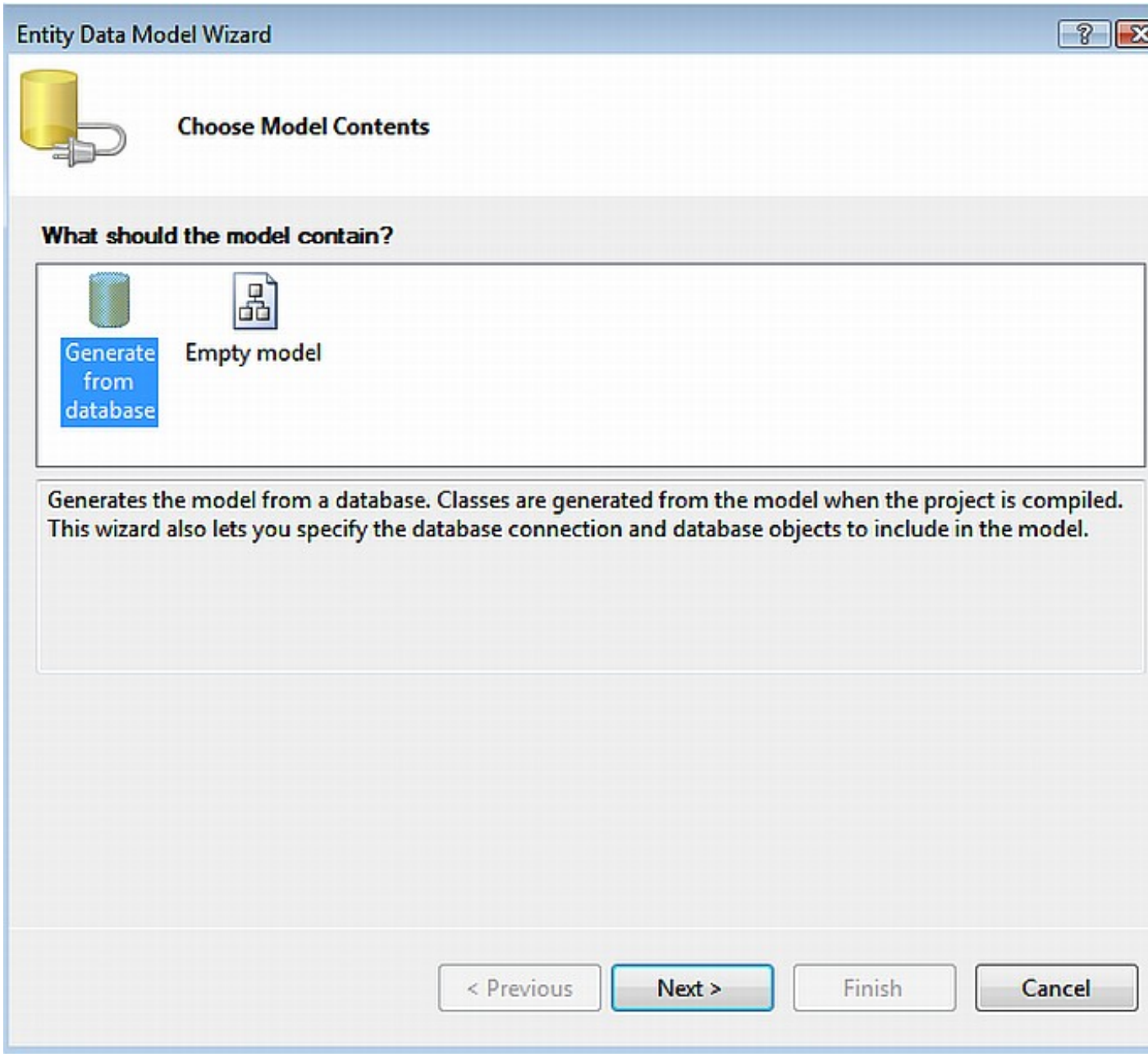
Generate from Database

model type and click

Next

.

Figure 2.122. Model Contents



10. In the

    Entity Data Model Wizard

    dialog

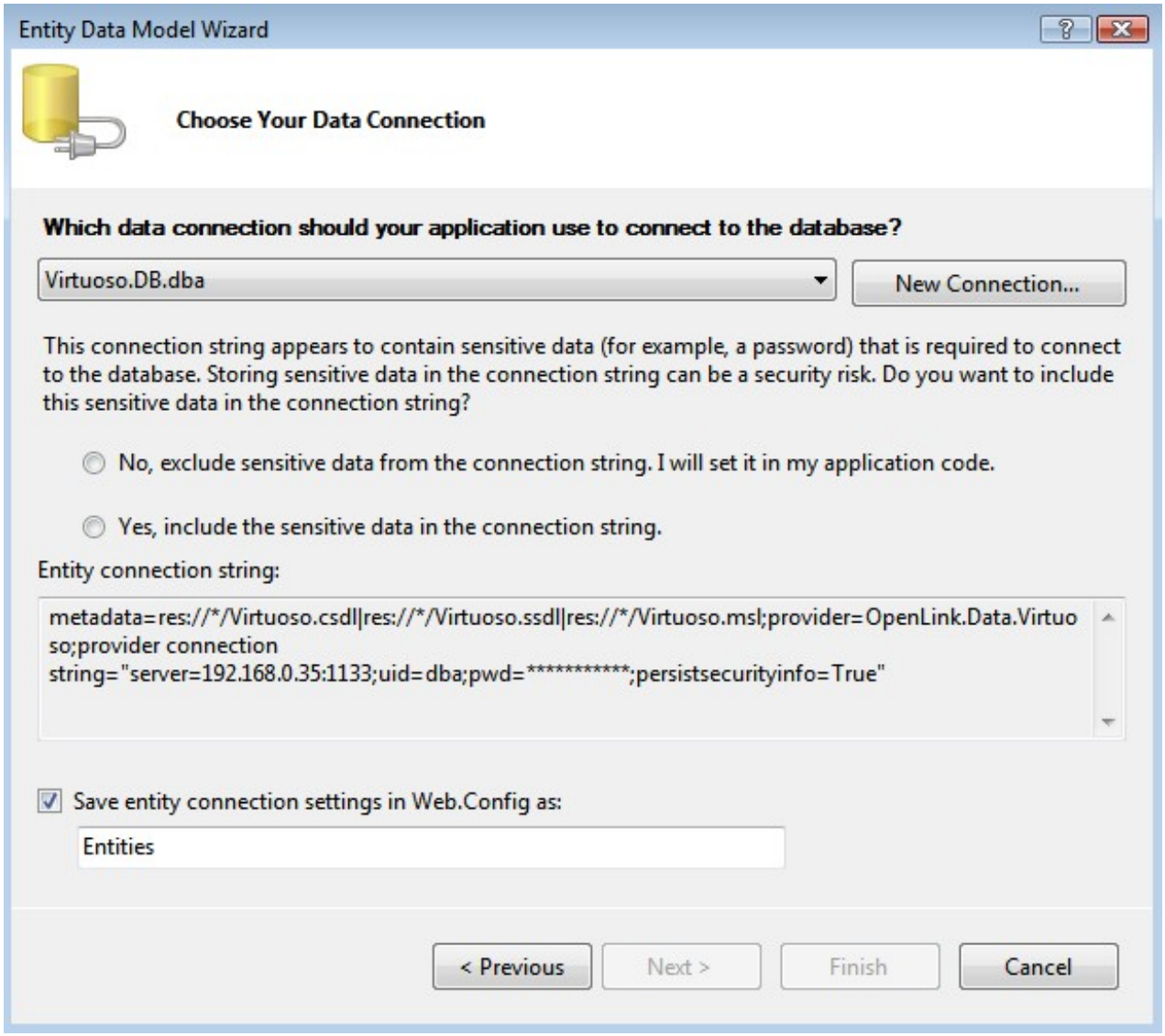
    Choose your Data Connection

    page select the

    New Connection

    button

Figure 2.123. Data Connection



11. In the

    Choose Data Source

    dialog, select the OpenLink

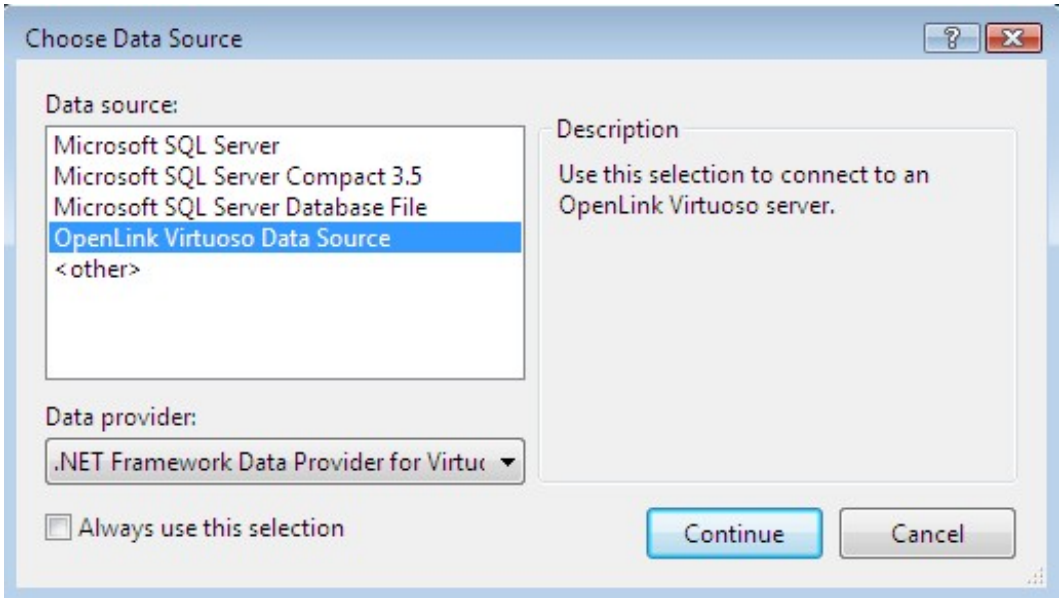
    Virtuoso Data Source

    from the list displayed and click

    Continue

    .

Figure 2.124. Data Source



12. In the

    Add Connection

    dialog, specify the

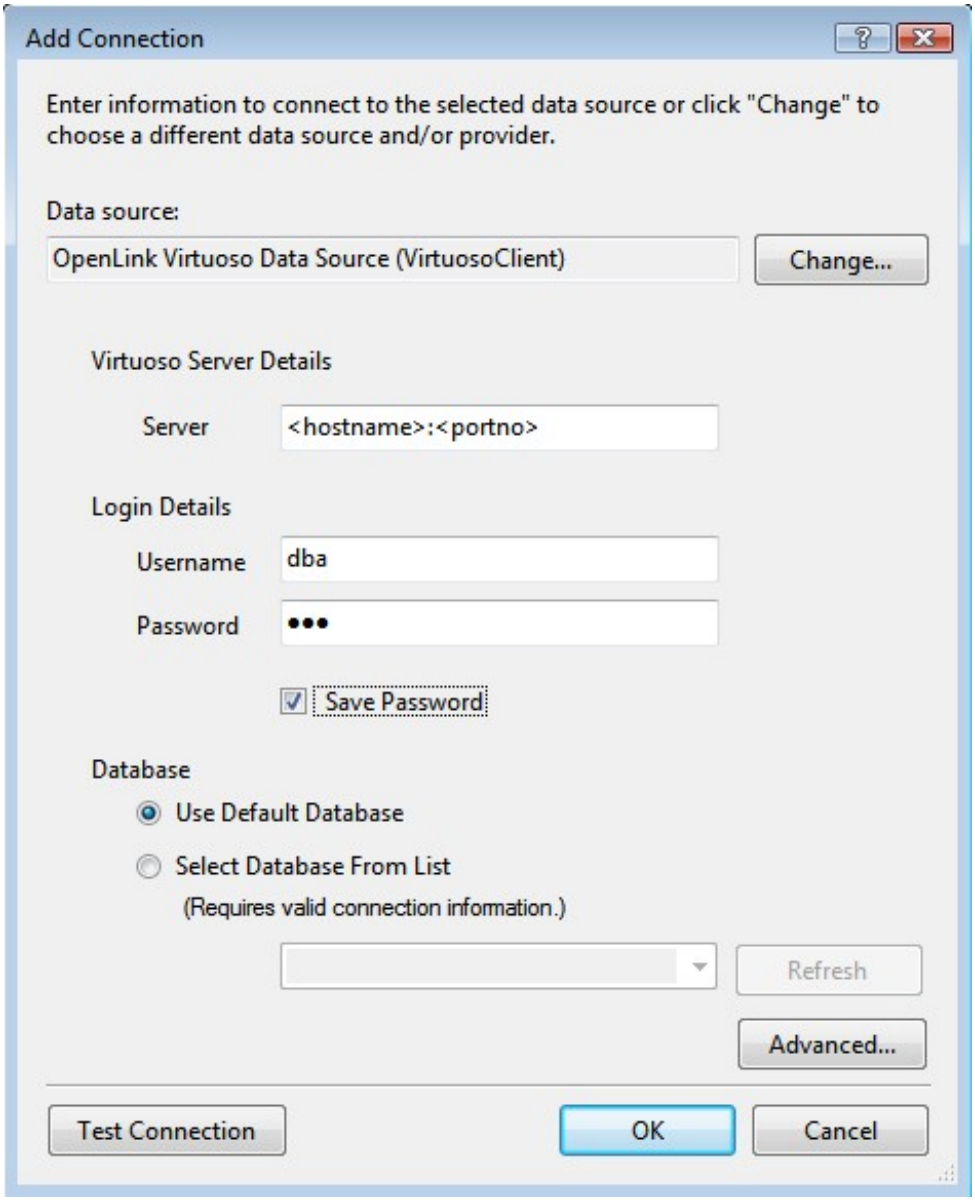
    hostname, portno, username and password

    for the target Virtuoso Server and check the

    Save Password

    check box.

    Figure 2.125. Connection Properties



13. Select the

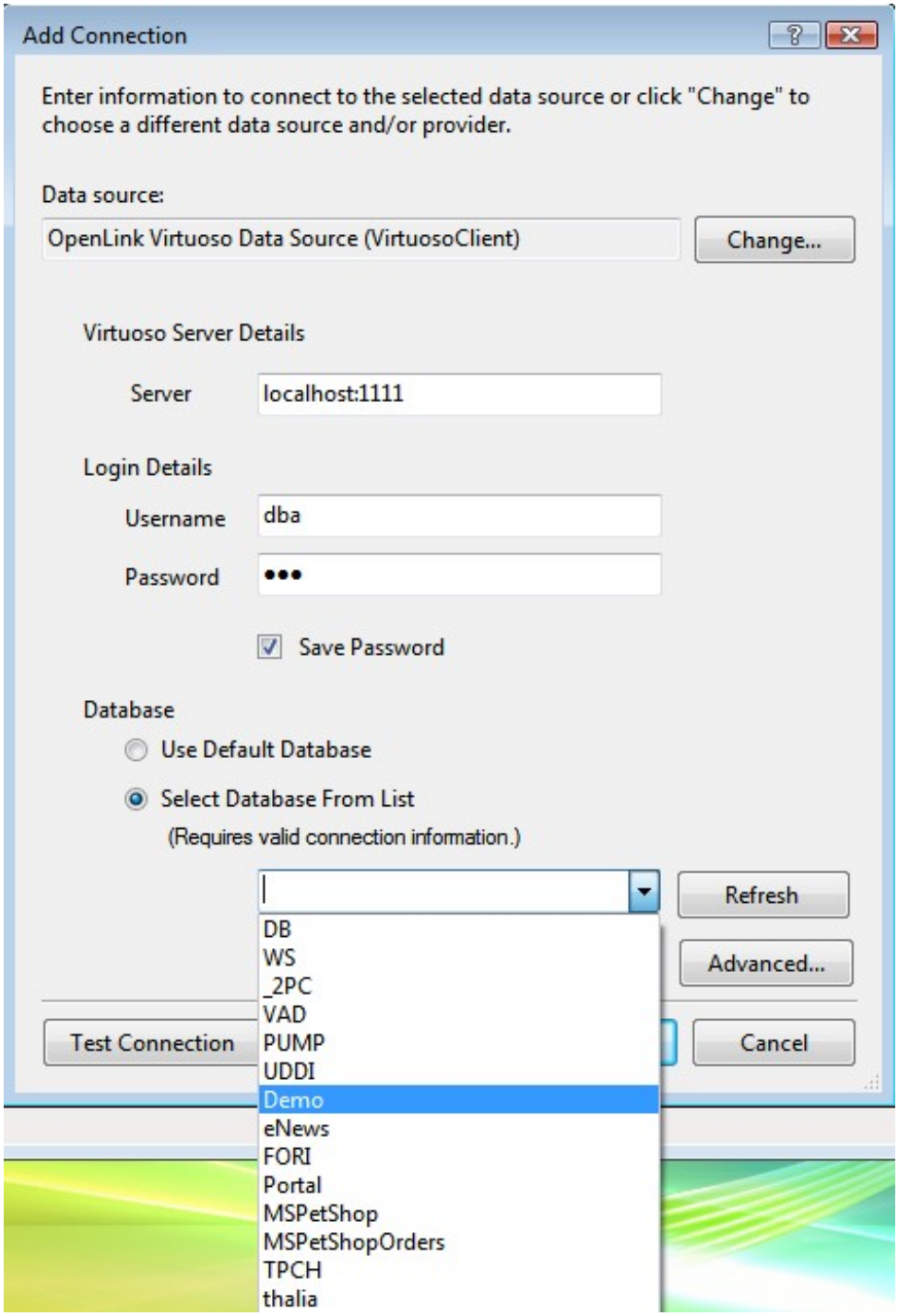
    Select Database From List

    radio button and choose

    Demo

    from the drop down list, assuming the Virtuoso Demo Database is installed.

Figure 2.126. Advanced Properties

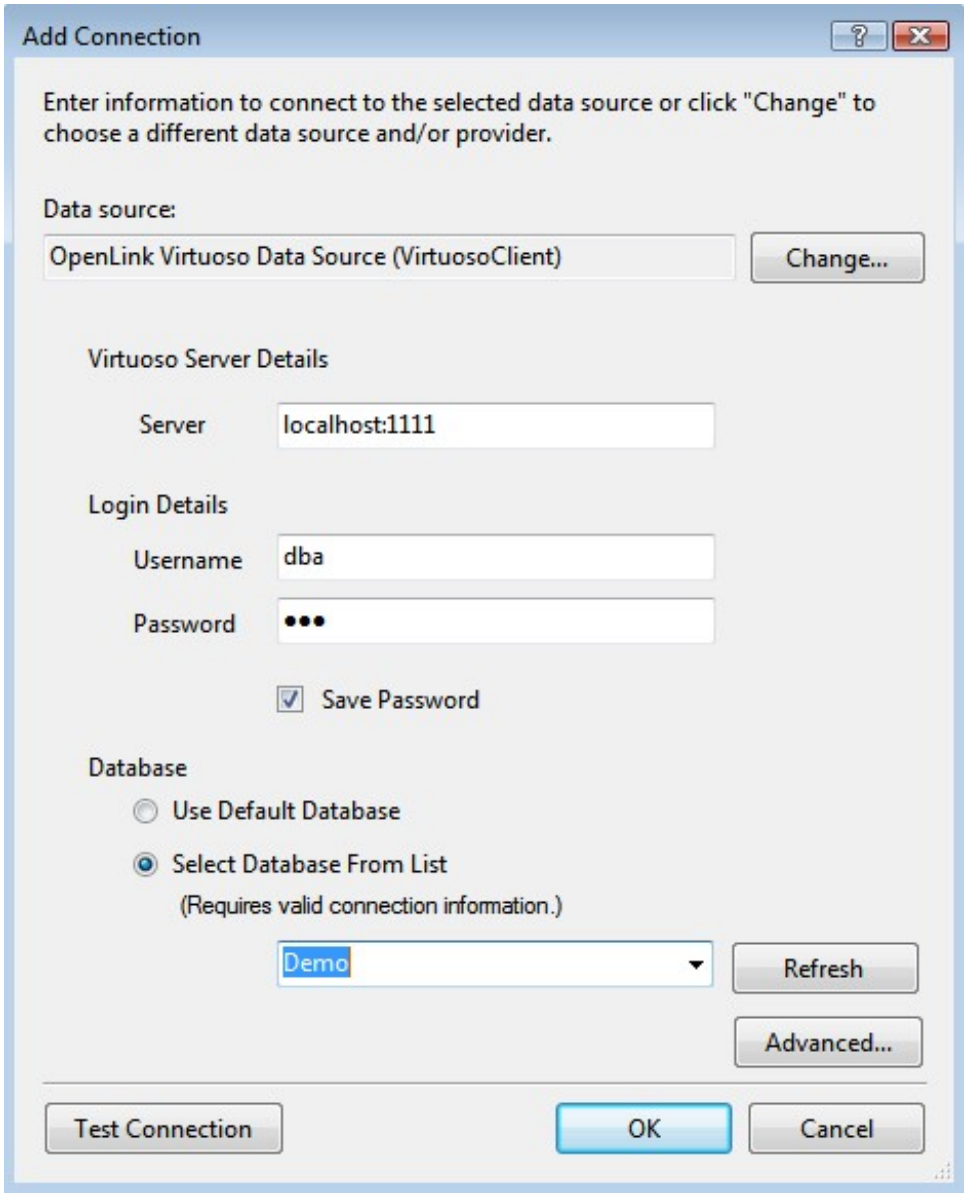


14. Click the

    Test Connection

    button to verify the connection is successful and then click OK to add the connection.

    Figure 2.127. Test Connection



15. Set the

    entity connect string

    name to

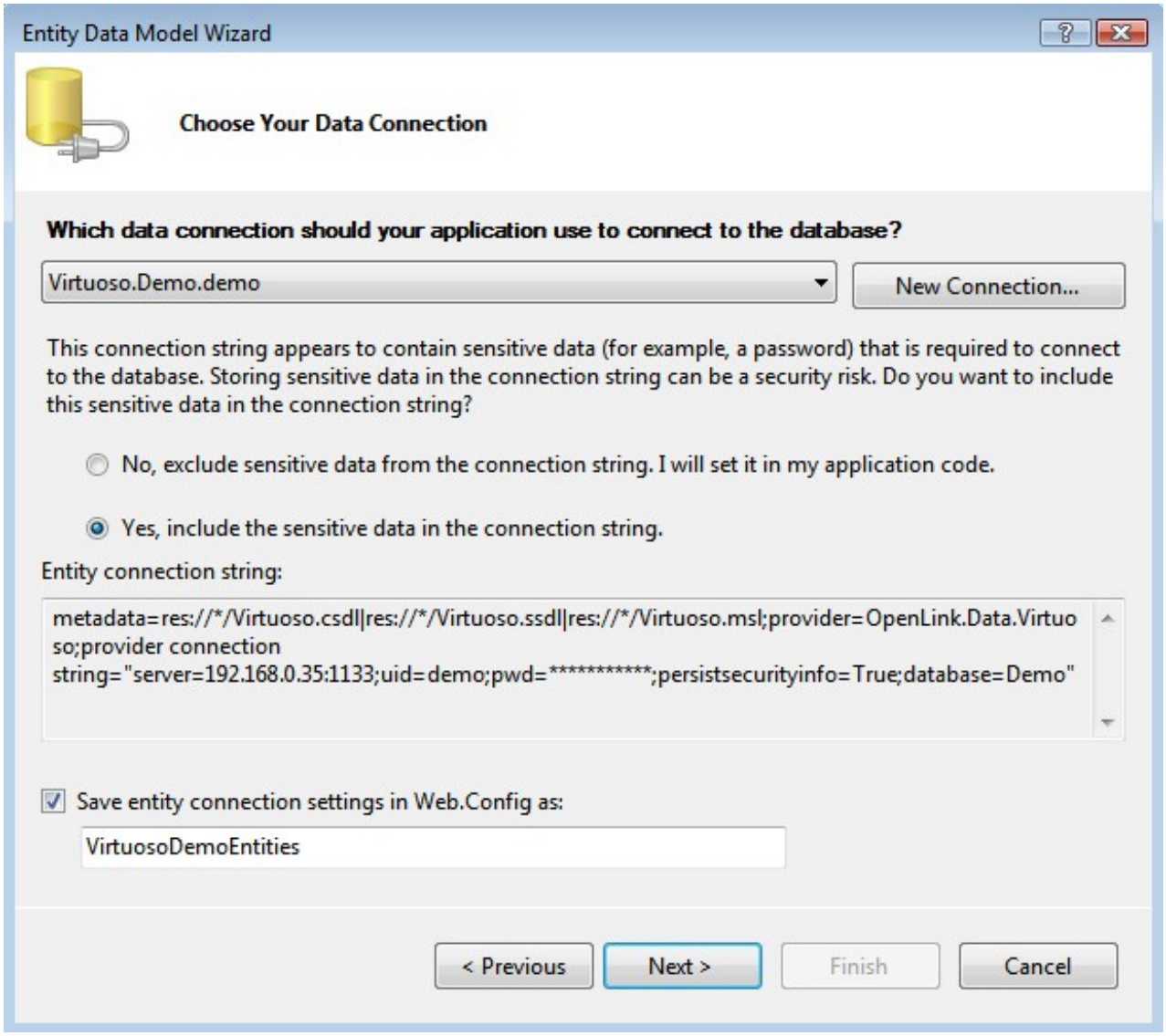
    VirtuosoDemoEntities

    (note this name as it is required in step 17 below) and click

    Next

    .

    Figure 2.128. entity connect string



16. In the

    Choose your Database Objects

    page select the

    Tables

    check box to select all tables in the Demo database for addition to the Entity Data Model, set the

    Model Namespace

    to

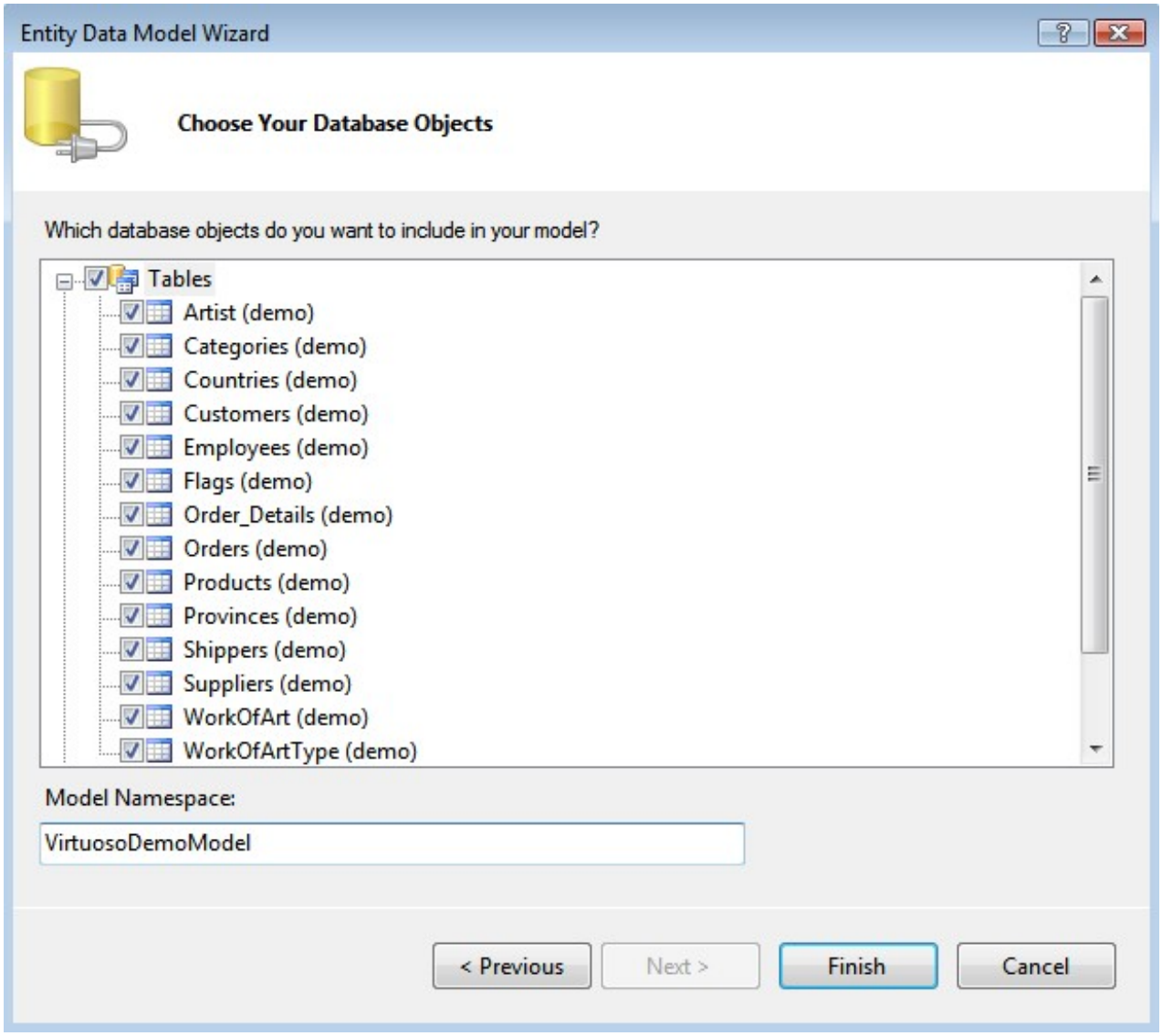
    VirtuosoDemoModel

    and click

    Finish

    .

Figure 2.129. Database Objects

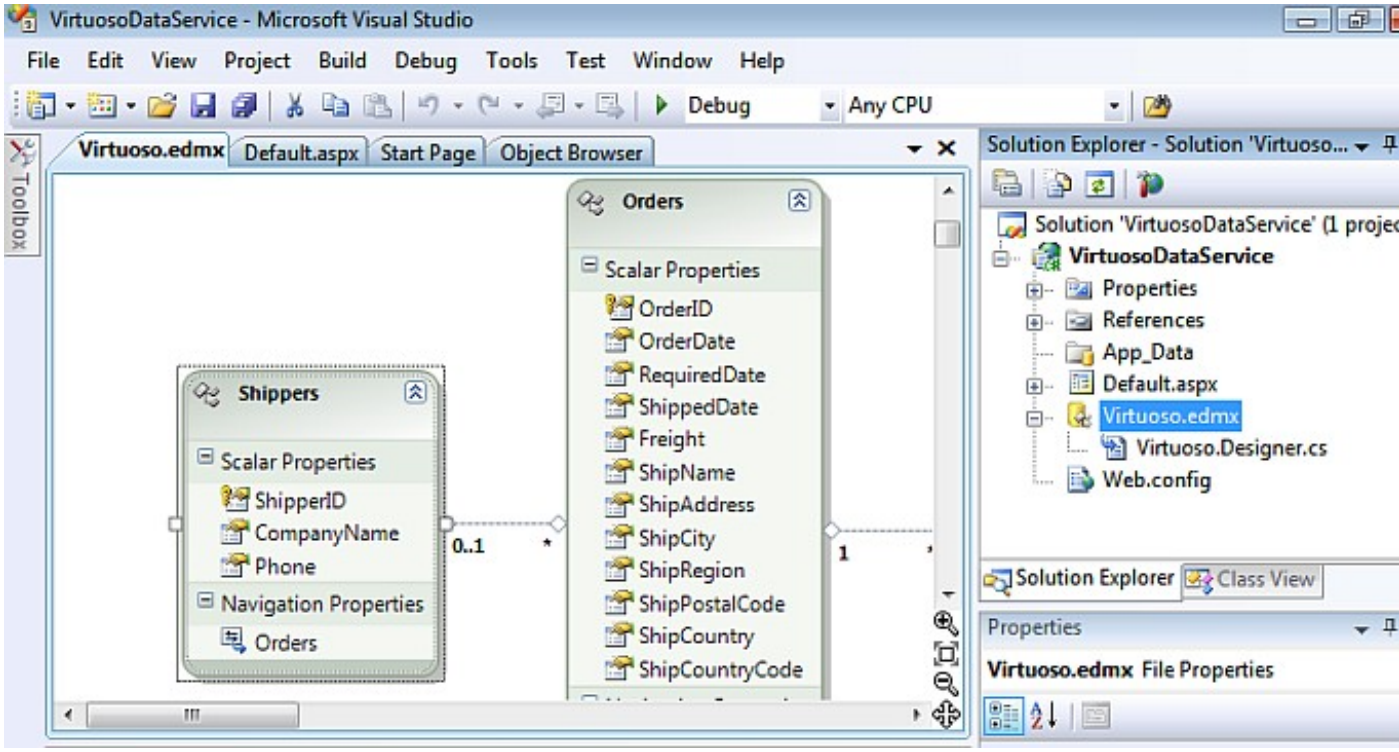


17. The

    Virtuoso.edmx

    EDM will be created with the tables and relationships displayed in the Visual Studio IDE

    Figure 2.130. Virtuoso.edmx



18. Right click on the

    VirtuosoDataService

    project name of the

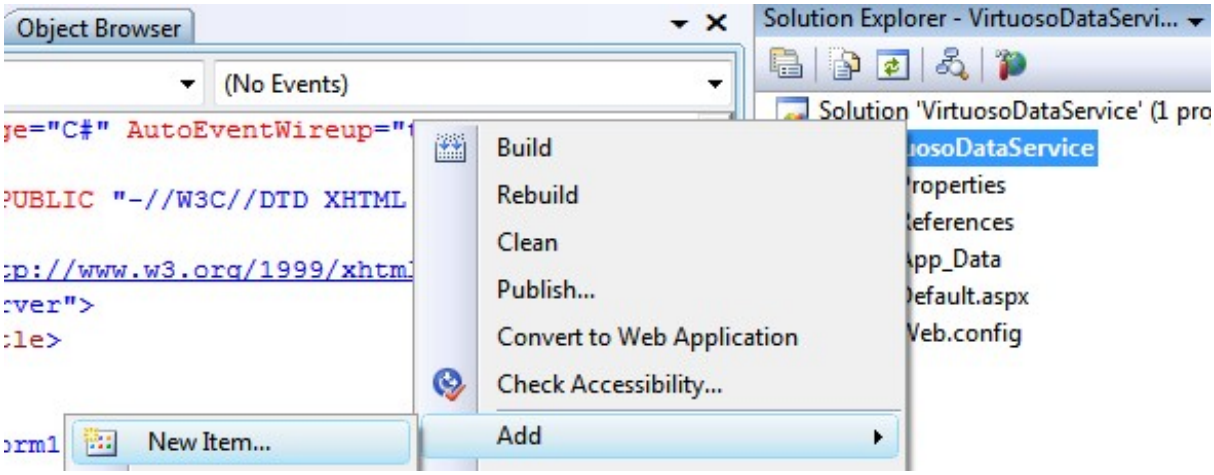
    Solution Explorer pane

    , then select the

    Add -> New Item

    menu options.

Figure 2.131. New Item



19. The

    Add New Item

    dialog will appear, choose the

ADO.NET Data Service

template, give it the name

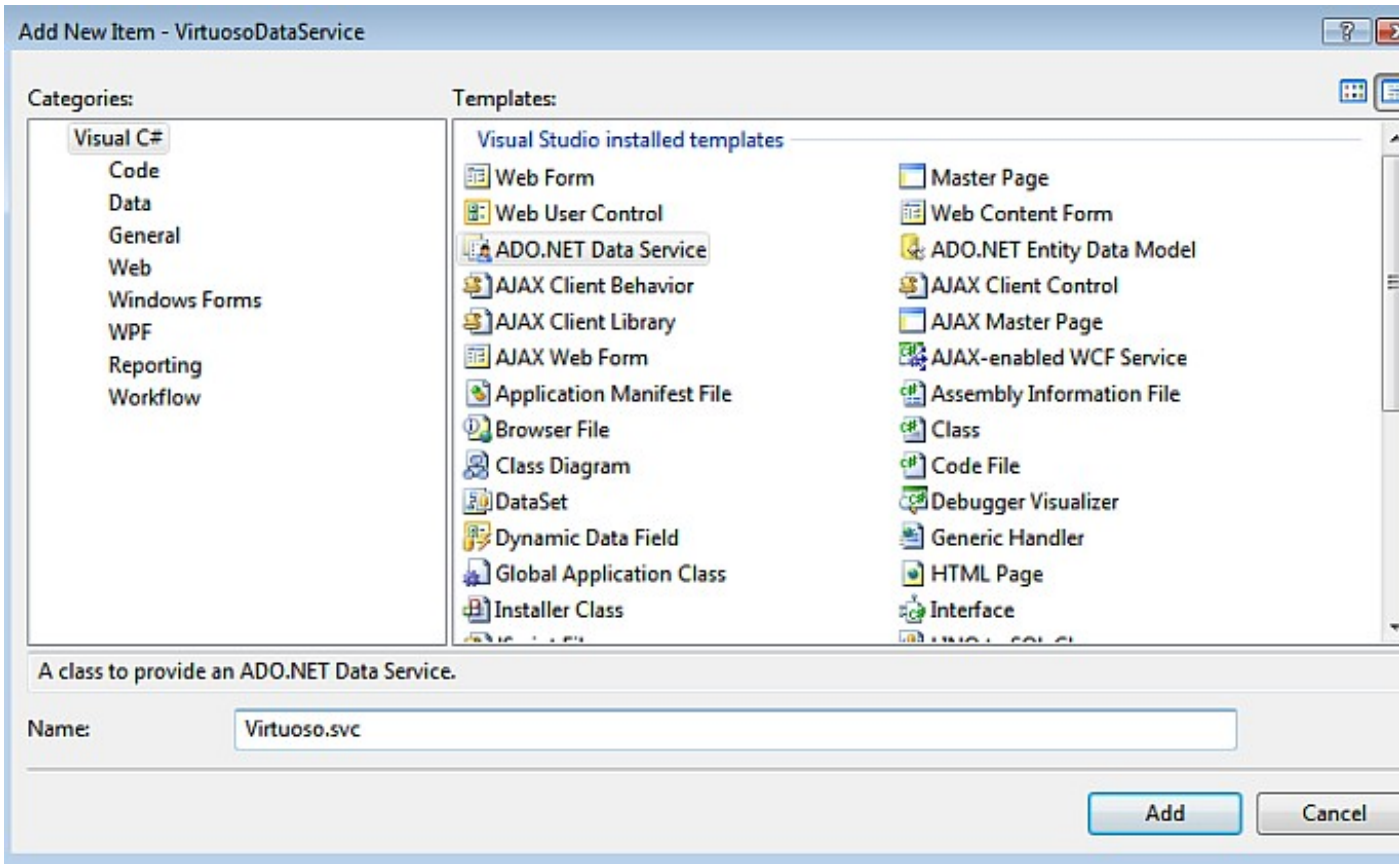
Virtuoso.svc

and click

Add

to create the ADO.Net Data Service.

Figure 2.132. ADO.NET Data Service



20. In the

Virtuoso.svc.cs

Data Service file created add the data source class name of

VirtuosoDemoEntities

(note this is the name set in step 12) as the

DataService

name and enable the access to the Data Service by adding the entry

config.SetEntitySetAccessRule("", EntitySetRights.All);*

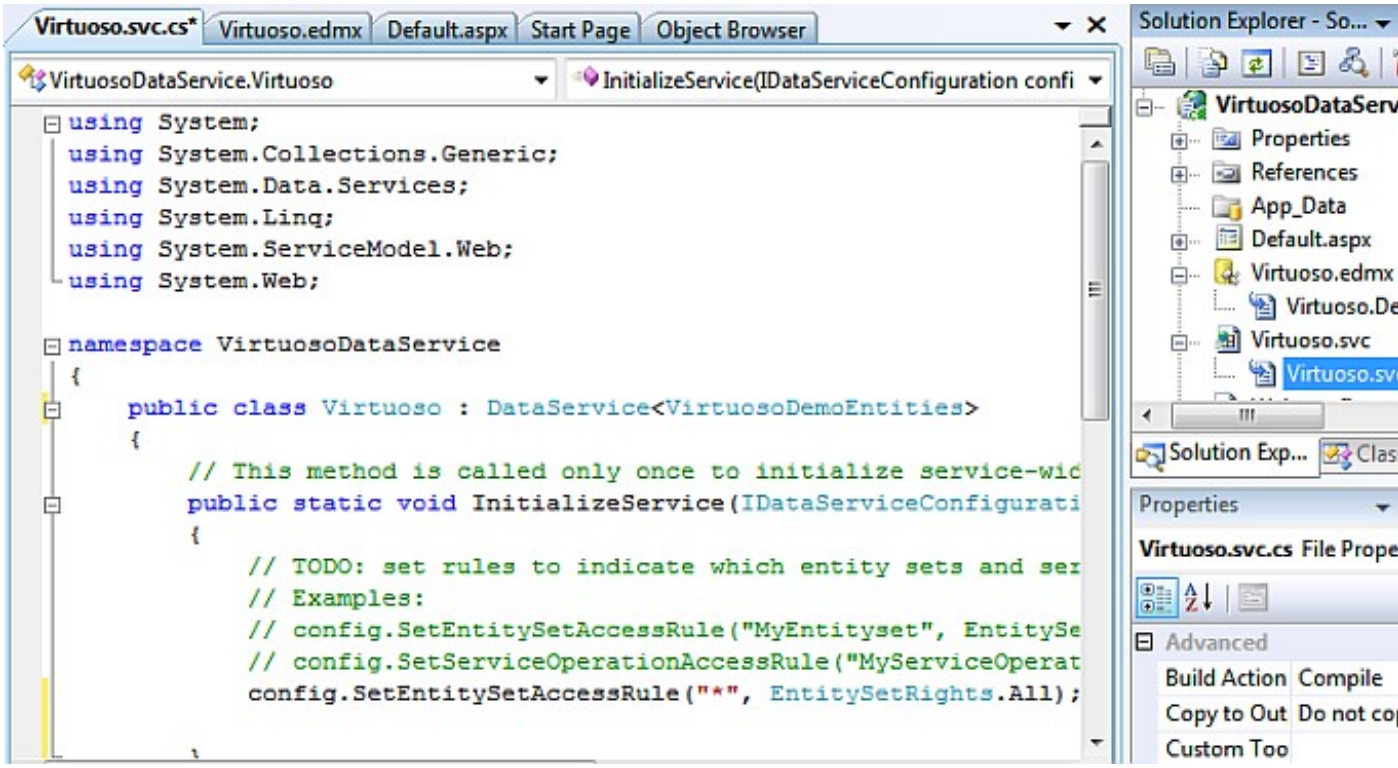
in the

InitializeService

method.

```
// C#

using System;
using System.Web;
using System.Collections.Generic;
using System.ServiceModel.Web;
using System.Linq;
using System.Data.Services;

namespace SimpleDataService
{
    public class Northwind : DataService<VirtuosoDemoEntities>
    {
        public static void InitializeService(IDataServiceConfiguration  config)
        {
            config.SetEntitySetAccessRule("*", EntitySetRights.All);
        }
    }
}
```

Figure 2.133. Data Service

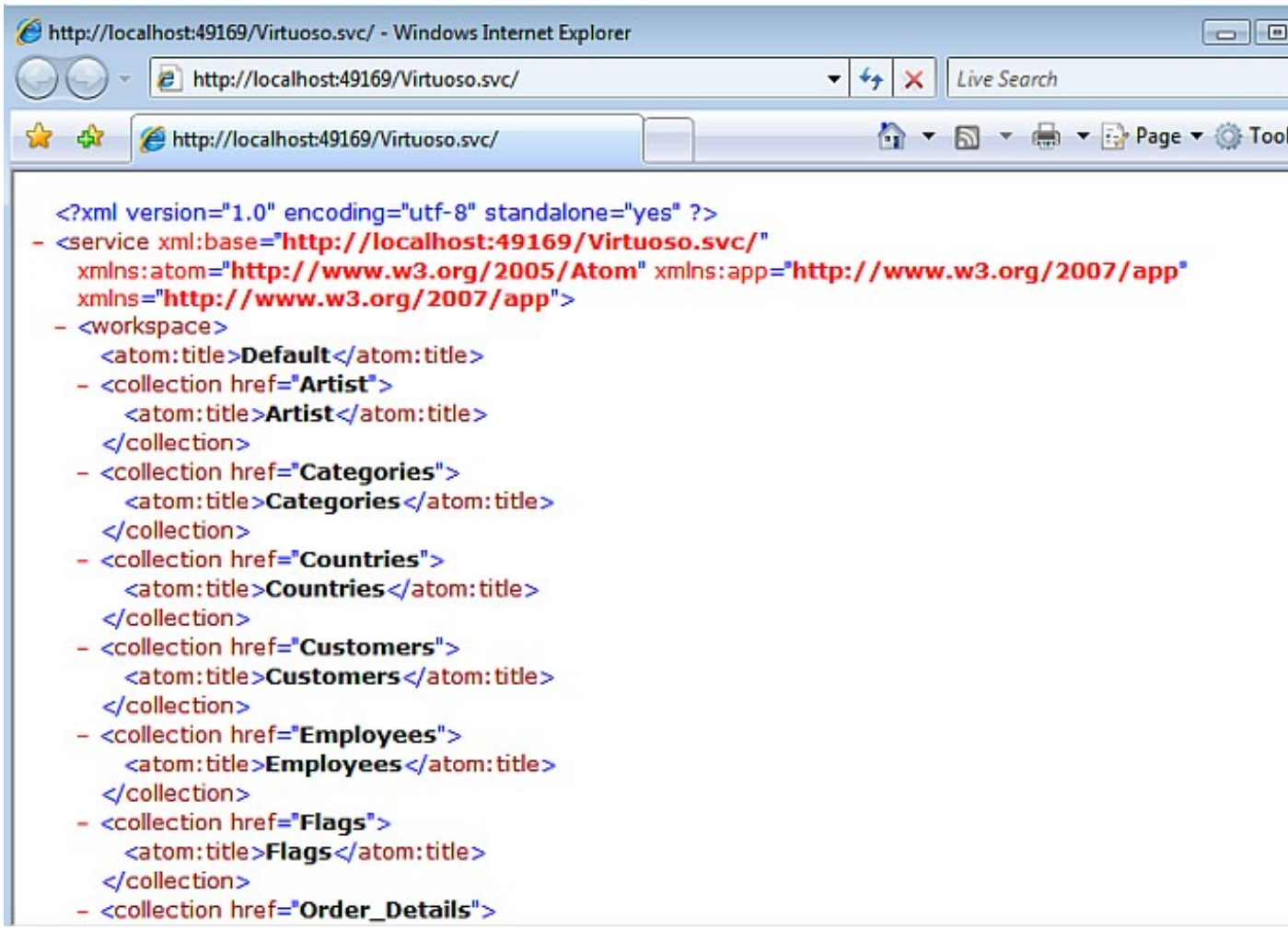


21. To test the Data Service, simply hit

    Ctrl+F5

    within Visual Studio, which will start the development web server, run the Data Services server inside and load a Web browser page displaying the list of available tables/entities of the Demo database.

Figure 2.134. test the Data Service



22. To access a specific entity instance like the

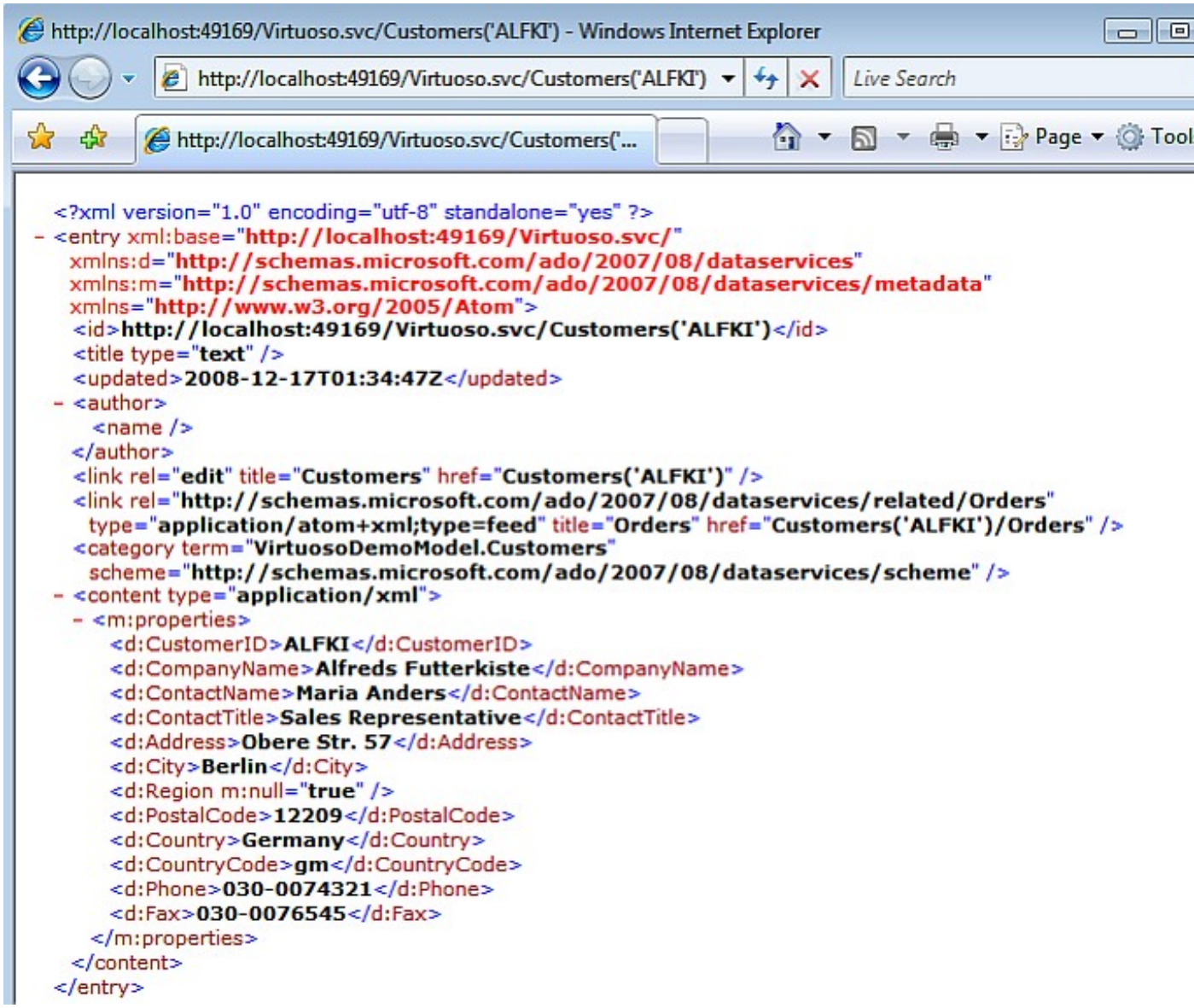
    Customers

    table

    ALFKI

    record, this would be specified as http://host/vdir/Virtuoso.svc/Customers('ALFKI') .

    Figure 2.135. Access a specific entity instance



NOTES

1. *Important*

   - To view

   Atom

   (the default format returned by an ADO.NET Data Service) in Internet Explorer, you must first ensure that

   Feed Reading View

   is turned

   off

   . This can be done on the

   Content tab

   of

Tools in Internet Options

.

2. If a Data Services entity instance URI page fails to load you can turn

   Verbose

   errors on by adding

   config.UseVerboseErrors = true;

   in the

   virtuoso.svc.cs InitializeService

   method to obtain more detailed information from the server as to why the page failed to load:

```
public static void InitializeService(IDataServiceConfiguration config)
{
config.UseVerboseErrors = true;
config.SetEntitySetAccessRule("*", EntitySetRights.All);
}
```

2.9. Windows Form Application for accessing Virtuoso RDF data via SPASQL using the Virtuoso ADO.Net Provider

This section will guide you through creating a simple application that allows you to access RDF data in a Virtuoso database as an Entity DataSet and explore that RDF data in an intuitive way by clicking on dereferenceable IRIs .

2.9.1. Pre-requisites

1. Microsoft Visual Studio 2008
2. The Virtuoso ADO.Net provider for .Net 3.5 and the Entity Framework.
3. The example assumes that you have a local Virtuoso server with the Northwind demo database installed. If the demo database is not already installed then download the demo database VAD package (demo_dav.vad) and install it. The VAD package will create a new database in Virtuoso called demo containing the familiar Northwind tables. It will also creates Linked Data Views of the Northwind tables. In the example we assume the database is accessible on a hostname of "demo.openlinksw.com" on the default port 80, where an actually live instance of the Virtuoso Demo database is hosted. Users would use the appropriate hostname and port number of their Virtuoso installation to create the sample application, and would be would be example.com for a default installation or whatever the URIQA DefaultHost Virtuoso configuration parameter is set to when the demo database VAD package is installed.

2.9.2. Creating the Application

Step 1 - Create a view of the RDF data.

We want to be able to access the RDF data in Visual Studio and the easiest way to do this is to create a view of the data that we are interested in and bind that view to a DataSet. This can be considered as using server side SPARQL . Virtuoso supports an extension to standard SQL that allows execution of SPARQL. If a SQL query begins with the keyword SPARQL then the rest of the query is interpreted by as SPARQL. If a SPARQL query is used as the definition of a view then that view can be manipulated using SQL like any other view. In this way the result set from a SPARQL query can be easily accessed from Visual Studio using ADO.Net and the Entity Framework.

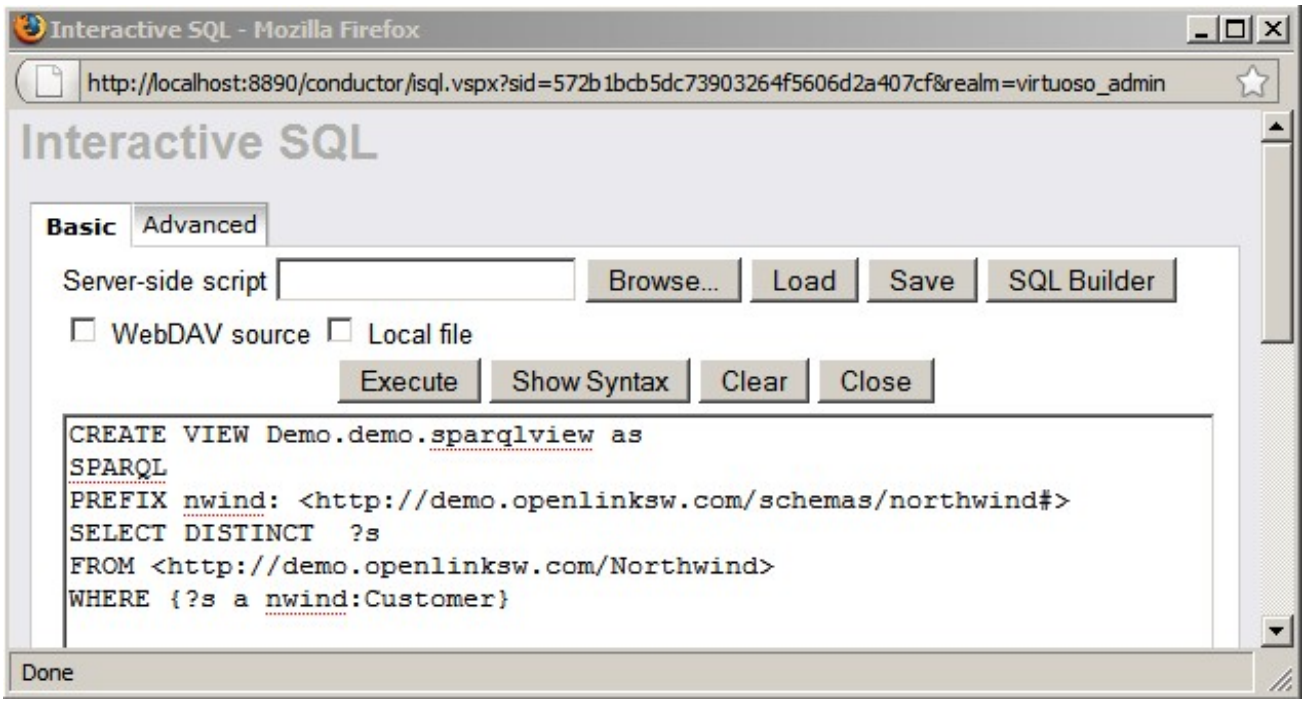
To create a view of the customers in the Northwind first open the Virtuoso Conductor and log in as dba. Then open iSQL from the menu on the left and execute the following statement.

```
CREATE VIEW Demo.demo.sparqlview as
```

```
SPARQL
PREFIX nwind: <http://demo.openlinksw.com/schemas/northwind#>
SELECT DISTINCT  ?s
FROM <http://demo.openlinksw.com/Northwind>
WHERE {?s a nwind:Customer}
```

Note: If the view is added to the Visual Studio project as user "demo" (or any other than "dba'), then it must be ensured that the "SPARQL_SELECT" and "SPARQL_SPONGE" roles are assigned to this user, which can be done via the Virtuoso Conductor in the "System Admin" -> "User Accounts" tab.

Figure 2.136. SPARQL_SPONGE



Step 2 - Create a simple grid form in Visual Studio

1. Open

   Visual Studio

   and create a new

   Windows Forms Application

   called RDFDemo.
2. In the

   Form Designer

   drag a

   DataGridView

   on to the form.
3. Click the

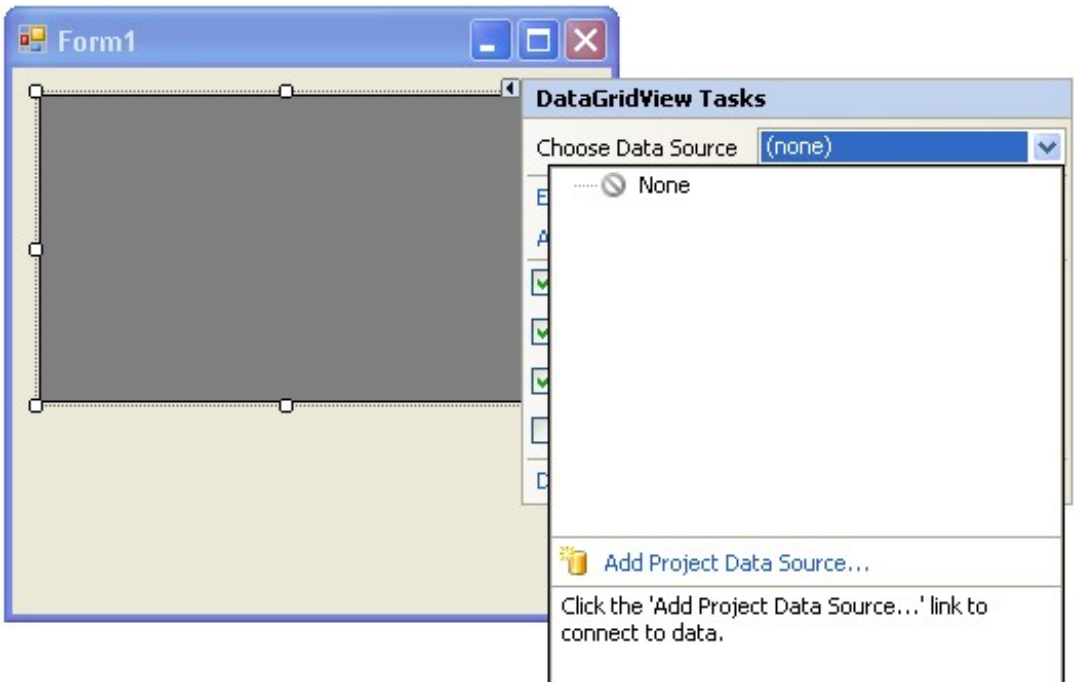
   Choose Data Source

   drop down and select

   Add Project Data Source

.

Figure 2.137. Data Source



4. In the

   Data Source Configuration Wizard

   choose Database and then set up a connection to the demo database on your local Virtuoso server.
5. On the

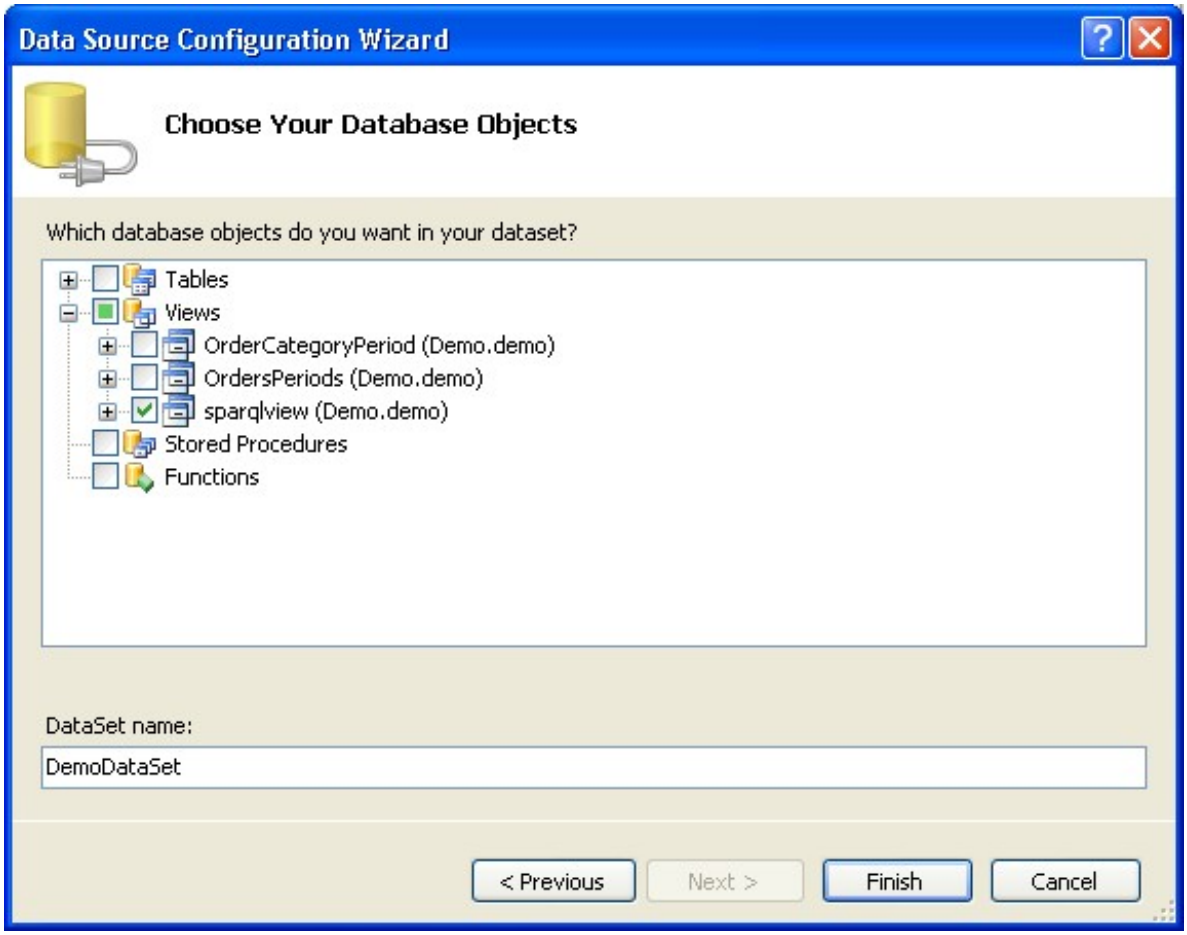
   Choose Your Data Objects

   page expand the

   Views

   and select sparqlview.

Figure 2.138. Data Source Configuration Wizard



6. Click

   Finish

   .
7. In the

   Form Designer

   select dataGridView1 and change the

   AutoSizeColumnsMode

   to AllCellsExceptHeader.
8. Select the

   DefaultCellStyle

   and click on the ellipsis. This will open the

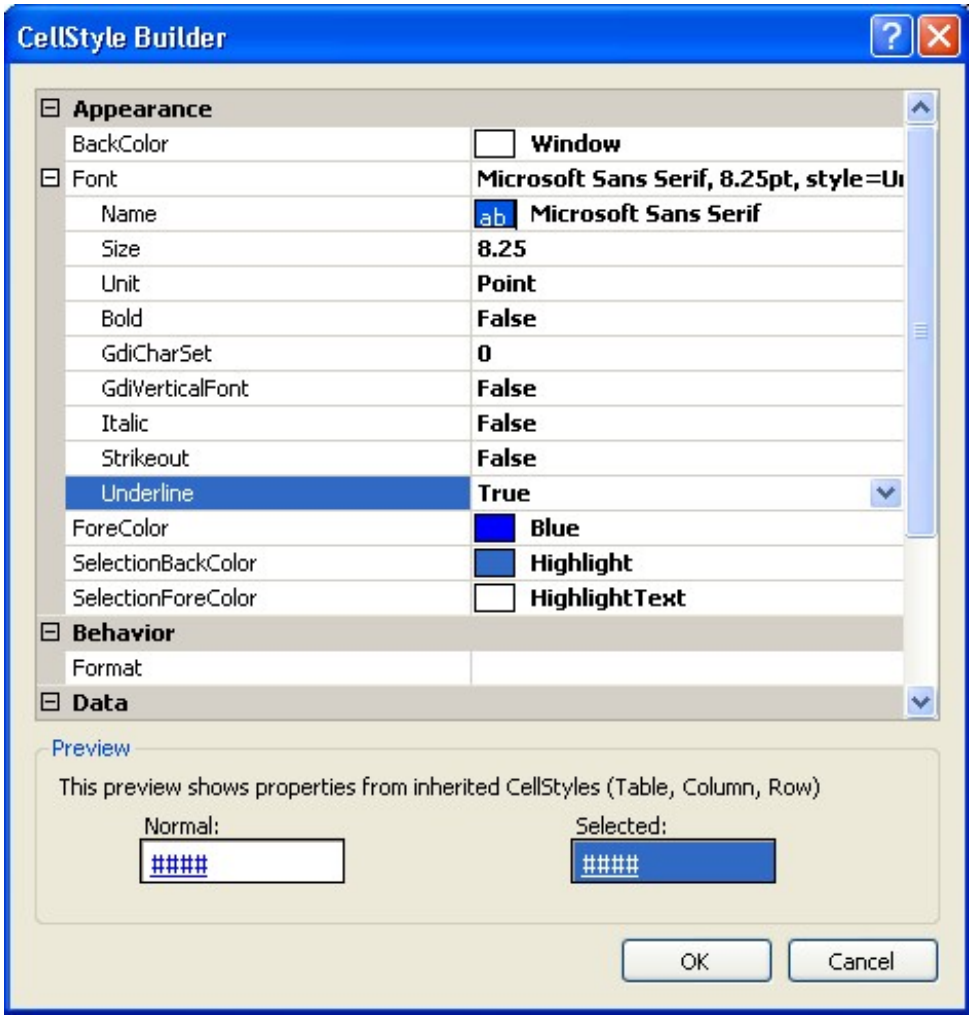
   CellStyleBuilder

   . Change the

   ForeColor

   to Blue.

   Figure 2.139. CellStyleBuilder



9. Expand

   Font

   and change

   Underline

   to True. Click

   OK

   .

Step 3 - Change the mapping of the DataSet.

In the Solution Explorer you will now have a DataSet called DemoDataSet.xsd. If you double click on this it opens the DataSet Designer. Select the column called s in the sparqlview table and in the Properties pane change the DataType from System.String to System.Object.

The data returned by a SPARQL query can either be an IRI or a literal value. In order to distinguish between the two the Virtuoso ADO.Net provider defines an additional data type, SQLExtendedString. By setting the column type to System.Object we can cast the fetched data back to SQLExtendedString and find out if an individual value is an IRI or a literal and handle it appropriately.

Step 4 - Create the on_click event handler for the cells in the DataGridView.

Return to the *Form Designer* and double click on the cell of the *DataGridView* . This creates the *dataGridView1_CellContentClick* method in Form1.cs. This is the method that handles clicking on IRI objects in the grid.

Paste in the following block of code into the body of the *dataGridView1_CellContentClick* method.

```
int column = e.ColumnIndex;
 object o = dataGridView1.Rows[e.RowIndex].Cells[e.ColumnIndex].Value;
 Type t = dataGridView1.Rows[e.RowIndex].Cells[e.ColumnIndex].ValueType;

 if (o is SqlExtendedString)
 {
     SqlExtendedString se = (SqlExtendedString) o;
     ExtendedStringHandler seHandler = new ExtendedStringHandler(se, this.sparqlviewTableAdapter.Connecti
     seHandler.displayData();
 }
 else if (o is SqlRdfBox)
 {
     //doesn't do anything at the moment
 }
```

As we are using the SQLExtendedString extension from the Virtuoso ADO.Net provider you will also need to add

```
using OpenLink.Data.Virtuoso;
```

at the top of the file.

Step 5 - Create a class to handle exploring the RDF data.

- Add a new C# class to the project called ExtendedStringHandler, by Right clicking on RDFDemo in the

  Solution Explorer

  and

  Add

  a

  Class

  .
- Add the following using statements to the top of the file

  ```
  using OpenLink.Data.Virtuoso;
  using System.Data;
  using System.Windows.Forms;
  using System.Drawing;
  using System.Data.Mapping;
  using System.Data.Common;
  ```
- Paste the following block of code into the class definition.

  ```
  StringBuilder DescribeCommand;
  VirtuosoConnection ParentConnection;
  List<Label> labelList = new List<Label>();
  List<TextBox> textBoxList = new List<TextBox>();
  DescribeDataSet describeDataSet = new DescribeDataSet();
  Boolean isIRI = false;

  public ExtendedStringHandler(SqlExtendedString iri, VirtuosoConnection parentConnection)
   {
       ParentConnection = parentConnection;
       if (iri.IriType == SqlExtendedStringType.IRI)
       {
           isIRI = true;
           DescribeCommand = new StringBuilder("sparql select * from <http://demo.openlinksw.com/Nor
       }
   }

   public string describeCommandText
   {
  ```

```
    get
    {
        return DescribeCommand.ToString();
    }
}

public void getDescribeData()
{
     VirtuosoCommand myCommand = new VirtuosoCommand(this.describeCommandText, this.ParentConnect
    VirtuosoDataAdapter myAdapter = new VirtuosoDataAdapter();
    myAdapter.SelectCommand = myCommand;
    myAdapter.Fill(describeDataSet.DataTable1);
}

public void displayData()
{
    if (isIRI)
    {
        getDescribeData();
        Form describeForm = new Form();
        describeForm.AutoScroll = true;
        describeForm.Width = 840;

        Label label1 = new Label();
        label1.AutoSize = true;
        label1.Font = new Font(label1.Font.FontFamily, label1.Font.Size + 3.0F, label1.Font.Style
        describeForm.Controls.Add(label1);

        DataTable table1 = describeDataSet.Tables[0];
        if (table1.Rows.Count == 0)
            label1.Text = "No Details Available";
        else
        {
            foreach (DataRow row in table1.Rows)
                if (row[0].ToString() == "http://www.w3.org/1999/02/22-rdf-syntax-ns#type")
                {
                    StringBuilder title = new StringBuilder(row[1].ToString() + " details");
                    label1.Text = title.ToString();
                    break;
                }

            foreach (DataRow row in table1.Rows)
            {
                Label propertyLabel = new Label();
                TextBox valueBox = new TextBox();
                valueBox.Width = 400;

                object property = row[0];
                object value = row[1];

                if (value is SqlExtendedString)
                {
                    valueBox.ForeColor = Color.Blue;
                    valueBox.Font = new Font(valueBox.Font.FontFamily, valueBox.Font.Size, valueB
                }
                propertyLabel.Text = row[0].ToString();
                propertyLabel.AutoEllipsis = true;
                propertyLabel.AutoSize = false;
                propertyLabel.Width = propertyLabel.PreferredWidth > 380 ? 380 : propertyLabel.Pr

                Binding bind = new Binding("Text", row[1], "");
                valueBox.DataBindings.Add(bind);

                labelList.Add(propertyLabel);
                textBoxList.Add(valueBox);
            }

            for (int i = 0; i < table1.Rows.Count; i++)
            {
                textBoxList[i].Click += new EventHandler(this.iri_Click);
                labelList[i].Location = new Point(10, i * 20 + 50);
                textBoxList[i].Location = new Point(400, i * 20 + 50);
                describeForm.Controls.Add(labelList[i]);
```

```
                    describeForm.Controls.Add(textBoxList[i]);
                }

                describeForm.Height = labelList.Count * 20 + 100 > 500 ? 500 : labelList.Count * 20 +
            }
            describeForm.ShowDialog();
        }
        else
        {
            Form blankForm = new Form();
            Label label1 = new Label();
            label1.Text = "Blank Node";
            label1.Font = new Font(label1.Font.FontFamily, label1.Font.Size + 3.0F, label1.Font.Style
            blankForm.ShowDialog();
        }
    }

    public void iri_Click(object sender, EventArgs e)
    {
        int boxNum = 0;

        for (int i = 0; i < textBoxList.Count; i++)
        {
            if (sender == textBoxList[i])
            {
                boxNum = i;
                break;
            }
        }

       Object o = describeDataSet.DataTable1.Rows[boxNum][1];
        if (o is SqlExtendedString)
        {
            SqlExtendedString se = (SqlExtendedString)o;
            ExtendedStringHandler seHandler = new ExtendedStringHandler(se, ParentConnection);
            seHandler.displayData();
        }
        else if (o is SqlRdfBox)
        {
           //doesn't do anything at the moment
       }
    }
```

The ExtendedStringHandler class creates a new SPARQL query based on the IRI that was clicked. This query is executed against Virtuoso using the ADO.Net connection in the same way that any SQL statement would be executed across an ADO.Net connection. This can be considered as Client Side SPARQL. The result set from the query describes the selected object and is returned as an ADO.Net DataAdapter. The DataAdapter is used to fill a DataTable which is displayed on a new form. We now need to add the new DataSet to the project and define the DataTable that will hold the query results.

Step 6 - Add a new DataSet to hold the query results.

1. Right click RDFDemo in the

   Solution Explorer

   and add a new

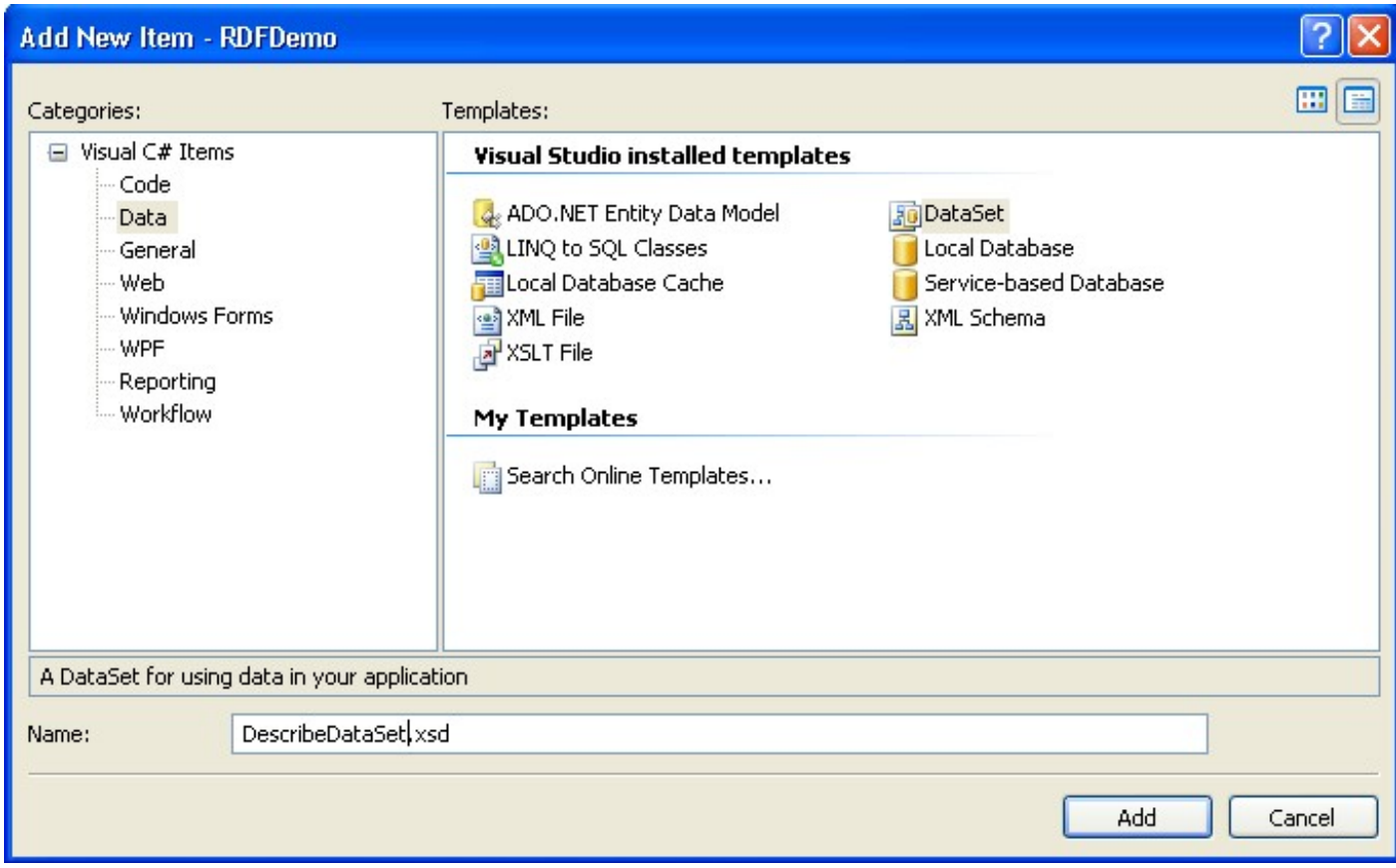
   DataSet

   . Call the new

   DataSet

   DescribeDataSet.

   Figure 2.140. Add a new DataSet



2. Double click on DescribeDataSet in the

   Solution Explorer

   to open the

   DataSet

   Designer and drag a

   DataTable

   from the

   Toolbox

   into it.
3. Add two columns, p and o, to the

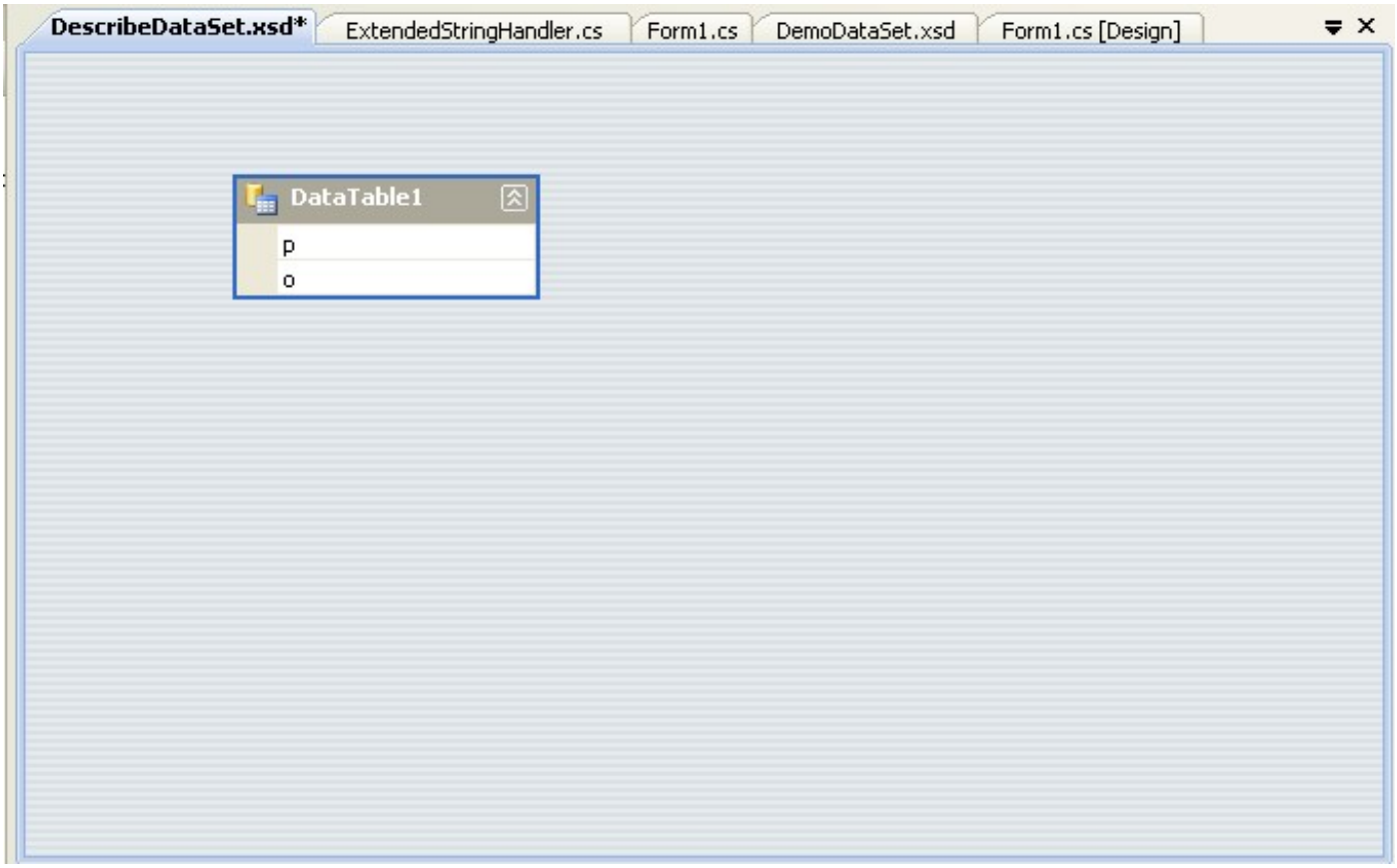
   DataTable

   and set the

   DataType

   of each column to System.Object.

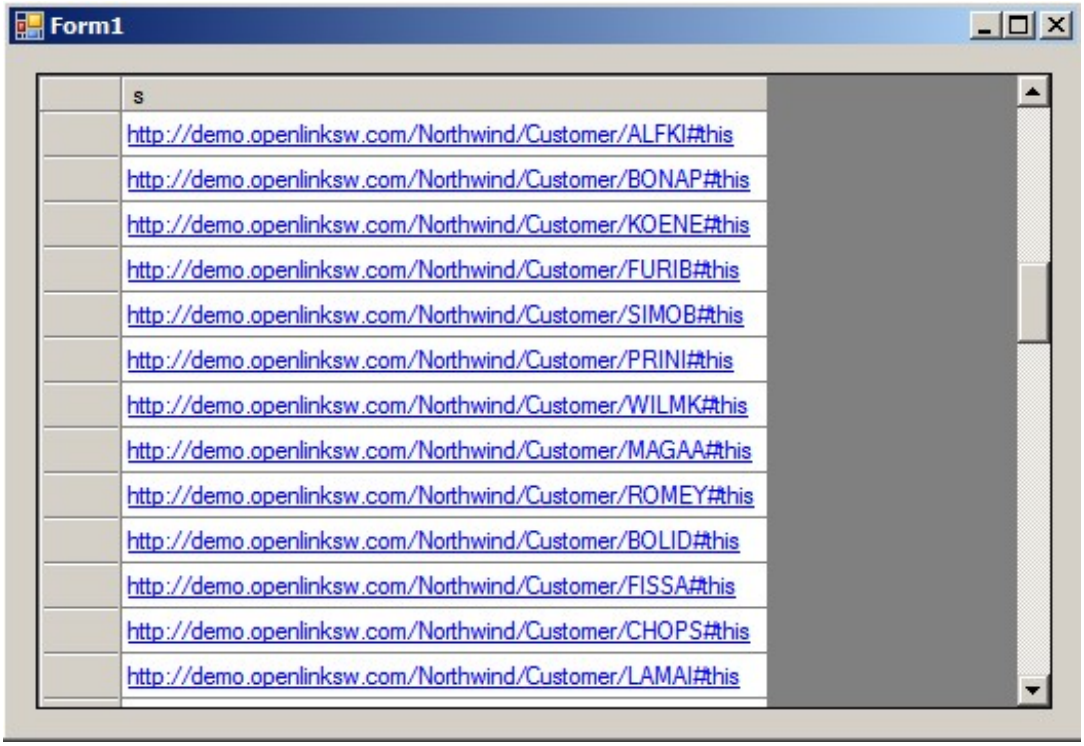
Figure 2.141. Add two columns



Step 7 - Build and run the application.

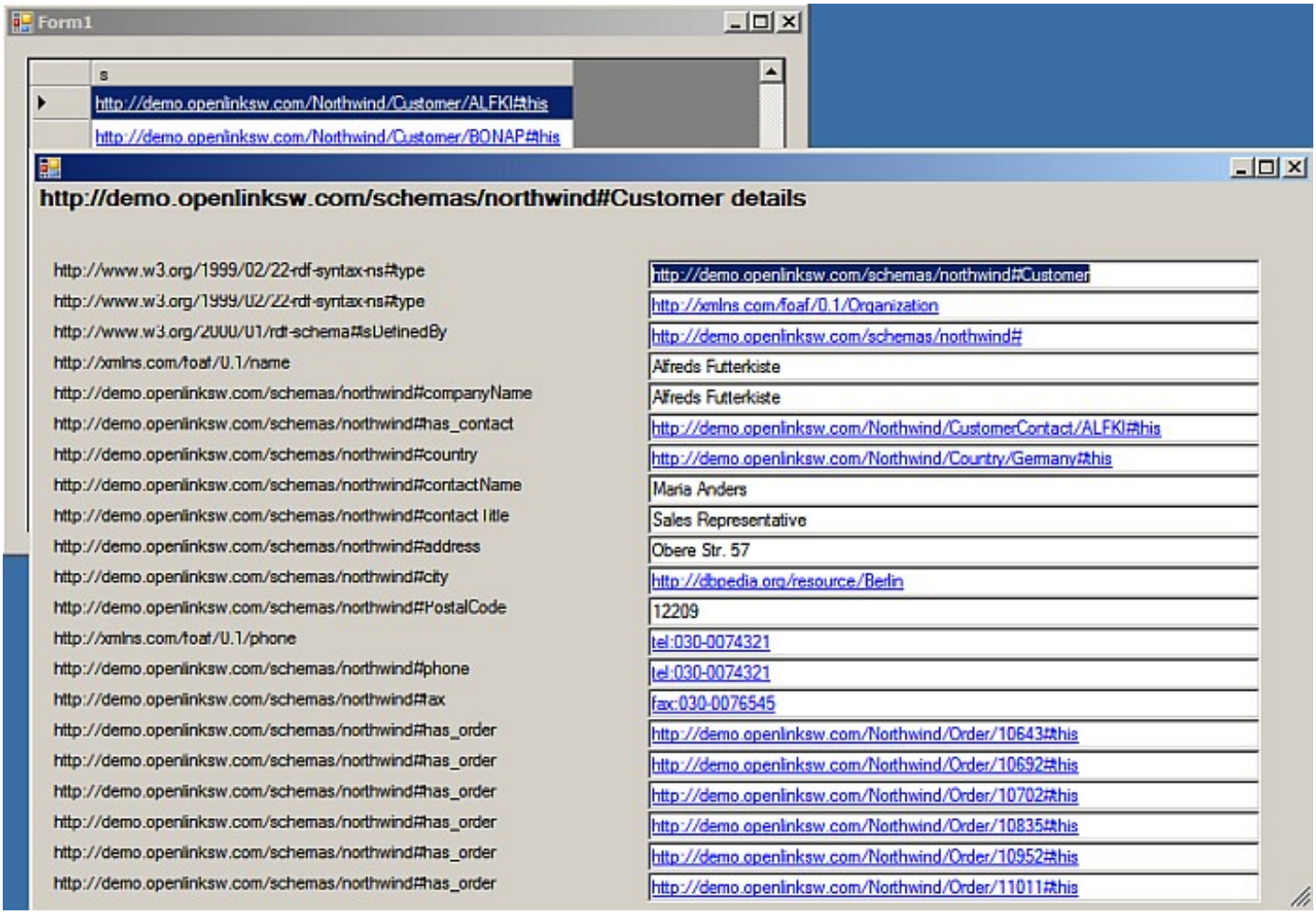
You should see a form displaying all the Northwind customers, like this.

Figure 2.142. Northwind customers



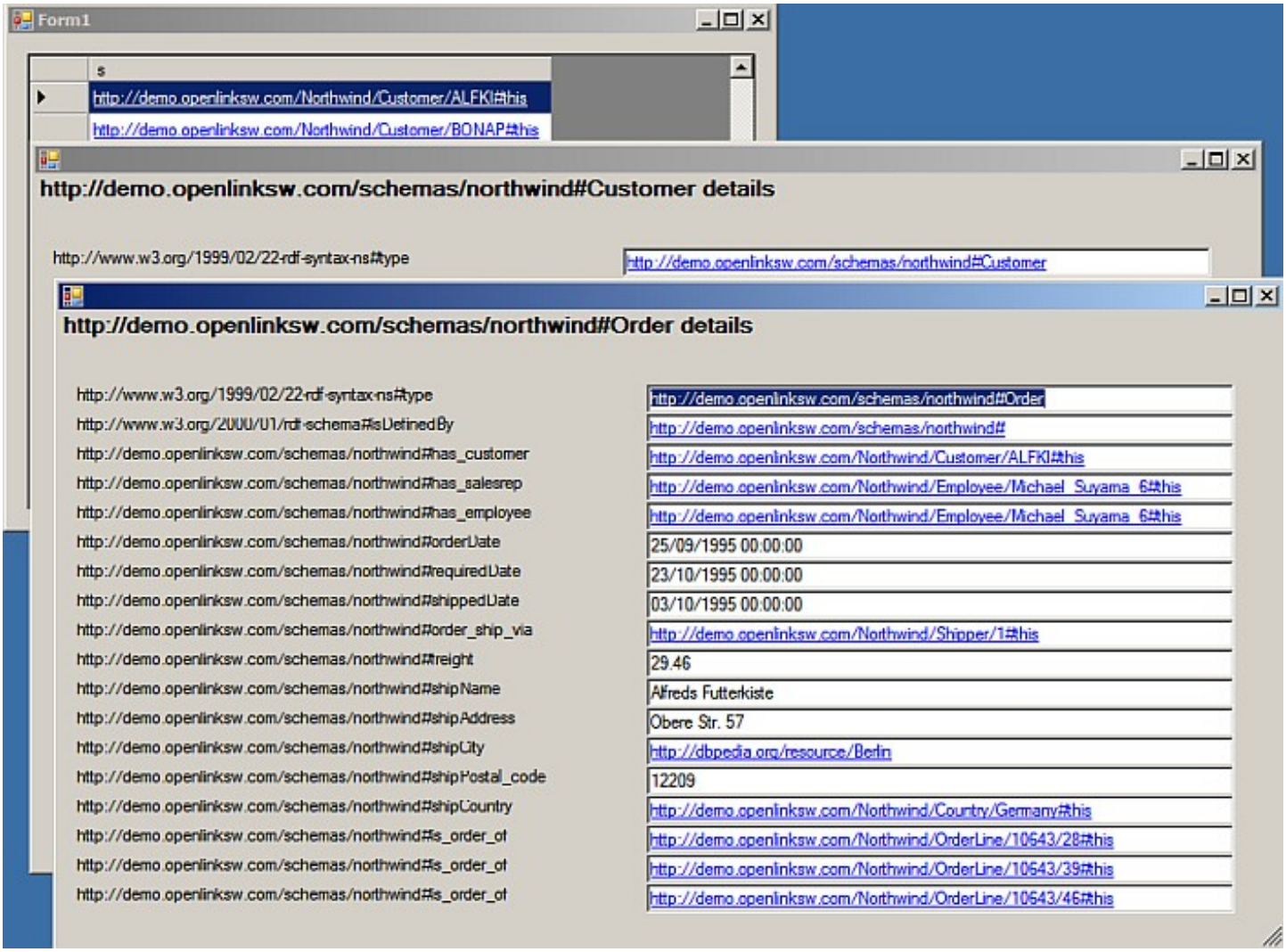
When any customer is clicked it opens a new form showing customer details.

Figure 2.143. Customer details



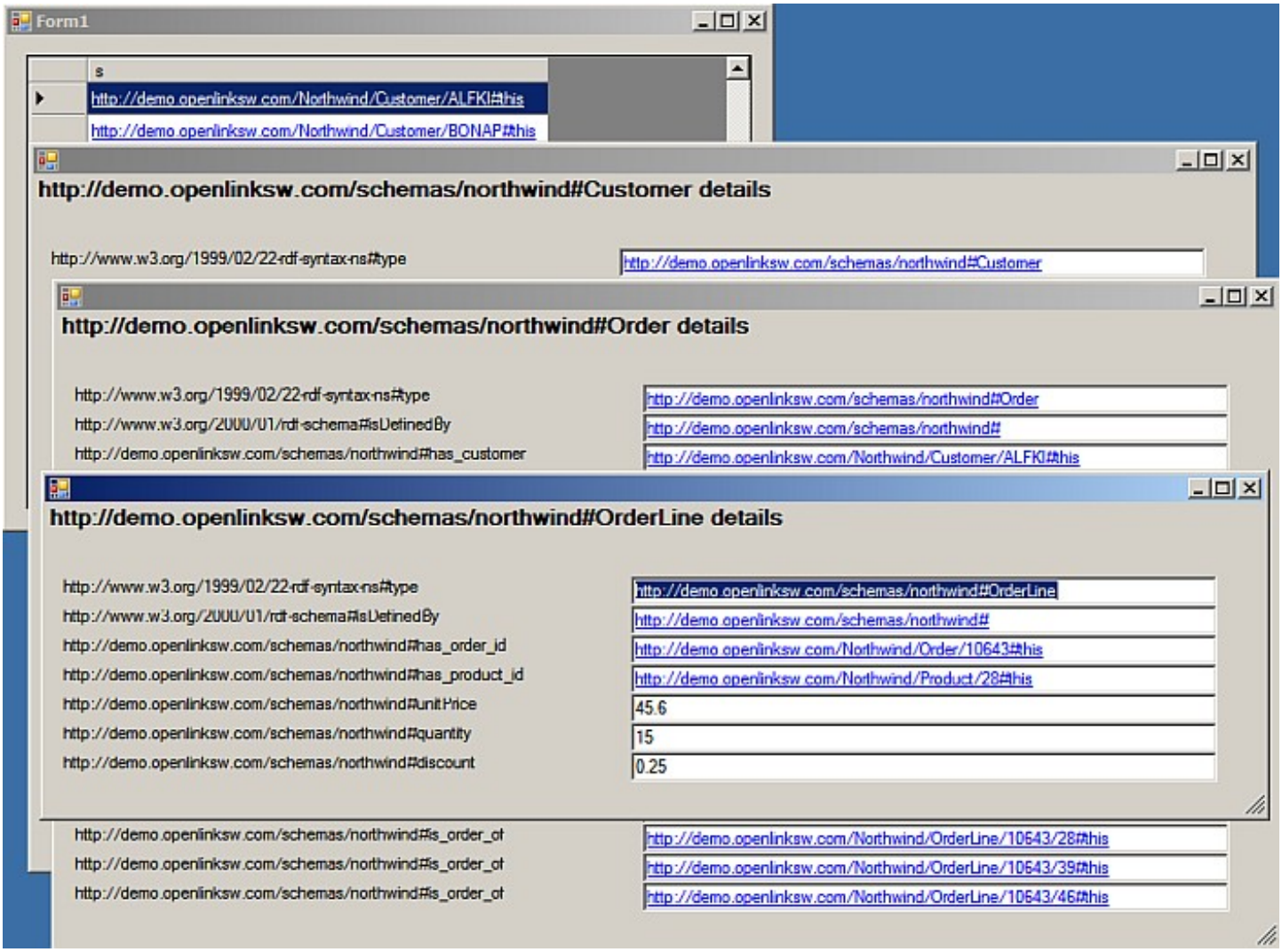
Clicking on the links in the new form allows you to drill down further to get order, product, location details etc.

Figure 2.144. Order, product, location details



and

Figure 2.145. Order, product, location details



Next Steps

You will notice if you keep clicking on the links that this application will only display data that is held in the Northwind graph. Clicking on an external link, for example the link to Berlin in dbpedia, http://dbpedia.org/resource/Berlin, results in a empty window and an error message. The next step is to extend this application so that it can handle dereferencing external IRIs.

2.9.3. Extending RDFDemo to Allow Dereferencing of External IRIs

This section will guide you through extending the application created in Creating a Windows Forms Application To Access RDF Data Using The Virtuoso ADO.Net Provider so that it will dereference external IRIs.

Pre-requisites

1. A working copy of the RDFDemo application created in Creating a Windows Forms Application To Access RDF Data Using The Virtuoso ADO.Net Provider.

Extending the Application

In RDFDemo when the sparql endpoint is queried to get the description of the selected item it executes a query that is restricted to the local Northwind dataset. The query is something like

```
SPARQL
PREFIX nwind: <http://demo.openlinksw.com/schemas/northwind#>
SELECT DISTINCT  ?s
FROM <http://demo.openlinksw.com/Northwind>
WHERE {?s a nwind:Customer}
```

If you examine the ExtendedStringHandler class you will see that the dataset clause, from <http://example.com/Northwind>, is hard coded. This means that when when the selected IRI is a link to an external data store, such as dbpedia, there is no matching data and an error is displayed. If the application is to be able to dereference external IRIs then the hard coded dataset clause needs to be removed and then we can use a Virtuoso extension to SPARQL, get:soft, that tells Virtuoso that it needs to go and look elsewhere for the graph. However, this will result in a loss of performance when exploring the local Northwind dataset. To minimize the impact on performance we will first query the local Northwind dataset and if there are no matching triples returned then we will use a more generic query that will look elsewhere for matching data.

Step 1 - Add the alternative query to the ExtendedString Class.

- Open the RDFDemo project in Visual Studio
- Open the ExtendedStringHandler class.
- Remove DescribeCommand by removing the line

```
StringBuilder DescribeCommand;
```

  and substitute the following:

```
StringBuilder DescribeCommandSimple, DescribeCommandGeneral;
```

- In the ExtendedStringHandler constructor the sparql query that was DescribeCommand becomes DescribeCommandSimple and we define a new query for DescribeCommandGeneral.

```
DescribeCommandSimple = new StringBuilder("sparql select * from <http://demo.openlinksw.com/Northw
DescribeCommandGeneral = new StringBuilder("sparql define get:soft " + '"'.ToString() + "soft" + "
```

- The single describeCommand property needs to be replaced with the two new properties, DescribeCommandSimple and DescribeCommandGeneral

```
public string describeCommandSimpleText
{
    get
    {
        return DescribeCommandSimple.ToString();
    }
}
public string describeCommandGeneralText
{
    get
    {
        return DescribeCommandGeneral.ToString();
    }
}
```

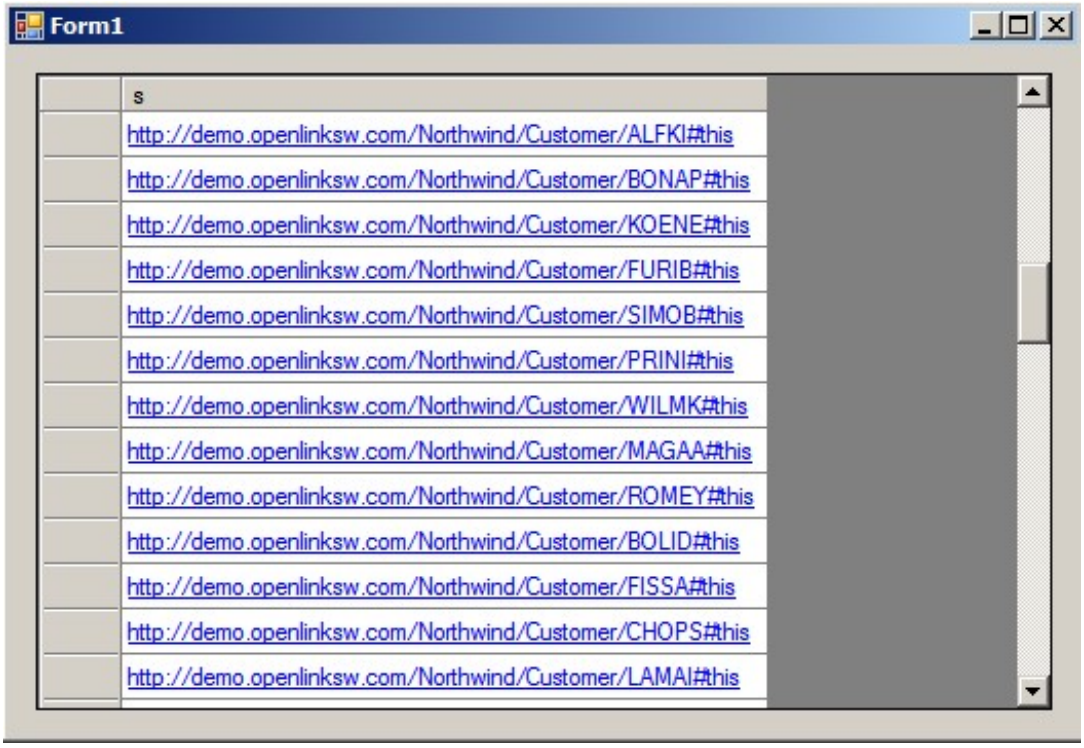
- Finally, the getDescribeData method needs changing to:

```
public void getDescribeData()
{
    VirtuosoCommand myCommand = new VirtuosoCommand(this.describeCommandSimpleText, this.ParentCon
    VirtuosoDataAdapter myAdapter = new VirtuosoDataAdapter();
    myAdapter.SelectCommand = myCommand;
    myAdapter.Fill(describeDataSet.DataTable1);
    // Tried the simple version if fails to get the data try
    // to look elsewhere.
    if (describeDataSet.DataTable1.Rows.Count == 0)
    {
        myCommand.CommandText = describeCommandGeneralText;
        myAdapter.Fill(describeDataSet.DataTable1);
    }
}
```

Step 2 - Build and Run the Application

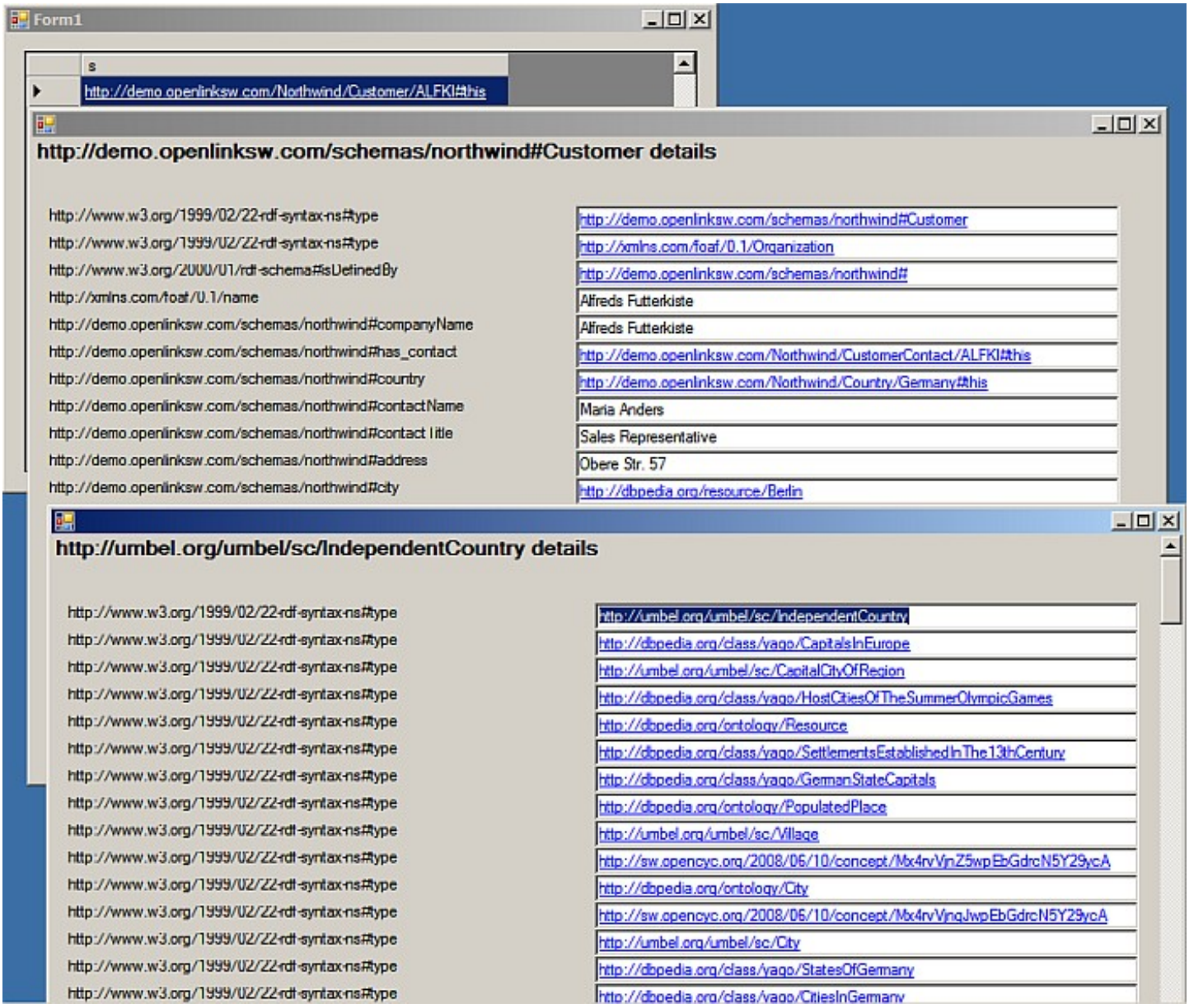
You will see the same starting form:

Figure 2.146. Build and Run the Application



Select a Customer and then select the link to the City in dbpedia. This will now open up another window displaying information about the city from dbpedia. Be patient as it may take a little while to open.

Figure 2.147. Customer



Step 3 - Changing the Form Title

Notice that in displayData method that we look for a http://www.w3.org/1999/02/22-rdf-syntax-ns#type and create a title for the form from it.

```
foreach (DataRow row in table1.Rows)
  if (row[0].ToString() == "http://www.w3.org/1999/02/22-rdf-syntax-ns#type")
  {
      StringBuilder title = new StringBuilder(row[1].ToString() + " details");
      label1.Text = title.ToString();
      break;
  }
```

This worked well for the Northwind subjects but less well now we are getting data from other graphs. To change the title of the forms used to display the data:

- Add an new member variable to hold the IRI that we exploring to the bock of member variables

```
StringBuilder DescribeCommandSimple, DescribeCommandGeneral;
VirtuosoConnection ParentConnection;
List<Label> labelList = new List<Label>();
List<TextBox> textBoxList = new List<TextBox>();
DescribeDataSet describeDataSet = new DescribeDataSet();
Boolean isIRI = false;
SqlExtendedString ParentIRI;
```

- Assign a value to ParentIRI in the constructor:

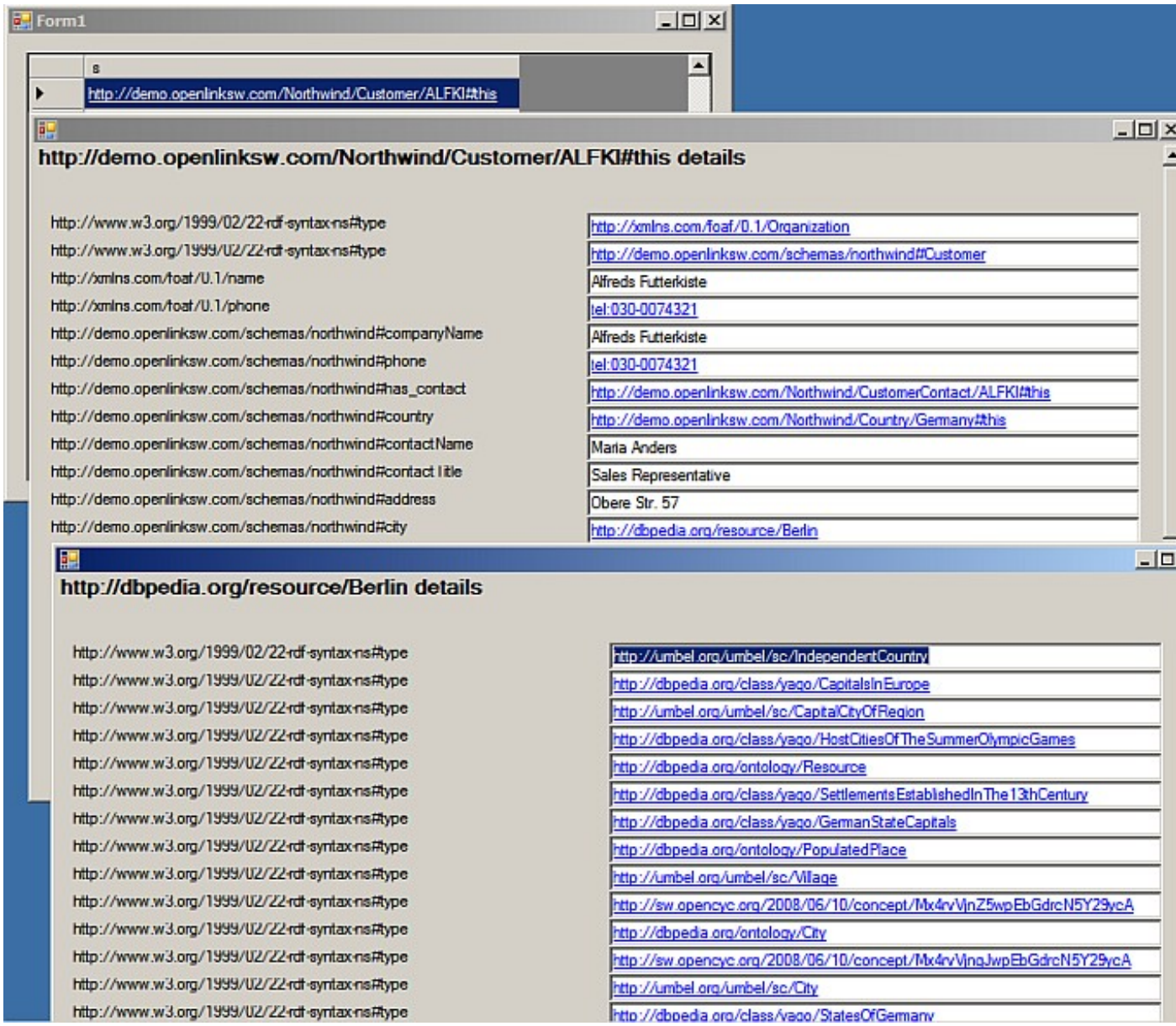
```
public ExtendedStringHandler(SqlExtendedString iri, VirtuosoConnection parentConnection)
  {
      ParentConnection = parentConnection;
      if (iri.IriType == SqlExtendedStringType.IRI)
      {
          ParentIRI = iri;
          isIRI = true;
          DescribeCommandSimple = new StringBuilder("sparql select * from <http://demo.openlinksw.
          DescribeCommandGeneral = new StringBuilder("sparql define get:soft " + '"'.ToString() +
      }
  }
```

- Remove the existing foreach block that sets the form title and replace with the following lines:

```
StringBuilder title = new StringBuilder(ParentIRI.ToString() + " details");
label1.Text = title.ToString();
```

- Build and run the application.

Figure 2.148. Build and run the application



Next Steps

The application now allows you to explore data and follow links from your locally held data into the external web of data. Looking at the data displayed in the form it would be nice to make the labels for the properties more compact. The label http://dbpedia.org/property/population is a very precise definition but for our purposes it would be clearer to label the property just population. In the next step will be to modify the application so that it displays more readable labels.

2.9.4. Extending RDFDemo to Display More Compact Labels

This section will guide you through extending the application created in Extending RDFDemo to Allow Dereferencing of External IRIs so that the data is displayed in a more readable form.

Pre-requisites

1. A working copy of the RDFDemo application created in Extending RDFDemo to Allow Dereferencing of External IRIs

Extending the Application

The RDF demo application presents the user with a list of Customers from the Northwind database in the form of dereferenceable IRIs. When a customer is selected from the list the application uses a sparql query to describe that customer and the results are displayed in a form as rows of labels and data. The labels correspond to RDF predicates and the data corresponds to RDF objects while the subject is the customer initially selected. In many cases the objects are dereferencable IRIs which are then used as the subject when the 'drilling down' into the data. However, the predicates are also IRIs so it is possible to gain more information about these as well.

The RDF Schema defines a property http://www.w3.org/2000/01/rdf-schema#label that may be used to provide a human-readable version of a resource's name. We can obtain further details of each of the predicates in a resultset and check to see if one of the properties is an http://www.w3.org/2000/01/rdf-schema#label. If it is we can use the associated text as the label in our form instead of the the predicate IRI. The benefit should be a more human readable form.

Step 1 - Add a New Method to Get the Label Text

This method takes the predicate IRI and issues a sparql query to get its description. It then cycles through the returned dataset to find a http://www.w3.org/2000/01/rdf-schema#label. If there is one then the associated text is returned by the method. Otherwise the method returns the IRI string.

- Add the following method to the ExtendedStringHandlerClass

```
private string getLabelText(Object label)
  {
      string labelText = label.ToString();
      if (label is SqlExtendedString)
      {
          SqlExtendedString se = (SqlExtendedString)label;
          StringBuilder getLabelCommandText = new StringBuilder("sparql define get:soft \"soft\" s
          VirtuosoCommand getLabelCommand = new VirtuosoCommand(getLabelCommandText.ToString(), Pa
          VirtuosoDataAdapter getLabelAdapter = new VirtuosoDataAdapter();
          getLabelAdapter.SelectCommand = getLabelCommand;
          DataSet getLabelDataSet = new DataSet();
          try
          {
              getLabelAdapter.Fill(getLabelDataSet);
              foreach (DataRow getLabelRow in getLabelDataSet.Tables[0].Rows)
              {
                  if (getLabelRow[0].ToString() == "http://www.w3.org/2000/01/rdf-schema#label")
                  {
                      labelText = getLabelRow[1].ToString();
                      break;
                  }
              }
          }
          catch
          {
          }
      }
      return labelText;
  }
```

- Change the line in displayData from

```
propertyLabel.Text = row[0].ToString();
```

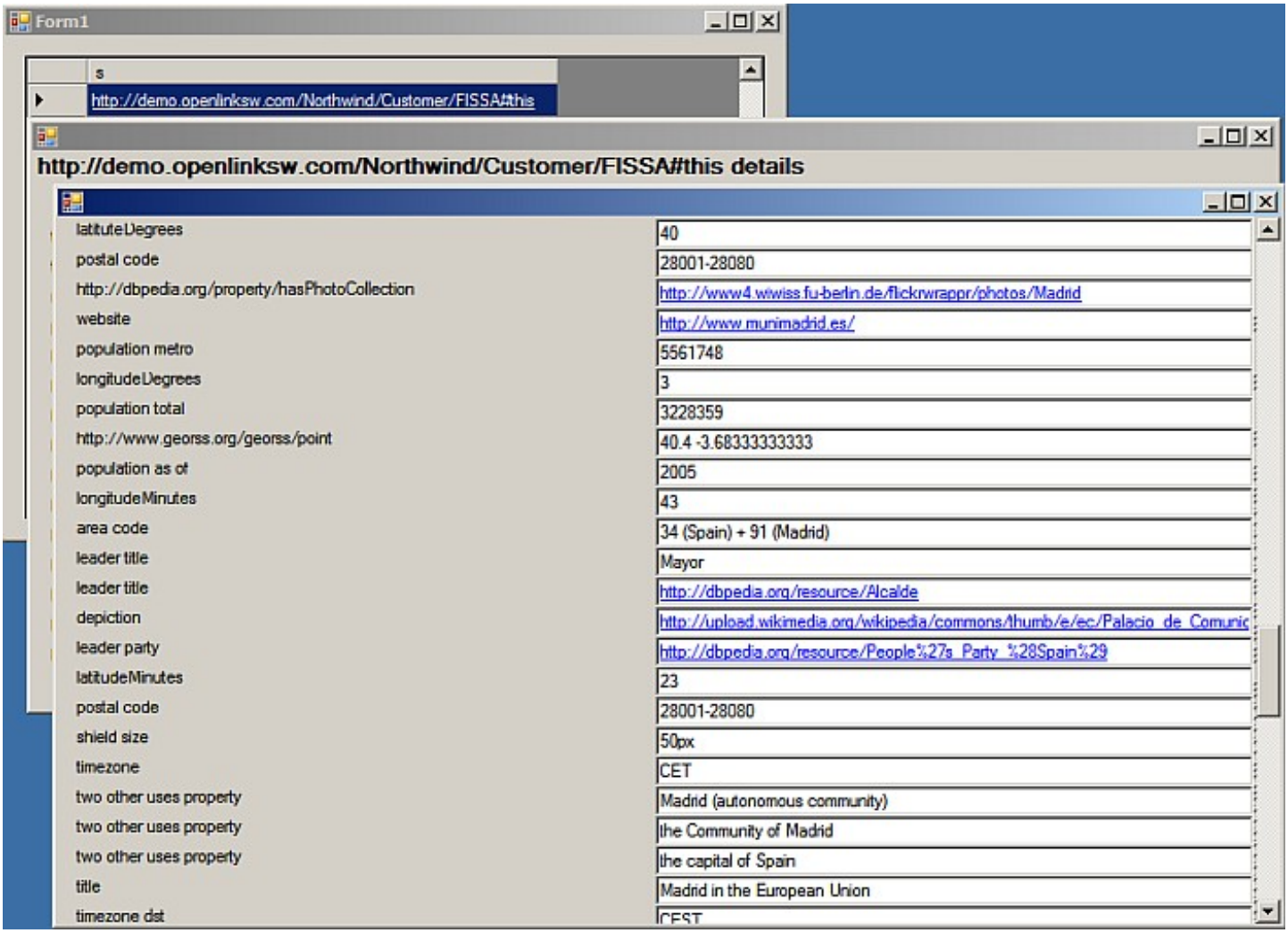
- to

```
propertyLabel.Text = getLabelText(row[0]);
```

Step 1 - Add a New Method to Get the Label Text

When you run the application you will see that the initial form is the same. In fact, when you select the Customer you will also see that the customer details are also the same. It is only when you start exploring data outside the Northwind graph that you will see the labels in the form change.

Figure 2.149. Northwind graph



Next Steps

It is clear from running the application that the Northwind ontology does not define an http://www.w3.org/2000/01/rdf-schema#label for its members. In order to benefit from this modified version of RDFDemo we need to update our Northwind ontology so that http://www.w3.org/2000/01/rdf-schema#label is defined for each resource. The next step will be to modify our Northwind ontology.

2.9.5. Modifying the Northwind Ontology to Add Labels

This section will guide you through modifying the Northwind Ontology created when you installed the demo database VAD package so that each resources is identified by an http://www.w3.org/2000/01/rdf-schema#label. This will improve the readability of the information displayed by the application created in Extending RDFDemo to Display More Compact Labels .

Pre-requisites

1. A working copy of the RDFDemo application created in Extending RDFDemo to Display More Compact Labels

Editing the Ontology

Get a Working Copy of the Northwind Ontology

The the file describing the Northwind Ontology, nw.owl, is installed in the DAV when the demo vad is loaded. To get a working copy open the Virtuoso Conductor and log in as dba. Select WebDAV Browser in the navigation panel on the left. This will open a window that allows you to browse the WebDAV Repository. The Northwind Ontology file can be found in DAV/VAD/demo/sql. Take a copy of the file.

Editing nw.owl

In the first instance is edited nw.owl so that the property name consistently begin with a lower case letter. This matches the results for describing resources held in the Northwind database. Also are added missing properties so that there should be a label in all cases.

Registering the Changes in Virtuoso

There are two methods for registering the changes in Virtuoso:

1. Method I:

   - Copy the edited version of nw.owl back into the DAV.
   - In isql, load the script load_ontology_dav.sql and execute it. The new version of nw.owl will then be used
2. - Ensure that your new version of nw.owl is in a folder accessible by Virtuoso. You may need to edit your virtuoso.ini file and restart the server.
   - In isql, load the script load_ontology_file.sql and edit it so that it has the full path to the new version of nw.owl
   - Run the script. The new version of nw.owl will then be used.

Modify RDFDemo so that it looks for the graph used to describe the Northwind data and searches that graph for the predicate details:

1. Add a new member variable to the ExtendedStringHandler class to hold the graphs that we need to search for the predicate information.

```
StringBuilder DescribeCommandSimple, DescribeCommandGeneral;
VirtuosoConnection ParentConnection;
List<Label> labelList = new List<Label>();
List<TextBox> textBoxList = new List<TextBox>();
List<String> graphList = new List<String>();
DescribeDataSet describeDataSet = new DescribeDataSet();
Boolean isIRI = false;
SqlExtendedString ParentIRI;
```

2. In displayData, after we have set the title of the form, add the following block of code:

```
// Later we will want to get property labels and for that
 // we will need the graph where the resource is defined.
foreach (DataRow row in table1.Rows)
     if (row[0].ToString() == "http://www.openarchives.org/ore/terms/isDescribedBy"
    && row[1].ToString() != ParentIRI.ToString())
     {
         String graph = row[1].ToString();
         graphList.Add(graph);
     }
```

3. Replace the existing getLabelText method with an extended version

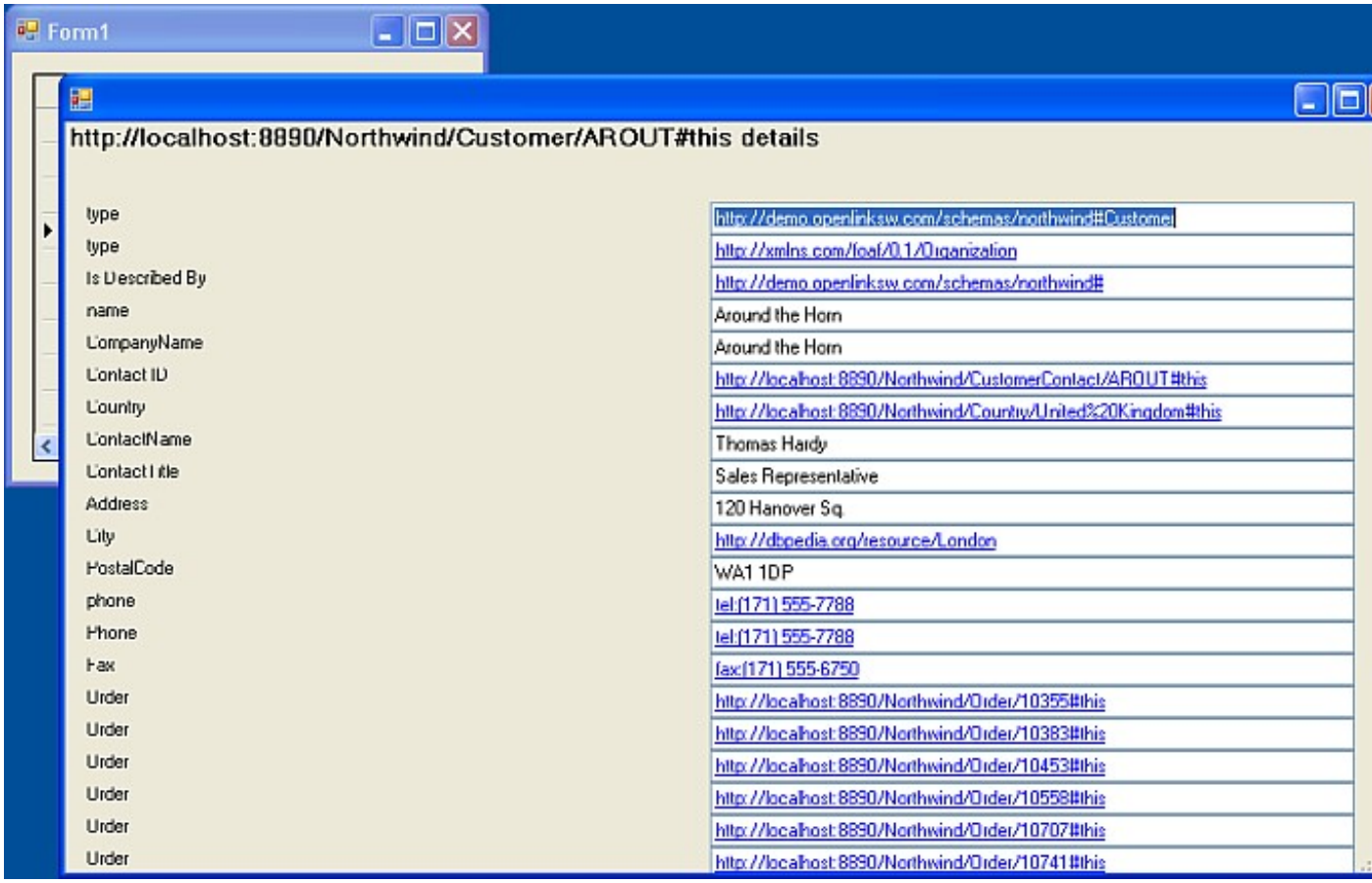
```
 private string getLabelText(Object label)
  {
      string labelText = label.ToString();
      if (label is SqlExtendedString)
      {
          Boolean foundLabel = false;
          SqlExtendedString se = (SqlExtendedString)label;
          VirtuosoDataAdapter getLabelAdapter = new VirtuosoDataAdapter();
```

```
            DataSet getLabelDataSet = new DataSet();

            //Try finding it in resources graph first
            foreach (String graph in graphList)
            {
                StringBuilder getLabelCommandText = new StringBuilder("sparql select * from <" + gra
                VirtuosoCommand getLabelCommand = new VirtuosoCommand(getLabelCommandText.ToString()
                getLabelAdapter.SelectCommand = getLabelCommand;

                try
                {
                    getLabelAdapter.Fill(getLabelDataSet);
                    foreach (DataRow getLabelRow in getLabelDataSet.Tables[0].Rows)
                    {
                        if (getLabelRow[0].ToString() == "http://www.w3.org/2000/01/rdf-schema#label
                        {
                            labelText = getLabelRow[1].ToString();
                            foundLabel = true;
                            break;
                        }
                    }
                }
                catch
                {
                }
                if (foundLabel)
                    break;
            }
            // If we still have no label try the predicate itself as the graph
            if (!foundLabel)
            {
                StringBuilder getLabelCommandText = new StringBuilder("sparql define get:soft \"soft
                VirtuosoCommand getLabelCommand = new VirtuosoCommand(getLabelCommandText.ToString()
                getLabelAdapter.SelectCommand = getLabelCommand;
                try
                {
                    getLabelAdapter.Fill(getLabelDataSet);
                    foreach (DataRow getLabelRow in getLabelDataSet.Tables[0].Rows)
                    {
                        if (getLabelRow[0].ToString() == "http://www.w3.org/2000/01/rdf-schema#label
                        {
                            labelText = getLabelRow[1].ToString();
                            break;
                        }
                    }
                }
                catch
                {
                }
            }
        }
        return labelText;
    }
```

This extended method first checks the graphs in the graph list for the predicate information then if no label has been found tries the predicate itself as the graph.

4. Build and run, the Northwind resources should now be correctly and concisely labeled

Figure 2.150. Northwind resources



Improving The Appearance of the Form

The following changes are not strictly necessary but improve the appearance of the form:

- Line up the right hand edge of the labels with the text boxes by setting TextAlign to MiddleRight and reduce the width of the labels.

```
propertyLabel.Text = getLabelText(row[0]);
propertyLabel.AutoEllipsis = true;
propertyLabel.AutoSize = false;
propertyLabel.Width = 130;
propertyLabel.TextAlign = ContentAlignment.MiddleRight;
```

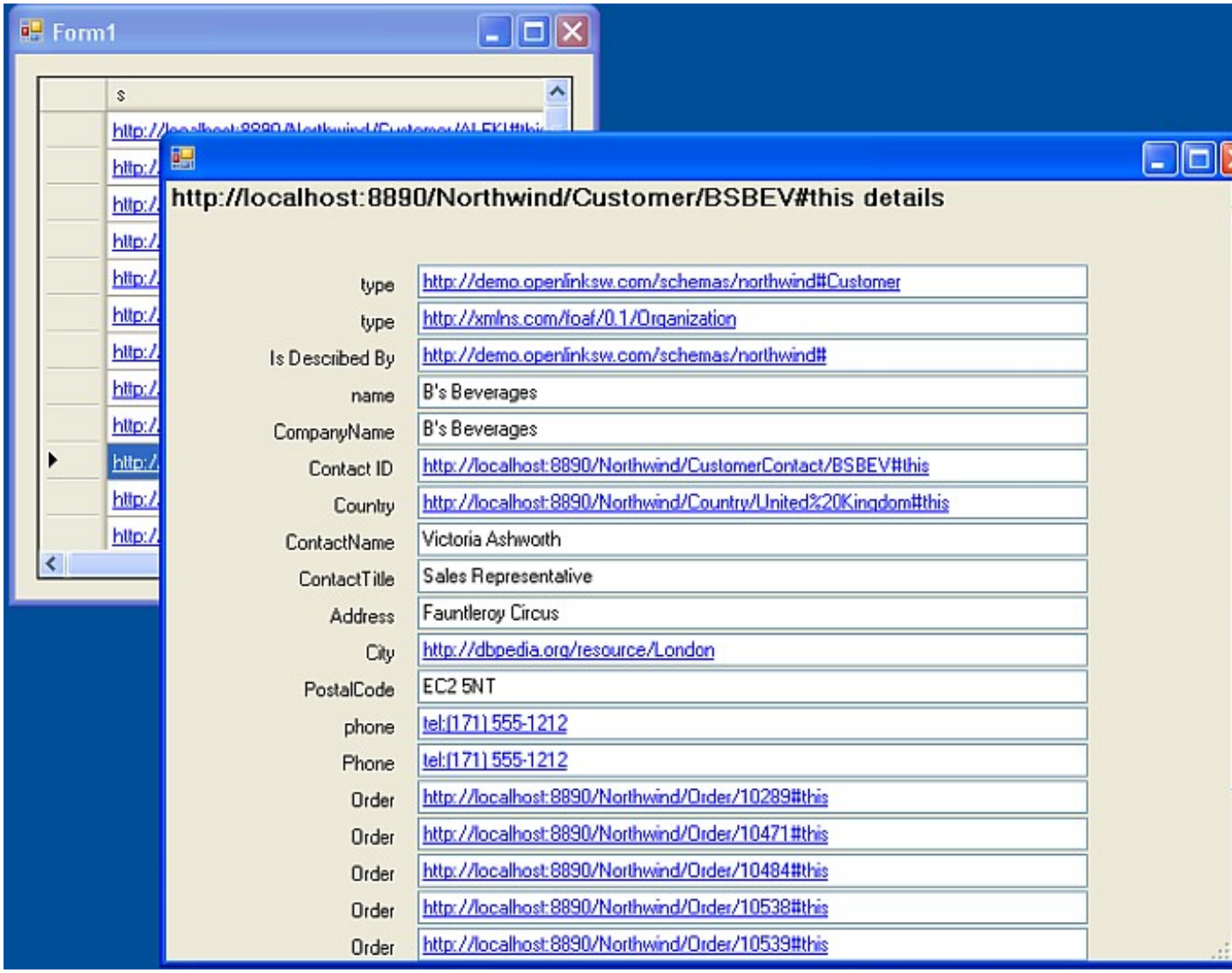
- Make the form narrower:

```
describeForm.Width = 660;
```

- Alter the positioning of the labels and TextBoxes on the form:

```
textBoxList[i].Click += new EventHandler(this.iri_Click);
labelList[i].Location = new Point(10, i * 22 + 50);
textBoxList[i].Location = new Point(150, i * 22 + 50);
describeForm.Controls.Add(labelList[i]);
describeForm.Controls.Add(textBoxList[i]);
```

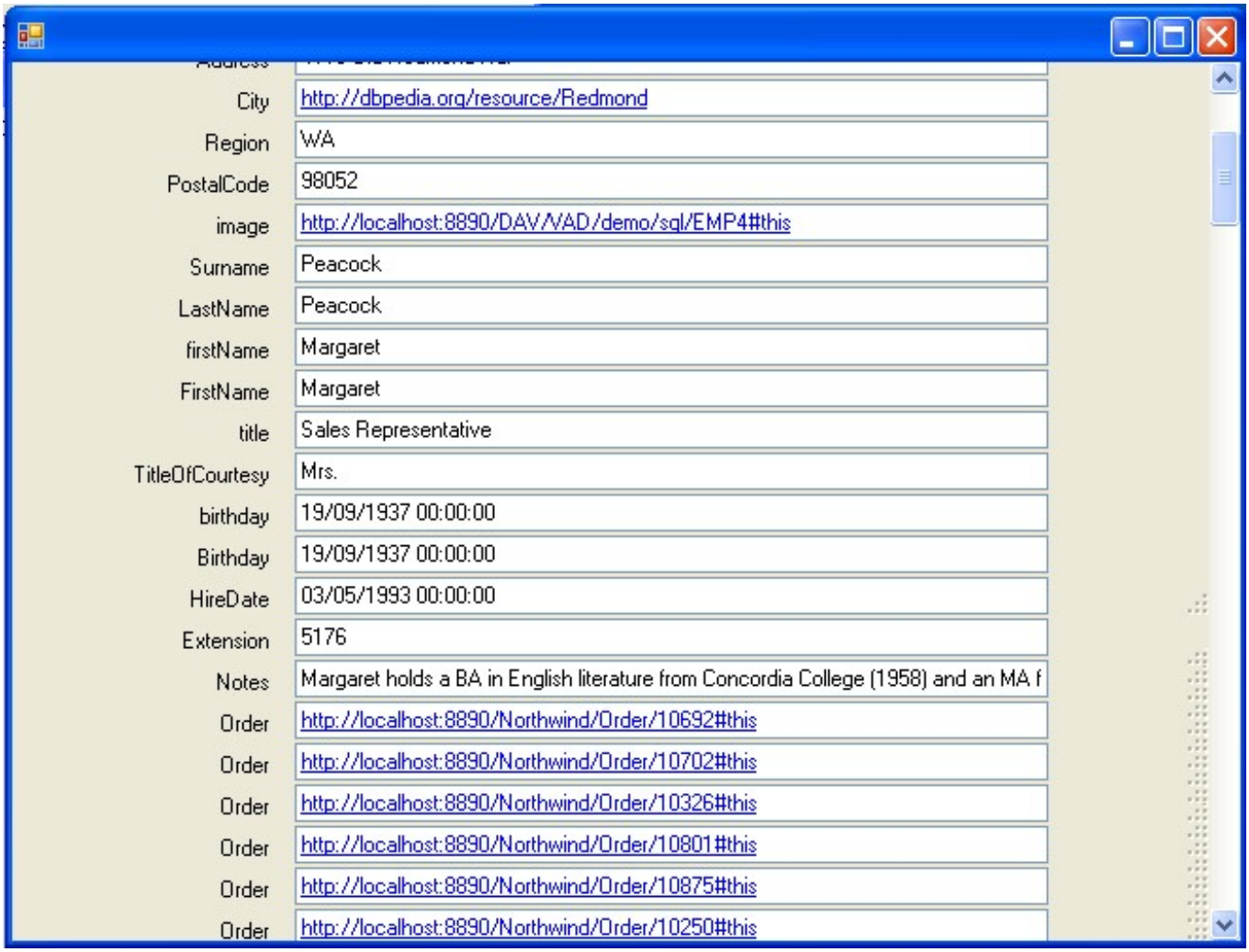
Figure 2.151. Alter the positioning



Next Steps

The image below shows some of the information about an employee in the Northwind dataset.

Figure 2.152. employee



In the next step we will extend the application so the images and web pages can be viewed and long text fields are displayed in full.

2.9.6. Extending RDFDemo to Display Images and Longer Text Fields.

This section will guide you through extending RDFDemo so that longer text fields can be displayed as a block of text and so that links to images and web pages can be viewed in a browser window.

Pre-requisites

1. A working copy of the RDFDemo application created in Modifying the Northwind Ontology to Add Labels

Modifying the Application

Displaying Text

We will modify the form that show the details of the selected item so that when the text in the boxes is too long to be seen in full a button will appear beside the box on the form and if you click the button the complete text will be displayed in a separate window.

1. Add a new class, MoreButton that extends System.Windows.Forms.Button.

    - In the

      Solution Explorer

      right click on the RDFDemo project and select

      Add

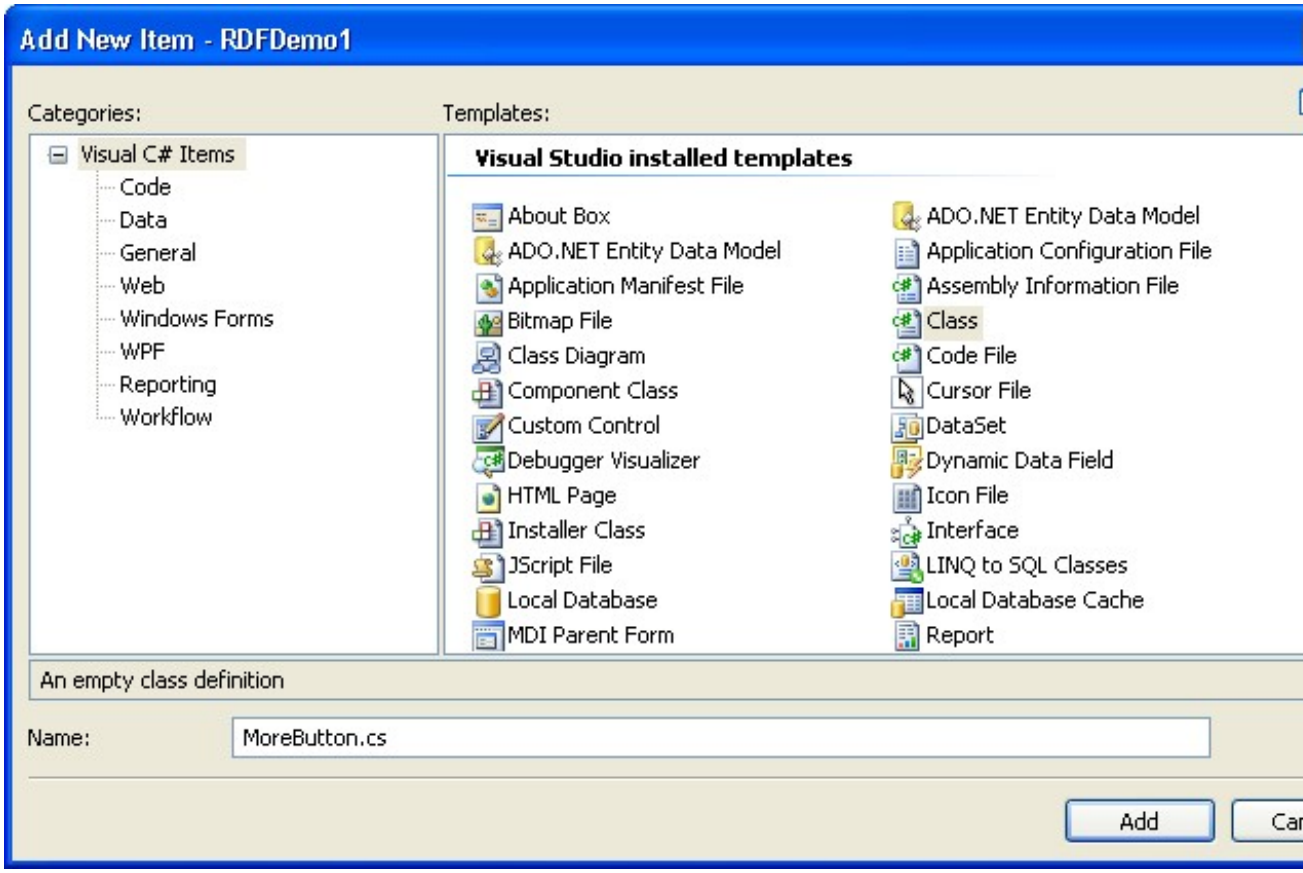
then

New Item

.

- Add a new class called MoreButton.cs.

Figure 2.153. Add a new class



- The following using statement goes at the top of the file:

```
using System.Windows.Forms;
```

- The MoreButton class must inherit from System.Windows.Forms.Button so the class definition line should look like this:

```
class MoreButton : Button
```

- Add the following code to the body of the MoreButton class:

```
String longText;

public MoreButton(String text)
{
    longText = text;
    this.Text = "More";
}

protected override void  OnClick(EventArgs e)
{
    Form moreForm = new Form();
    TextBox moreBox = new TextBox();
    moreBox.Text = longText;
    moreBox.Width = 300;
    moreBox.Height = 250;
    moreBox.ScrollBars = ScrollBars.Vertical;
    moreBox.Multiline = true;
    moreBox.WordWrap = true;
    moreBox.Select(0, 0);
```

```
            moreBox.ReadOnly = true;
            moreForm.Controls.Add(moreBox);
            moreForm.Width = 320;
            moreForm.Height = 280;
            moreForm.ShowDialog();
        }
```

2. In displayData in the ExtendedStringHandler class, when the labels and TextBoxes are added to the form check if the text is too big for the box. If it is then add a MoreButton that will display all the text in a separate window.

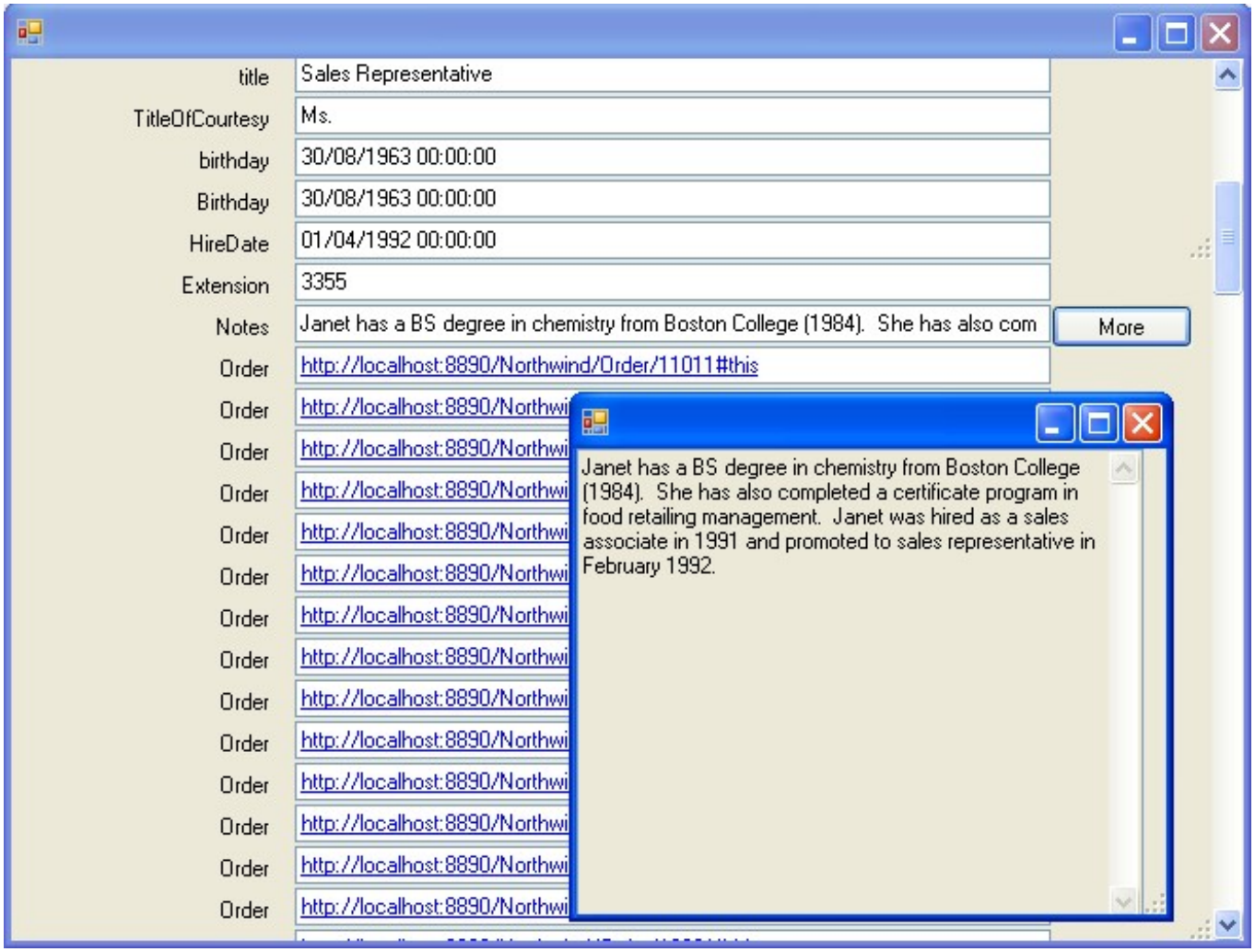
```
if (textBoxList[i].DataBindings[0].DataSource.ToString().Length > 80
    && !(textBoxList[i].DataBindings[0].DataSource is SqlExtendedString))
{
    moreButtonList.Add(new MoreButton(textBoxList[i].DataBindings[0].DataSource.ToString()));
    moreButtonList[moreButtonList.Count - 1].Location = new Point(550, i * 22 + 50);
    describeForm.Controls.Add(moreButtonList[moreButtonList.Count -1]);
}
```

3. We will also need a list to hold the buttons as they are created so the following needs to be added to the member variables at the top of the ExtendedStringHandler class.

```
List<MoreButton> moreButtonList = new List<MoreButton>();
```

4. Build and run the application. If you click through to an Employee page you will see the Notes field now has a button labeled More next to it. If you click on that button the text from the Notes field is displayed in a new window.

Figure 2.154. Notes



Displaying Images and Web Pages

Next we will modify the form so that item identified as images or web pages will be opened in a browser window. Again we will do this by adding a button beside the box on the form that will open the browser window.

1. Add a new class, OpenButton that extends System.Windows.Forms.Button.

   - In the

     Solution Explorer

     right click on the RDFDemo project and select

     Add

     then

     New Item
   - Add a new class called OpenButton.cs.
   - The following using statement goes at the top of the file:

     ```
     using System.Windows.Forms;
     ```
   - The OpenButton class must inherit from System.Windows.Forms.Button so the class definition line should look like this:

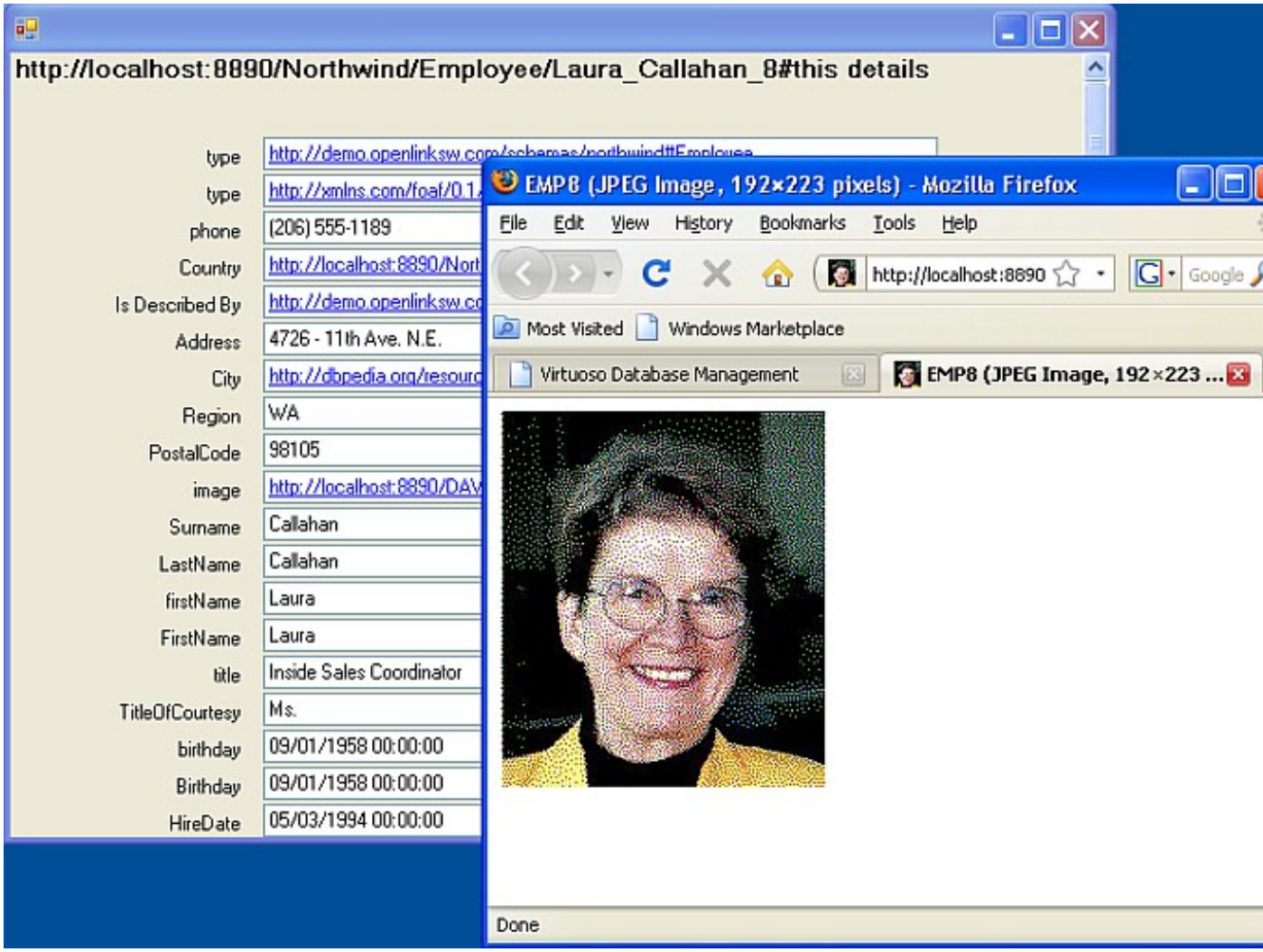
     ```
     class OpenButton : Button
     ```
   - Add the following code to the body of the OpenButton class:

     ```
     String urlText;

     public OpenButton(String text)
     {
         urlText = text;
         this.Text = "Open";
     }

     protected override void  OnClick(EventArgs e)
     {
         System.Diagnostics.Process.Start(urlText);
     }
     ```
2. In displayData in the ExtendedStringHandler class, where we added the code to check for long text fields we now need to check for IRIs that identify images and web pages. As a simple first attempt we will check for matching labels. So for example, if a property label is 'image' or 'webpage', we will assume it can be opened in a browser window and put an OpenButton beside it.

   ```
   if (labelList[i].Text == "website"
       || labelList[i].Text == "image"
       || labelList[i].Text == "depiction"
       || labelList[i].Text == "page"
       || labelList[i].Text == "url"
       || labelList[i].Text == "image skyline")
   {
       openButtonList.Add(new OpenButton(textBoxList[i].DataBindings[0].DataSource.ToString()));
       openButtonList[openButtonList.Count - 1].Location = new Point (550, i * 22 + 50);
       describeForm.Controls.Add(openButtonList[openButtonList.Count - 1]);
   }
   ```
3. We will also need a list to hold the buttons as they are created so the following needs to be added to the member variables at the top of the ExtendedStringHandler class.

   ```
   List<OpenButton> openButtonList = new List<OpenButton>();
   ```
4. Build and run the application. If you click through to an Employee page now you will see that the Image field now has a button labeled Open next to it. If you click on that button the image is opened in your default browser.

Figure 2.155. Image



Next Steps

It has already been mentioned that the property labels are also dereferenceable IRIs. We used this feature to find a short name to display rather that the complete IRI. The next step is to make the labels clickable so the ontology itself can also be explored.

2.9.7. Extending RDFDemo To Make The Property Labels Clickable

This section will guide you through extending RDFDemo to make the property labels clickable. Clicking on the property label will take you to a page describing that property.

Pre-requisites

1. A working copy of the RDFDemo application created in Extending RDFDemo to Display Images and Longer Text Fields

Modifying the Application

Making the Labels Clickable

Making the property labels clickable is fairly straight forward. We will use basically the same code as is used to make the values clickable.

1. First we need to add a new event handler. In the ExtendedStringHandler class modify displayData to add an event handler for each of the property labels.

```
for (int i = 0; i < table1.Rows.Count; i++)
{
    textBoxList[i].Click += new EventHandler(this.iri_Click);
```

```
    labelList[i].Location = new Point(10, i * 22 + 50);
    textBoxList[i].Location = new Point(150, i * 22 + 50);
    describeForm.Controls.Add(labelList[i]);
    describeForm.Controls.Add(textBoxList[i]);
```

becomes

```
for (int i = 0; i < table1.Rows.Count; i++)
{
    textBoxList[i].Click += new EventHandler(this.iri_Click);
    labelList[i].Location = new Point(10, i * 22 + 50);
    labelList[i].Click += new EventHandler(this.label_Click);
    textBoxList[i].Location = new Point(150, i * 22 + 50);
    describeForm.Controls.Add(labelList[i]);
    describeForm.Controls.Add(textBoxList[i]);
```

2. Then we need to add the EventHandler method. Add the following to the ExtendedStringHandler Class:

```
public void label_Click(object sender, EventArgs e)
  {
      int labelNum = 0;

      for (int i = 0; i < labelList.Count; i++)
      {
          if (sender == labelList[i])
          {
              labelNum = i;
              break;
          }
      }

      Object o = describeDataSet.DataTable1.Rows[labelNum][0];
      if (o is SqlExtendedString)
      {
          SqlExtendedString se = (SqlExtendedString)o;
          ExtendedStringHandler seHandler = new ExtendedStringHandler(se, ParentConnection);
          seHandler.displayData();
      }
      else if (o is SqlRdfBox)
      {
          //doesn't do anything at the moment
      }
  }
```

If you compare this method to the EventHandler for the values, iri_Click, you will see that it is basically the same. The only difference is that it uses the property element from the data table rather than value.

3. Finally, to make it clear that the labels are now active links, we will change the label colour to blue and underline them.

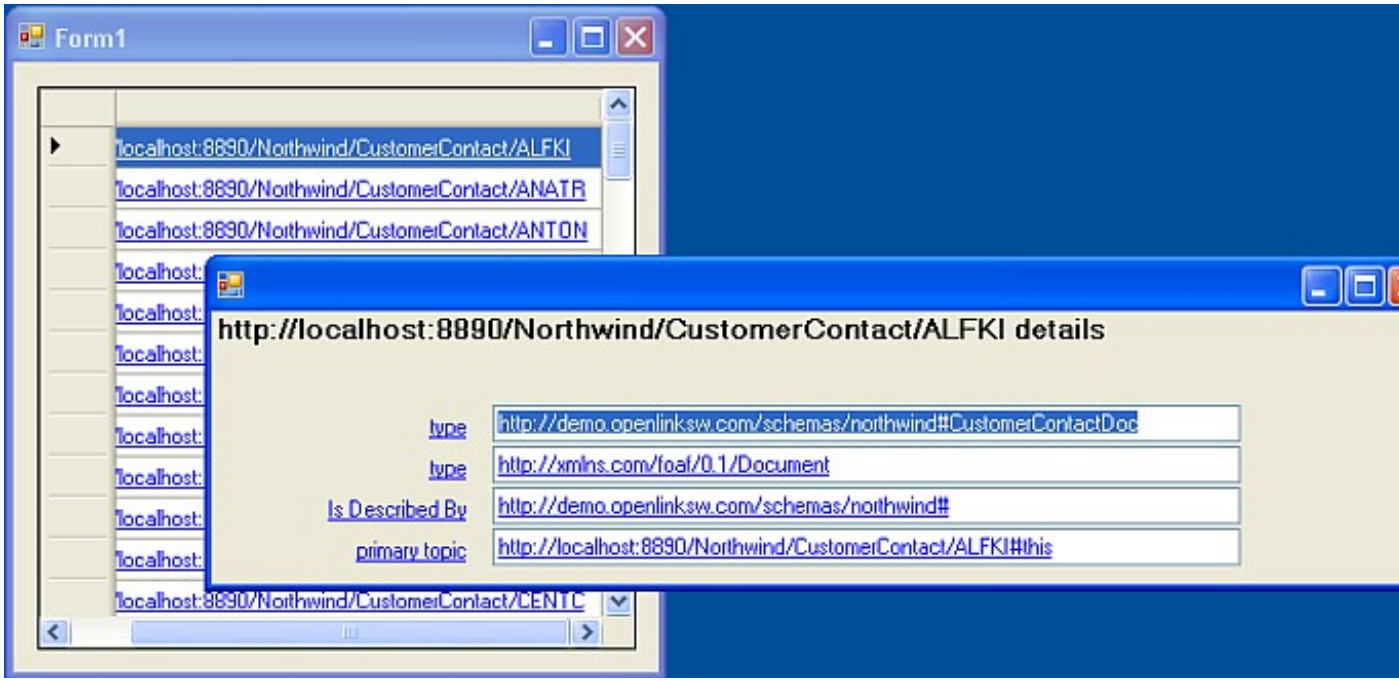
```
 propertyLabel.Text = getLabelText(row[0]);
 propertyLabel.AutoEllipsis = true;
 propertyLabel.AutoSize = false;
 propertyLabel.Width = 130;
 propertyLabel.TextAlign = ContentAlignment.MiddleRight;
```

becomes:

```
propertyLabel.Text = getLabelText(row[0]);
propertyLabel.ForeColor = Color.Blue;
propertyLabel.Font = new Font(propertyLabel.Font.FontFamily, propertyLabel.Font.Size, propertyLabe
propertyLabel.AutoEllipsis = true;
propertyLabel.AutoSize = false;
propertyLabel.Width = 130;
propertyLabel.TextAlign = ContentAlignment.MiddleRight;
```

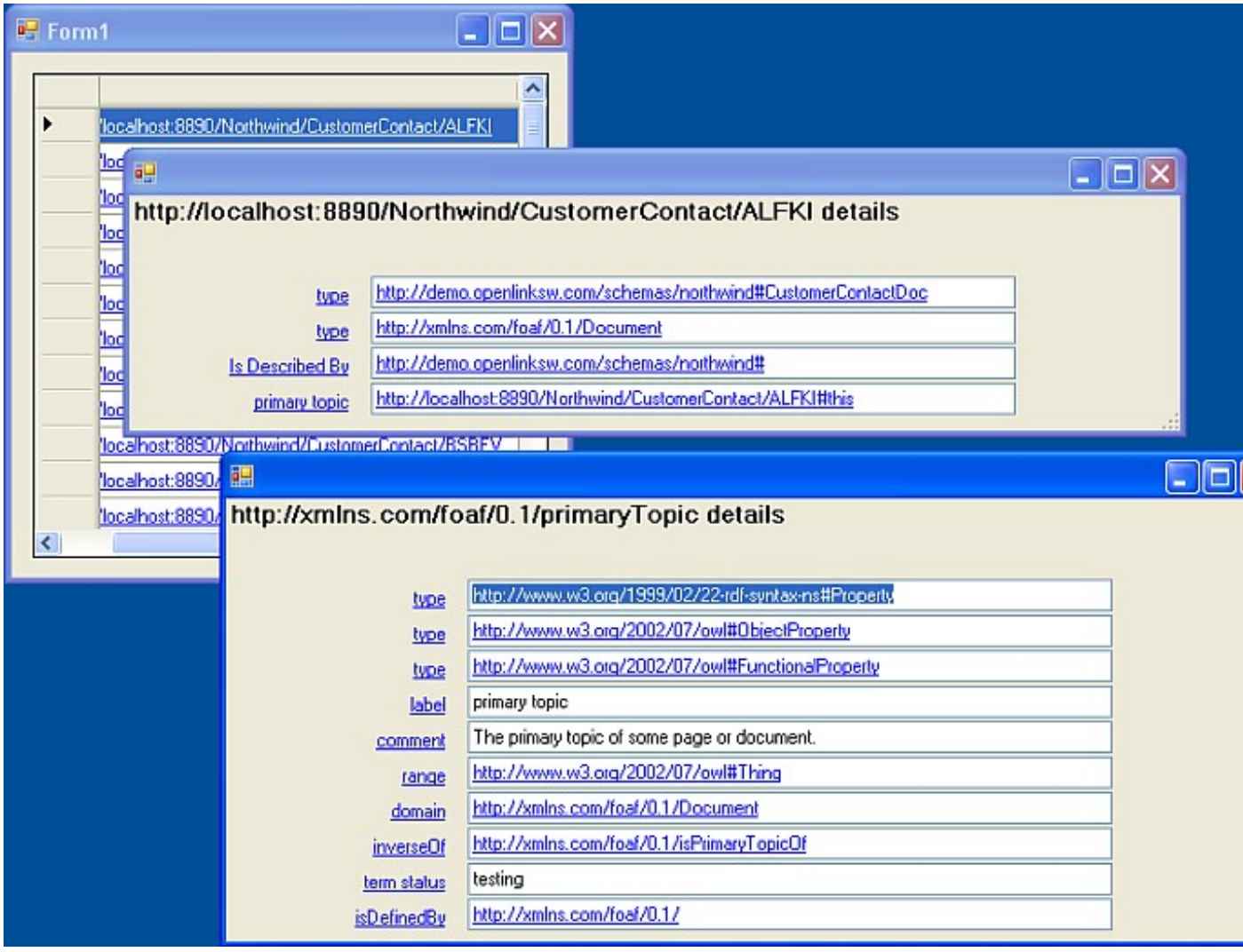
4. Build and run the application. You will see the familiar starting page. If you then select one of the Customers you will notice the property labels now look like hyperlinks.

Figure 2.156. hyperlinks



If you click on one of the label hyperlinks you will see a new form showing detailed information about the property which can itself be explored further by clicking on labels and values.

Figure 2.157. labels and values



These simple changes work up to a point but are not robust. If you explore the properties used by the local Northwind graph you quickly find that the property details are not found. The application needs some further changes to work consistently. The problem is finding the graph where the property information is defined. The general handler for Extended Strings first checks the local Northwind graph, http://example.com/Northwind, where the Northwind data is held, and then uses the IRI itself as the graph and tries to load that dynamically. This is not working for information about the Northwind properties. These properties are defined in the Northwind ontology, http://demo.openlinksw.com/schemas/northwind. We have already had to find this graph when getting the short label name. We need to be able to associate the property label with the graph where its definition is stored. Then we can search this graph for details about the property when the label is clicked.

Using the Graph Where the Property Label was Found to Find the Property Details

1. Create a new class, IRILabel, that inherits from System.Windows.Forms.Label.

   - In the

     Solution Explorer

     right click on the RDFDemo project and select

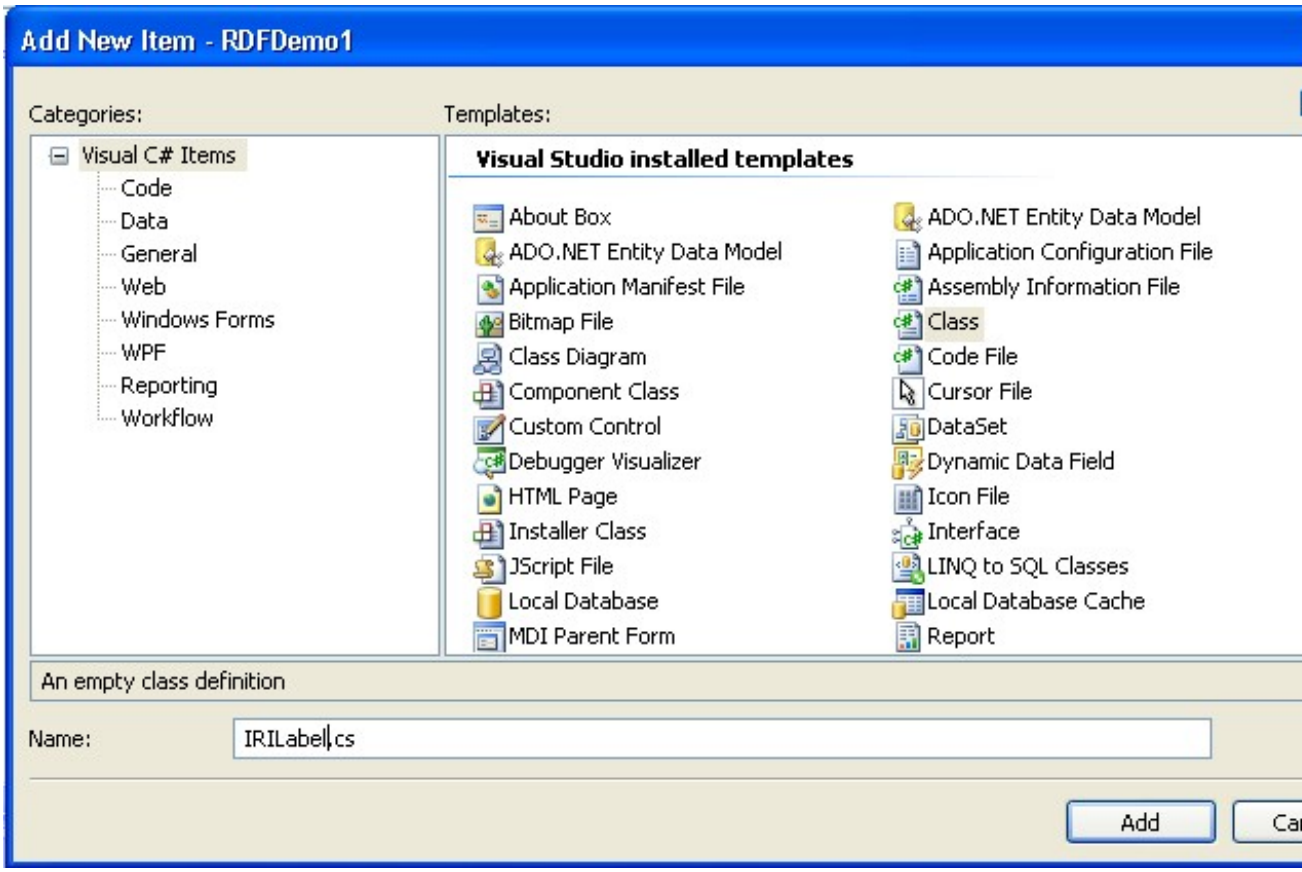
     Add

     then

     New Item

     .
   - Add a new class called IRILabel.cs.

     Figure 2.158. Add a new class

2. We need to reference System.Windows.Forms, System.Data and OpenLink.Data.Virtuoso so add

```
using System.Windows.Forms;
using OpenLink.Data.Virtuoso;
```

```
using System.Data;
```

to the using block at the top of the class file. The class definition should look like this:

```
class IRILabel : Label
```

3. Paste the following code into the body of the class:

```
SqlExtendedString SourceIRI;
String GraphUsed;
VirtuosoConnection ParentConnection;

public IRILabel(Object iri, List<String> graphList, VirtuosoConnection parentConnection)
{
    ParentConnection = parentConnection;
    if (iri is SqlExtendedString)
    {
        SourceIRI = (SqlExtendedString)iri;
        Text = this.getLabelText(graphList);
    }
    else
        Text = iri.ToString();
}

public SqlExtendedString iri
{
    get
    {
        return SourceIRI;
    }
}

public String graph
{
    get
    {
        return GraphUsed;
    }
}

private string getLabelText(List<String> graphList)
{
    string labelText = SourceIRI.ToString();
    Boolean foundLabel = false;
    VirtuosoDataAdapter getLabelAdapter = new VirtuosoDataAdapter();
    DataSet getLabelDataSet = new DataSet();

    //Try finding it in resources graph first
    foreach (String graph in graphList)
    {
        StringBuilder getLabelCommandText = new StringBuilder("sparql select * from <" + graph + "
        VirtuosoCommand getLabelCommand = new VirtuosoCommand(getLabelCommandText.ToString(), Pare
        getLabelAdapter.SelectCommand = getLabelCommand;

        try
        {
            getLabelAdapter.Fill(getLabelDataSet);
            foreach (DataRow getLabelRow in getLabelDataSet.Tables[0].Rows)
            {
                if (getLabelRow[0].ToString() == "http://www.w3.org/2000/01/rdf-schema#label")
                {
                    labelText = getLabelRow[1].ToString();
                    foundLabel = true;
                    break;
                }
            }
        }
        catch
        {
        }
        if (foundLabel)
        {
            GraphUsed = graph;
```

```
                break;
            }
        }

        // If we still have no label try the predicate itself as the graph
        if (!foundLabel)
        {
            GraphUsed = SourceIRI.ToString();
            StringBuilder getLabelCommandText = new StringBuilder("sparql define get:soft \"soft\" sel
            VirtuosoCommand getLabelCommand = new VirtuosoCommand(getLabelCommandText.ToString(), Pare
            getLabelAdapter.SelectCommand = getLabelCommand;
            try
            {
                getLabelAdapter.Fill(getLabelDataSet);
                foreach (DataRow getLabelRow in getLabelDataSet.Tables[0].Rows)
                {
                    if (getLabelRow[0].ToString() == "http://www.w3.org/2000/01/rdf-schema#label")
                    {
                        labelText = getLabelRow[1].ToString();
                        break;
                    }
                }
            }
            catch
            {
            }
        }
        return labelText;
    }
```

Notice that the getLabelText method has been moved into this new class and is now called from the constructor. When the IRILabel is constructed the label text is found using the list of graphs provided to the constructor. The graph containing the label is noted. We need to alter ExtendedStringHandler so that the labels are the new IRILabel type and so that the correct information is supplied to the constructor.

4. Change the labelList member variable declaration so it looks like this:

```
List<IRILabel> labelList = new List<IRILabel>();
```

5. Each propertyLabel created must be the new IRILabel type so the line:

```
Label propertyLabel = new Label();
```

becomes:

```
IRILabel propertyLabel = new IRILabel(row[0], graphList, ParentConnection);
```

6. As getLabelText is now called by the IRILabel constructor we can remove the line:

```
propertyLabel.Text = getLabelText(row[0]);
```

from describeData in ExtendedStringHandler.

7. Now we modify the label_Click EventHandler so that it uses the graph information. Replace the existing method with:

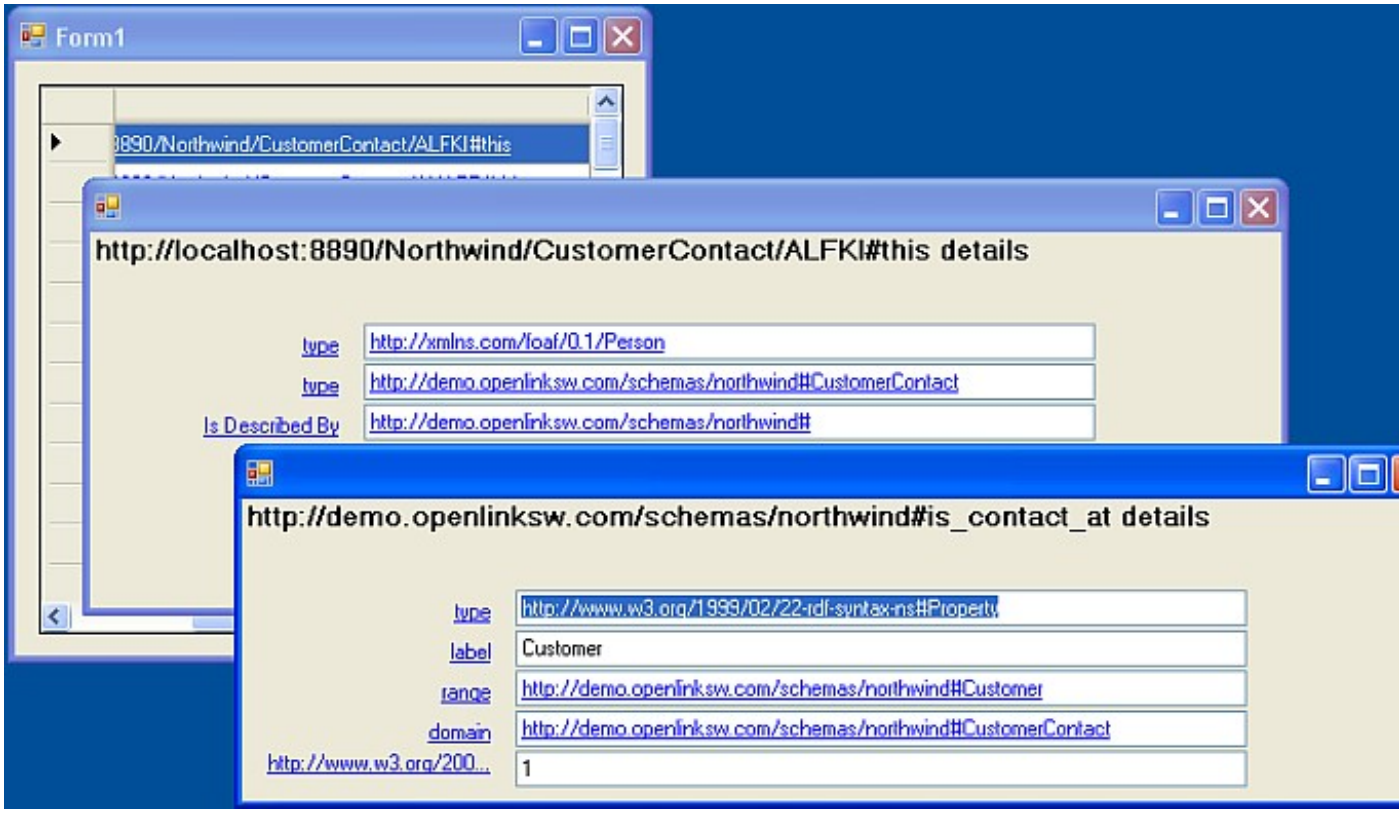
```
public void label_Click(object sender, EventArgs e)
  {
      int labelNum = 0;

      for (int i = 0; i < labelList.Count; i++)
      {
          if (sender == labelList[i])
          {
              labelNum = i;
              break;
          }
      }

      SqlExtendedString se = labelList[labelNum].iri;
      ExtendedStringHandler seHandler = new ExtendedStringHandler(se, ParentConnection, labelList[
          seHandler.displayData();

  }
```

8. Note that this method uses a new constructor for the ExtendedStringHandler that takes the graph as an additional argument. We need to add this new constructor.

```
public ExtendedStringHandler(SqlExtendedString iri, VirtuosoConnection parentConnection, String gr
  {
      ParentConnection = parentConnection;
      if (iri.IriType == SqlExtendedStringType.IRI)
      {
          ParentIRI = iri;
          isIRI = true;
          DescribeCommandSimple = new StringBuilder("sparql select * from <http://example.com/Nort
          DescribeCommandGeneral = new StringBuilder("sparql define get:soft " + '"'.ToString() +
      }
  }
```

   This new constructor uses the supplied graph to build the alternative sparql select statement that looks for the details about the supplied IRI. With these changes in place the application will find the description of the Northwind properties.
9. Build and run. As you explore the data you will see that you can find descriptions of the properties used to describe the entities in the Northwind dataset:

Figure 2.159. Northwind dataset



2.10. Creating a Web Browser Application to Access RDF Data Using The Virtuoso ADO.Net Provider

This section will guide you through creating first a Web Service that exposes RDF data from Virtuoso and then a simple web browser application that consumes the Web Service and allowing you to access and explore the RDF data by clicking on dereferenceable IRIs .

2.10.1. Pre-requisites

1. The example assumes that you have a local Virtuoso server with the Northwind demo database installed. If the demo database is not already installed then download the demo database VAD package (demo_dav.vad) and install it. The VAD package will create a new database in Virtuoso called demo containing the familiar Northwind tables. It will also creates Linked Data Views of the Northwind tables. In the example we assume the database is accessible on a hostname

of "demo.openlinksw.com" on the default port 80, where an actually live instance of the Virtuoso Demo database is hosted. Users would use the appropriate hostname and port number of their Virtuoso installation to create the sample application, and would be would be example.com for a default installation or whatever the URIQA DefaultHost Virtuoso configuration parameter is set to when the demo database VAD package is installed.

2. The Virtuoso ADO.Net provider for .Net 3.5 and the Entity Framework.
3. Microsoft Visual Studio 2008
4. The Virtuoso Cartridges VAD package .

2.10.2. Creating the Web Service

Step 1 - Create a view of the RDF data

To create a view of the customers in the Northwind first open the Virtuoso Conductor and log in as dba. Then open iSQL from the menu on the left and execute the following statement.

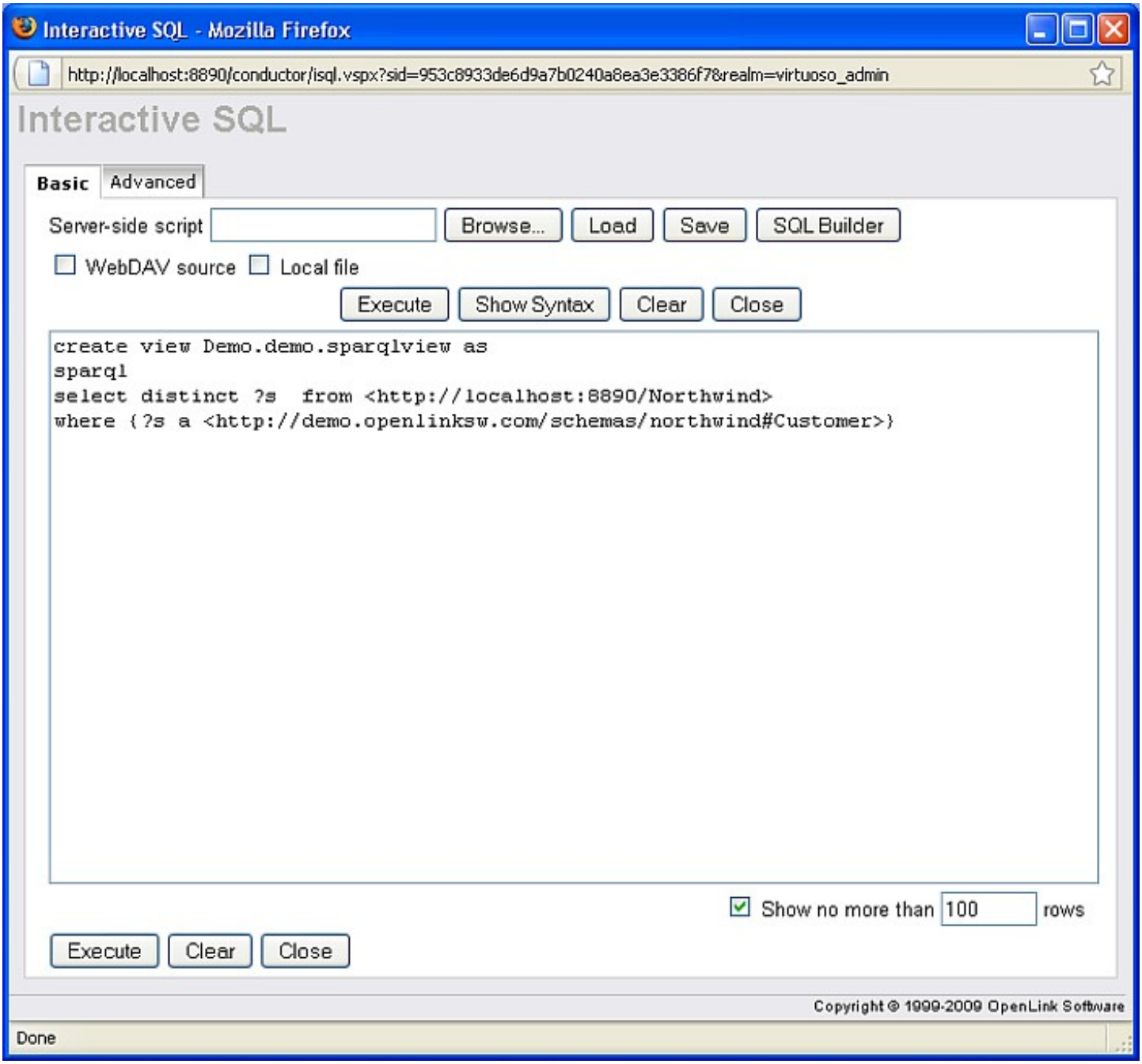
```
create view Demo.demo.sparqlview as
sparql
select distinct ?s  from <http://example.com/Northwind>
where {?s a <http://demo.openlinksw.com/schemas/northwind#Customer>}
```

Note:

- If the view is added to the Visual Studio project as user "demo" (or any other than "dba'), then it must be ensured that the "SPARQL_SELECT" and "SPARQL_SPONGE" roles are assigned to this user, which can be done via the Virtuoso Conductor in the "System Admin" -> "User Accounts" tab.
- Execute permissions will also need to be granted to the following procedure:

```
grant execute on DB.DBA.RDF_MAKE_LONG_OF_SQLVAL to "demo"
```

Figure 2.160. create a view



Step 2 - Create the Visual Studio Project and Add the Model

1. Open

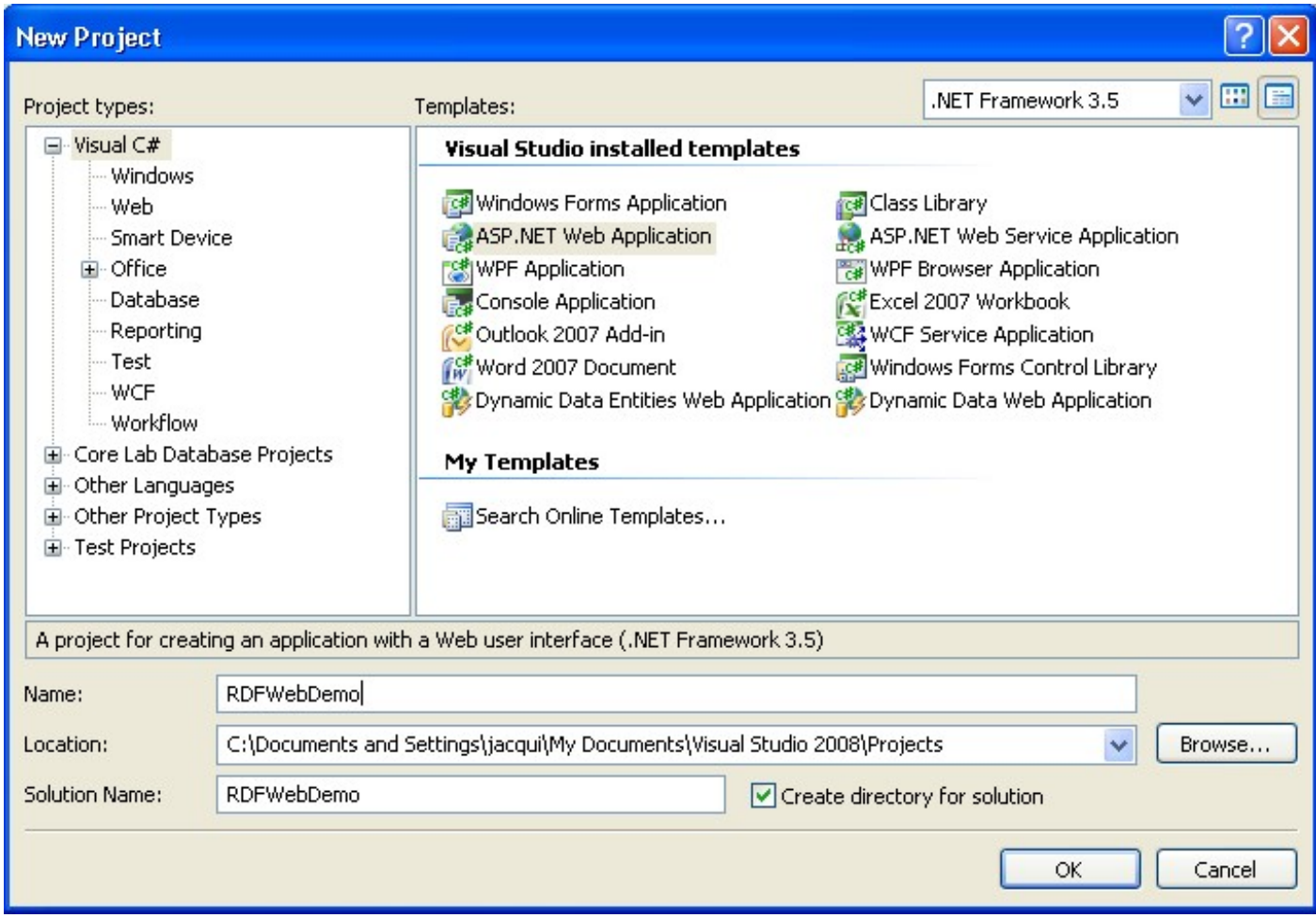
   Visual Studio

   and create a new

   ASP .NET Web Application

   called RDFWebDemo.

   Figure 2.161. create new application



2. Right click RDFWebDemo in the

   Solution Explorer

   and add a new

   ADO.NET Entity Data Model

   called Model1.edmx. This will open the Entity Data Model Wizard.
3. Choose

   Generate From Database

   and click

   Next

   .
4. Set up a connection to the Demo database on your local Virtuoso Server, select

   Yes, include the sensitive data in the connection string

   and set the name of the entities to DemoEntities. Click

   Next

   .
5. On the

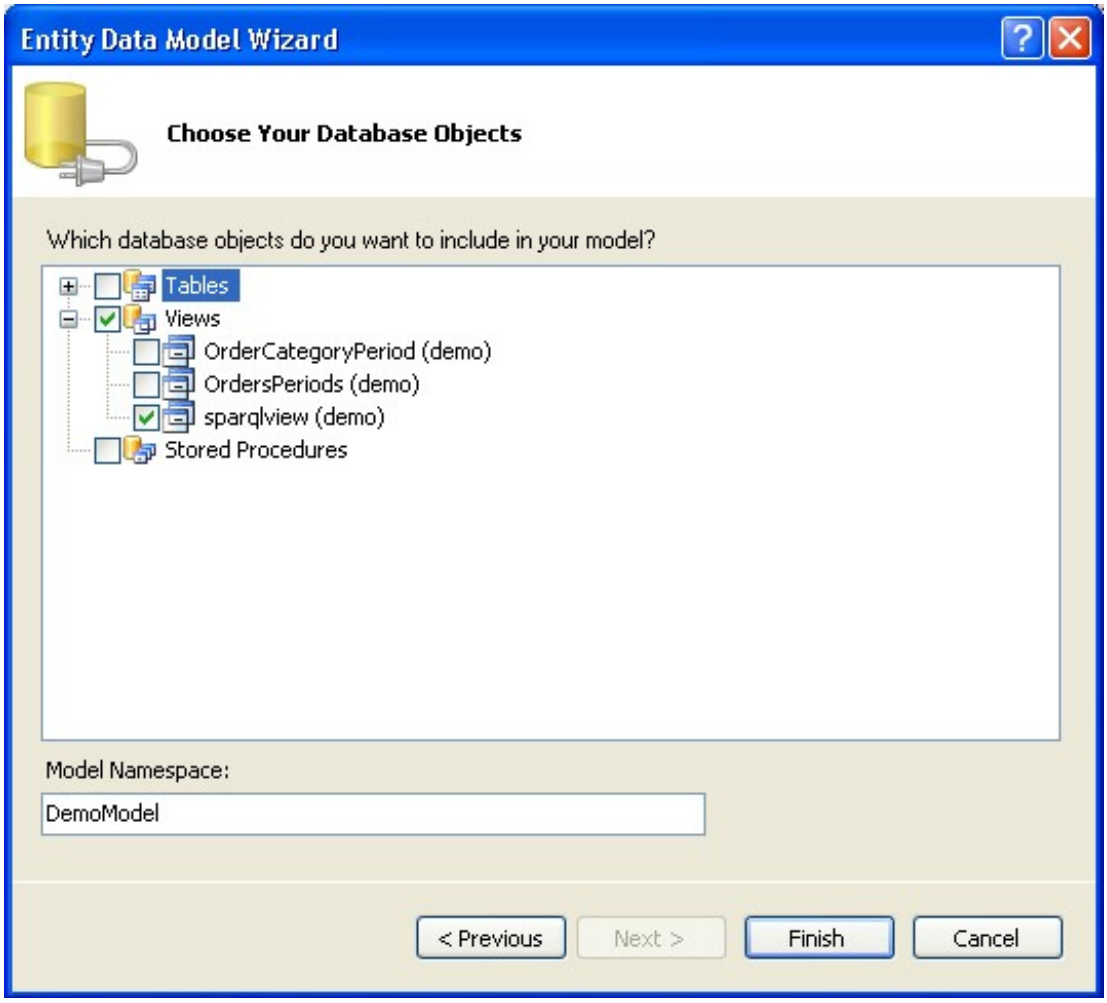
   Choose Your Database Objects

page expand

Views

and select sparqlview. Check that the Model Namespace is DemoModel and click Finish.

Figure 2.162. Model Namespace



Step 3 - Add the Web Service

1. Right click RDFWebDemo in the

   Solution Explorer

   and add a new

   ADO.NET Data Service

   called WebDataService1.svc. Click

   Add

   .
2. In the class definition of WebDataService1 in the newly created file WebDataService1.svc.cs replace /* TODO: put your data source class name here */ with the name of our model, DemoEntities.

   ```
   public class WebDataService1 : DataService<DemoEntities>
   ```
3. In the InitializeService method add the line:

```
config.SetEntitySetAccessRule("*", EntitySetRights.All);
```

The method should look like this:

```
public static void InitializeService(IDataServiceConfiguration config)
  {
      // TODO: set rules to indicate which entity sets and service operations are visible, updata
      // Examples:
      // config.SetEntitySetAccessRule("MyEntityset", EntitySetRights.AllRead);
      // config.SetServiceOperationAccessRule("MyServiceOperation", ServiceOperationRights.All);

      config.SetEntitySetAccessRule("*", EntitySetRights.All);
  }
```

Step 4 - Compile and Run

Hit *F5* to compile and run the service. Select *OK* when prompted to enable debugging. The default browser will be launched showing a page like:

```
  <?xml version="1.0" encoding="utf-8" standalone="yes" ?>
- <service xml:base="http://example.com/WebDataService1.svc/" xmlns:atom="http://www.w3.org/2005/Atom" xm
- <workspace>
  <atom:title>Default</atom:title>
- <collection href="sparqlview">
  <atom:title>sparqlview</atom:title>
  </collection>
  </workspace>
  </service>
```

The service is now running.

Note the address on which the service is made available. You will need to know this when creating the app to consume the service. Look in the Address Bar of the browser. It will be something like: http://example.com/WebDataService1.svc/

2.10.3. Creating the Browser Application

Step 1 - Create the Visual Studio Project.

1. Open

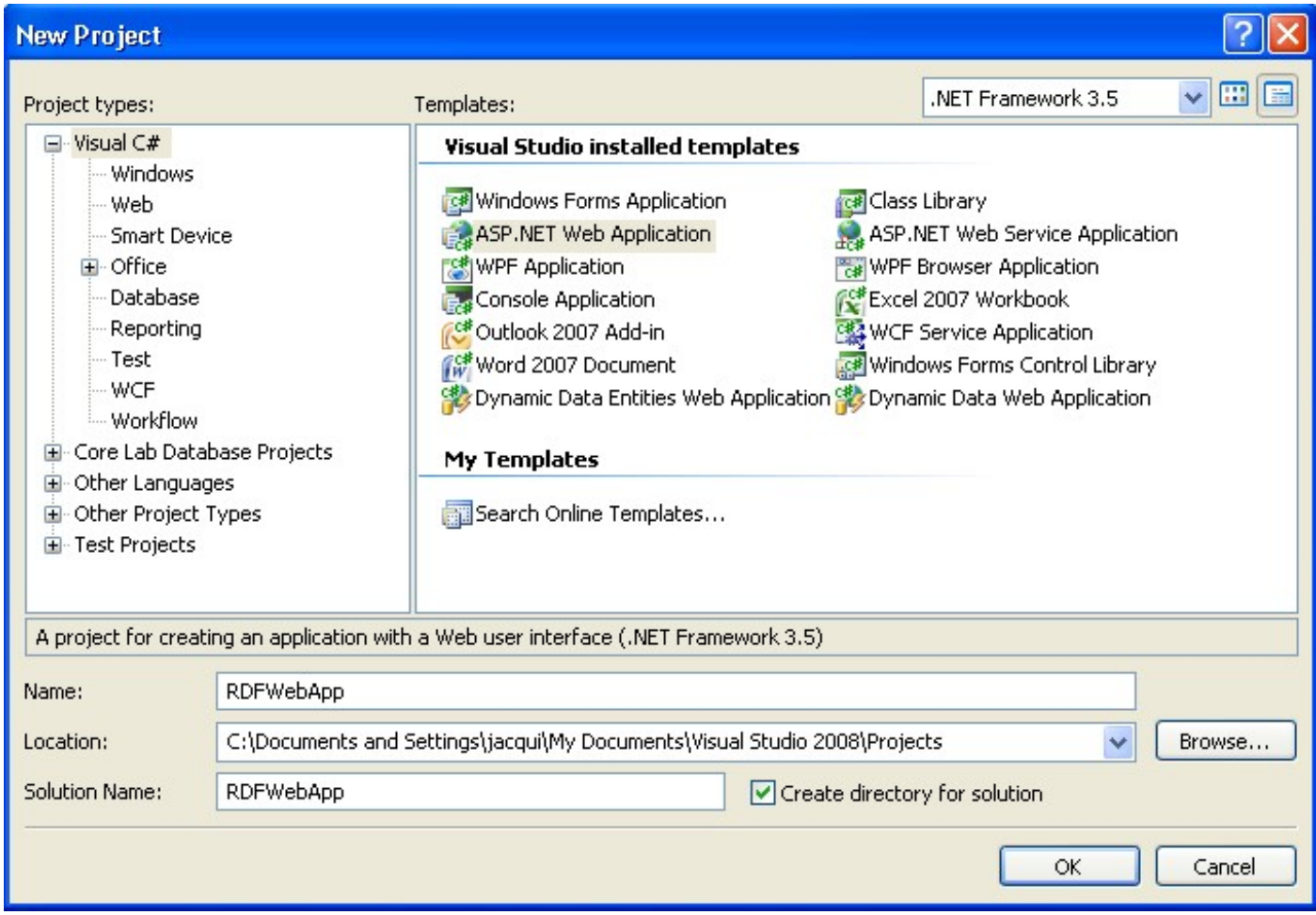
   Visual Studio

   and create a new

   ASP.NET Web Application

   called RDFWebApp.

   Figure 2.163. New Web Application



2. Create client side entities with datasvcutil.exe

    - Open a command prompt.
    - Navigate to *C:\WINDOWS\Microsoft.NET\Framework\v3.5*.
    - Generate the client classes using the following command:

        ```
        datasvcutil.exe /uri:http://example.com/WebDataService1.svc /out:DemoEntities.cs
        ```

        Note the address of the service - you may need to change the port number to match the one seen in the address at the end of Step 4 in Creating the Web Service.
3. Add the classes to RDFWebApp.

    - Right click RDFWebApp
    - Choose to add an existing item and add c:\WINDOWS\Microsoft.NET\Framework\v3.5\DemoEntities.cs.
4. Add a reference to System.Data.Services.Client to the project.

Step 2 - Display the contents of sparqlview as a table on the page

To display the RDF data on the web page we create a table with a row for each item in sparqlview. We then use each IRI from sparqlview to create a hyperlink. The hyperlinks are displayed in the table cells. To do this add the following block of code to the page_load method in Default.aspx.cs.

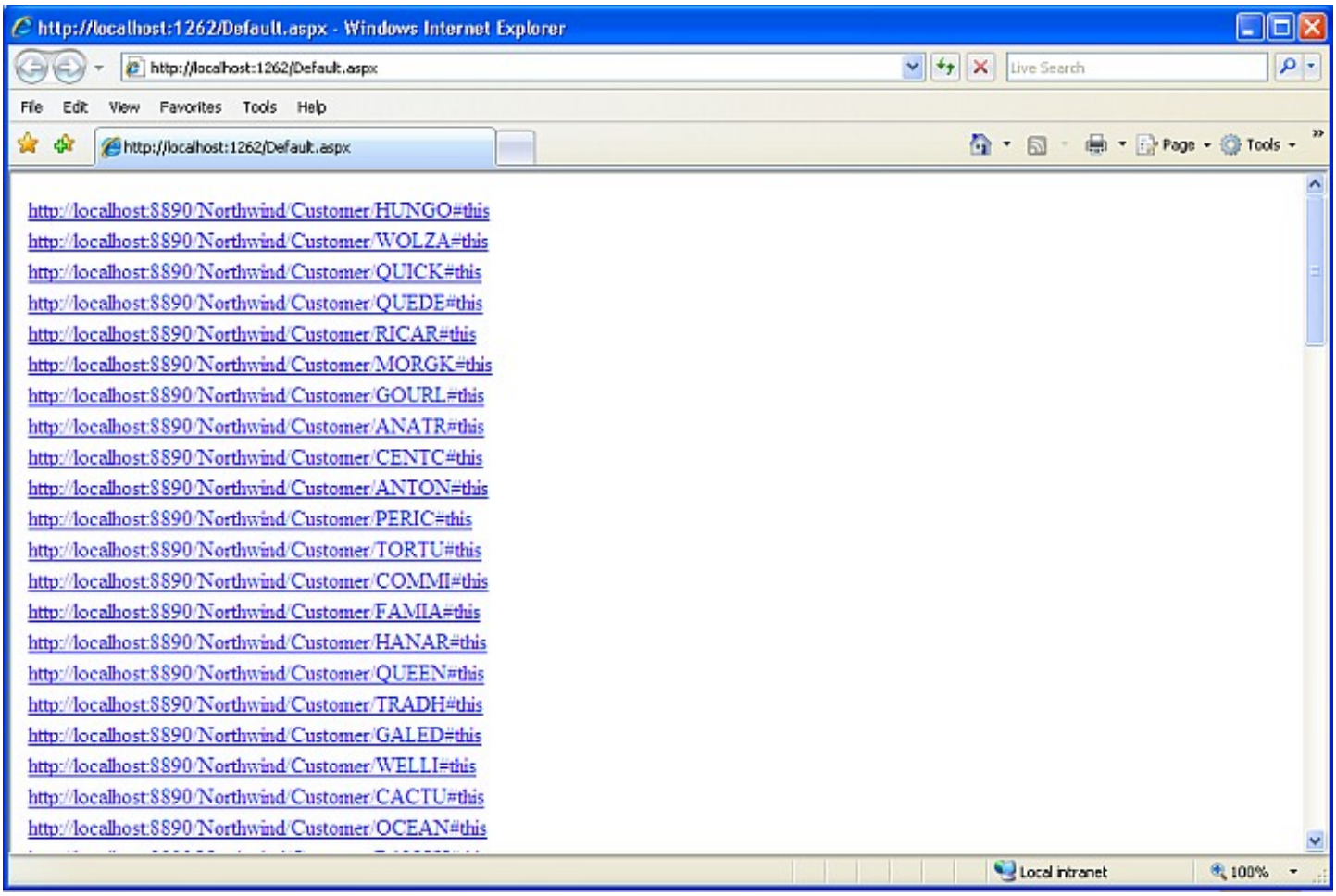
```
DemoModel.DemoEntities svc = new DemoModel.DemoEntities(new Uri("http://example.com/WebDataService1.svc"

 var query = svc.sparqlview;
 Table iriTable = new Table();
 this.Controls.Add(iriTable);

 foreach (DemoModel.sparqlview sv in query)
 {
     TableRow tRow = new TableRow();
     iriTable.Rows.Add(tRow);
```

```
        TableCell tCell = new TableCell();
        tRow.Cells.Add(tCell);
        HyperLink h = new HyperLink();
        h.Text = sv.s;
        h.NavigateUrl = sv.s;
        tCell.Controls.Add(h);
    }
```

Note the address of the service in the first line - you may need to change the port number to match the one seen in the address at the end of Step 4 in Creating the Web Service.

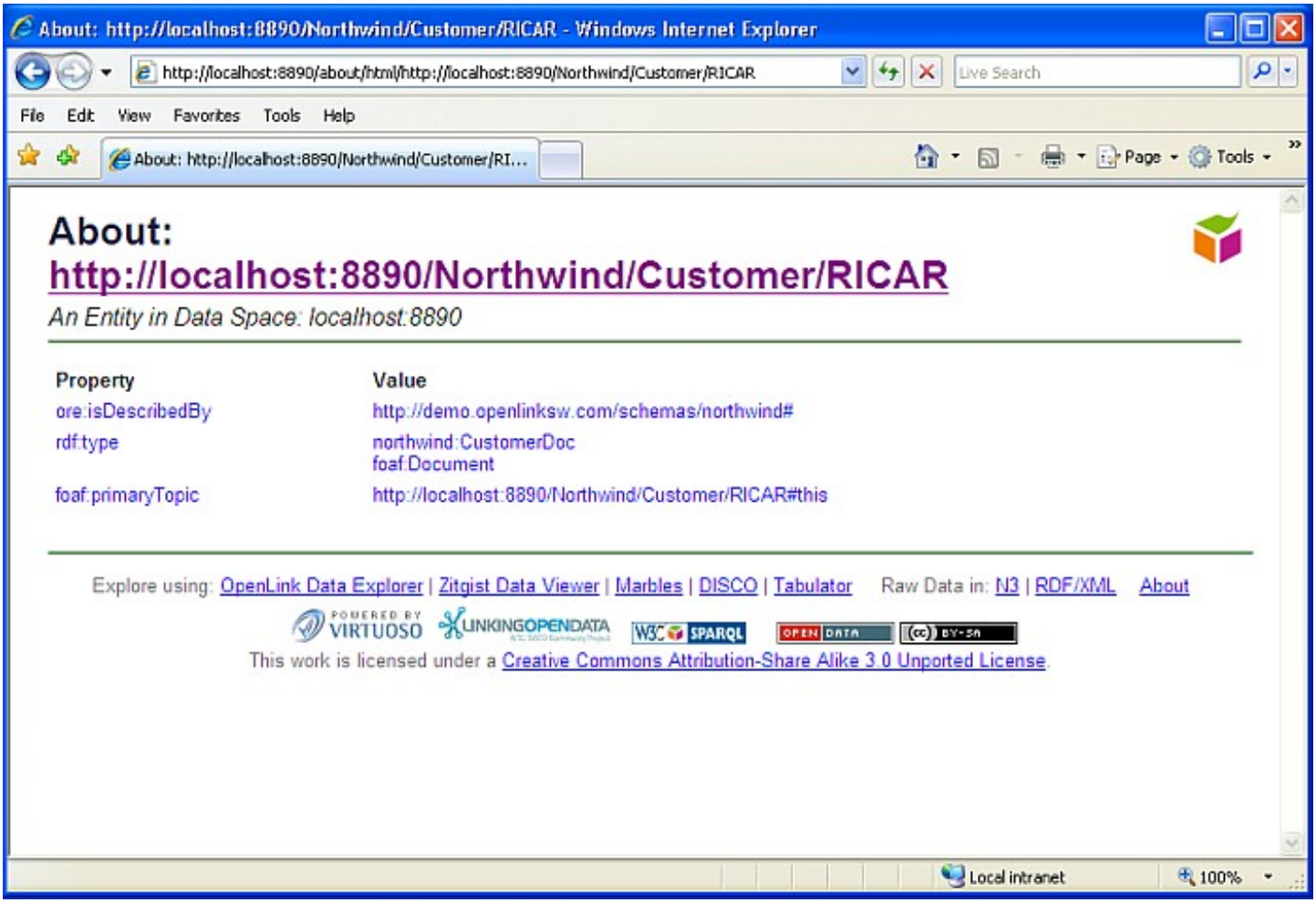
Compile and run RDFWebApp (ensuring that the service created above is still running). This will launch a browser and display the IRIs from sparqlview as a list of hyperlinks.

Figure 2.164. list of hyperlinks



With the Cartridges VAD package installed in Virtuoso, clicking on these links will take you to a description page of the referenced resource. The description page is created using description.vsp .

Figure 2.165. Description page



2.10.4. Deploy With IIS

To create and test this simple Web Service we have used the Visual Studio Development Server. This section describes how to deploy the service using IIS.

Web Service

To deploy the service using IIS:

- Open the RDFWebDemo project in Visual Studio, go to the

  Project

  menu and select RDFWebDemo

  Properties

  .
- Select the

  Web

  tab and scroll down to the

  Servers

  section. Select

  Use local IIS Server

. The project URL defaults to using localhost.
- Click the

  Create Virtual Directory

  button then check that the Service works on localhost.
- Build the project then start without debugging (ctrl F5).

The start page that you see when you test the service will look the same as before but the address in the browser bar will be something like http://example.com/RDFWebDemo1/WebDataService1.svc/. You can now access your service remotely using the hostname or IP address of your server.

If at this point you get an Access is denied error, 401.3, then you will need to add the Internet Guest Account (IUSR_XXX where XXX is your computer name) to the users allowed to access the folder containing the RDFWebDemo project.

Web Application

You will now need to modify RDFWebApp to access the service at the new address. At the same time we will also change RDFWebApp so that it too is deployed using IIS

- Open The RDFWebApp project in Visual Studio.
- Go to the

  Project

  menu and select RDFWebApp

  Properties

  .
- Select the

  Web

  tab and scroll down to the

  Servers

  section. Select

  Use local IIS Server

  . The project URL defaults to using localhost.
- Click the

  Create Virtual Directory

  button. The web application will then run on the local IIS.

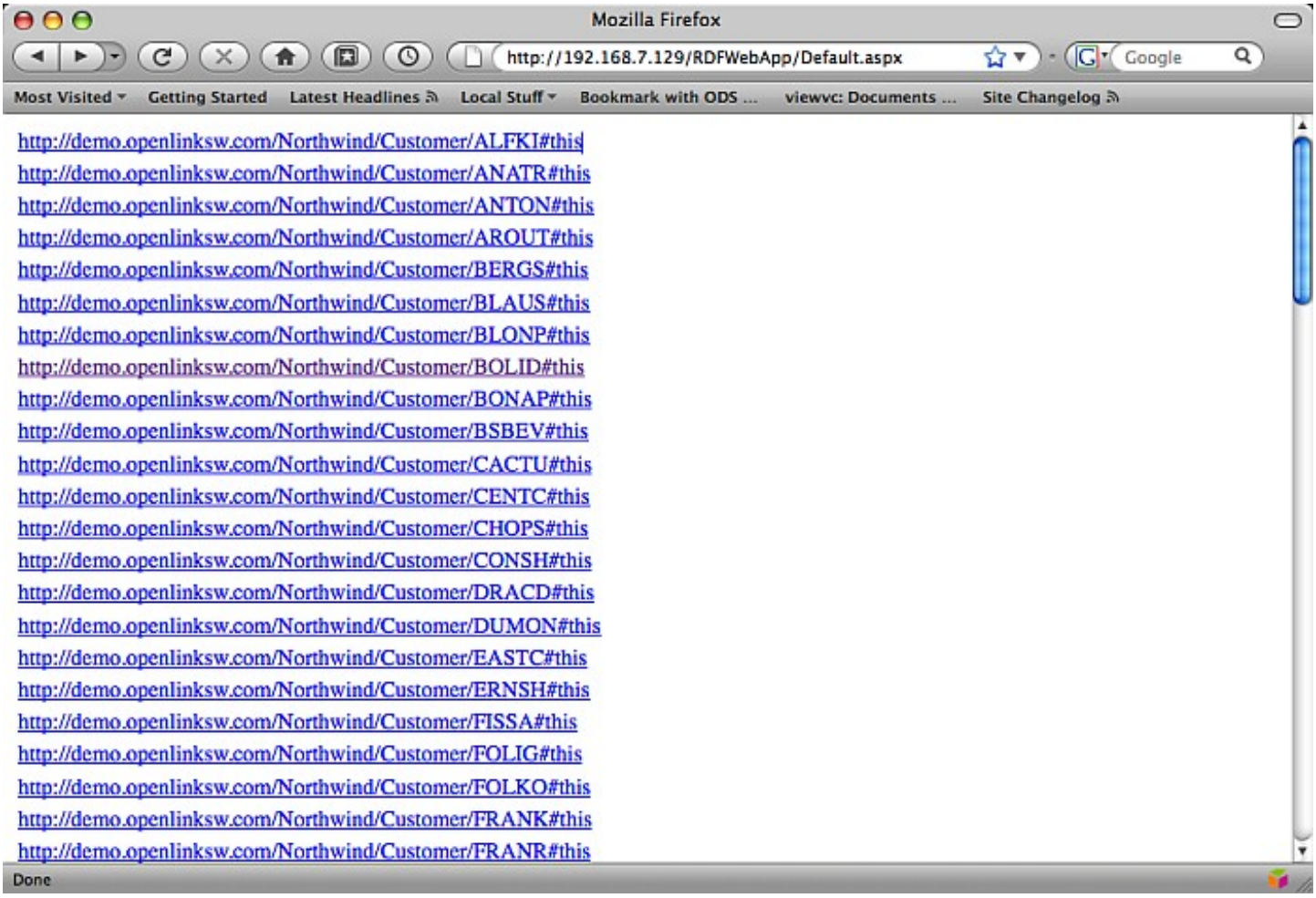
  ```
  DemoModel.DemoEntities svc = new DemoModel.DemoEntities(new Uri("http://example.com/WebDataService
  ```

  to

  ```
  DemoModel.DemoEntities svc = new DemoModel.DemoEntities(new Uri("http://example.com/RDFWebDemo/Web
  ```
- To reference the web service running on IIS you will need to edit Default.aspx.cs. Change
- Build the project then start without debugging (ctrl F5).

The web application is accessible on http://example.com/RDFWebApp/Default.aspx and can also be accessed using the hostname or IP address of you server e.g. http://192.168.7.129/RDFWebApp/Default.aspx

Figure 2.166. Default.aspx



Next Steps

The next example shows you how to quickly create an ADO.Net Data Service that exposes RDF data in Virtuoso and how to create a basic Web application to consume that service. The next step is to create a Silverlight Application to consume the same service .

2.11. Creating a Silverlight Application to consume the service

This section will guide you through creating an application for Silverlight that will consume the ADO.Net Data Service created in Creating a Web Browser Application to Access RDF Data Using The Virtuoso ADO.Net Provider.

2.11.1. Pre-requisites

1. The Microsoft Silverlight 2 Tools for Visual Studio 2008 SP1
2. The ADO.Net Data Service created in Creating a Web Browser Application to Access RDF Data Using The Virtuoso ADO.Net Provider.
3. The Visual Studio project used to create the ADO.Net Data Service.

2.11.2. Creating the Application for Silverlight.

1. Open the ADO.Net Data Service project in

   Visual Studio
2. In the

   Solution Explorer

   right click on the RDFWebDemo solution and add a new Project.
3. In the

Add New Project

dialog select

Silverlight Application

and click

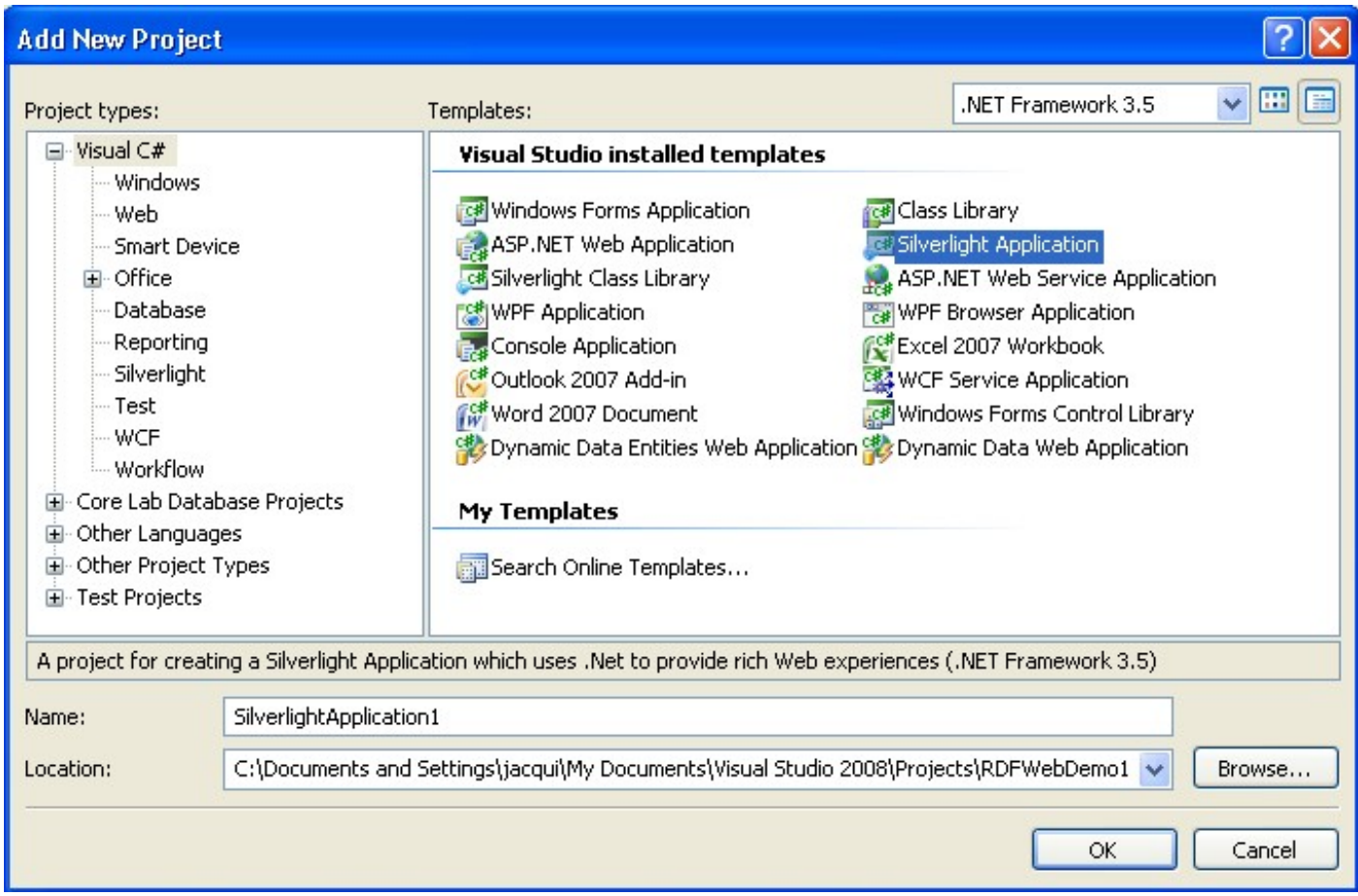
OK

. This will open the

Add Silverlight Application

dialog.

Figure 2.167. Add Silverlight Application



4. Select

   Link this Silverlight Control into an existing Web Site

   and make sure the Web Site selected is RDFWebDemo. Select

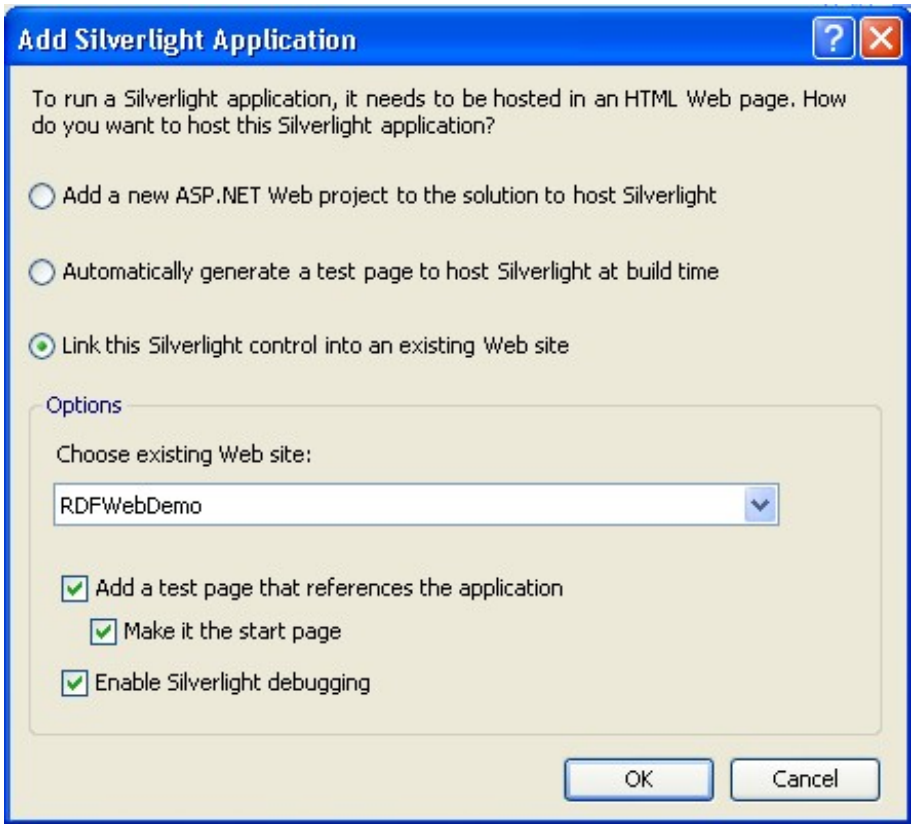
   Add a test page that references the application

   and

   Make it the start page

   .

Figure 2.168. Add a test page



5. In Solution Explorer, select RDFWebDemo, open the Project menu and select Properties.
6. Select the

   Web

   tab, and select

   Specific Page

   in the

   Start Action

   section. Click on the ellipsis and select SilverlightApplication1TestPage.html as the start page.
7. Add a reference to the data service. In the

   Solution Explorer

   right click on SilverlightApplication1 and select

   Add Service Reference
8. In the

   Add Service Reference

   dialog click the

   Discover

   button. Your ADO.Net Data Service should appear in the Address box and the Services box.
9. Select the service and click

OK

. ServiceReference1 will now be added to the ServiceReferences.

10. Open page.xaml.cs and add references to the service and to the System.Data.Services.Client assembly by adding the following using statements at the top of the file:

```
using System.Data.Services.Client;
using SilverlightApplication1.ServiceReference1;
```

11. We need at create a data service context to reference the data from the service and to load data from the view, sparqlview, exposed by the service. Add the following lines to the page constructor after InitializeComponent()

```
DataServiceContext svcCtx = new DataServiceContext(new Uri("WebDataService1.svc", UriKind.Relative
svcCtx.BeginExecute<sparqlview>(new Uri("sparqlview", UriKind.Relative), loadSCallback, svcCtx);
```

12. Add the loadSCallback method to the page class. The method loads the data from sparqlview and puts it in a List. This List populates a list box on the page.

```
private void loadSCallback(IAsyncResult asyncResult)
{   List<Uri> uList = new List<Uri>();
    DataServiceContext ctx = asyncResult.AsyncState as DataServiceContext;
    foreach (sparqlview sv in ctx.EndExecute<sparqlview>(asyncResult))
     uList.Add(new Uri(sv.s));
    listBox1.DataContext = uList;
}
```

13. Add the list box to the page. In the

Solution Explorer

double click on page.xaml to open it in the editor. Add the following code between the <grid> and </grid> tags.

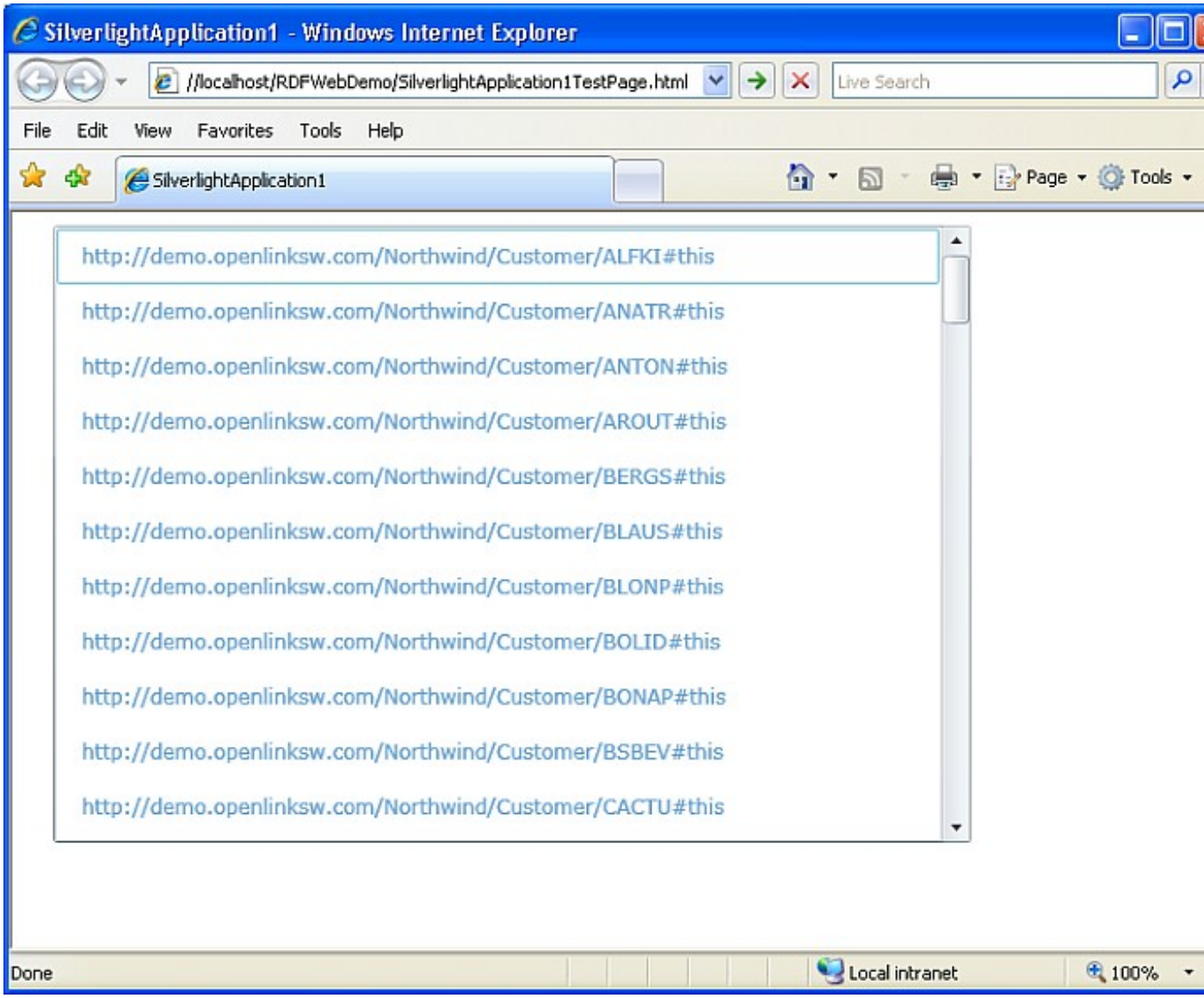
```
<ListBox x:Name="listBox1"
    HorizontalAlignment="Stretch"
    Margin="25,8,26,-78" Grid.RowSpan="1"
    Grid.Row="0" VerticalAlignment="Stretch"
    ItemsSource="{Binding Mode=OneWay}" >
    <ListBox.ItemTemplate>
        <DataTemplate>
            <StackPanel x:Name="DisplayListData"
             Orientation="Horizontal"
             VerticalAlignment="Bottom"
             Margin="5" >
                <HyperlinkButton
                 Content ="{Binding}"
                 NavigateUri="{Binding}"
                 Margin="5,0,0,0"
                 VerticalAlignment="Bottom"
                 HorizontalAlignment="Left"
                 FontSize="12">
                </HyperlinkButton>
            </StackPanel>
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
```

14. Build the Silverlight application and launch without debugging using

Ctrl F5

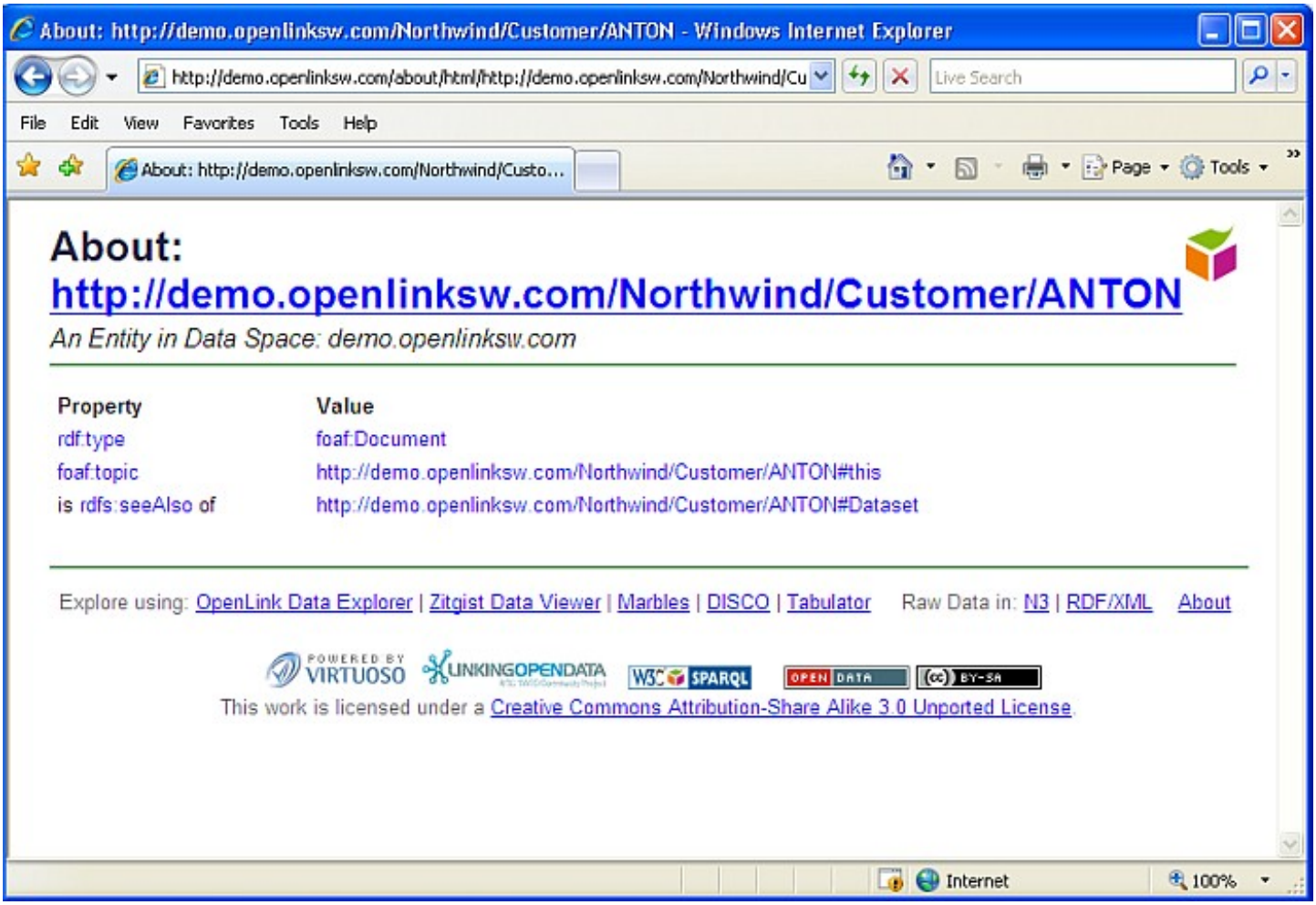
. This will launch the browser and open SilverlightApplication1TestPage.aspx.

Figure 2.169. SilverlightApplication1TestPage.aspx



Clicking on one of the IRIs will open the page using description.vsp.

Figure 2.170. using description.vsp



2.12. Creating A Simple .NET RIA Services Application To Display Data From Virtuoso

.NET RIA Services is a new framework from Microsoft that simplifies the development of n-tier web applications. A typical application will consist of a presentation layer, application logic and a data access layer. In these examples a combination of Silverlight 3 and .NET RIA Services will provide the presentation layer and application layer while the data will reside in Virtuoso and be accessed using the Virtuoso ADO.NET provider. The first example demonstrates how to display data from the Employee table in the Demo database in a grid on a web page. The second example shows how to display IRIs from the Linked Data Views of the demo data. The IRIs are used to create hyperlinks that are the starting point for exploring the linked data.

2.12.1. Pre-requisites

1. The example assumes that you have a local Virtuoso server with the Northwind demo database installed. If the demo database is not already installed then download the demo database VAD package (demo_dav.vad) and install it. The VAD package will create a new database in Virtuoso called demo containing the familiar Northwind tables. It will also creates Linked Data Views of the Northwind tables. In the example we assume the database is accessible on a hostname of "demo.openlinksw.com" on the default port 80, where an actually live instance of the Virtuoso Demo database is hosted. Users would use the appropriate hostname and port number of their Virtuoso installation to create the sample application, and would be would be example.com for a default installation or whatever the URIQA DefaultHost Virtuoso configuration parameter is set to when the demo database VAD package is installed.
2. The Virtuoso ADO.Net provider for .Net 3.5 and the Entity Framework.
3. Microsoft Visual Studio 2008
4. The Virtuoso Cartridges VAD package .
5. Silverlight 3 Tools for Visual Studio 2008 SP1 .
6. Microsoft .NET RIA Services July 2009 Preview .

2.12.2. Creating the Application

Step 1 - Create the Visual Studio Projects.

1. Open

   Visual Studio

   and create a new

   Silverlight Application

   project. Call the project DemoApplication.
2. In the New Silverlight Application dialog ensure that

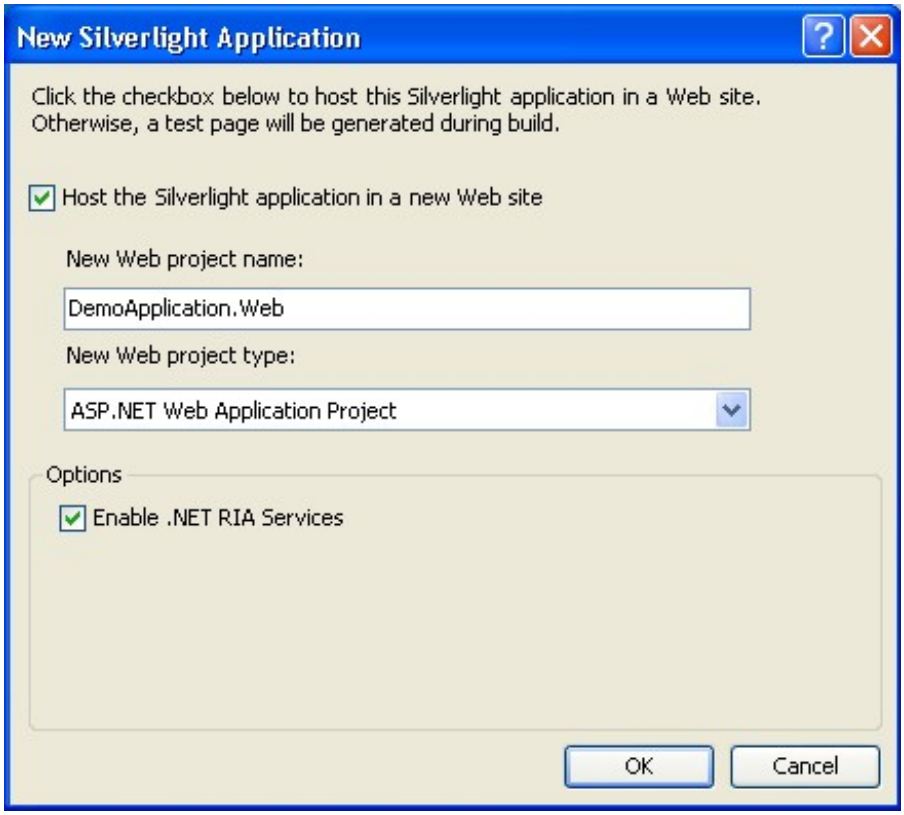
   Enable .NET RIA Services

   is checked. Click the

   OK

   button.

   Figure 2.171. .NET RIA Services Application



At this point a skeleton solution is created that consists of a client project called DemoApplication and a server project called DemoApplication.Web. This application will use data from the Virtuoso database. We add the data and its schema to the application by adding an ADO.NET entity data model to the server project.

Step 2 - Add the Data Model

1. Right click the server project in the

   Solution Explorer

and

Add New Item

. In the dialog box select

ADO.NET Entity Data Model

and call it demo.edmx. Click the

Add

button. This will open the

Entity Data Model Wizard

.

2. Choose

   Generate From Database

   and click

   Next

   .
3. Set up a connection to the Demo database on your local Virtuoso Server, select

   Yes, include the sensitive data in the connection string

   and set the name of the entities to DemoEntities. Click

   Next

   .
4. On the

   Choose Your Database Objects

   page expand

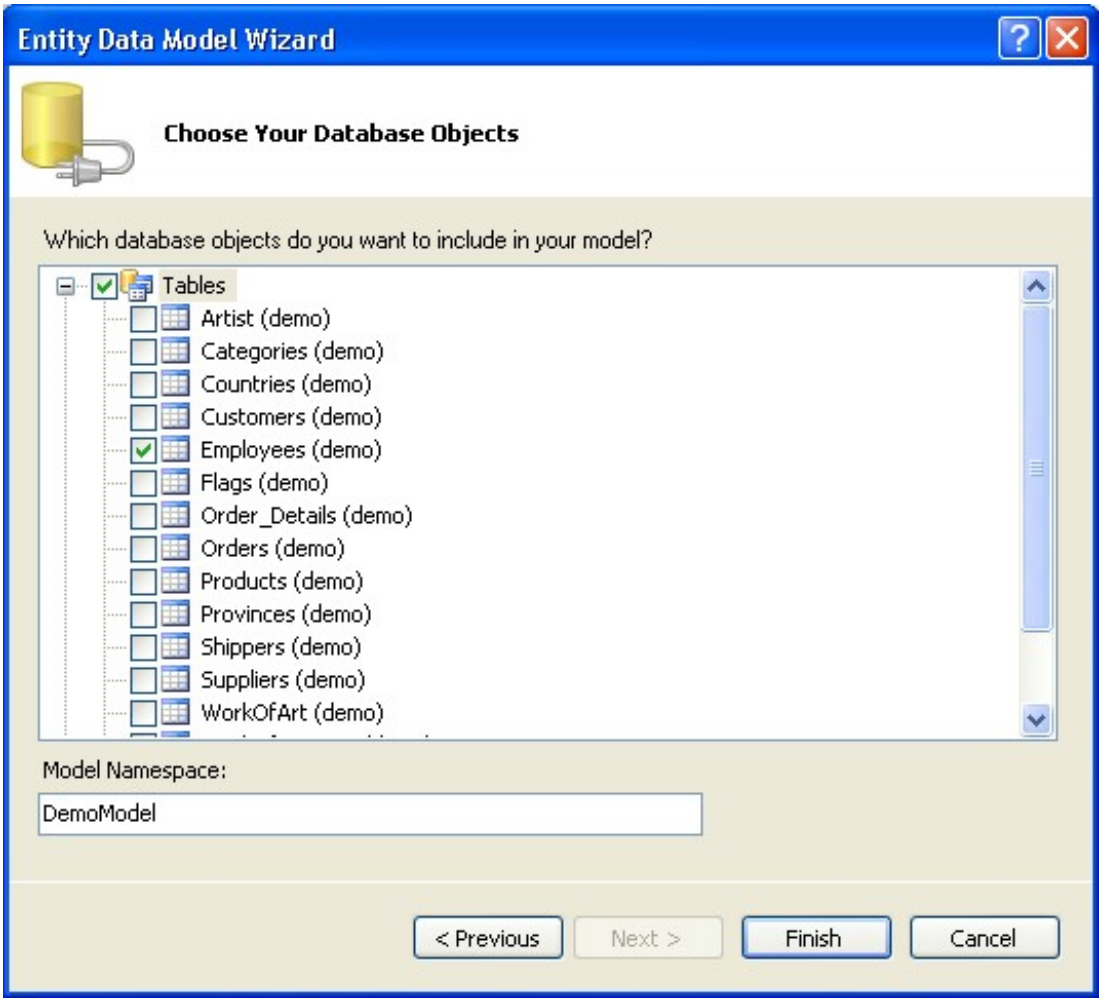
   Tables

   and select Employees. Check that the Model Namespace is DemoModel and click

   Finish

   .

Figure 2.172. .NET RIA Services Application



We want to make the entities in the model available to both the client and server parts of the solution. To do this we need to add a DomainService to the solution. However, to make the entities from the data model available to the domain service we must first build the solution.

Step 3 - Add a Domain Service.

1. First build the solution.
2. Right click the server project in the

   Solution Explorer

   and

   Add New Item

   . In the dialog box choose

   Domain Service Class

   from the Templates pane and call it EmployeeService.cs. Click

   Add

   . This will open the

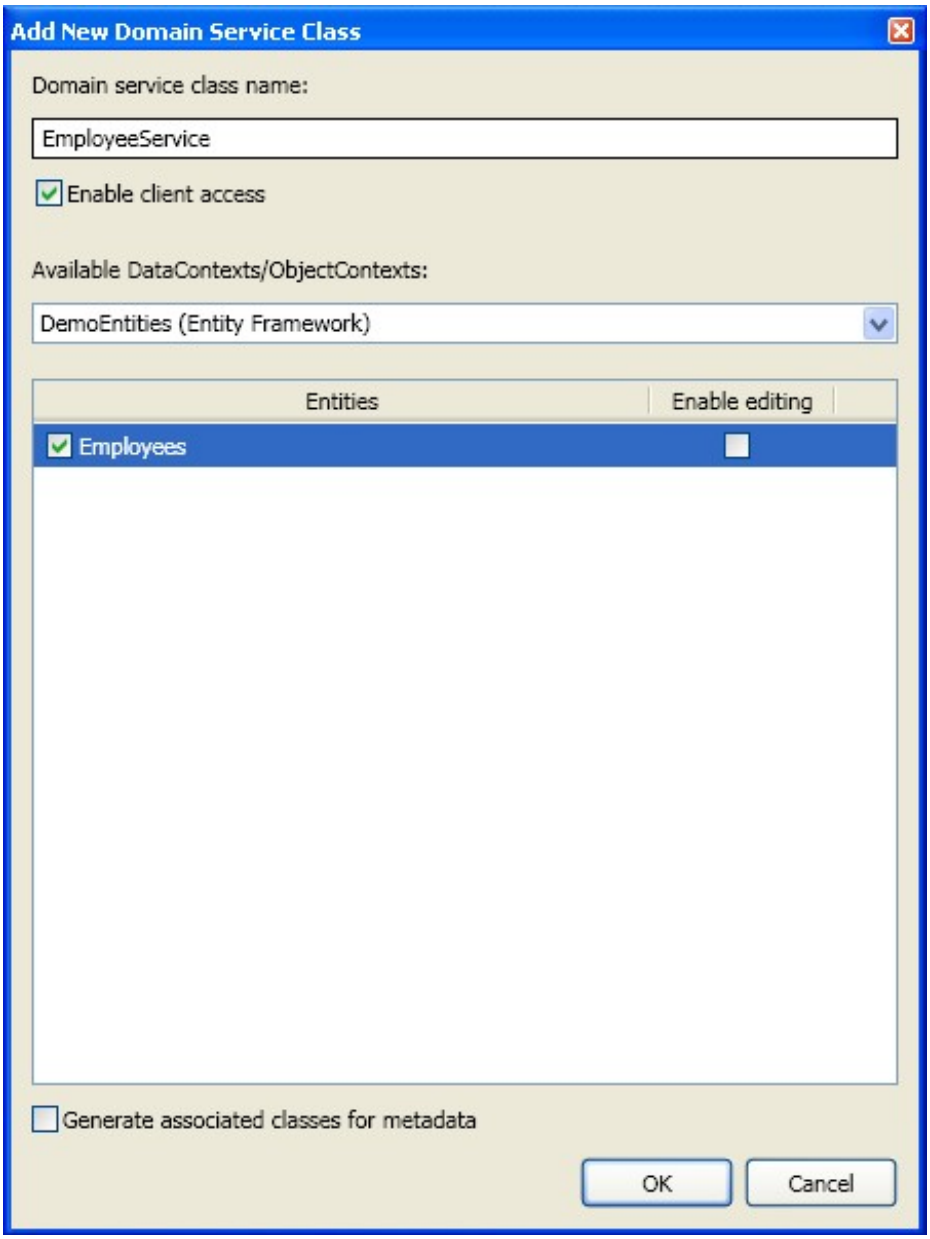
   Add New Domain Service Class

   dialog.
3. The entities from the model we have just added to the project are listed under

Entities

. Tick the box next to Employees. and click OK.

Figure 2.173. .NET RIA Services Application



This will create the DomainService class and generated code in both the client and server parts of the application. The Silverlight client can now interact with the data through the DomainContext class in the client project. At this point you need to build the solution again.

Step 4 - Display The Data

1. From the

   Silverlight XAML Controls

   in the

   Toolbox

   drag a

DataGrid

between the <Grid> </Grid> tags on MainPage.xaml in the client. Call the grid EmployeeGrid.

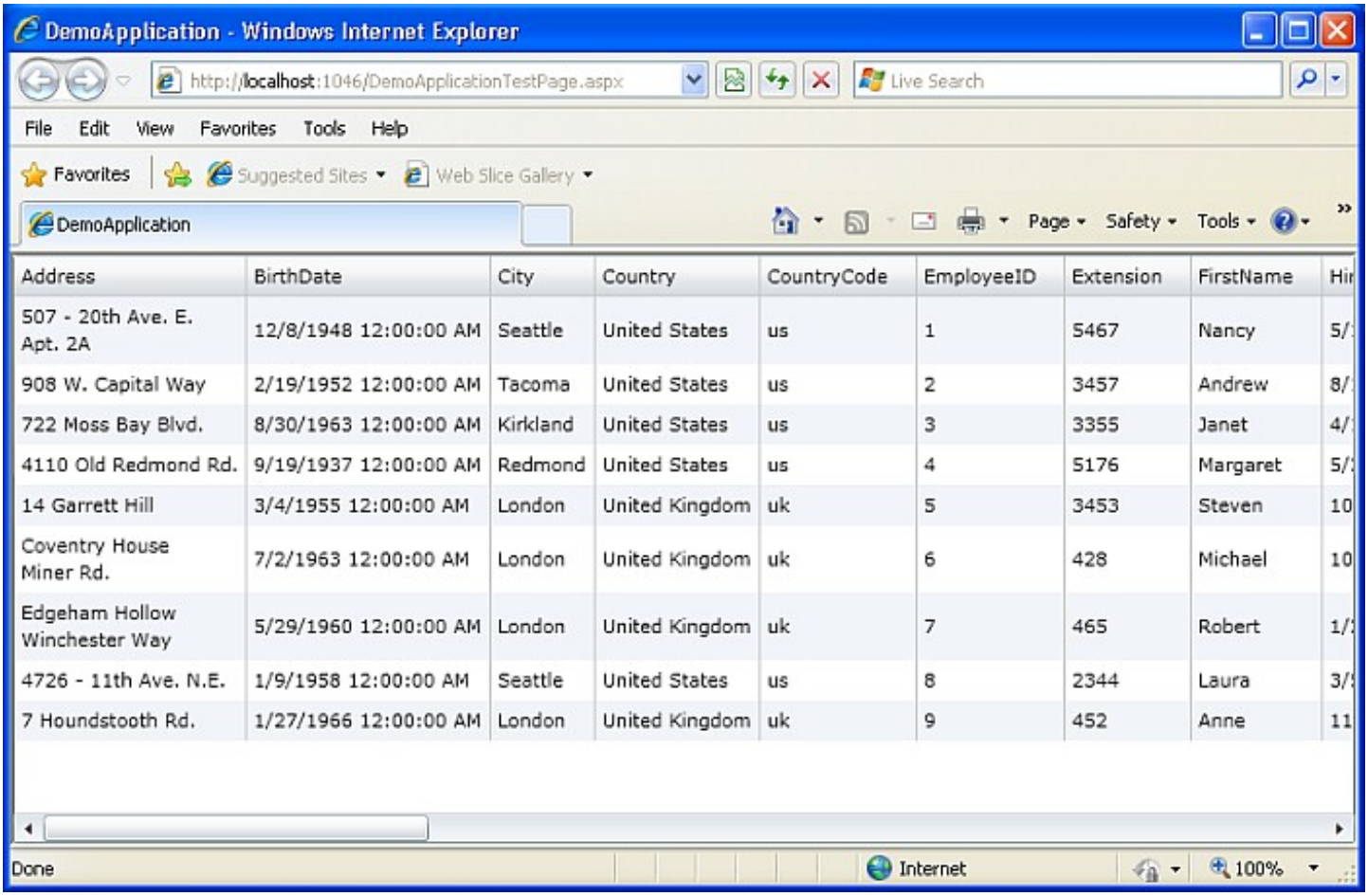
```
<data:DataGrid Name="EmployeeGrid"></data:DataGrid>
```

2. Instantiate the DomainContext to get the list of employees and add them to the grid by adding code to MainPage.xaml.cs so it looks like this:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using DemoApplication.Web;
using System.Windows.Ria.Data;

namespace DemoApplication
{
    public partial class MainPage : UserControl
    {
        private EmployeeContext _employeeContext = new EmployeeContext();
        public MainPage()
        {
            InitializeComponent();
            LoadOperation<Employees> LoadOp =
                this._employeeContext.Load(this._employeeContext.GetEmployeesQuery());
            this.EmployeeGrid.ItemsSource = LoadOp.Entities;
        }
    }
}
```

3. Build and run the application. Internet Explorer will be launched and you will see the data displayed on the page as a grid.

Figure 2.174. .NET RIA Services Application



Address	BirthDate	City	Country	CountryCode	EmployeeID	Extension	FirstName	Hir
507 - 20th Ave. E. Apt. 2A	12/8/1948 12:00:00 AM	Seattle	United States	us	1	5467	Nancy	5/
908 W. Capital Way	2/19/1952 12:00:00 AM	Tacoma	United States	us	2	3457	Andrew	8/
722 Moss Bay Blvd.	8/30/1963 12:00:00 AM	Kirkland	United States	us	3	3355	Janet	4/
4110 Old Redmond Rd.	9/19/1937 12:00:00 AM	Redmond	United States	us	4	5176	Margaret	5/
14 Garrett Hill	3/4/1955 12:00:00 AM	London	United Kingdom	uk	5	3453	Steven	10
Coventry House Miner Rd.	7/2/1963 12:00:00 AM	London	United Kingdom	uk	6	428	Michael	10
Edgeham Hollow Winchester Way	5/29/1960 12:00:00 AM	London	United Kingdom	uk	7	465	Robert	1/
4726 - 11th Ave. N.E.	1/9/1958 12:00:00 AM	Seattle	United States	us	8	2344	Laura	3/
7 Houndstooth Rd.	1/27/1966 12:00:00 AM	London	United Kingdom	uk	9	452	Anne	11

2.12.3. Displaying RDF Data

One advantage of using Virtuoso as the data store is the seamless way in which we can use this use this application to display dereferencable IRIs and explore RDF linked data.

The Virtuoso SPASQL interface allows RDF data in the Virtuoso Quad store to be queried using SPARQL from any SQL interface by simply prefixing the SPARQL query with the keyword SPARQL. We will use the SPASQL interface to create a view containing the IRIs of the Employees in the the Demo RDF data in Virtuoso.

Step 1 - Create the View in Virtuoso

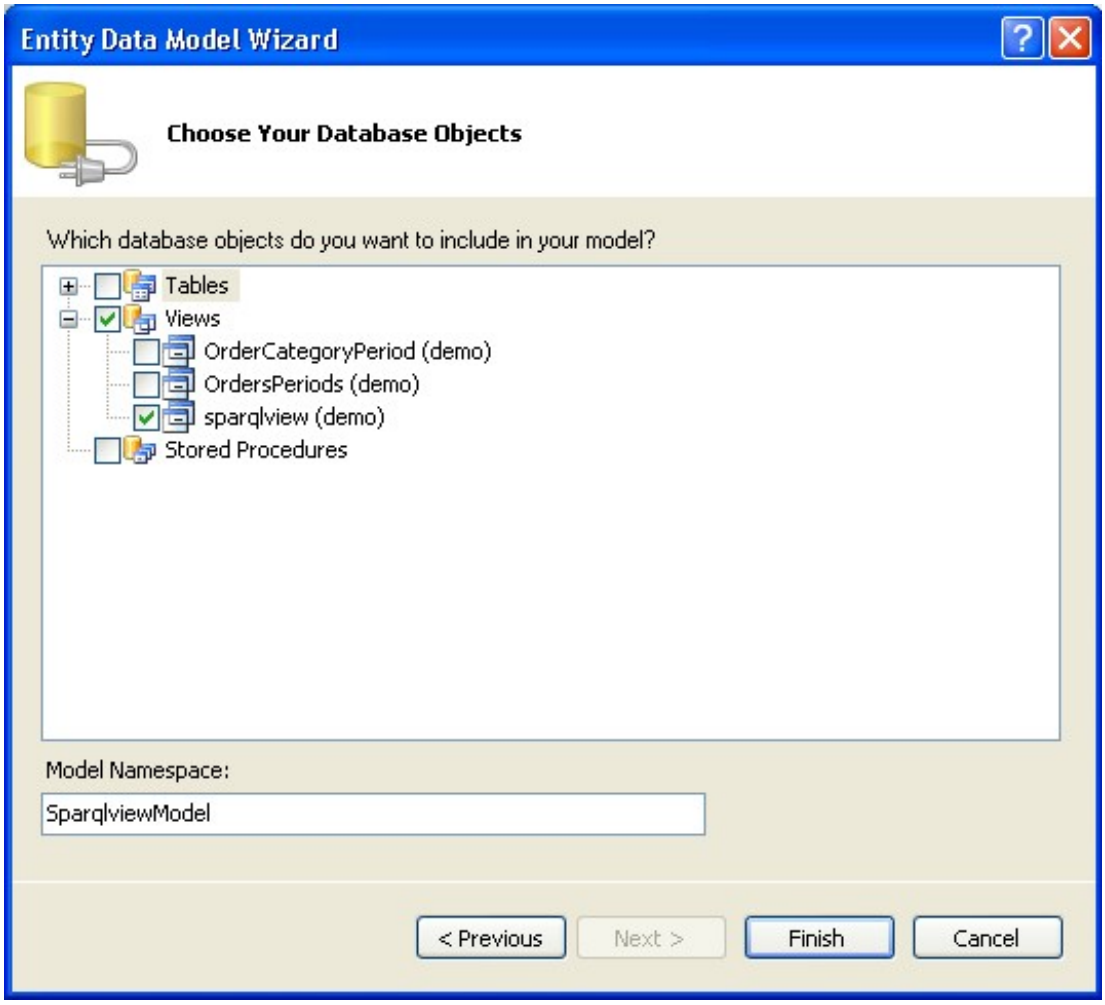
1. Open the Virtuoso Conductor.
2. In isql execute the following statement. Remember to use the appropriate hostname and port number of your Virtuoso installation, typically example.com for a default installation or whatever the URIQA DefaultHost Virtuoso configuration parameter was set to when the demo database VAD package was installed.

```
CREATE VIEW Demo.demo.sparqlview as
SPARQL
PREFIX nwind: <http://demo.openlinksw.com/schemas/northwind#>
SELECT DISTINCT  ?s
FROM <http://demo.openlinksw.com/Northwind>
WHERE
  {
    ?s a nwind:Employee
  }
```

Step 2 - Modify the Solution To use the View

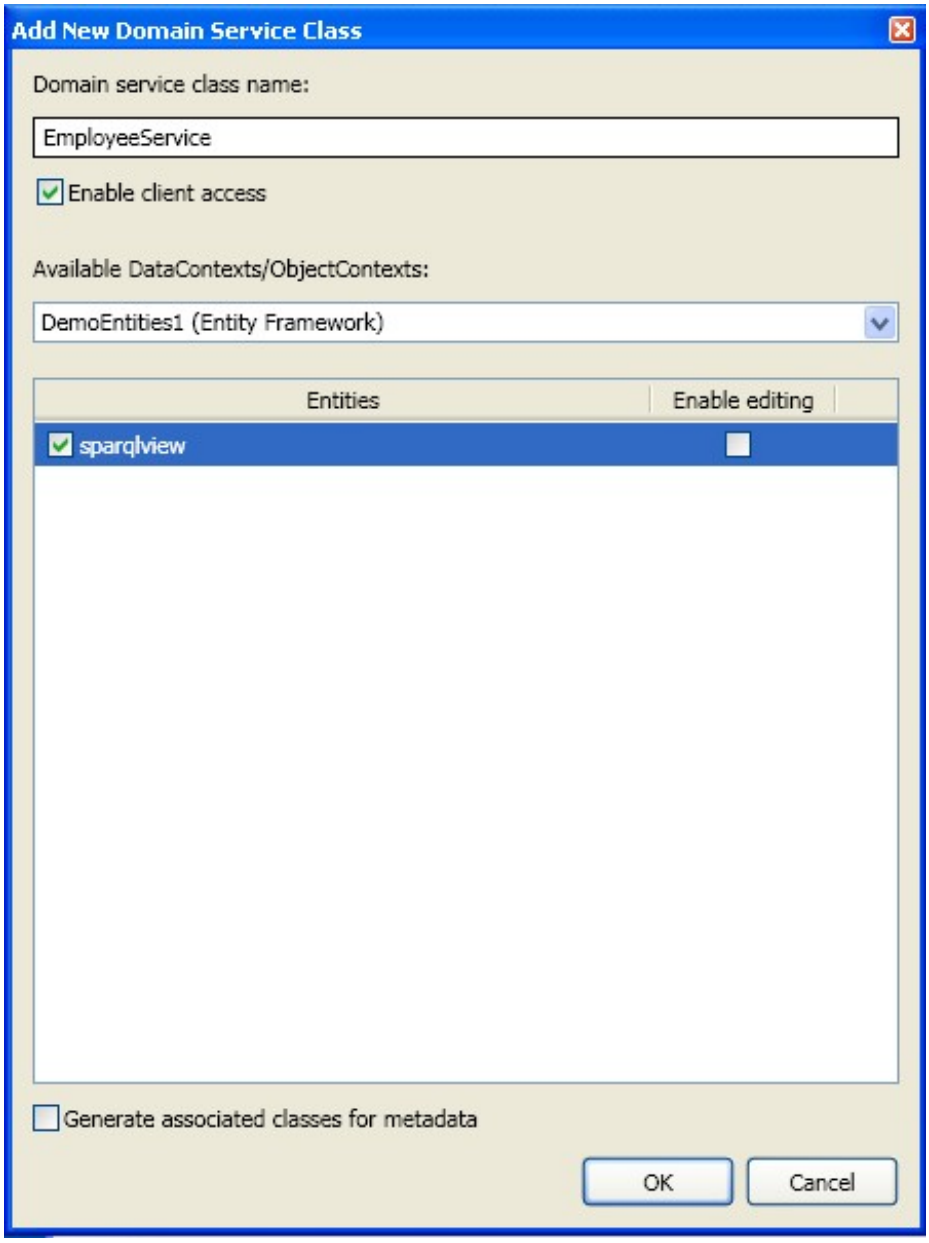
1. Delete the existing Employee model and add a new one that comprises this new view.

Figure 2.175. .NET RIA Services Application



2. Delete the DomainService. Build the solution and add a new DomainService called EmployeeService. Select the sparqlview entity. Build the solution.

Figure 2.176. .NET RIA Services Application



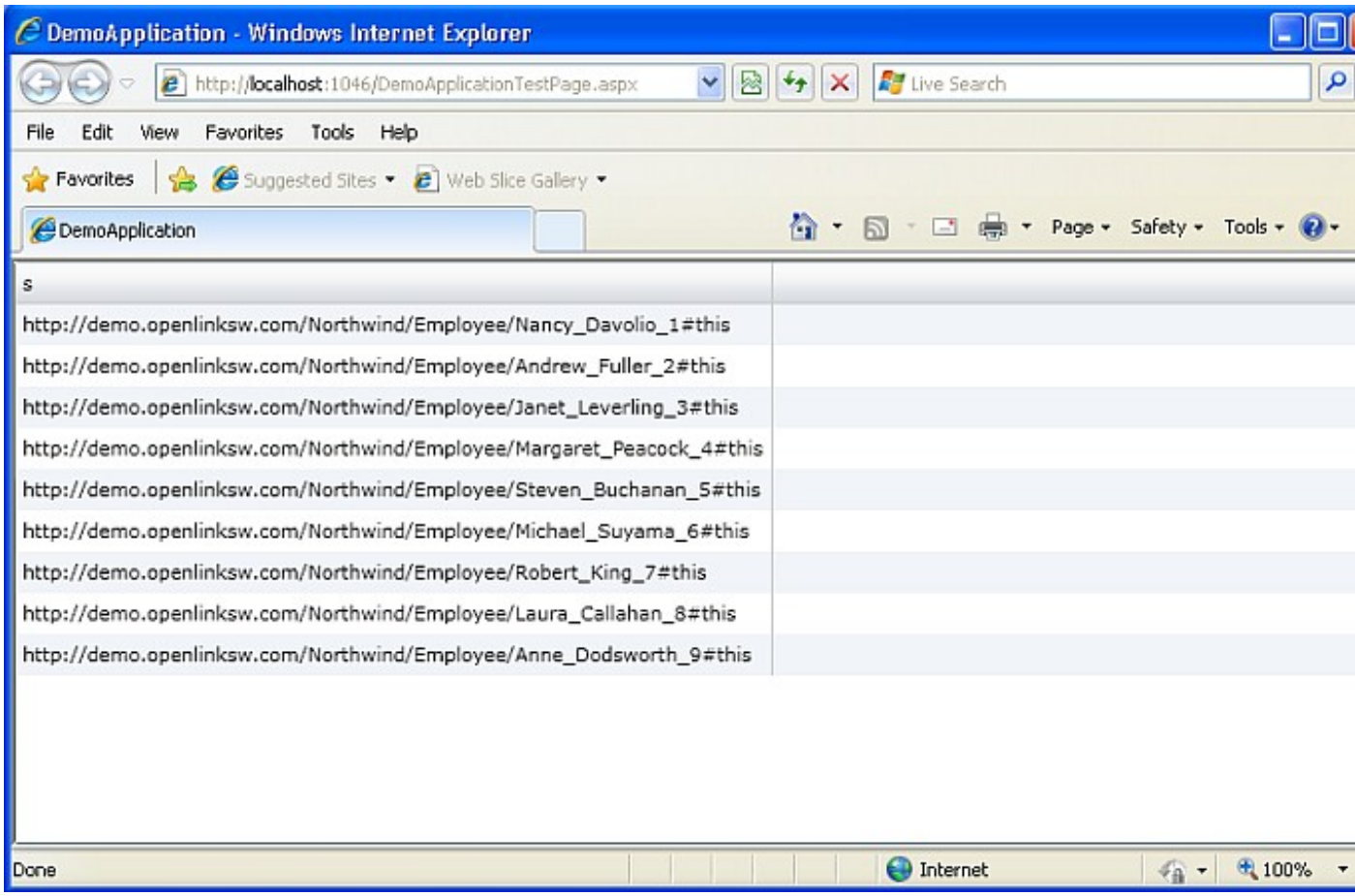
3. Modify the code in mainpage.xaml.cs so it uses the sparqview entity. It should look like this:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using DemoApplication.Web;
using System.Windows.Ria.Data;

namespace DemoApplication
{
    public partial class MainPage : UserControl
    {
        private EmployeeContext _employeeContext = new EmployeeContext();
        public MainPage()
        {
            InitializeComponent();
            LoadOperation<sparqlview> LoadOp =
                this._employeeContext.Load(this._employeeContext.GetSparqlviewQuery());
```

```
                this.EmployeeGrid.ItemsSource = LoadOp.Entities;
            }
        }
    }
```

4. Build and run the application. You will see a list of IRIs that identify the Northwind employees.

Figure 2.177. .NET RIA Services Application



To realize the power of linked data we would now liked to begin exploring this data by clicking on these IRIs.

Step 3 - Make Hyperlinks From IRIs

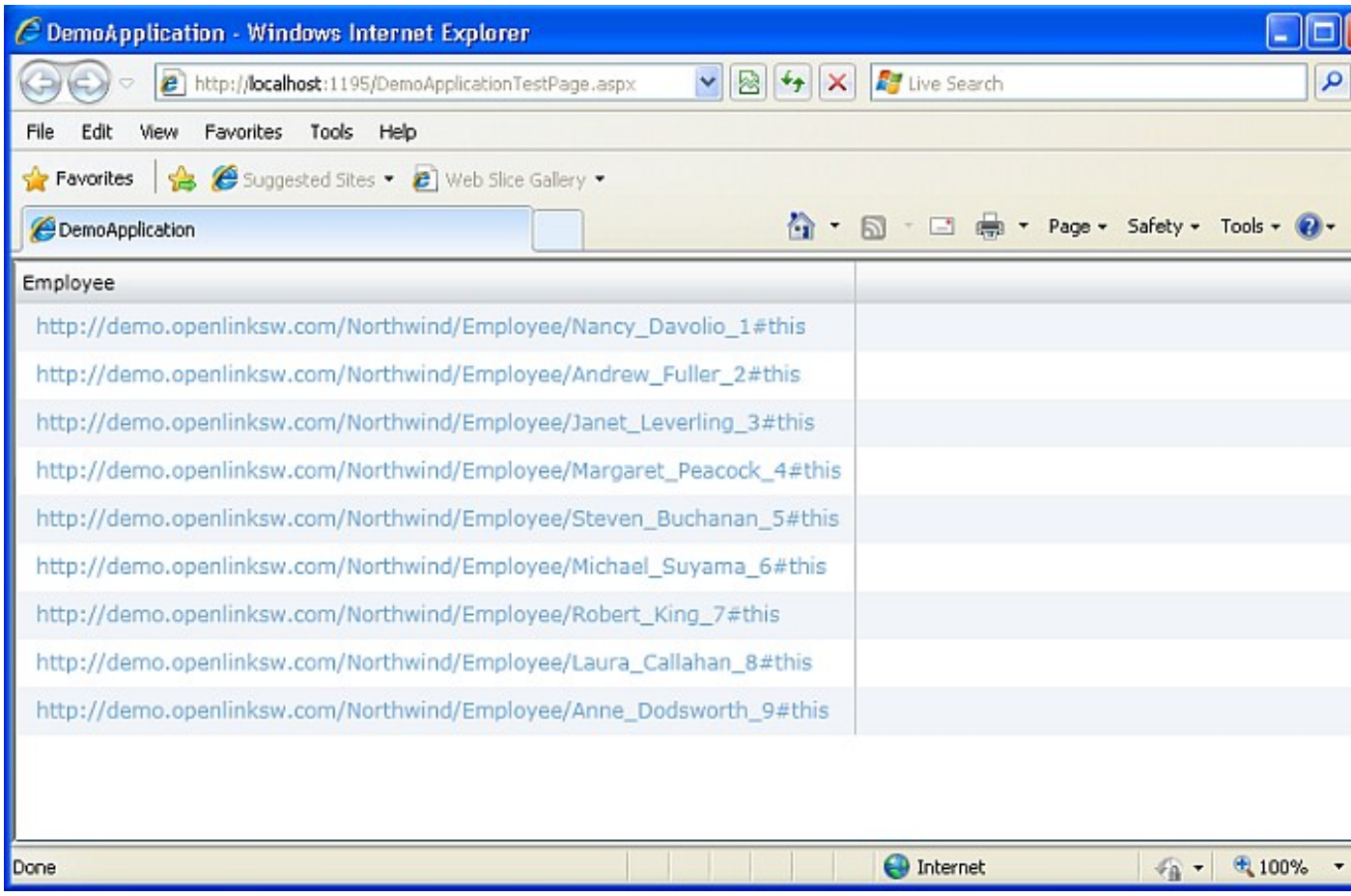
1. Modify the DataGrid in MainPage.xaml to bind the IRI in each cell of the grid to a Hyperlink button. We set the AutoGenerateColumns property of the DataGrid to False then add our own Template for the column. MainPage.xaml should look like this:

```
<UserControl xmlns:data="clr-namespace:System.Windows.Controls;assembly=System.Windows.Controls.Da
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008" xmlns:mc="http://schemas.openxmlf
    mc:Ignorable="d" d:DesignWidth="640" d:DesignHeight="480">
  <Grid x:Name="LayoutRoot">
      <data:DataGrid Name="EmployeeGrid" AutoGenerateColumns="False">
            <data:DataGrid.Columns>
                <data:DataGridTemplateColumn Header="Employee">
                    <data:DataGridTemplateColumn.CellTemplate>
                        <DataTemplate>
                            <StackPanel x:Name="DisplayEmployeeData"
                                        Orientation="Horizontal"
                                        VerticalAlignment="Bottom"
                                        Margin="5" >
                                <HyperlinkButton Content ="{Binding s}"
                                                 NavigateUri="{Binding s}"
                                                 Margin="5,0,0,0"
```

```
                                                VerticalAlignment="Bottom"
                                                HorizontalAlignment="Left"
                                                FontSize="12">
                                    </HyperlinkButton>
                                </StackPanel>
                            </DataTemplate>
                        </data:DataGridTemplateColumn.CellTemplate>
                    </data:DataGridTemplateColumn>
                </data:DataGrid.Columns>
            </data:DataGrid>
        </Grid>
    </UserControl>
```

2. If we build and run the project now each IRI in the list appears as a hyperlink.

Figure 2.178. .NET RIA Services Application



3. Clicking on each of the links takes you to a new document that contains further links to information about each Employee. The Demo dataset can be explored further by following the dereferenceable IRIs in the description pages.

Figure 2.179. .NET RIA Services Application

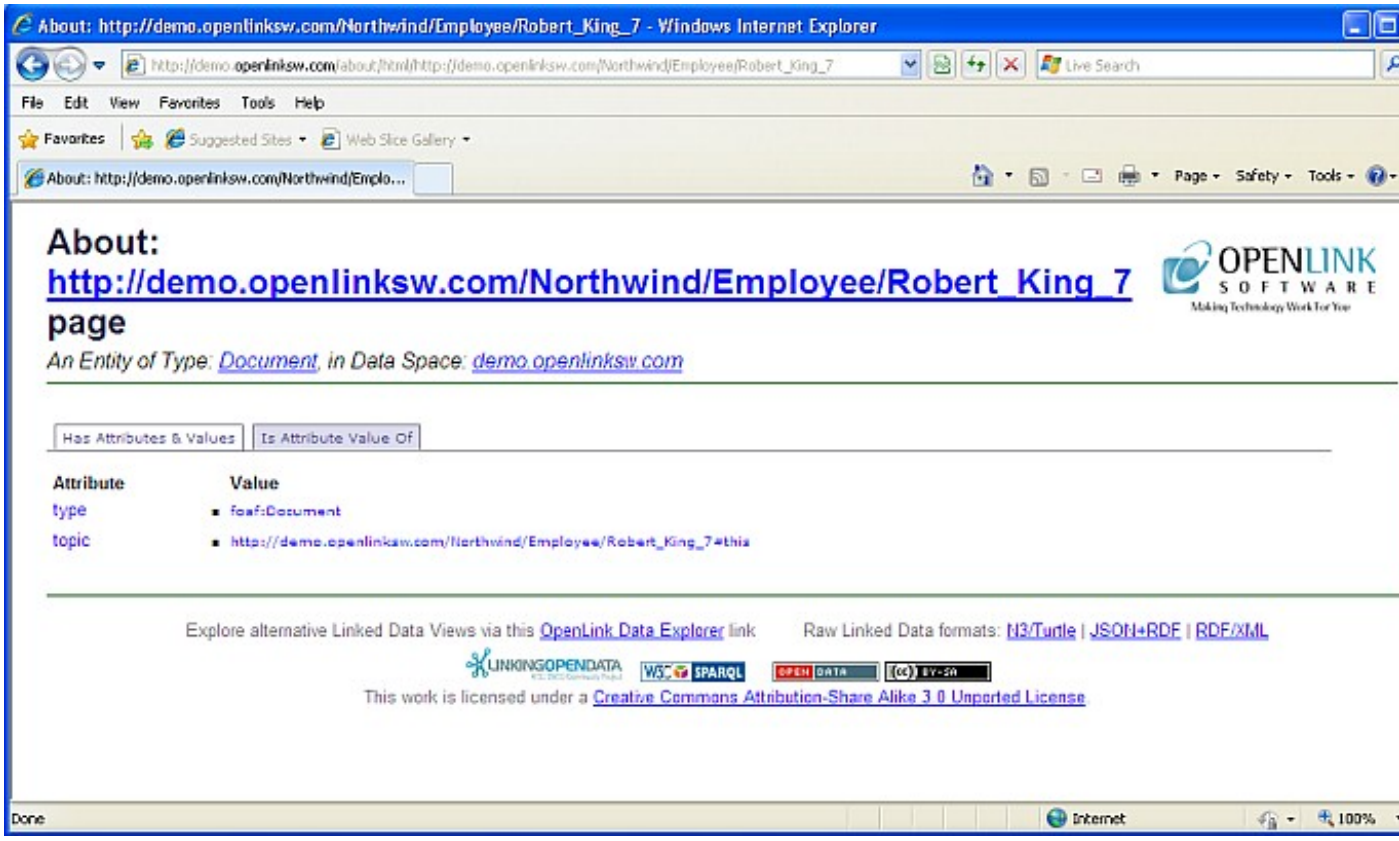
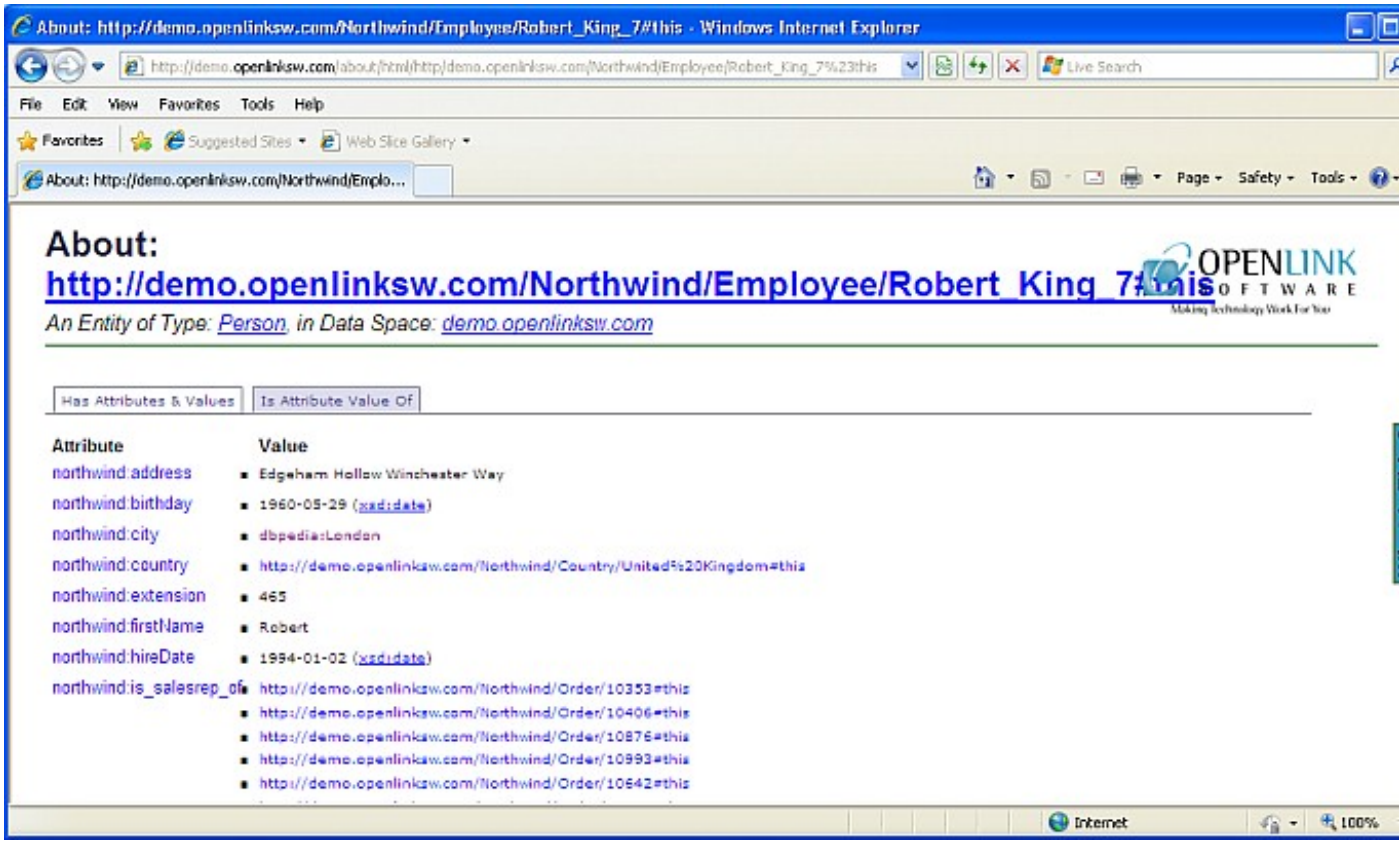


Figure 2.180. .NET RIA Services Application



2.12.4. Next Steps

The examples in this document show you how to simply display data in a browser using Silverlight 3 and .NET RIA Services. See an example of a more complicated application.

2.13. Creating a .Net RIA Services Application That Will Update Virtuoso Data

This example demonstrates how data in Virtuoso can be updated from a Microsoft .NET RIA Services application. The example is a continuation of the first example in Creating a Simple .Net RIA Services Application to Display Data From Virtuoso .

2.13.1. Pre-requisites

1. A working copy of the application created in Creating a Simple .Net RIA Services Application to Display Data From Virtuoso .

2.13.2. Creating the Application

Step 1 - Add A New Domain Service Class

The EmployeeService Domain Service Class was only used to display data so was created read only. In this example we need to be able to update the data so we need to remove the read only Domain Service Class and create a new on.

1. In the

   Server Explorer

   right click EmployeeService and select

   Delete
2. Right click the server project and

   Add New Item

   . In the dialog box choose Domain Service Class from the Templates pane and again call it EmployeeService.cs. Click Add.
3. Select the Employees entity and tick

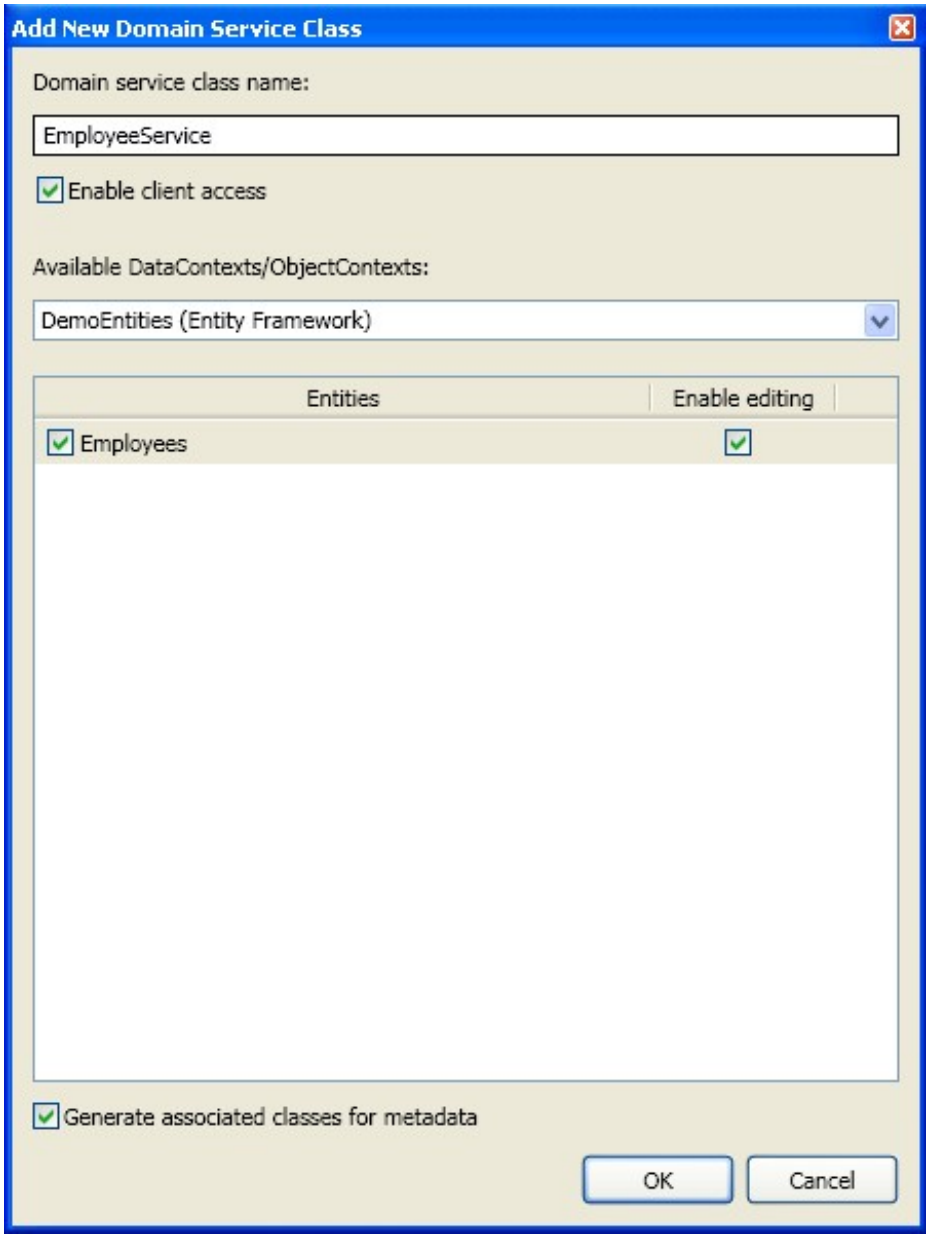
   Enable editing

   . Also tick

   Generated associated classes for metadata

   .

   Figure 2.181. .NET RIA Services Application



4. Update MainPage.xaml.cs to use EmployeeService2 the new domain service rather than EmployeeContext.

We want to create a master detail style page. To do this we will use the DomainDataSourceComponent from the Silverlight Components. But first we will add a pager so only 5 records are displayed at a time.

Step 2 - Add a DataPager

1. Add two new namespaces to MainPage.xaml

```
xmlns:riaControls="clr-namespace:System.Windows.Controls;assembly=System.Windows.Ria.Controls"
xmlns:ds="clr-namespace:DemoApplication.Web"
```

2. Use a DomainDataSource to provide the data to fill the grid. Add the following to MainPage.xaml before the DataGrid.

```
<riaControls:DomainDataSource x:Name="EmployeeDataSource"
                              QueryName="GetEmployeesQuery"
                              LoadSize="10"
                              AutoLoad="True">
    <riaControls:DomainDataSource.DomainContext>
        <ds:EmployeeService2/>
    </riaControls:DomainDataSource.DomainContext>
</riaControls:DomainDataSource>
```

3. Update the MainPage.xaml.cs code behind file. Using the DomainDataSource means you no longer need to instantiate the context and load the grid in MainPage.xaml.cs so it becomes:

```
public partial class MainPage : UserControl
{
    public MainPage()
    {
        InitializeComponent();
    }
}
```

4. Set the binding source of the DataGrid to the DomainDataSource

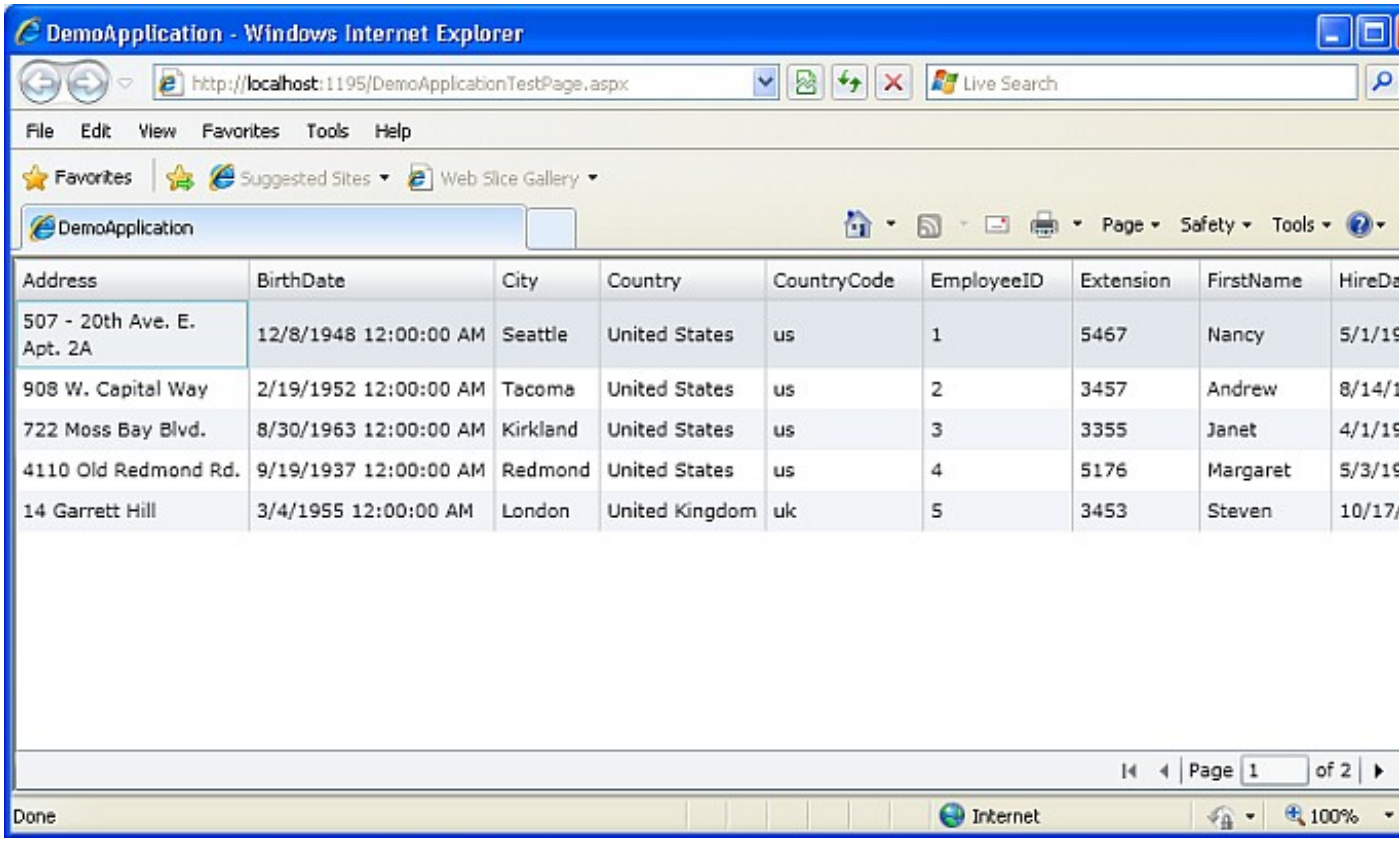
```
<data:DataGrid MinHeight="100"  IsReadOnly="True" ItemsSource="{Binding Data, ElementName=Employee
```

5. Drag a DataPager form the tool box onto MainPage.xaml just after the DataGrid.
6. Add a page size and binding element to the DataPager.

```
<data:DataPager PageSize="5" Source="{Binding Data, ElementName=employeeDataSource}"
                    Margin="0,-1,0,0"></data:DataPager>
```

7. Build and run the application. The data should be displayed 5 rows at a time.

Figure 2.182. .NET RIA Services Application



We now need to add a DataForm to display the details.

Step 3 - Add a DataForm

1. Add the following namespace to MainPage.xaml

```
xmlns:dataForm="clr-namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data.DataFo
```

2. Add the form to MainPage.xaml

```
<dataForm:DataForm x:Name="dataForm1" Header="Employee Information"  AutoGenerateFields="False" Au
   <dataForm:DataForm.EditTemplate>
       <DataTemplate>
           <StackPanel>
               <dataForm:DataField Label="Employee ID">
                   <TextBox Text="{Binding EmployeeID, Mode=TwoWay}" />
               </dataForm:DataField>
               <dataForm:DataField Label="First Name">
```

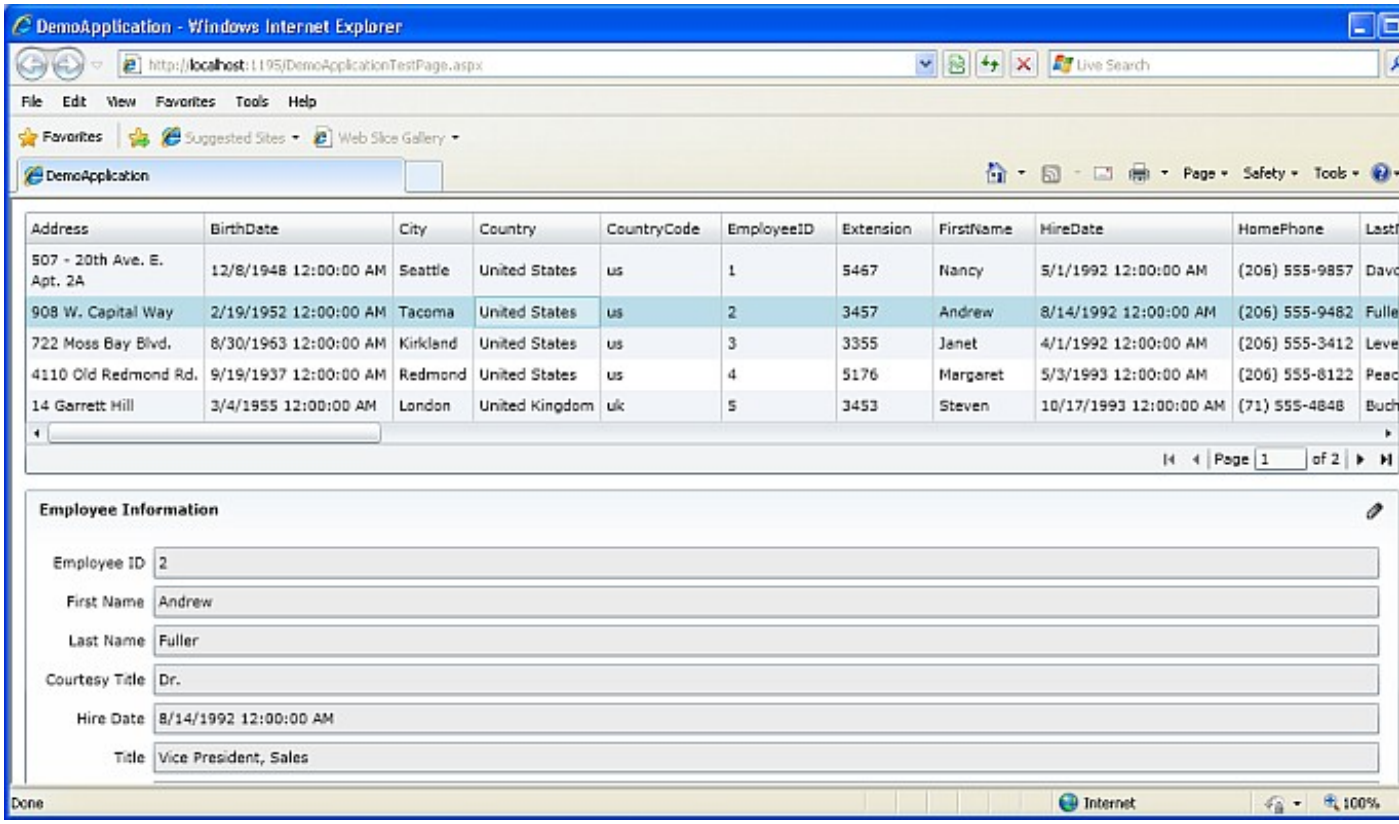
```
                    <TextBox Text="{Binding FirstName, Mode=TwoWay}" />
                </dataForm:DataField>
                <dataForm:DataField Label="Last Name">
                    <TextBox Text="{Binding LastName, Mode=TwoWay}" />
                </dataForm:DataField>
                <dataForm:DataField Label="Courtesy Title">
                    <TextBox Text="{Binding TitleOfCourtesy, Mode=TwoWay}" />
                </dataForm:DataField>
                <dataForm:DataField Label="Hire Date">
                    <TextBox Text="{Binding HireDate, Mode=TwoWay}" />
                </dataForm:DataField>
                <dataForm:DataField Label="Title">
                    <TextBox Text="{Binding Title, Mode=TwoWay}"  />
                </dataForm:DataField>
                <dataForm:DataField Label="Reports To">
                    <TextBox Text="{Binding ReportsTo, Mode=TwoWay}"  />
                </dataForm:DataField>
                <dataForm:DataField Label="Region">
                    <TextBox Text="{Binding Region, Mode=TwoWay}"  />
                </dataForm:DataField>
                <dataForm:DataField Label="Address">
                    <TextBox Text="{Binding Address, Mode=TwoWay}"  />
                </dataForm:DataField>
                <dataForm:DataField Label="City">
                    <TextBox Text="{Binding City, Mode=TwoWay}"  />
                </dataForm:DataField>
                <dataForm:DataField Label="Country Code">
                    <TextBox Text="{Binding CountryCode, Mode=TwoWay}"  />
                </dataForm:DataField>
                <dataForm:DataField Label="Postal Code">
                    <TextBox Text="{Binding PostalCode, Mode=TwoWay}"  />
                </dataForm:DataField>
                    </StackPanel>
            </DataTemplate>
        </dataForm:DataForm.EditTemplate>
    </dataForm:DataForm>
```

3. Surround the DomainDataSource, DataGrid and DataForm with

```
        <ScrollViewer BorderThickness="0"  VerticalScrollBarVisibility="Auto" Padding="12,0,12,0"
            <StackPanel Margin="0,12,0,12" Orientation="Vertical" >
.
.
.
            </StackPanel>
        </ScrollViewer>
```

4. Build and run the application. As each employee is selected the data form fill with their details

Figure 2.183. .NET RIA Services Application



By clicking on the pencil symbol in the top right hand corner the data in the form can be edited but it is not propagated back to the database.

2.13.3. Propagate Updates to Virtuoso

1. Add a 'Submit' button just after the DataForm in MainPage.xaml by adding the following code.

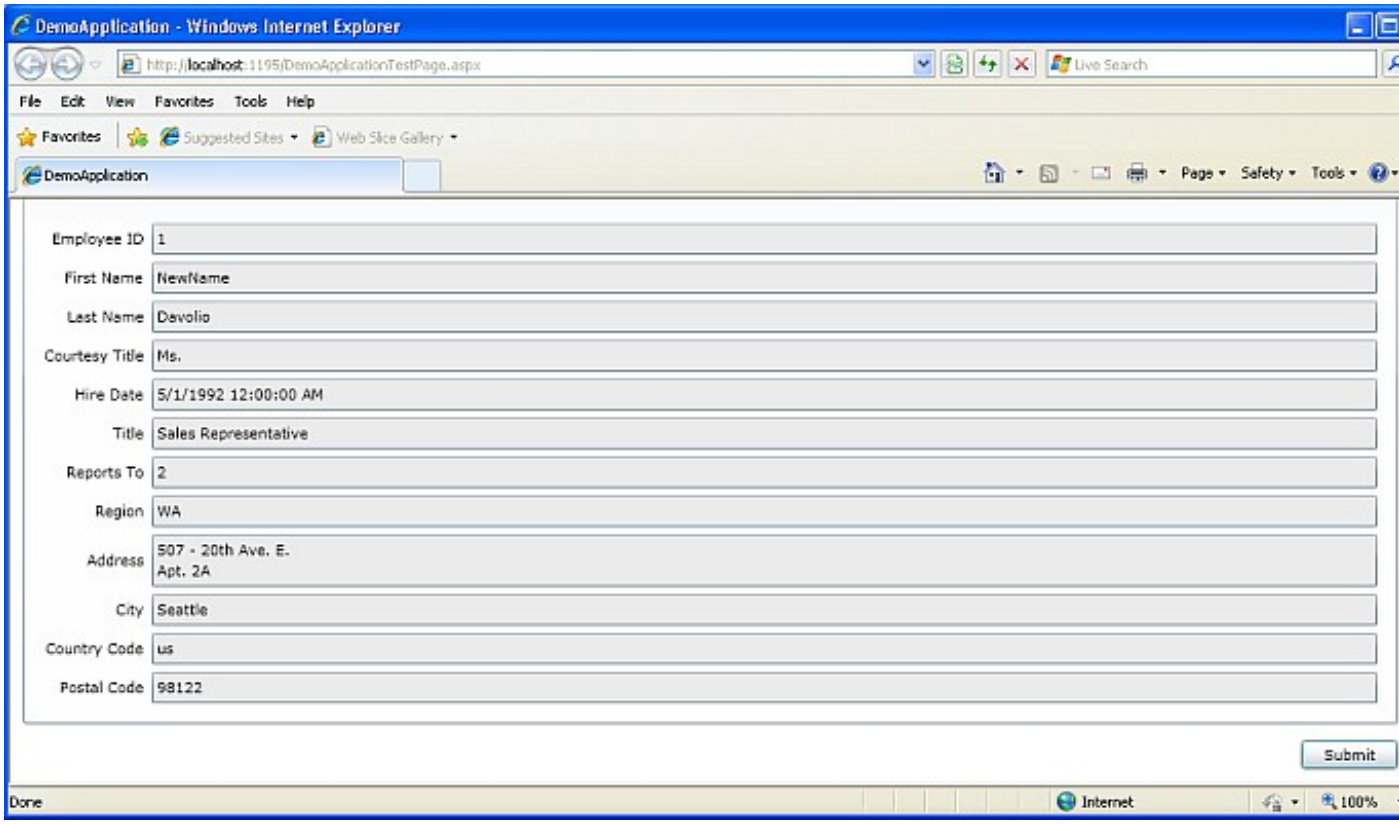
```
<StackPanel Orientation="Horizontal" HorizontalAlignment="Right" Margin="0,12,0,0">
  <Button x:Name="submitButton" Width="75" Height="23"  Content="Submit"  Margin="4,0,0,0" Click="
</StackPanel>
```

2. Handle the button click event in MainPage.xaml.cs by adding the following code:

```
private void submitButton_Click(object sender, RoutedEventArgs e)
        {
            EmployeeDataSource.SubmitChanges();
        }
```

3. Build and run the application. If you now edit data in the form and click the submit button the data in Virtuoso will be updated.

Figure 2.184. .NET RIA Services Application



2.14. Cluster Installation and Configuration

2.14.1. Virtuoso Elastic Cluster Installation & Configuration -- Version 7.x

These sections apply to Virtuoso as of version 7.x .

What

Virtuoso 7 and later versions can be configured in Elastic scale out cluster mode. The data is sharded in a large number of self-contained partitions. These partitions are divided among a number of database server processes and can migrate between them.

Why

An Elastic cluster can be dynamically resized with new partitions or shards dynamically added to the same or new hardware resources as the need to increase the size of the scale cluster or relocation of partitions is required.

How

This documentation details the steps for the installation and configuration of a Virtuoso Elastic Cluster on Unix:

1. Perform Virtuoso 7 Unix installation
2. Determine Elastic Cluster size
3. Enable Elastic Cluster mode
4. Start Elastic Cluster
5. Splitting Cluster nodes across different machines.

Virtuoso 7 Unix installation

Step 1 : Perform Virtuoso 7 Unix installation

```
$ tar xvf virtuoso-universal-server-7.x.tar
x install.sh
x universal-server.taz
```

```
$ sh install.sh

- Extracting Virtuoso Universal Server v7.x

- Creating default environment settings

- Creating default database settings
Configuring: database
Creating directory $VIRTUOSO_HOME/database/backup
Installing new virtuoso.ini in $VIRTUOSO_HOME/database
Installing new php.ini in $VIRTUOSO_HOME/database
Creating symlink to $VIRTUOSO_HOME/bin/virtuoso-iodbc-t

- Registering ODBC drivers

- Registering .NET provider for Mono

- Installing VAD packages in database (this can take some time)

- Checking where license should be stored

- Starting OpenLink License Manager

- Checking for initial Virtuoso license

- Starting Virtuoso server instance

- Finalizing installation

This concludes the first part of the installation.

- Attempting to start browser

Please start a browser manually and open the following URL to finalize
the installation process:

     http://cname:8890/install/

Installation completed
$
```

Determine Elastic Cluster size

Step 2 : Determine Elastic Cluster size

1. Determine how many nodes you want to start the elastic cluster with: 2, 3, 4, 5, 6, 7, 8 etc. Note: the number of the nodes must be >= 2
2. Setup Virtuoso environment and stop default database:

```
$ . ./virtuoso-environment.sh
$ virtuoso-stop.sh
Shutting down Virtuoso instance in [database]
$
```

3. Optionally remove the default database such that is does not get started with the cluster:

```
rm  -rf database
```

4. Full list of supported options for the script "virtuoso-mkcluster.sh":

```
  -cluster-size       Number of nodes in cluster
  -cluster-node       Node number in a cluster
  -cluster-port       Base portnumer for cluster
  -cluster-ipc-port   Base portnumer for cluster IPC
  -cluster_size       Size of the cluster
  -virtuoso_home      Virtuoso home path
```

5. Run the virtuoso-mkcluster.sh script to create the cluster, note the default number of nodes is 4, but this can be changed by setting the environment variable CLUSTER_SIZE to the required number of nodes to be created before running the script:

```
$ virtuoso-mkcluster.sh
Configuring cluster node: 1/4
```

```
Creating directory $VIRTUOSO_HOME/cluster_01
Creating directory $VIRTUOSO_HOME/cluster_01/backup
Creating directory $VIRTUOSO_HOME/cluster_01/logs
Installing new virtuoso.ini in $VIRTUOSO_HOME/cluster_01
Installing new php.ini in $VIRTUOSO_HOME/cluster_01
Creating symlink to $VIRTUOSO_HOME/bin/virtuoso-iodbc-t
Configuring cluster node: 2/4

Creating directory $VIRTUOSO_HOME/cluster_02
Creating directory $VIRTUOSO_HOME/cluster_02/backup
Installing new virtuoso.ini in $VIRTUOSO_HOME/cluster_02
Removing unneeded sections from virtuoso.ini
Creating symlink to $VIRTUOSO_HOME/bin/virtuoso-iodbc-t
Configuring cluster node: 3/4

Creating directory $VIRTUOSO_HOME/cluster_03
Creating directory $VIRTUOSO_HOME/cluster_03/backup
Installing new virtuoso.ini in $VIRTUOSO_HOME/cluster_03
Removing unneeded sections from virtuoso.ini
Creating symlink to $VIRTUOSO_HOME/bin/virtuoso-iodbc-t
Configuring cluster node: 4/4

Creating directory $VIRTUOSO_HOME/cluster_04
Creating directory $VIRTUOSO_HOME/cluster_04/backup
Installing new virtuoso.ini in $VIRTUOSO_HOME/cluster_04
Removing unneeded sections from virtuoso.ini
Creating symlink to $VIRTUOSO_HOME/bin/virtuoso-iodbc-t
$
```

6. In each cluster_XX directory the following the `cluster.ini` files are created for cluster internal communication:

```
$ more cluster_01/cluster.ini
[Cluster]
Threads             = 20
Master              = Host1
ThisHost            = Host1
ReqBatchSize        = 10000
BatchesPerRPC       = 4
BatchBufferBytes    = 20000
LocalOnly           = 2
MaxKeepAlivesMissed = 1000
Host1               = localhost:22201
Host2               = localhost:22202
Host3               = localhost:22203
Host4               = localhost:22204
MaxHosts            = 5
$ more cluster_02/cluster.ini
[Cluster]
Threads             = 20
Master              = Host1
ThisHost            = Host2
ReqBatchSize        = 10000
BatchesPerRPC       = 4
BatchBufferBytes    = 20000
LocalOnly           = 2
MaxKeepAlivesMissed = 1000
Host1               = localhost:22201
Host2               = localhost:22202
Host3               = localhost:22203
Host4               = localhost:22204
MaxHosts            = 5
$ more cluster_03/cluster.ini
[Cluster]
Threads             = 20
Master              = Host1
ThisHost            = Host3
ReqBatchSize        = 10000
BatchesPerRPC       = 4
BatchBufferBytes    = 20000
LocalOnly           = 2
MaxKeepAlivesMissed = 1000
Host1               = localhost:22201
Host2               = localhost:22202
```

```
Host3               = localhost:22203
Host4               = localhost:22204
MaxHosts            = 5
$ more cluster_04/cluster.ini
[Cluster]
Threads             = 20
Master              = Host1
ThisHost            = Host4
ReqBatchSize        = 10000
BatchesPerRPC       = 4
BatchBufferBytes    = 20000
LocalOnly           = 2
MaxKeepAlivesMissed = 1000
Host1               = localhost:22201
Host2               = localhost:22202
Host3               = localhost:22203
Host4               = localhost:22204
MaxHosts            = 5
$
```

Enable Elastic Cluster mode

Step 3 : Enable Elastic Cluster mode

1. The `cluster.ini` files need to be reconfigured as detailed below for elastic cluster and file slicing/sharding to be enabled.
2. A common file called `cluster.global.ini` can be created and placed in the home directory of the Virtuoso installation:

```
[Cluster]
Threads             = 300
ReqBatchSize        = 10000
BatchesPerRPC       = 4
BatchBufferBytes    = 20000
LocalOnly           = 2
MaxKeepAlivesMissed = 100
RDFLoadBytes        = 52428800
Host1               = localhost:22201
Host2               = localhost:22202
Host3               = localhost:22203
Host4               = localhost:22204
```

3. This file can then be sym-linked to each of the `cluster_XX` directories of the cluster, and its content will be merged with the `cluster.ini` in the respective directory of each node when starting the cluster.

```
ln -s cluster.global.ini cluster_01/cluster.global.ini
ln -s cluster.global.ini cluster_02/cluster.global.ini
ln -s cluster.global.ini cluster_03/cluster.global.ini
ln -s cluster.global.ini cluster_04/cluster.global.ini
```

4. Edit each of the the `cluster.ini` files in the `cluster_XX` directories as follows to enable, elastic cluster mode:

```
$ more cluster_01/cluster.ini
[Cluster]
Master              = Host1
ThisHost            = Host1

[ELASTIC]
Segment1 = 4G, database.db = q1
Slices = 6
MaxSlices = 2048

$ more cluster_02/cluster.ini

[Cluster]
Master              = Host1
ThisHost            = Host2

[ELASTIC]
Segment1 = 4G, database.db = q1
Slices = 6
MaxSlices = 2048
```

```
$ more cluster_03/cluster.ini

[Cluster]
Master              = Host1
ThisHost            = Host3

[ELASTIC]
Segment1 = 4G, database.db = q1
Slices = 6
MaxSlices = 2048

$ more cluster_04/cluster.ini

[Cluster]
Master              = Host1
ThisHost            = Host4

[ELASTIC]
Segment1 = 4G, database.db = q1
Slices = 6
MaxSlices = 2048
```

5. The `[ELASTIC]` section in the `cluster.ini` files above, enables the elastic cluster mode, where multiple `segments` and `stripes` as detailed in the standard Virtuoso documentation database striping .
6. The `Slices` parameter above should be set to the number of hardware threads on the CPUs running on. Thus in the example above where it is set to 6, this assumes all nodes are running on the same physical machine with 12 cores with hyper-threading enabled i.e. 24 threads, thus 6 per cluster node.
7. The `MaxSlices` parameter above sets the the maximum number of physical slices in the cluster.

Start Elastic Cluster

Step 4 : Start Elastic Cluster

1. Start the Elastic cluster using the standard `virtuoso-start.sh` script run from the home directory of the Virtuoso installation,which automatically detects the number for nodes to be started:

```
$ virtuoso-start.sh
Starting Virtuoso instance in [cluster_01]
Starting Virtuoso instance in [cluster_02]
Starting Virtuoso instance in [cluster_03]
Starting Virtuoso instance in [cluster_04]
$
```

2. The default SQL port of the master node is 12201, as indicated in the virtuoso.ini file of the cluster_01 directory, and can then be used for connecting to the newly created cluster and check its status to ensure all nodes are online:

```
$ isql 12201
Connected to OpenLink Virtuoso
Driver: 07.10.3211 OpenLink Virtuoso ODBC Driver
OpenLink Interactive SQL (Virtuoso), version 0.9849b.
Type HELP; for help and EXIT; to exit.
SQL> status('cluster_d');
REPORT
VARCHAR
_______________________________________________________________________________

Cluster: No samples, Please refresh

1 Rows. -- 22 msec.
SQL> status('cluster_d');
REPORT
VARCHAR
_______________________________________________________________________________

Cluster 4 nodes, 2 s. 5 m/s 0 KB/s  5% cpu 0%  read 0% clw threads 1r 0w 0i buffers 349144 2 d 0 w
cl 1: 2 m/s 0 KB/s  0% cpu 0%  read 0% clw threads 1r 0w 0i buffers 45391 2 d 0 w 0 pfs
cl 2: 0 m/s 0 KB/s  0% cpu 0%  read 0% clw threads 0r 0w 0i buffers 43367 0 d 0 w 0 pfs
cl 3: 0 m/s 0 KB/s  0% cpu 0%  read 0% clw threads 0r 0w 0i buffers 50129 0 d 0 w 0 pfs
cl 4: 0 m/s 0 KB/s  0% cpu 0%  read 0% clw threads 0r 0w 0i buffers 39383 0 d 0 w 0 pfs
```

```
5 Rows. -- 22 msec.
SQL>
SQL> cl_ping(1,500, 1000);

Done. -- 7 msec.
SQL> cl_ping(2,500, 1000);

Done. -- 52 msec.
SQL> cl_ping(3,500, 1000);

Done. -- 60 msec.
SQL> cl_ping(4,500, 1000);

Done. -- 51 msec.
SQL>
```

3. The `database.log` of the master node can be checked to verify the Elastic cluster mode has been enable on server startup, which will contain the entry *PL LOG: Elastic cluster setup* :

```
$ more cluster_01/database.log

                Thu Oct 09 2014
05:11:49 { Loading plugin 1: Type `plain', file `wikiv' in `../hosting'
05:11:49   WikiV version 0.6 from OpenLink Software
05:11:49   Support functions for WikiV collaboration tool
05:11:49   SUCCESS plugin 1: loaded from ../hosting/wikiv.so }
05:11:49 { Loading plugin 2: Type `plain', file `mediawiki' in `../hosting'
05:11:49   MediaWiki version 0.1 from OpenLink Software
05:11:49   Support functions for MediaWiki collaboration tool
05:11:49   SUCCESS plugin 2: loaded from ../hosting/mediawiki.so }
05:11:49 { Loading plugin 3: Type `plain', file `creolewiki' in `../hosting'
05:11:49   CreoleWiki version 0.1 from OpenLink Software
05:11:49   Support functions for CreoleWiki collaboration tool
05:11:49   SUCCESS plugin 3: loaded from ../hosting/creolewiki.so }
05:11:49 { Loading plugin 4: Type `plain', file `im' in `../hosting'
05:11:49   IM version 0.61 from OpenLink Software
05:11:49   Support functions for Image Magick 6.8.1
05:11:49   SUCCESS plugin 4: loaded from ../hosting/im.so }
05:11:49 { Loading plugin 5: Type `plain', file `wbxml2' in `../hosting'
05:11:49   WBXML2 version 0.9 from OpenLink Software
05:11:49   Support functions for WBXML2 0.9.2 Library
05:11:49   SUCCESS plugin 5: loaded from ../hosting/wbxml2.so }
05:11:49 { Loading plugin 6: Type `attach', file `libphp5.so' in `../hosting'
05:11:49   SUCCESS plugin 6: loaded from ../hosting/libphp5.so }
05:11:49 { Loading plugin 7: Type `Hosting', file `hosting_php.so' in `../hosting'
05:11:49   Hosting version 3208 from OpenLink Software
05:11:49   PHP engine version 5.3.21
05:11:49   SUCCESS plugin 7: loaded from ../hosting/hosting_php.so }
05:11:49 { Loading plugin 8: Type `plain', file `qrcode' in `../hosting'
05:11:49   QRcode version 0.1 from OpenLink Software
05:11:49   Support functions for ISO/IEC 18004:2006, using QR Code encoder (C) 2006 Kentaro Fukuch
05:11:49   SUCCESS plugin 8: loaded from ../hosting/qrcode.so }
05:11:49 OpenLink Virtuoso Universal Server
05:11:49 Version 07.10.3211-pthreads for Linux as of Oct  6 2014
05:11:49 uses parts of OpenSSL, PCRE, Html Tidy
05:11:51 SQL Optimizer enabled (max 1000 layouts)
05:11:52 Compiler unit is timed at 0.000403 msec
05:12:03 Checkpoint started
05:12:04 Roll forward started
05:12:04 Roll forward complete
05:12:07 PL LOG: Elastic cluster setup
05:12:08 Checkpoint started
05:12:09 Checkpoint finished, log reused
05:12:11 Checkpoint started
05:12:12 Checkpoint finished, log reused
05:12:12 PL LOG: new clustered database:Init of RDF
05:12:23 Checkpoint started
05:12:25 Checkpoint finished, log reused
05:12:50 PL LOG: Installing Virtuoso Conductor version 1.00.8727 (DAV)
05:12:51 Checkpoint started
05:12:53 Checkpoint finished, log reused
05:13:23 Checkpoint started
05:13:25 Checkpoint finished, log reused
05:13:26 Checkpoint started
```

```
05:13:26 Checkpoint finished, log reused
05:13:28 HTTP/WebDAV server online at 8890
05:13:28 Server online at 12201 (pid 15211)
05:13:29 ZeroConfig registration CLUSTER (MASALA)
```

4. The cluster node directories can also be checked, where the database slice/shard files i.e. `database.db.X, database.db.Y, database.db.Z` can be seen:

```
$ ls -ltr cluster_01
total 2322804
drwxr-xr-x. 2 virtuoso virtuoso      4096 Oct  9 04:26 backup
drwxr-xr-x. 2 virtuoso virtuoso      4096 Oct  9 04:26 logs
-rwxr-xr-x. 1 virtuoso virtuoso     70607 Oct  9 04:26 php.ini
lrwxrwxrwx. 1 virtuoso virtuoso        24 Oct  9 04:26 virtuoso -> ..//bin/virtuoso-iodbc-t
-rw-r--r--. 1 virtuoso virtuoso         0 Oct  9 04:26 database.pxa
-rwxr-xr-x. 1 virtuoso virtuoso      6594 Oct  9 04:33 virtuoso.ini
-rw-r--r--. 1 virtuoso virtuoso       137 Oct  9 05:03 cluster.ini
lrwxrwxrwx. 1 virtuoso virtuoso        21 Oct  9 05:04 cluster.global.ini -> ../cluster.global.ini
drwxr-xr-x. 2 virtuoso virtuoso    131072 Oct  9 05:37 dump
-rw-r--r--. 1 virtuoso virtuoso     32915 Oct  9 06:55 database.2pc
-rw-r--r--. 1 virtuoso virtuoso        25 Oct 10 02:24 database.map
-rw-r--r--. 1 virtuoso virtuoso  56623104 Oct 10 03:23 database-temp.db
-rw-r--r--. 1 virtuoso virtuoso 824180736 Oct 10 05:10 database.db.16
-rw-r--r--. 1 virtuoso virtuoso 723517440 Oct 10 05:10 database.db.8
-rw-r--r--. 1 virtuoso virtuoso 740294656 Oct 10 05:10 database.db.0
-rw-r--r--. 1 virtuoso virtuoso  33554432 Oct 10 05:10 database.db
-rw-r--r--. 1 virtuoso virtuoso         0 Oct 10 05:10 database.trx
-rw-r--r--. 1 virtuoso virtuoso     34234 Oct 10 05:10 database.log
$
```

Splitting Cluster nodes across different machines

Step 5 : Splitting Cluster nodes across different machines

1. To split the node across across physical machines for better scale out performance, scalability and growth, simply perform a parallel Virtuoso installation on the additional physical machines and move the cluster nodes required to the designated machine. Example, for the default 4 node cluster to be split across two identical machines it would make sense to split 2 nodes across each machine, thus you would move say the cluster_03 and cluster_04 directory nodes to the new machine (removing them from the original). The `cluster.global.ini` file on each node would then need to be updated to set the HostXX parameters to point to the new locations for nodes 03 and 04:

```
$ more cluster.global.ini

[Cluster]
Threads             = 300
ReqBatchSize        = 10000
BatchesPerRPC       = 4
BatchBufferBytes    = 20000
LocalOnly           = 2
MaxKeepAlivesMissed = 100
RDFLoadBytes        = 52428800
Host1               = hostname1:22201
Host2               = hostname1:22202
Host3               = hostname2:22203
Host4               = hostname2:22204

Machine 1 (hostname1)

$ more cluster_01/cluster.ini
[Cluster]
Master              = Host1
ThisHost            = Host1

[ELASTIC]
Segment1 = 4G, database.db = q1
Slices = 12
MaxSlices = 2048

$ more cluster_02/cluster.ini

[Cluster]
```

```
Master              = Host1
ThisHost            = Host2

[ELASTIC]
Segment1 = 4G, database.db = q1
Slices = 12
MaxSlices = 2048

Machine 2 (hostname2)

$ more cluster_03/cluster.ini

[Cluster]
Master              = Host1
ThisHost            = Host3

[ELASTIC]
Segment1 = 4G, database.db = q1
Slices = 12
MaxSlices = 2048

$ more cluster_04/cluster.ini

[Cluster]
Master              = Host1
ThisHost            = Host4

[ELASTIC]
Segment1 = 4G, database.db = q1
Slices = 12
MaxSlices = 2048
```

2. Note assuming the same number of CPU threads on each machine i.e. 24, then the `Slices` param can be doubled to 12 for each node in `cluster.ini` , as above.
3. The cluster nodes can then be started on each machine, with 2 nodes being started on each in this case to form the cluster:

```
Machine 1 (hostname1)

$ virtuoso-start.sh
Starting Virtuoso instance in [cluster_01]
Starting Virtuoso instance in [cluster_02]
$

Machine 2 (hostname2)

$ virtuoso-start.sh
Starting Virtuoso instance in [cluster_03]
Starting Virtuoso instance in [cluster_04]
$
```

4. To stop the cluster use the standard `virtuoso-stop.sh` script on each machine, which automatically detects the number for nodes to be stopped:

```
Machine 1 (hostname1)

$ virtuoso-stop.sh
Stopping Virtuoso instance in [cluster_01]
Stopping Virtuoso instance in [cluster_02]

Machine 2 (hostname2)

$ virtuoso-stop.sh
Stopping Virtuoso instance in [cluster_03]
Stopping Virtuoso instance in [cluster_04]
$
```

5. The `cl_exec('shutdown')` command can also be run from `isql` on any node of the cluster to shutdown all nodes at once:

```
$ isql 12201
Connected to OpenLink Virtuoso
Driver: 07.10.3211 OpenLink Virtuoso ODBC Driver
OpenLink Interactive SQL (Virtuoso), version 0.9849b.
Type HELP; for help and EXIT; to exit.
```

```
SQL> cl_exec ('checkpoint');

Done. -- 2487 msec.
SQL> cl_exec ('shutdown');

*** Error 08S01: VD CL065: Lost connection to server
at line 2 of Top-Level:
cl_exec ('shutdown')
$
```

2.14.2. Virtuoso default Cluster Installation and Configuration

These sections apply to Virtuoso as of version 6.x and higher.

1. Run the Virtuoso 6 Unix (MacOsX) installer to perform a default installation:

```
$ tar xvf virtuoso-universal-server-6.1.tar
x install.sh
x universal-server.taz
$ sh install.sh

- Extracting Virtuoso Universal Server v6.1

- Creating default environment settings

- Creating default database settings
Configuring: database
Creating directory $VIRTUOSO_HOME/database/backup
Installing new virtuoso.ini in $VIRTUOSO_HOME/database
Installing new php.ini in $VIRTUOSO_HOME/database
Creating symlink to $VIRTUOSO_HOME/bin/virtuoso-iodbc-t

- Registering ODBC drivers

- Registering .NET provider for Mono

- Installing VAD packages in database (this can take some time)

- Checking where license should be stored

- Starting OpenLink License Manager

- Checking for initial Virtuoso license

- Starting Virtuoso server instance

- Finalizing installation

This concludes the first part of the installation.

- Attempting to start browser

Please start a browser manually and open the following URL to finalize
the installation process:

     http://cname:8890/install/

Installation completed
$
```

2. Determine how many nodes you want to start the cluster with: 2, 3, 4, 5, 6, 7, 8 etc.

   Note

   : the number of the nodes should be >= 2
3. Setup Virtuoso environment and stop default database:

```
$ . ./virtuoso-environment.sh
$ virtuoso-stop.sh
Shutting down Virtuoso instance in [database]
$
```

4. Optionally remove the default database such that is does not get started with the cluster:

```
rm  -rf database
```

5. Full list of supported options for the script

   "virtuoso-mkcluster.sh"

   :

```
  -cluster-size       Number of nodes in cluster
  -cluster-node       Node number in a cluster
  -cluster-port       Base portnumer for cluster
  -cluster-ipc-port   Base portnumer for cluster IPC
  -cluster_size       Size of the cluster
  -virtuoso_home      Virtuoso home path
```

6. Run the virtuoso-mkcluster.sh script to create the cluster, note the default number of nodes is 4, but this can be changed by setting the environment variable CLUSTER_SIZE to the required number of nodes to be created before running the script:

```
$ virtuoso-mkcluster.sh
Configuring cluster node: 1/4

Creating directory $VIRTUOSO_HOME/cluster_01
Creating directory $VIRTUOSO_HOME/cluster_01/backup
Creating directory $VIRTUOSO_HOME/cluster_01/logs
Installing new virtuoso.ini in $VIRTUOSO_HOME/cluster_01
Installing new php.ini in $VIRTUOSO_HOME/cluster_01
Creating symlink to $VIRTUOSO_HOME/bin/virtuoso-iodbc-t
Configuring cluster node: 2/4

Creating directory $VIRTUOSO_HOME/cluster_02
Creating directory $VIRTUOSO_HOME/cluster_02/backup
Installing new virtuoso.ini in $VIRTUOSO_HOME/cluster_02
Removing unneeded sections from virtuoso.ini
Creating symlink to $VIRTUOSO_HOME/bin/virtuoso-iodbc-t
Configuring cluster node: 3/4

Creating directory $VIRTUOSO_HOME/cluster_03
Creating directory $VIRTUOSO_HOME/cluster_03/backup
Installing new virtuoso.ini in $VIRTUOSO_HOME/cluster_03
Removing unneeded sections from virtuoso.ini
Creating symlink to $VIRTUOSO_HOME/bin/virtuoso-iodbc-t
Configuring cluster node: 4/4

Creating directory $VIRTUOSO_HOME/cluster_04
Creating directory $VIRTUOSO_HOME/cluster_04/backup
Installing new virtuoso.ini in $VIRTUOSO_HOME/cluster_04
Removing unneeded sections from virtuoso.ini
Creating symlink to $VIRTUOSO_HOME/bin/virtuoso-iodbc-t
$
```

7. For each of the cluster_XX directories created edit the cluster.ini file and set the HostXX parameter to the hostname:portno to be used by the cluster nodes for internal communication.

```
$ more cluster_01/cluster.ini
[Cluster]
Threads             = 20
Master              = Host1
ThisHost            = Host1
ReqBatchSize        = 10000
BatchesPerRPC       = 4
BatchBufferBytes    = 20000
LocalOnly           = 2
MaxKeepAlivesMissed = 1000
Host1               = hostname:22201
Host2               = hostname:22202
Host3               = hostname:22203
Host4               = hostname:22204
MaxHosts            = 5
$ more cluster_02/cluster.ini
[Cluster]
```

```
Threads             = 20
Master              = Host1
ThisHost            = Host2
ReqBatchSize        = 10000
BatchesPerRPC       = 4
BatchBufferBytes    = 20000
LocalOnly           = 2
MaxKeepAlivesMissed = 1000
Host1               = hostname:22201
Host2               = hostname:22202
Host3               = hostname:22203
Host4               = hostname:22204
MaxHosts            = 5
$ more cluster_03/cluster.ini
[Cluster]
Threads             = 20
Master              = Host1
ThisHost            = Host3
ReqBatchSize        = 10000
BatchesPerRPC       = 4
BatchBufferBytes    = 20000
LocalOnly           = 2
MaxKeepAlivesMissed = 1000
Host1               = hostname:22201
Host2               = hostname:22202
Host3               = hostname:22203
Host4               = hostname:22204
MaxHosts            = 5
$ more cluster_04/cluster.ini
[Cluster]
Threads             = 20
Master              = Host1
ThisHost            = Host4
ReqBatchSize        = 10000
BatchesPerRPC       = 4
BatchBufferBytes    = 20000
LocalOnly           = 2
MaxKeepAlivesMissed = 1000
Host1               = hostname:22201
Host2               = hostname:22202
Host3               = hostname:22203
Host4               = hostname:22204
MaxHosts            = 5
$
```

8. Start the cluster using the standard virtuoso-start.sh script which automatically detects the number for nodes to be started:

```
$ virtuoso-start.sh
Starting Virtuoso instance in [cluster_01]
Starting Virtuoso instance in [cluster_02]
Starting Virtuoso instance in [cluster_03]
Starting Virtuoso instance in [cluster_04]
$
```

9. The default SQL port of the master node is 12201, as indicated in the virtuoso.ini file of the cluster_01 directory, and can then be used for connecting to the newly created cluster and check its status to ensure all nodes are online:

```
$ isql 12201
Connected to OpenLink Virtuoso
Driver: 06.01.3127 OpenLink Virtuoso ODBC Driver
OpenLink Interactive SQL (Virtuoso), version 0.9849b.
Type HELP; for help and EXIT; to exit.
SQL> status ('cluster');
REPORT
VARCHAR
_______________________________________________________________________________

Cluster 4 nodes, 4 s. 1 m/s 0 KB/s  0% cpu 0%  read 0% clw threads 1r 0w 0i buffers 2981 0 d 0 w 0

1 Rows. -- 7 msec.
SQL> cl_ping(1,500, 1000);

Done. -- 7 msec.
SQL> cl_ping(2,500, 1000);
```

```
Done. -- 52 msec.
SQL> cl_ping(3,500, 1000);

Done. -- 60 msec.
SQL> cl_ping(4,500, 1000);

Done. -- 51 msec.
SQL>
```

10. To split the node across across physical machines for better performance and scalability, simply perform a parallel Virtuoso installation on the additional physical machines and move the cluster nodes required to the designated machine. Example, for the default 4 node cluster to be split across two identical machines it would make sense to split 2 nodes across each machine, thus you would move say the cluster_03 and cluster_04 directory nodes to the new machine (removing them from the original). The cluster.ini file on each node would then need to be updated to set the HostXX parameters to point to the new locations for nodes 03 and 04:

```
Machine 1 (hostname1)

$ more cluster_01/cluster.ini
[Cluster]
Threads             = 20
Master              = Host1
ThisHost            = Host1
ReqBatchSize        = 10000
BatchesPerRPC       = 4
BatchBufferBytes    = 20000
LocalOnly           = 2
MaxKeepAlivesMissed = 1000
Host1               = hostname1:22201
Host2               = hostname1:22202
Host3               = hostname2:22203
Host4               = hostname2:22204
MaxHosts            = 5
$ more cluster_02/cluster.ini
[Cluster]
Threads             = 20
Master              = Host1
ThisHost            = Host2
ReqBatchSize        = 10000
BatchesPerRPC       = 4
BatchBufferBytes    = 20000
LocalOnly           = 2
MaxKeepAlivesMissed = 1000
Host1               = hostname1:22201
Host2               = hostname1:22202
Host3               = hostname2:22203
Host4               = hostname2:22204
MaxHosts            = 5
$

Machine 2 (hostname2)

$ more cluster_03/cluster.ini
[Cluster]
Threads             = 20
Master              = Host1
ThisHost            = Host3
ReqBatchSize        = 10000
BatchesPerRPC       = 4
BatchBufferBytes    = 20000
LocalOnly           = 2
MaxKeepAlivesMissed = 1000
Host1               = hostname1:22201
Host2               = hostname1:22202
Host3               = hostname2:22203
Host4               = hostname2:22204
MaxHosts            = 5
$ more cluster_04/cluster.ini
[Cluster]
Threads             = 20
Master              = Host1
ThisHost            = Host4
```

```
ReqBatchSize        = 10000
BatchesPerRPC       = 4
BatchBufferBytes    = 20000
LocalOnly           = 2
MaxKeepAlivesMissed = 1000
Host1               = hostname1:22201
Host2               = hostname1:22202
Host3               = hostname2:22203
Host4               = hostname2:22204
MaxHosts            = 5
$
```

11. The nodes can then be started on each machine, with 2 nodes being started on each in this case to form the cluster:

```
Machine 1 (hostname1)

$ virtuoso-start.sh
Starting Virtuoso instance in [cluster_01]
Starting Virtuoso instance in [cluster_02]
$

Machine 2 (hostname2)

$ virtuoso-start.sh
Starting Virtuoso instance in [cluster_03]
Starting Virtuoso instance in [cluster_04]
$
```

12. To stop the cluster use the standard virtuoso-stop.sh script which automatically detects the number for nodes to be stopped:

```
$ virtuoso-stop.sh
Stopping Virtuoso instance in [cluster_01]
Stopping Virtuoso instance in [cluster_02]
Stopping Virtuoso instance in [cluster_03]
Stopping Virtuoso instance in [cluster_04]
$
```

2.14.3. Backup and Restore

Backup

There are 2 ways to backup Virtuoso Cluster DB:

1. Backup every node using `back_online` *For example, for every cluster in its backup folder execute:*

```
SQL> backup_online('dbp',10000000,0,vector('backup'));

Done. -- 272 msec.
```

2. Make backup of all nodes at once using cl_exec() . For example, execute:

```
SQL> cl_exec ('backup_online (''dbp'', 10000000, 0,  vector (''backup''))');

Done. -- 573 msec.
```

Restore from Backup

After backup is done for every node in its backup folder, in order to restore, execute from the ..\bin folder the virtuoso-restore.sh:

```
# . ./virtuoso-restore.sh all dbp
```

where the second parameter is the restore prefix.

2.14.4. Cluster Configuration

1. Set "FAST_START=1" in the viruoso-start.sh file and then run:

```
viruoso-start.sh
```

or

2. Execute the following line:

```
# /etc/init.d/virtuoso start
Starting OpenLink Virtuoso:                                    [  OK  ]
```

3. In order to check the nodes, connect to port 12201 using the ISQL tool:

```
isql 12201
```

4. To check the cluster status, execute the following command:

```
SQL> status('cluster');
REPORT
VARCHAR
_______________________________________________________________________________

Cluster 4 nodes, 293 s. 0 m/s 0 KB/s  0% cpu 0%  read 0% clw threads 1r 0w 0i buffers 1781 0 d 0 w

1 Rows. -- 4 msec.
```

2.14.5. HTTP Service Configuration on Subordinate Nodes of a Virtuoso Cluster

This section applies to Virtuoso as of version 6.x and higher.

What

This documentation details how to configure the Subordinate (also called Slave) Nodes of a Virtuoso Elastic Cluster to service HTTP clients.

Why

By default, only the Primary (also called Master) instance of a Virtuoso Elastic Cluster is configured to provide HTTP services.

The Subordinate (also called Slave) nodes of the cluster may also be configured to provide HTTP services, enabling load balancing by spreading HTTP requests across the cluster's nodes.

How

This documentation details the steps for the installation and configuration of a Virtuoso Elastic Cluster on Unix:

1. *Step 1*

   : Set up each instance as a HTTP Server;
2. *Step 2*

   : Install and configure HTTP services on each instance;
3. *Step 3*

   : Configure load balancing.

Set up each instance as an HTTP Server

Step 1 : Set up each instance as a HTTP Server

Each node can be configured to provide HTTP services as follows:

1. Copy the `[HTTP Server]` section from the Primary instance's configuration file (by default, `virtuoso.ini`) to the configuration file of each Subordinate instance:

```
[HTTPServer]
ServerPort                 = 8890
ServerRoot                 = ../vsp
DavRoot                    = DAV
EnabledDavVSP              = 0
HTTPProxyEnabled           = 0
```

```
TempASPXDir                 = 0
DefaultMailServer           = localhost:25
MaxClientConnections        = 5
MaxKeepAlives               = 10
KeepAliveTimeout            = 10
MaxCachedProxyConnections   = 10
ProxyConnectionCacheTimeout = 15
HTTPThreadSize              = 280000
HttpPrintWarningsInOutput   = 0
Charset                     = UTF-8
;HTTPLogFile                 = logs/http.log
MaintenancePage             = atomic.html
EnabledGzipContent          = 1
```

2. Edit the `ServerPort` parameter to make it unique on the machine hosting this instance; i.e., if a subordinate instance is running on same physical node as the primary instance, then the subordinate's HTTP port must to be changed (from 8890, for instance) to a unique port (e.g., 8891).
3. Install the Virtuoso Conductor to enable HTTP Administration of the instance being configured. Note: if the subordinate instance is not on the same machine as the primary instance, then the vad directory may also need to be copied from the primary instance to the subordinate instance.:

```
SQL> vad_install ('../vad/conductor_dav.vad', 0);
SQL_STATE  SQL_MESSAGE
VARCHAR  VARCHAR
_______________________________________________________________________________

00000    No errors detected
00000    Installation of "Virtuoso Conductor" is complete.
00000    Now making a final checkpoint.
00000    Final checkpoint is made.
00000    SUCCESS

6 Rows. -- 10263 msec.
SQL>
```

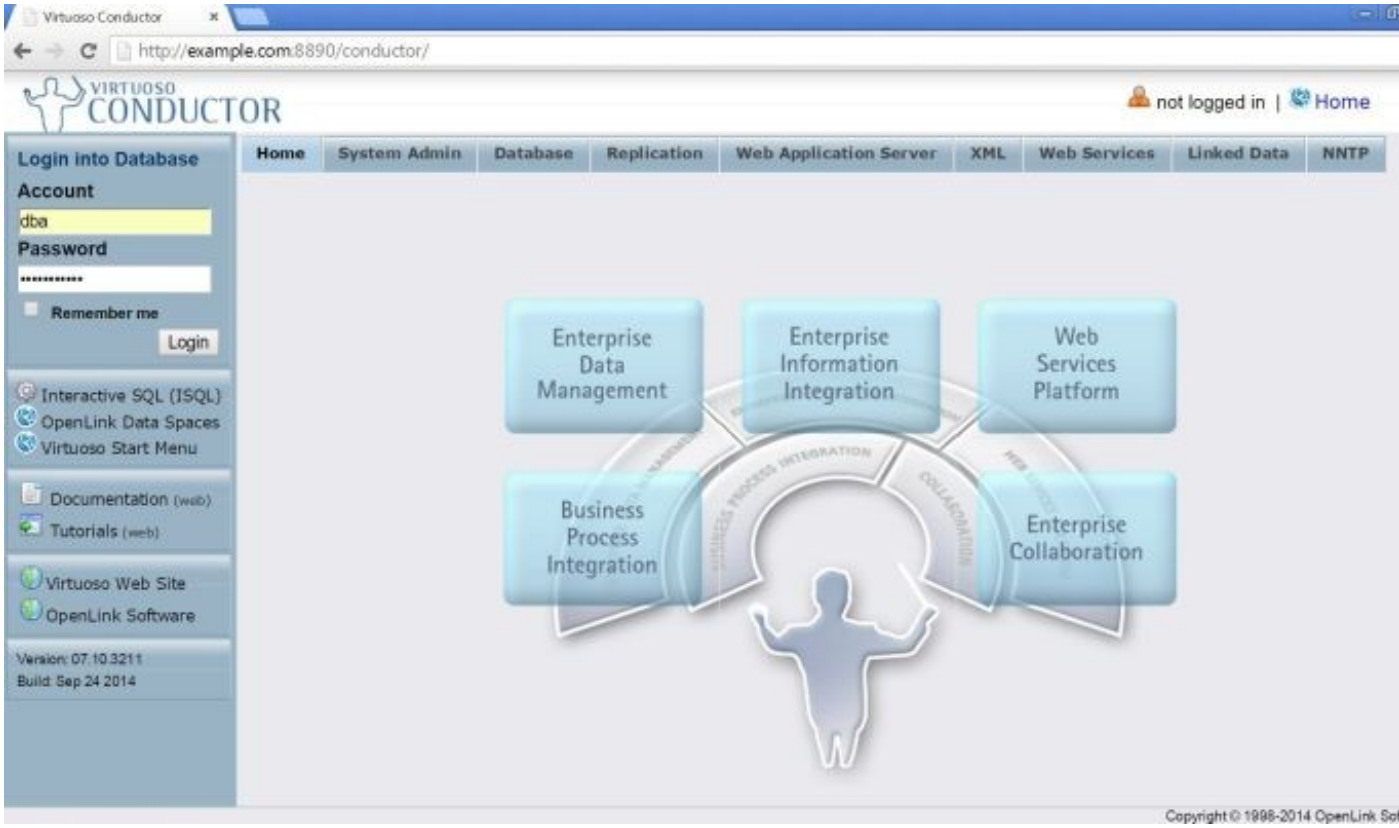
Install and configure HTTP services on each instance

Step 2 : Install and configure HTTP services on each instance

Any HTTP services required on the subordinate instance will need to specifically installed or configured on that physical node. For example, the Virtuoso default SPARQL endpoint (`/sparql`) may be configured by:

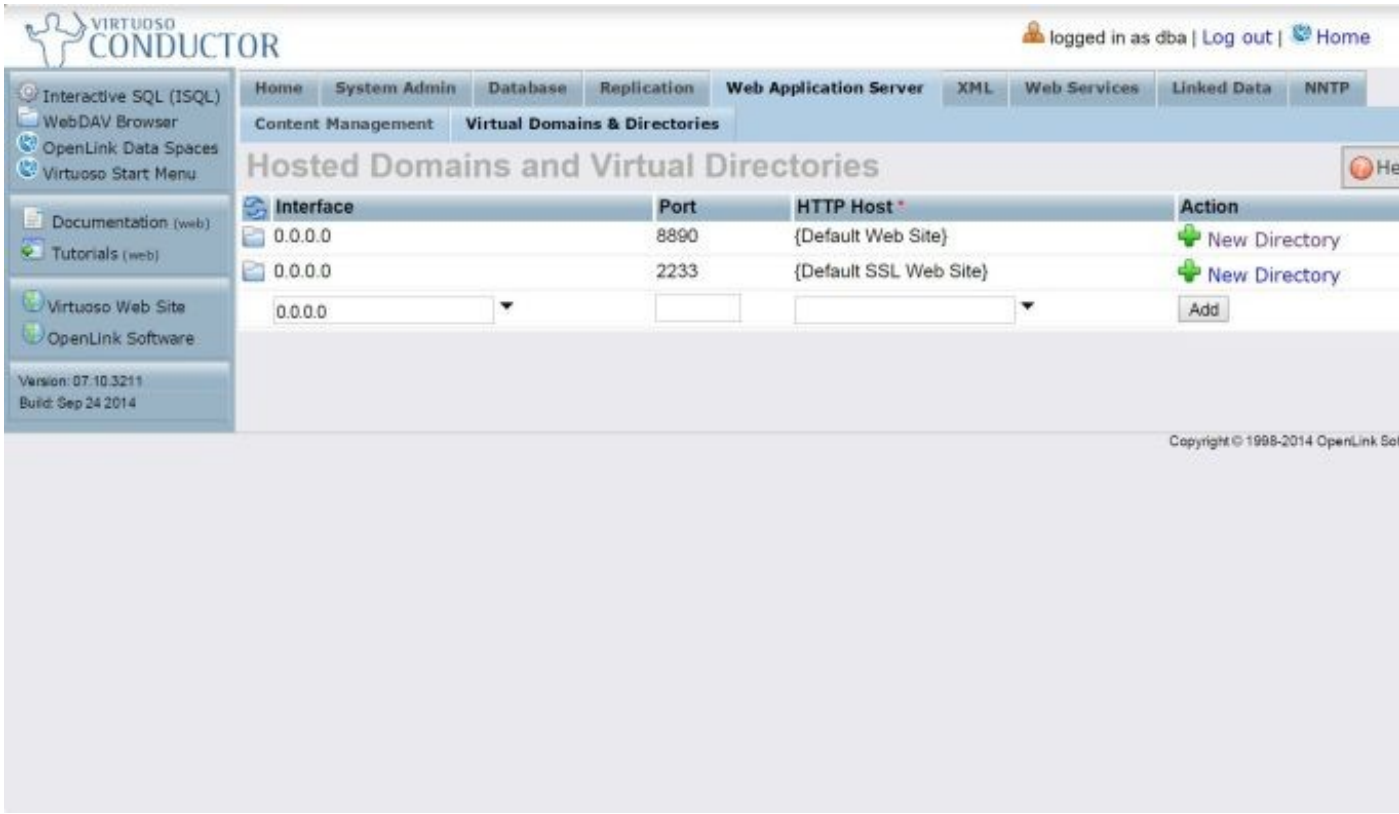
1. Log in into the Virtuoso Conductor http://hostname:port/conductor :

Figure 2.185. Configure SPARQL Endpoint: log in



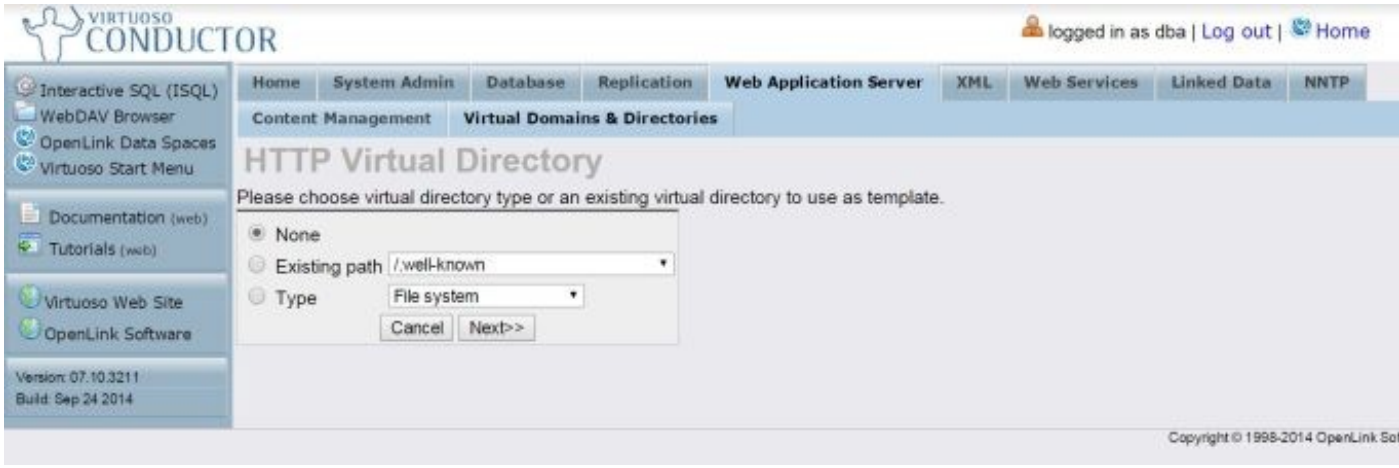
2. Go the the Web Application Server -> Virtual Domains & Directories tab:

Figure 2.186. Configure SPARQL Endpoint: Virtual Domains and Directories



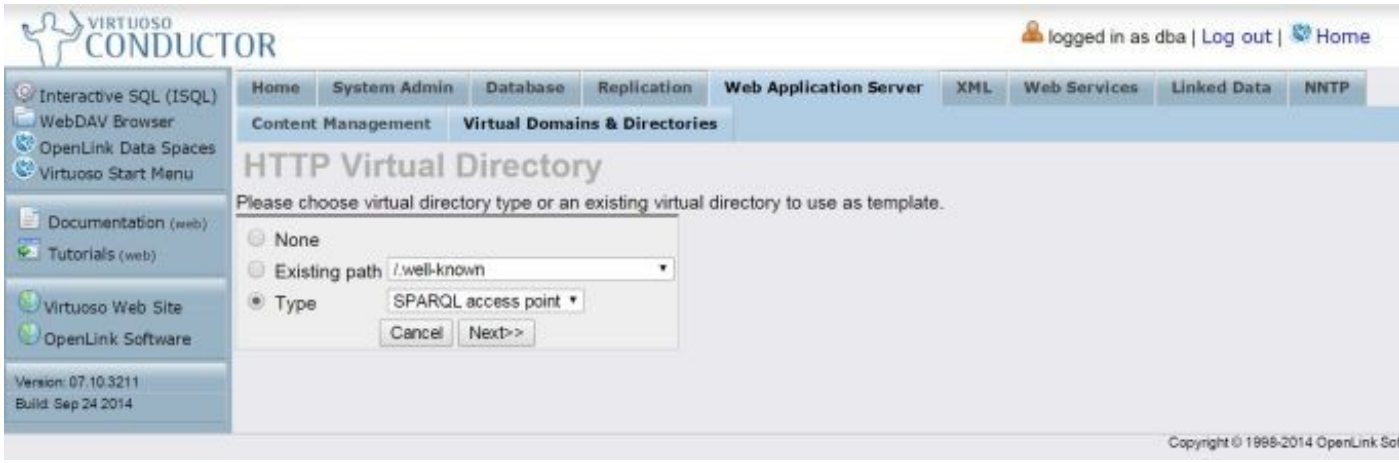
3. Select the New Directory Action for the Default Web Site HTTP host:

Figure 2.187. Configure SPARQL Endpoint: new directory



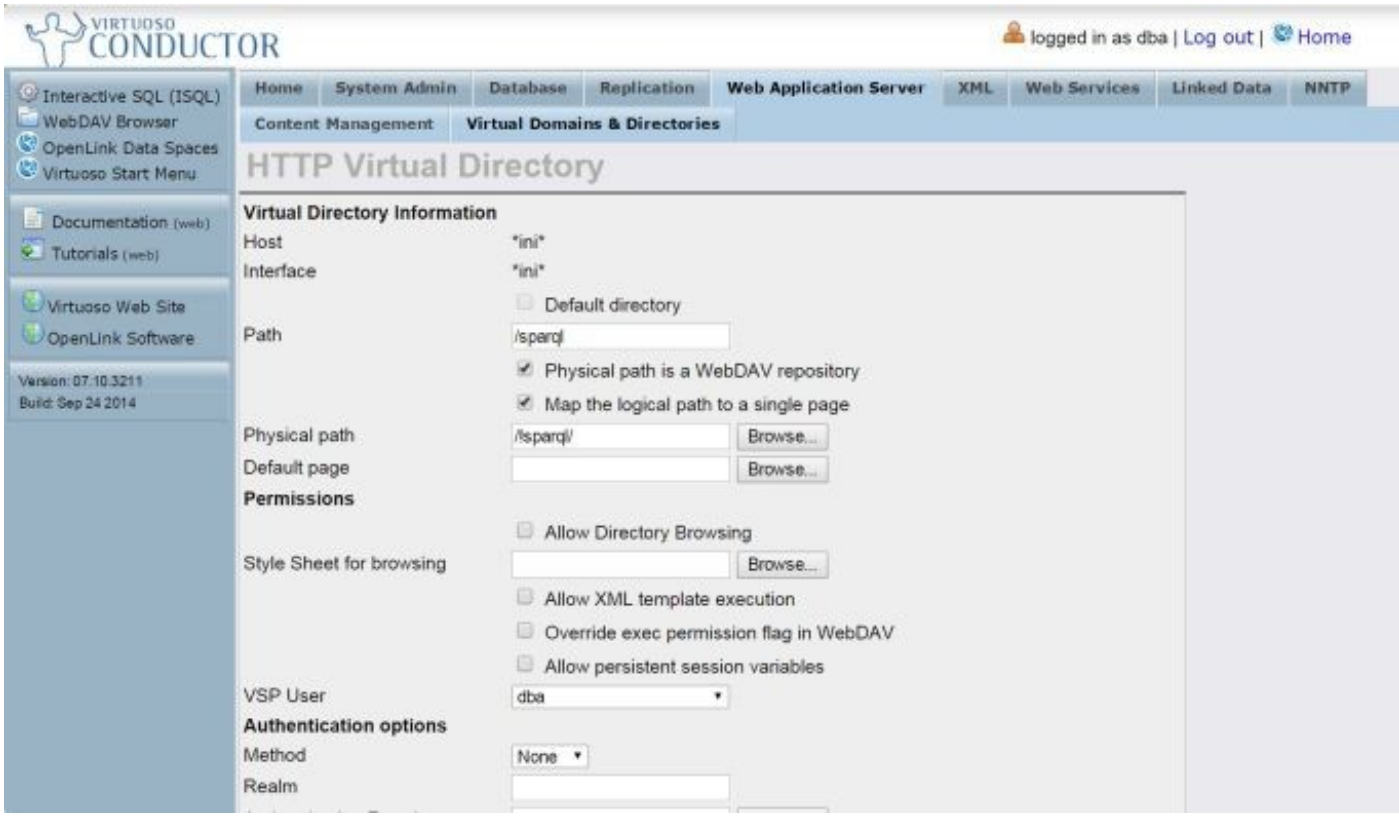
4. Select the Type radio button and SPARQL access point item from the drop down list box:

Figure 2.188. Configure SPARQL Endpoint: set type SPARQL



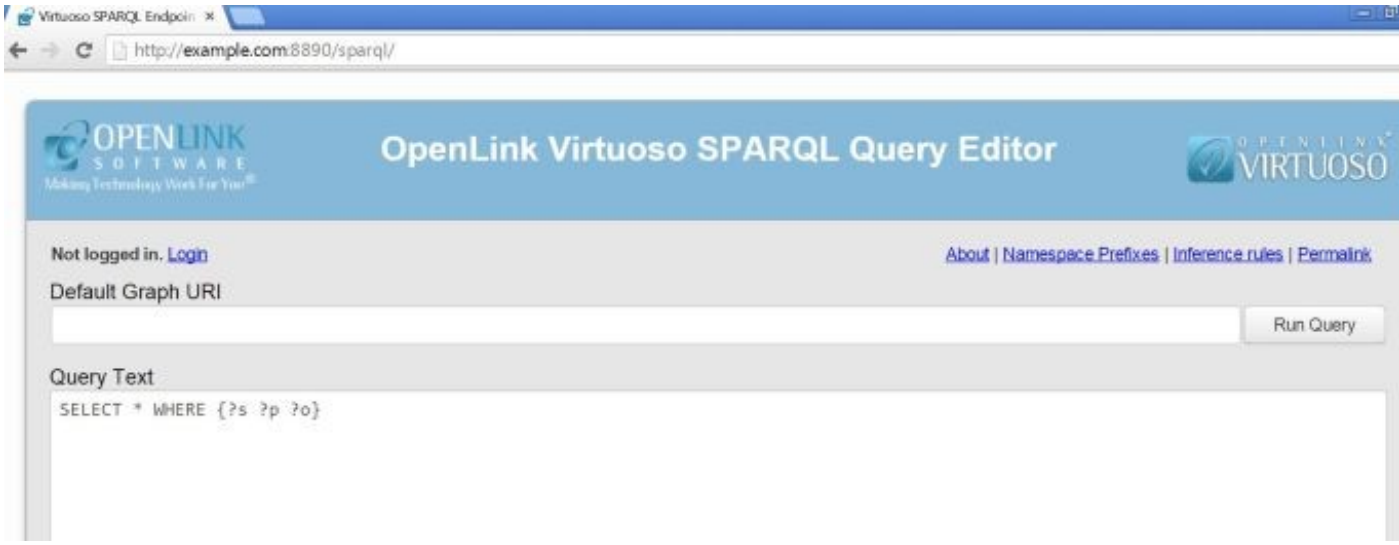
5. Click "Next".
6. Enter /sparql as the Path param in the Virtual Directory Information section and click Save Changes:

Figure 2.189. Configure SPARQL Endpoint: set /sparql virtual directory



7. The SPARQL endpoint will not be accessible on http://hostname:port/sparql the the newly configured slave nodes:

Figure 2.190. Configure SPARQL Endpoint: SPARQL Endpoint



8. Further details on SPARQL endpoint configuration can be found in Service Endpoint documentation section.
9. Typical Virtuoso server log output from a slave node when started, showing the HTTP server running on port 8890, being:

```
20:12:49 OpenLink Virtuoso Universal Server
20:12:49 Version 07.10.3209-pthreads for Linux as of Apr 26 2014
20:12:49 uses parts of OpenSSL, PCRE, Html Tidy
20:12:49 Registered to OpenLink Virtuoso (Internal Use)
20:12:49 Personal Edition license for 500 connections
20:12:49 Issued by OpenLink Software
20:12:49 This license will expire on Sun May 17 06:18:35 2015 GMT
20:12:49 Enabled Cluster Extension
20:12:49 Enabled Column Store Extension
20:12:57 Database version 3126
20:12:57 SQL Optimizer enabled (max 1000 layouts)
```

```
20:12:58 Compiler unit is timed at 0.000208 msec
20:12:58 Roll forward started
20:12:58 Roll forward complete
20:12:59 Checkpoint started
20:12:59 Checkpoint finished, log reused
20:12:59 HTTP/WebDAV server online at 8890
20:12:59 Server online at 12202 (pid 15969)
```

Configure load balancing

Step 3 : Configure load balancing

A reverse-proxy service (like Nginx or Apache) can then be configured such that requests are proxied across as any or all nodes of the cluster, to provide the desired load balancing.

Additional Information

- Only the Primary Node of an Elastic Cluster may be configured as a Publisher for Virtuoso Replication Cluster purposes.
- The Virtuoso 500 billion triple Berlin SPARQL Benchmark (BSBM) dataset runs were performed on a 24-node Elastic Cluster. Each node was configured to provide HTTP services and a SPARQL endpoint, and the query load was spread over the entire cluster.

2.14.6. Troubleshooting Tips

If an operation seems to hang, see the output of:

```
status ()
```

Check for the presence of the following conditions:

- The cluster line shows 0% CPU, no message traffic and an unchanging number of buffers wired, this is probably a bug. To reset, restart the cluster or the offending process if found. Restart is done by executing:

  ```
  raw_exit ();
  ```

  over an SQL connection to the process in question.
- The cluster line shows many threads waiting compared to total threads. If CPU is 0 and this does not change there could be a transaction that holds locks indefinitely. To clear, execute:

  ```
  txn_killall (1);
  ```

  Do this at a node that has local threads waiting. This is seen in the Lock Status paragraph of status (") when connected to the node in question.
- The cluster line shows a changing number in the pfs field. The system is swapping and slowed down.
- If the status () itself hangs, try another process of the cluster. See that there is no temporary atomic activity like a long checkpoint. If the situation persists there is a bug. The checkpoint can be seen by the presence of the

  checkpinmt_in_progress

  file in each server's working directory.
- To check the integrity of database files, do:

  ```
  cl_exec ('backup ''/dev/null''');
  ```

  If this returns, the databases are OK. If one is found to be corrupt the corresponding server exits.

2.14.7. See Also:

Tip

Setting up and operating Virtuoso on a cluster.

Virtuoso Cluster Programming

Tip

Virtuoso Cluster Fault Tolerance.

Chapter 3. Quick Start & Tours

Abstract

This chapter is aimed at getting Virtuoso up and running quickly with a few basic examples to demonstrate key concepts.

The Virtuoso quick start tour is designed to familiarize you with some of the components in Virtuoso. After completing the tour, you should be able to link new tables into Virtuoso, and query the tables.

3.1. Where to Start

3.1.1. Default Passwords

When you start up Virtuoso for the first time, there are 3 user accounts defined:

Table 3.1. Default users of Virtuoso

User Name	Default Password	Usage
dba	dba	Default Database Administrator account.
dav	dav	WebDAV Administrator account.
vad	vad	WebDAV account for internal usage in VAD (disabled by default).
demo	demo	Default demo user for the demo database. This user is the owner of the Demo catalogue of the demo database.
soap	soap	SQL User for demonstrating SOAP services.
fori	fori	SQL user account for 'Forums' tutorial application demonstration in the Demo database.

One database user and 2 WebDAV users. These users have their passwords set to default values. It is therefore important to change them immediately after the installation.

The one database user is the database administrator with username "dba" and password "dba". This can be changed using the Interactive SQL utility. When started without parameters, the ISQL tries to log on as dba with the default password. The SQL statement to change a user's password is:

```
set password <old password> <new password>
```

The password is an identifier, so take care to use proper quotation.

You can also use the graphical Virtuoso Administration Interface to administer Virtuoso database users.

The 2 WebDAV user accounts, dav and davuser also have their password set to their username. There are 2 easy ways to change them. Either use the GUI in Administration Interface under WebDAV Administration / WebDAV services / Users Administrator or use the SQL statement:

```
update WS.WS.SYS_DAV_USER set U_PASSWORD='<new password>'
where U_NAME='<username>'
```

Note quotation around varchar values. Please remember to perform these operations for all Virtuoso server instances installed. By default these are the Virtuoso with an empty database and Virtuoso [demo] with the demo database.

3.1.2. Post-Installation Sanity Check

Verify by ISQL

Verify usability of your Virtuoso server by executing the following command from your command line prompt:

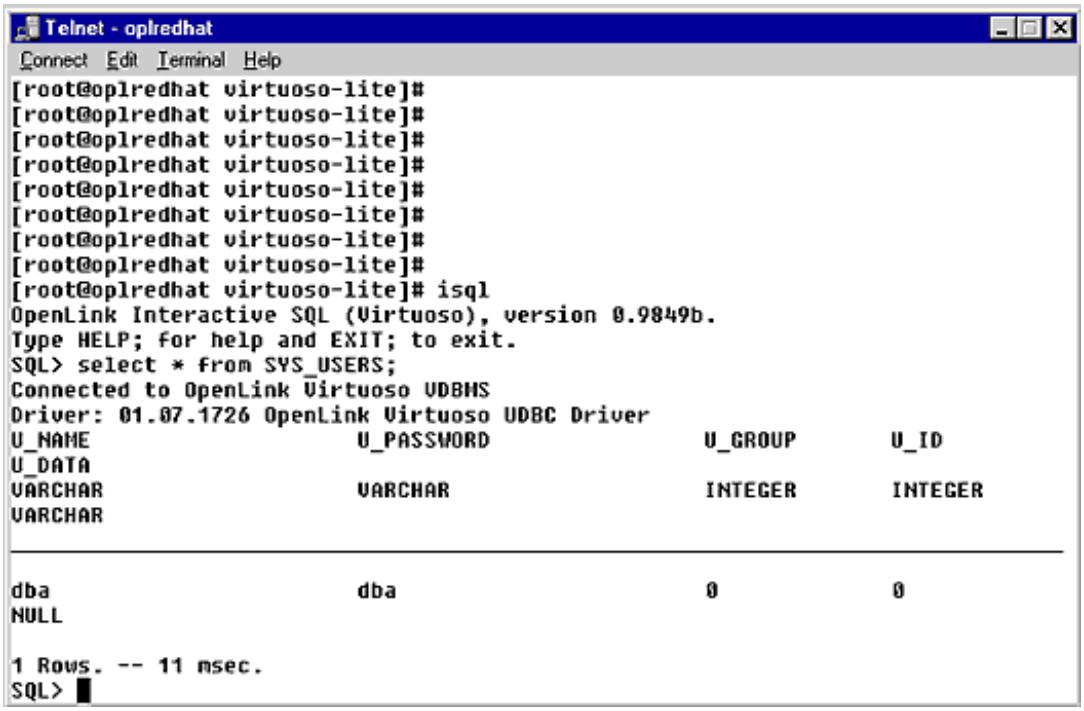
```
isql
```

From the ISQL prompt enter the following SQL command:

```
select * from DB.DBA.SYS_USERS;
```

This should produce a resultset containing one record if everything has been implemented correctly to this point.

Figure 3.1. ISQL in Telnet



Verify by HTTP

A quick way to check that the database is running, is to point a browser to the http port. The following example URLs will show the System Manager for the default, and the demo Virtuoso databases:

```
http://example.com:8889/
http://example.com:8890/
http://a_virtuoso_server.org:8890/
```

On a Windows Client there is a shortcut to the *OpenLink Virtuoso Conductor* in the OpenLink Virtuoso program group.

You will be presented with the OpenLink Virtuoso Conductor screen:

Figure 3.2. Virtuoso Conductor

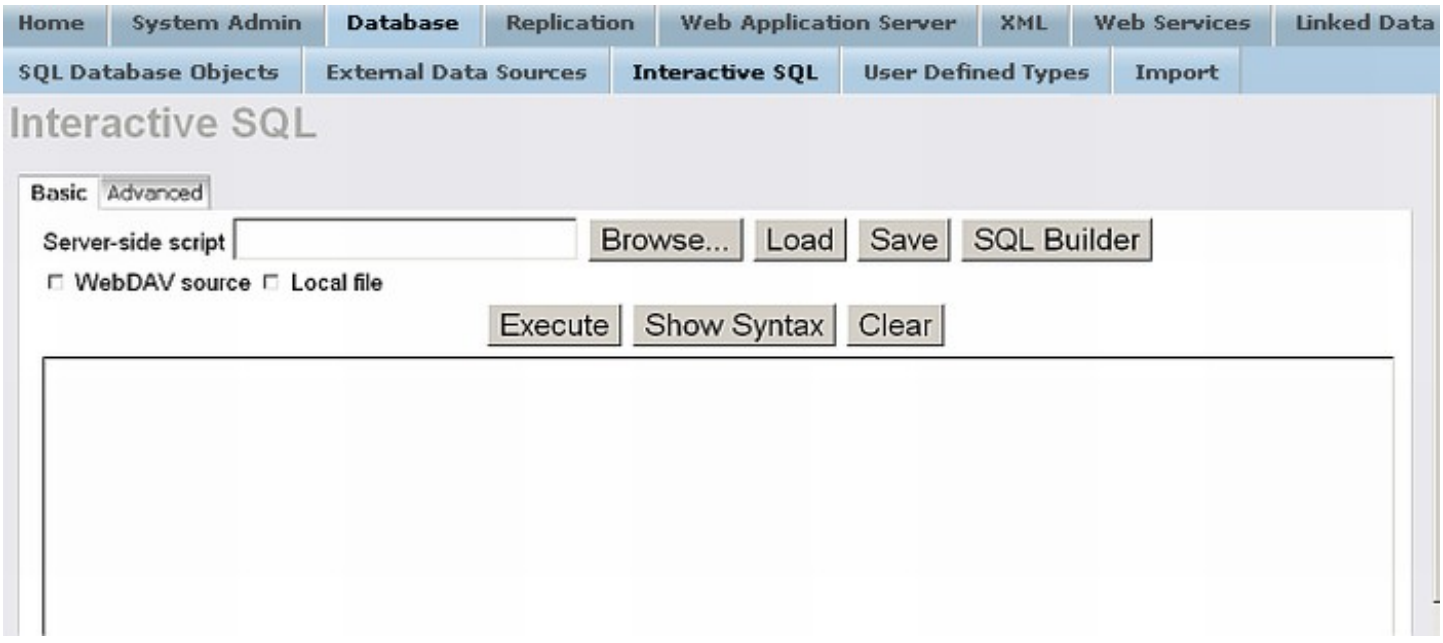
Verify by web based SQL query

Click on the *Conductor* link to enter the Virtuoso Server Administration Conductor Interface. You will be presented with a login form, type in the correct details for the database DBA user, by default this is username=dba; password=dba.

Figure 3.3. Virtuoso Conductor - Login Form

Got to tab "Database" and then go to tab "Interactive SQL".

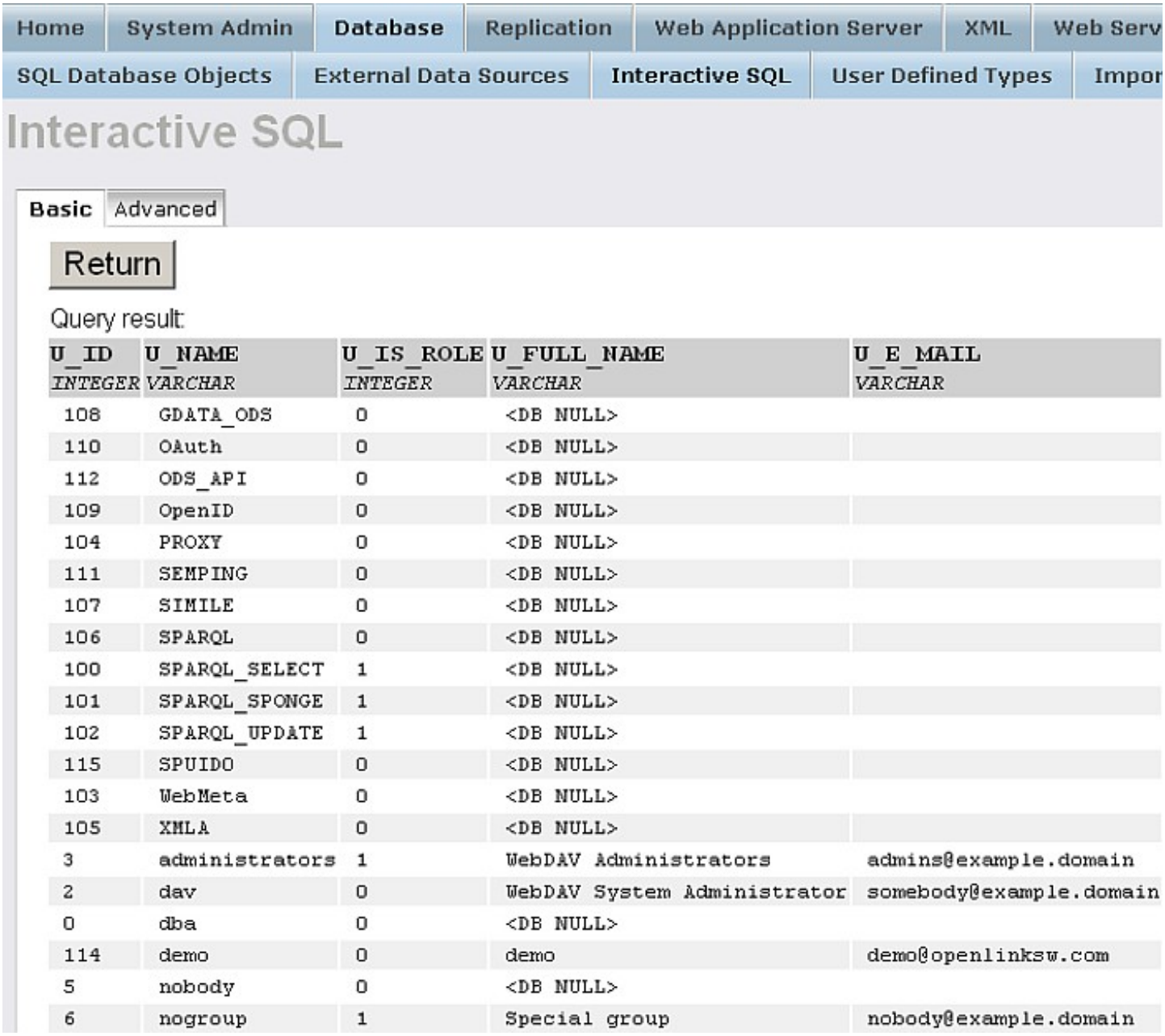
Figure 3.4. Virtuoso Conductor - Interactive SQL



Enter the SQL Statement command "SELECT * FROM SYS_USERS" in the SQL Statement text area. Note that only valid SQL can be supplied, so you cannot type a database command such as "tables;". Also, note that the ";" is not valid in this context. Press *Execute* .

You should see the SQL results, as shown below.

Figure 3.5. Virtuoso Conductor - SQL Results



U_ID *INTEGER*	U_NAME *VARCHAR*	U_IS_ROLE *INTEGER*	U_FULL_NAME *VARCHAR*	U_E_MAIL *VARCHAR*
108	GDATA_ODS	0	<DB NULL>	
110	OAuth	0	<DB NULL>	
112	ODS_API	0	<DB NULL>	
109	OpenID	0	<DB NULL>	
104	PROXY	0	<DB NULL>	
111	SEMPING	0	<DB NULL>	
107	SIMILE	0	<DB NULL>	
106	SPARQL	0	<DB NULL>	
100	SPARQL_SELECT	1	<DB NULL>	
101	SPARQL_SPONGE	1	<DB NULL>	
102	SPARQL_UPDATE	1	<DB NULL>	
115	SPUIDO	0	<DB NULL>	
103	WebMeta	0	<DB NULL>	
105	XMLA	0	<DB NULL>	
3	administrators	1	WebDAV Administrators	admins@example.domain
2	dav	0	WebDAV System Administrator	somebody@example.domain
0	dba	0	<DB NULL>	
114	demo	0	demo	demo@openlinksw.com
5	nobody	0	<DB NULL>	
6	nogroup	1	Special group	nobody@example.domain

3.1.3. Administering Your Virtuoso Installation

Virtuoso has been designed for Web based Administration. With your Virtuoso server running you will be able to point a Web browser at the servers listening address and port. The installation will include the default server and optionally the demo database server. The default server listens on port 8890 whilst the demo server listens on port 8889. If you are using the machine where Virtuoso was installed then the server address can be localhost or the hostname otherwise you will need to know the hostname or IP address of the machine where Virtuoso was installed. The URL that you will want to point your browser may be:

```
http://example.com:8890/
```

Note that that trailing / is important and may be required for older browsers

Figure 3.6. Visual Server Administration Interface

The "Conductor" link will take you to the Visual Server Administration Interface. Information about Visual Server Administration Interface can be found in the Server Administration chapter , which describes all the Interfaces available for configuring your server.

3.2. Client Connections

Virtuoso supports a number of data access API's such as ODBC and JDBC. They both provide high performance native connectivity to the Virtuoso database system.

3.2.1. ODBC

The Virtuoso driver for ODBC is available for Windows and most Unix operating systems. For Windows the you can configure a data source that utilizes the driver in the familiar way via the ODBC Administrator. This is very easy windows and wizards based process. On Unix the process has historically not been so simple. We have been trying to a achieve the same level of usability under Unix through our support of the iODBC project . For more information and details about how to configure the Virtuoso driver for ODBC please go to the Virtuoso Driver for ODBC section.

3.2.2. JDBC

The Virtuoso driver for JDBC is a type 4 native driver which is available for all Java platforms. This means that you have a single jar file, for which to include into your classpath, and then the driver is ready for use by your applications. The connection URL typically consists of:

```
jdbc:virtuoso://hostname:port/UID=uid/PWD=pwd
```

More information about the JDBC driver can be found in the Virtuoso Driver for JDBC section. The JDBC driver also supports SSL encrypted connections. This JDBC driver is an excellent companion to any web enabled applications especially when combined with Virtuoso's other features such as WSDL and SOAP.

3.2.3. OLEDB

Virtuoso gives application developers an opportunity to utilize OLE DB interfaces for their database programming needs. OLE DB is a Microsoft developed and promoted technology for uniform data access across diverse data sources. Many applications make indirect use of OLE DB through a higher level ADO interface.

VIRTOLEDB is the native OLE DB provider for Virtuoso . This enables applications to access a Virtuoso server using OLE DB interfaces. VIRTOLEDB provides implementation for all required and many optional OLE DB interfaces.

3.3. Virtual Database Server

Virtuoso's Virtual Database Engine enables you to produce Dynamic Web Content from any major database management system. This enables dynamic, real-time HTML and XML generation from any number of different database engines concurrently. The Visual Server Administration Interface of Virtuoso allows you to effortlessly remotely choose and configure remote data sources to be linked into Virtuoso. Once a table is linked into Virtuoso then it is usable like any native table leaving you free to perform join queries without ever worrying about the underlying data source.

The Virtual Database (VDB) Engine section of the Conceptual Overview chapter explains the concepts in more details.

Visit the Visual Server Administration Interface section to see how to link tables into Virtuoso, or follow the small example below to get you started:

3.3.1. Configuring Your ODBC Data Sources

Before you can link a table into Virtuoso, you need to configure an ODBC datasource. It is quite likely that you already have ODBC data sources defined. If so, you can skip this part of the tour.

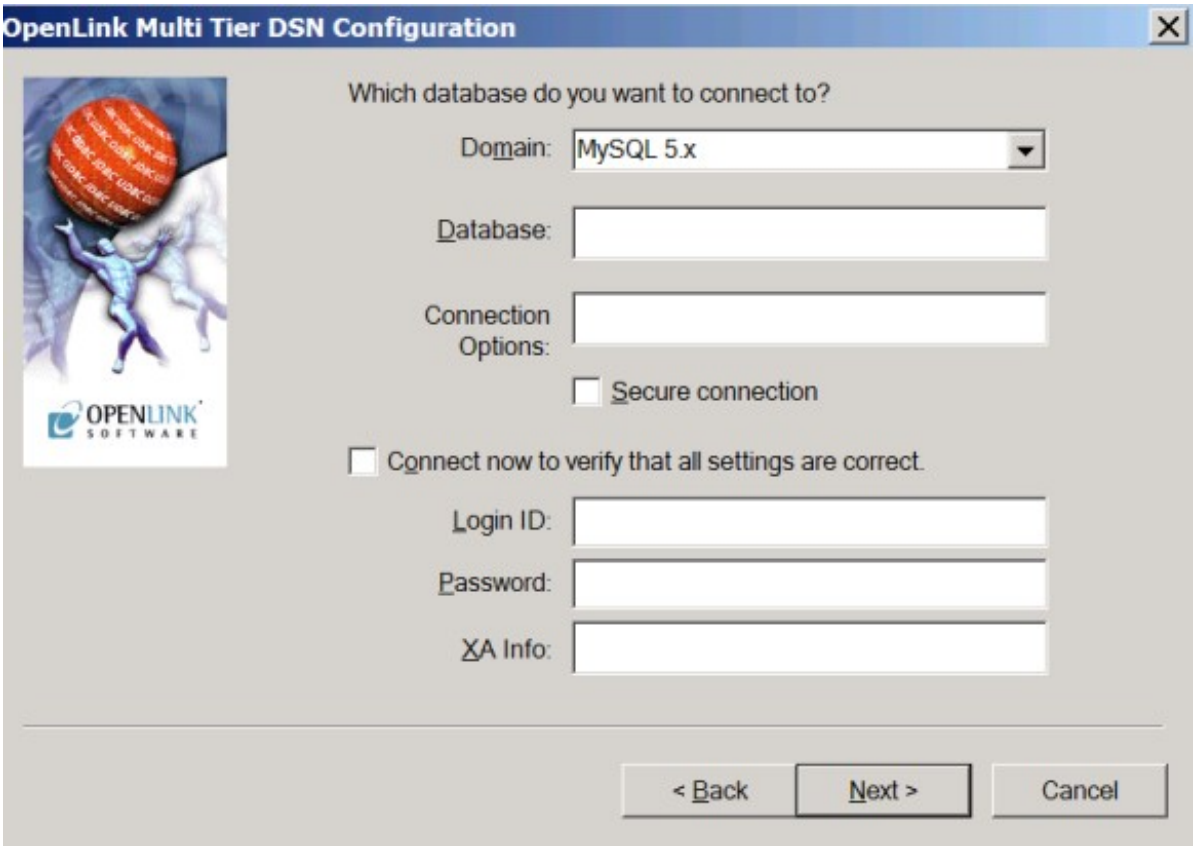
If you do not have any data sources defined, you should configure your datasource using an appropriate ODBC driver. The ODBC driver may come from the database vendor or a third-party middleware vendor, such as OpenLink Software. You will need to configure your ODBC datasource in accordance with the directions from your driver vendor.

In this part of the tour, we show sample DSN configuration information for some database vendor drivers. These examples of working data sources are not meant to replace the instructions from the driver vendor. In your specific installation other parameters may be necessary.

To define ODBC data sources on Windows XP/95/98/NT, in the Control Panel go to Administrative Tools and then click the Data Sources (ODBC) icon.

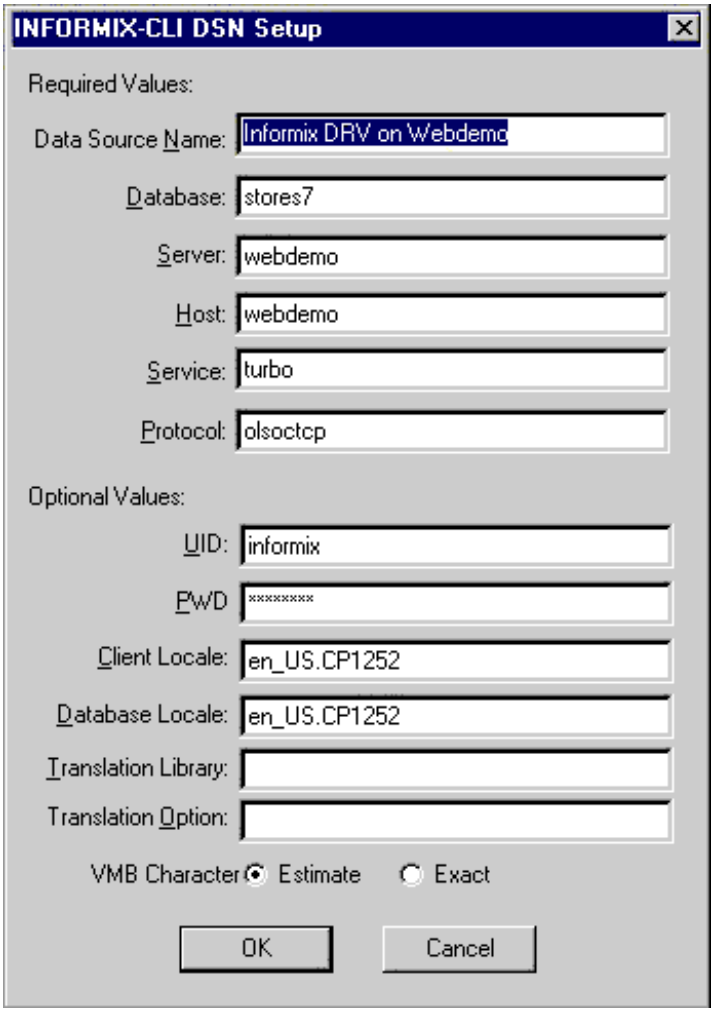
This first datasource uses OpenLink Generic ODBC Driver V6.0 to create DSN to a MySQL 5.x database.

Figure 3.7. OpenLink Mutli Tier DSN Configuration with database MySQL 5.x



The next example is an Informix 7 datasource using the driver from Informix Software, Inc.

Figure 3.8. Informix Driver DSN Configuration



The next few images show a Microsoft SQL Server 6.5 datasource using the Microsoft Corporation SQL Server driver.

Figure 3.9. MS SQL Server DSN Configuration

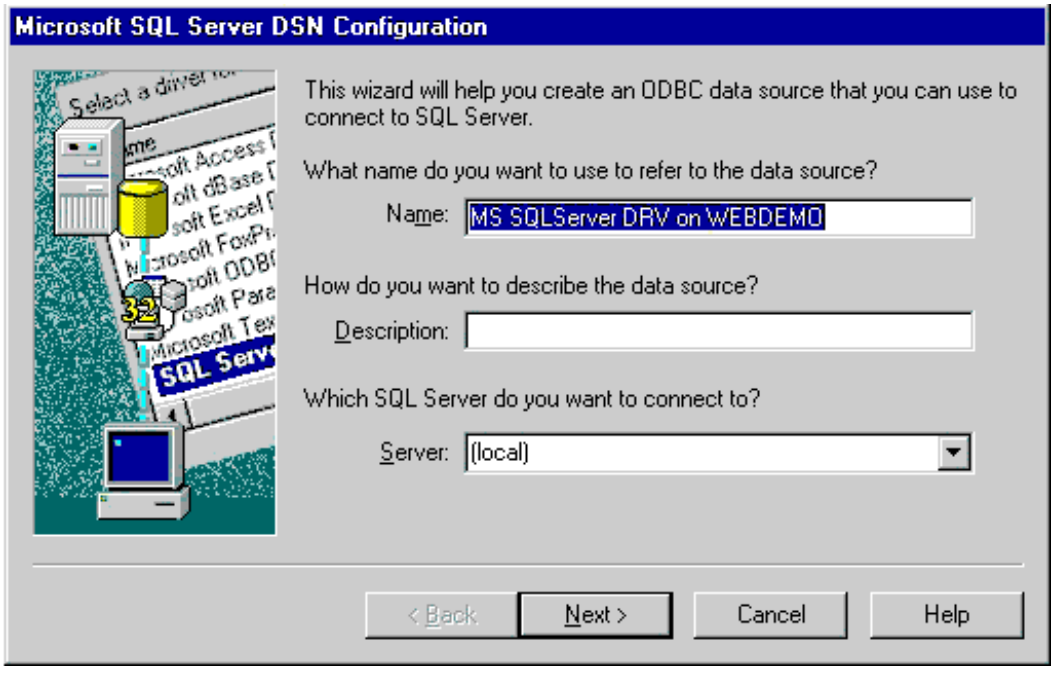


Figure 3.10. MS SQL Server DSN Configuration

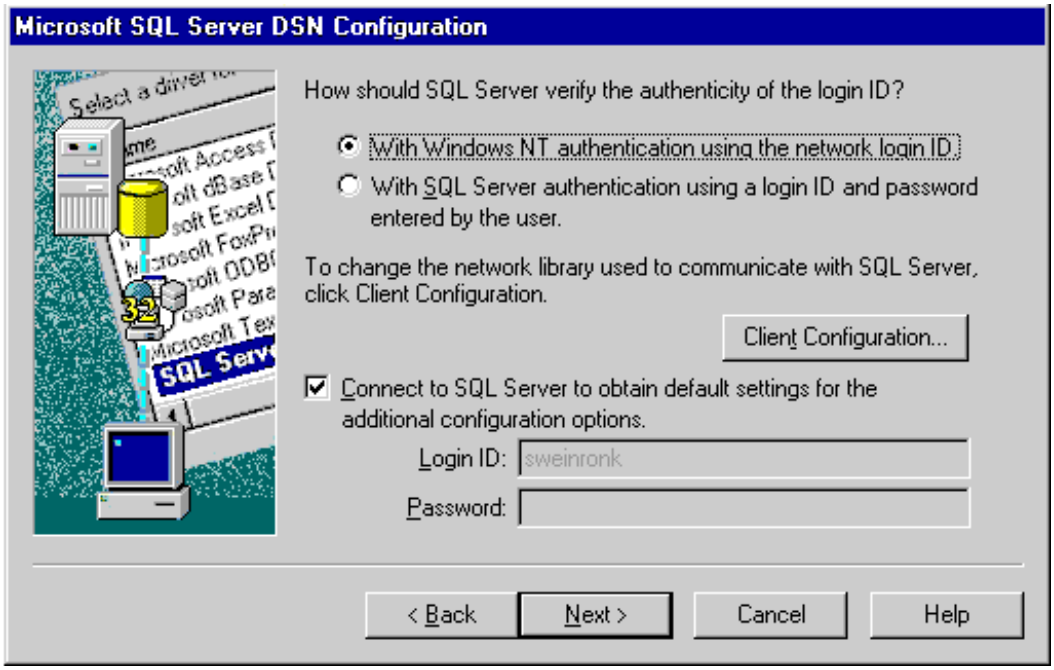


Figure 3.11. MS SQL Server DSN Configuration

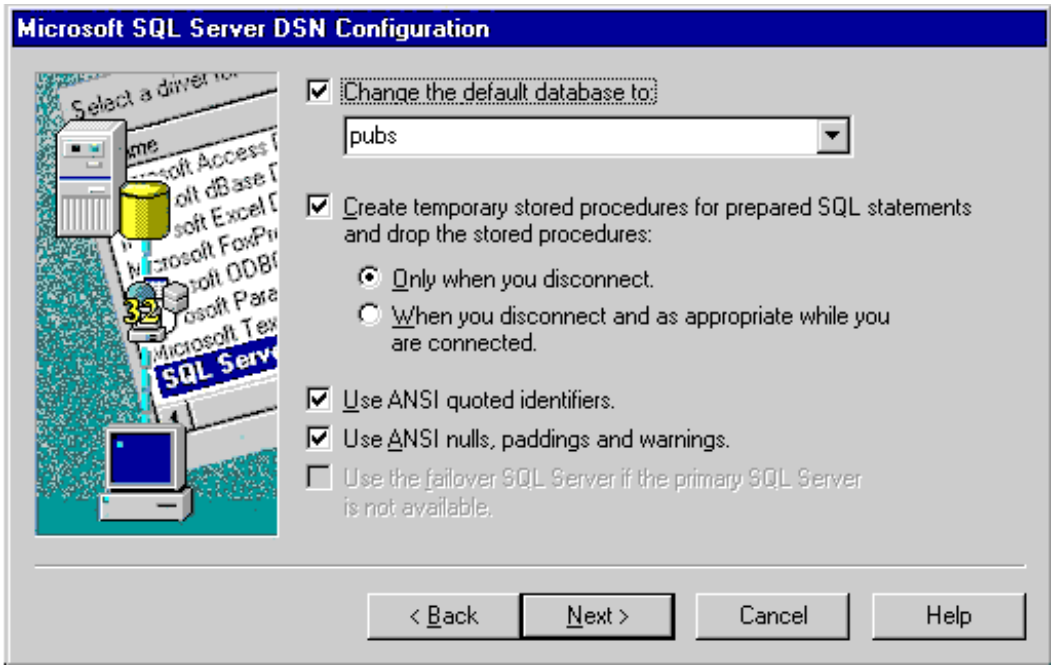


Figure 3.12. MS SQL Server DSN Configuration

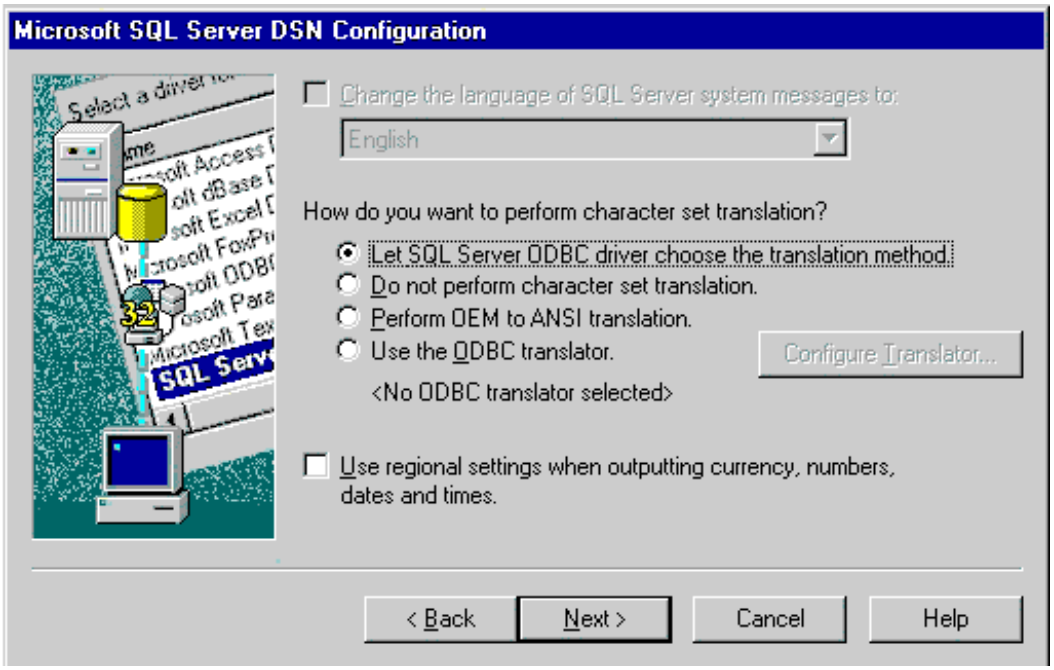
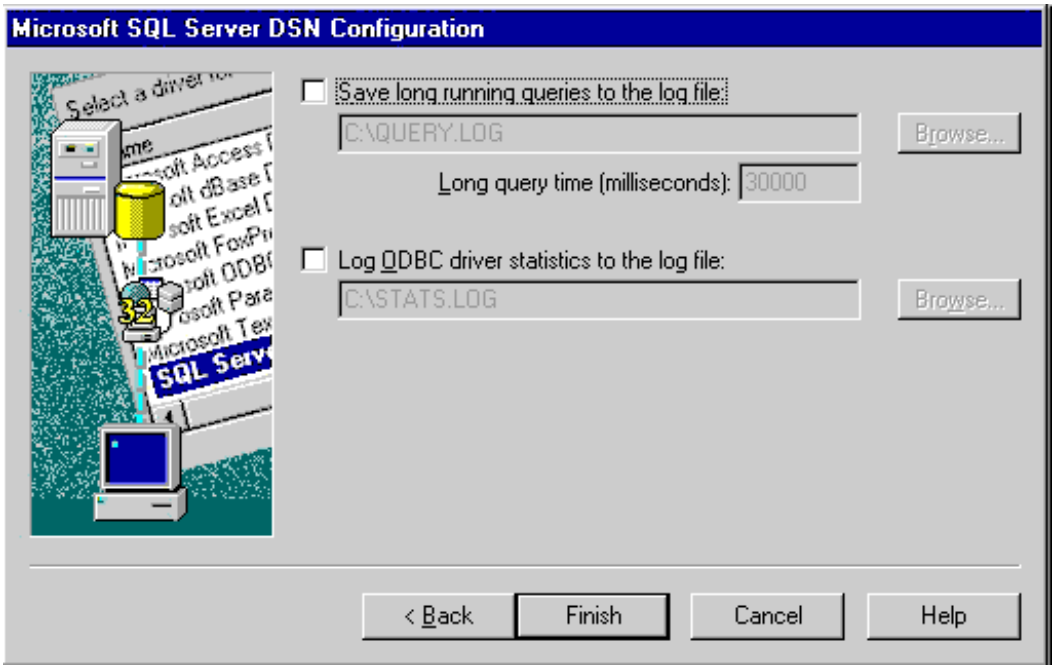
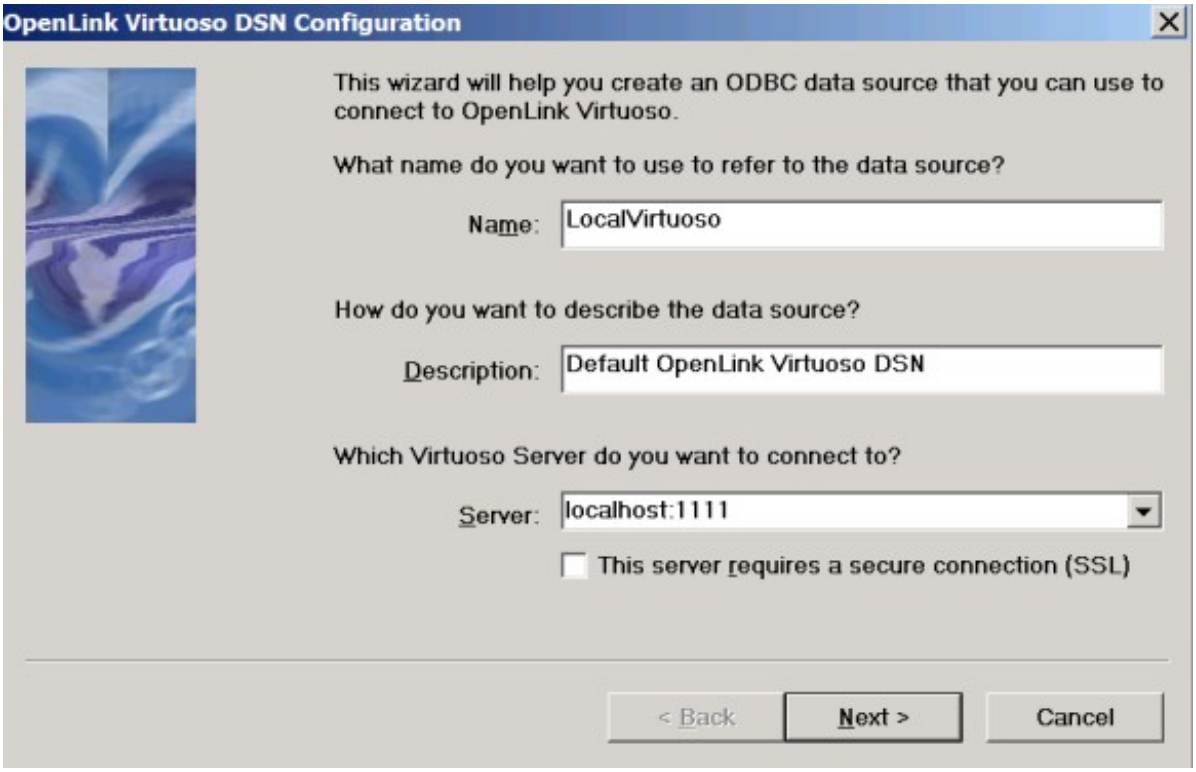


Figure 3.13. MS SQL Server DSN Configuration



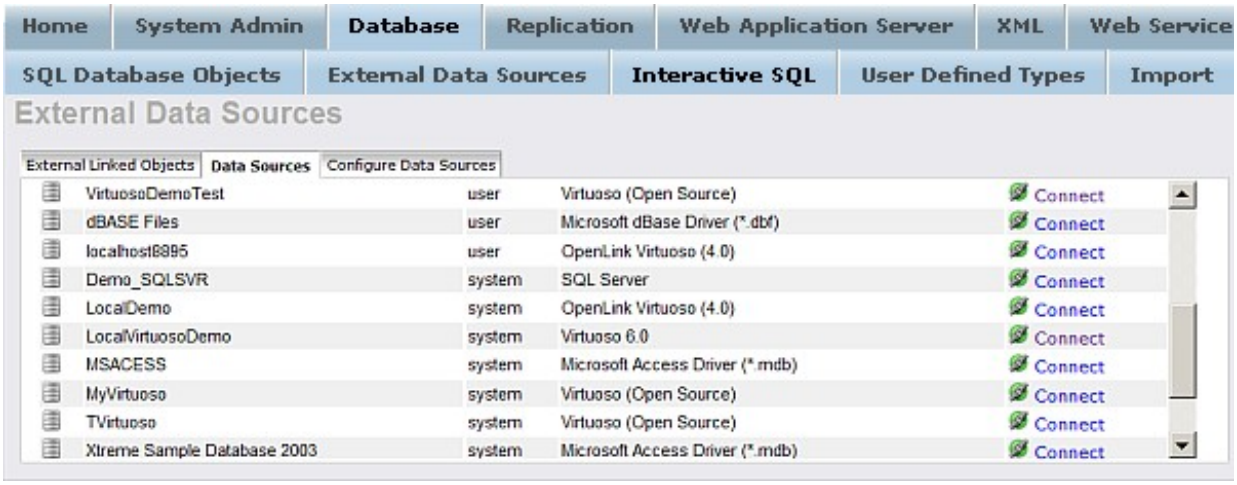
Finally, review the configuration information for your default local Virtuoso datasource, named "*Local Virtuoso* ".

Figure 3.14. OpenLink Virtuoso DSN Configuration



DSNs can be created and configured within the Virtuoso Conductor. Go to "Database", then to "External Data Sources" and then go to "Configure Data Sources" tab. Click the "Add System DSN", or "Add User DSN", or "Add File DSN" button.

Figure 3.15. Creating a new DSN



3.3.2. Datasource Check

Before trying to link a table into Virtuoso, you should verify that the datasource is working correctly. Use a tool such as Microsoft Access, or ODBCtest to test your datasource. Your sanity check will verify that the datasource is correctly configured, that all necessary software is installed, that the database is running, and that there are no permission related problems. In addition, it will verify that the driver that you are using works correctly with the underlying datasource.

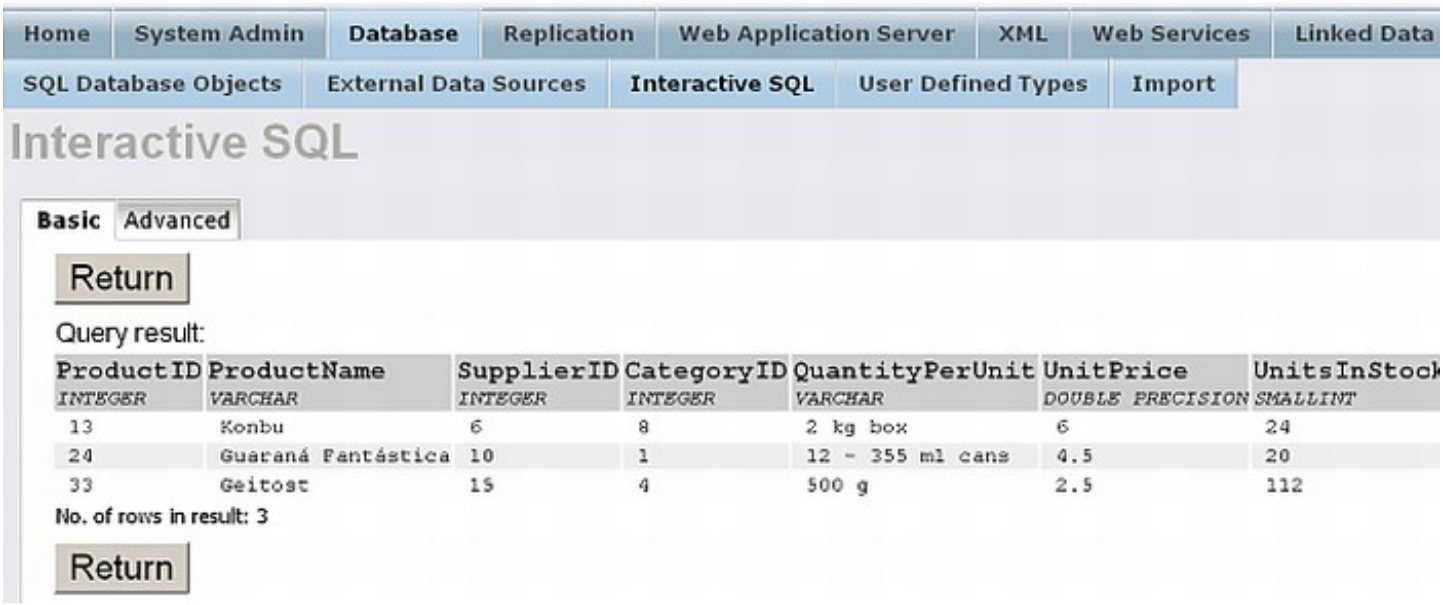
If there is a problem using a specific tool such as Microsoft Access with a specific driver, then that same problem will be manifested when the datasource is exposed via Virtuoso.

3.3.3. Demo Datasource Query

To query the sample data in the Demo Database make sure the web browser is pointing to the Demo database HTTP port which is typically 8890. From Conductor go to tab "Database" and then select the tab "Interactive SQL". Enter a SQL statement and click the button "Execute". For example:

```
SELECT * FROM DEMO..products WHERE UnitPrice < 7
```

Figure 3.16. Demo Database Query



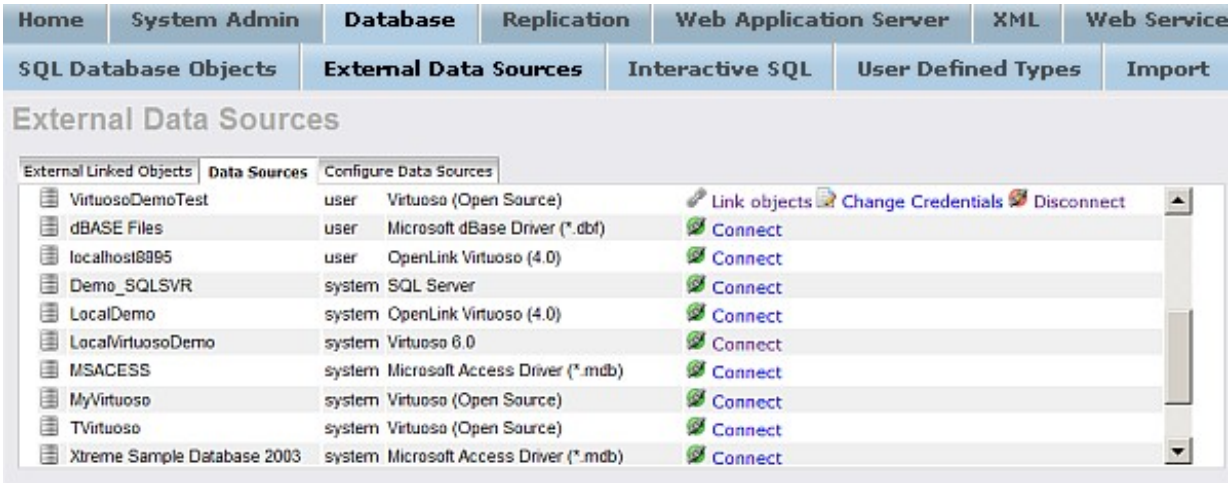
3.3.4. Linking Remote Tables Into Virtuoso

A table on a remote datasource may be linked into the Virtuoso database so that it appears as a local table. Start a browser and point to Conductor, for ex. http://example.com:8890/conductor/. Go to tab "Database", then select "External Data Sources" and then "Sata Sources". Specify the remote datasource that you wish to link to Virtuoso. For example, click the link "connect" for DSN "VirtuosoDemoTest".

Figure 3.17. Enter login name and password.

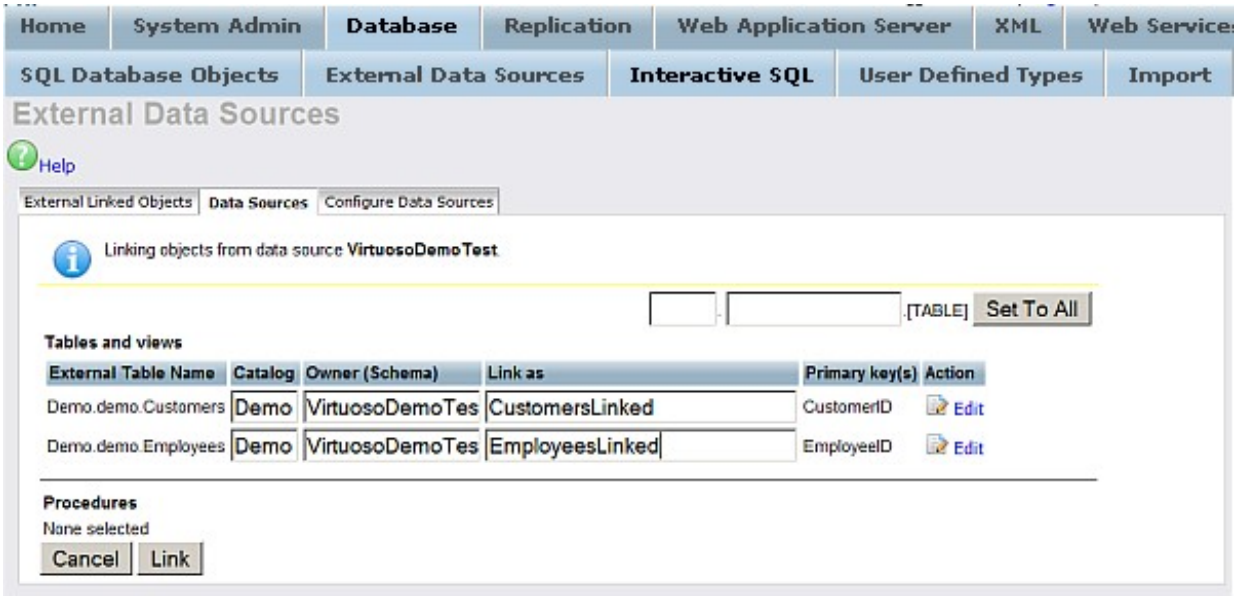
As result should be shown for DSN "VirtuosoDemoTest" the links "Link objects", "Change Credentials", "Disconnect". Click the "Link objects" link.

Figure 3.18. Connected to datasource.



Pick the tables to be linked, and define the names to use.

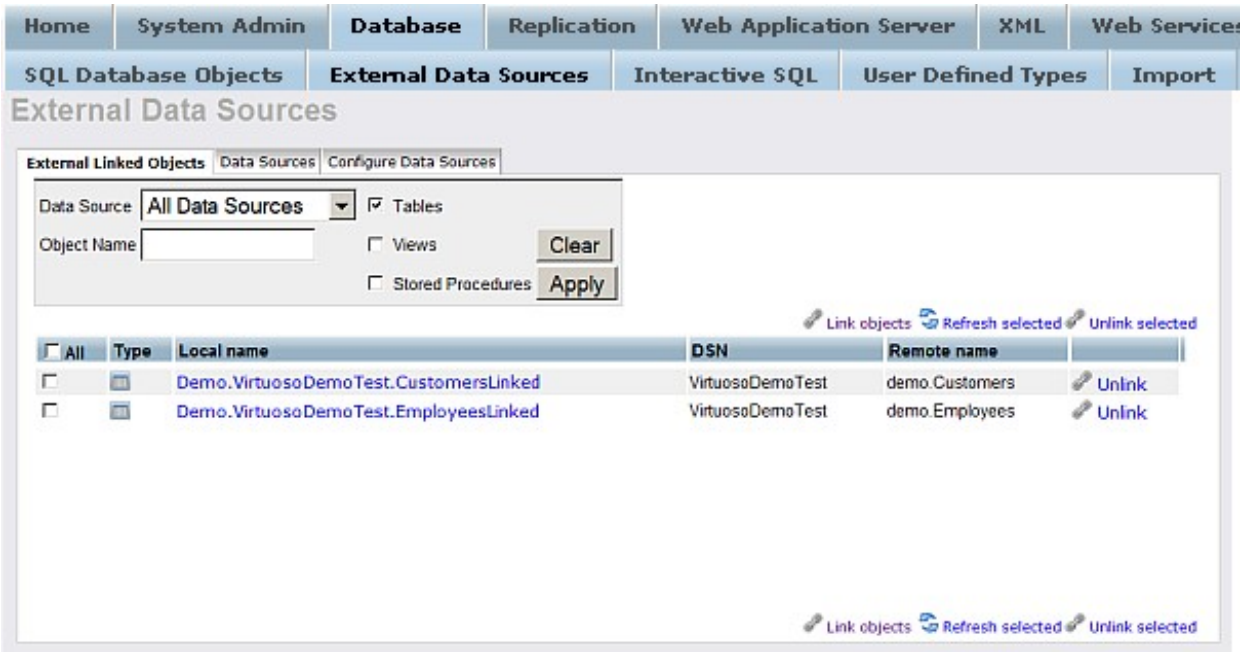
Figure 3.19. Define tables to link



3.3.5. Listing or Unlinking Tables

To list the tables that have been linked into Virtuoso, from Conductor go to *Databases/ External Data Sources/ External Linked Objects* . A table may be unlinked by pressing its "Unlink" icon.

Figure 3.20. List of Connected Data Sources

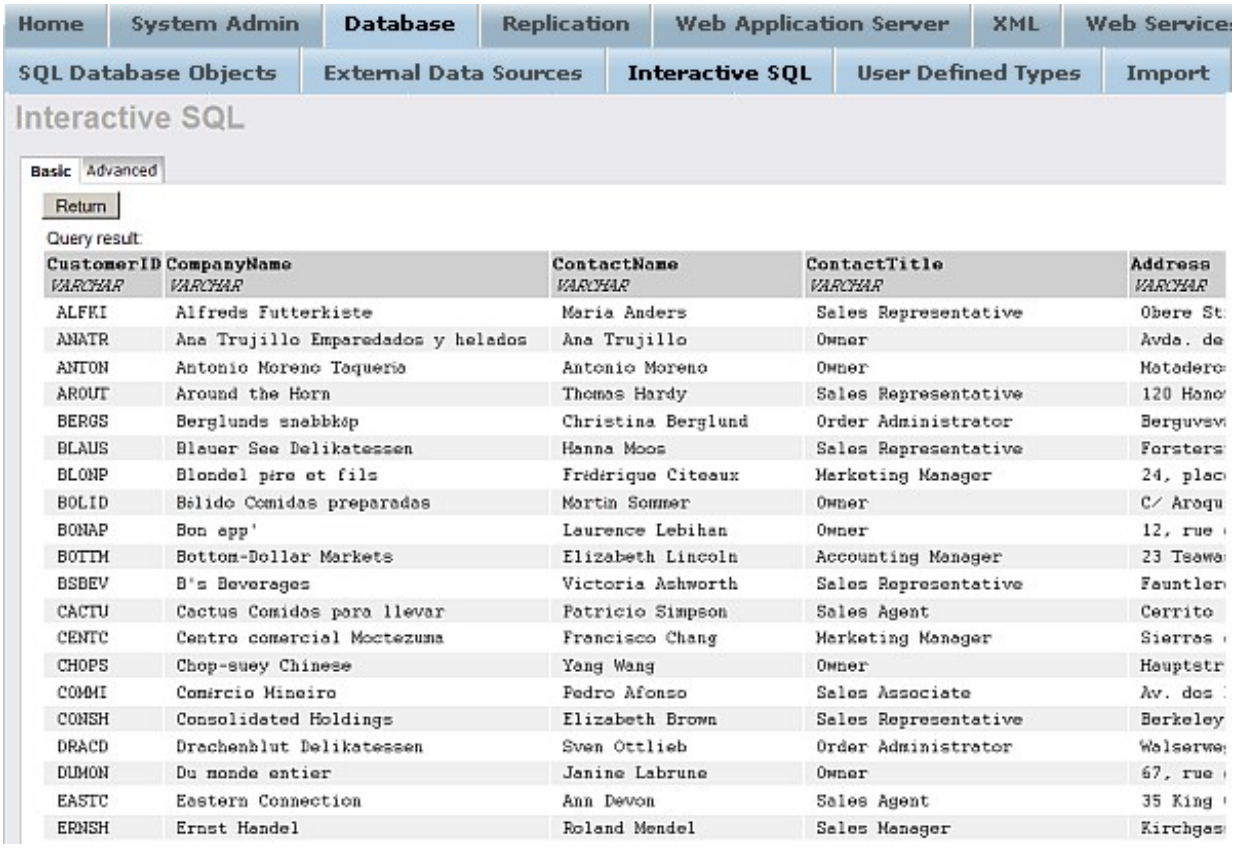


3.3.6. Querying Linked Tables

Once the tables have been linked into Virtuoso, they can be queried using the Interactive SQL query. From the Conductor UI go to *Database/Interactive SQL* . Type a SQL that accesses a remote table such as: SELECT * FROM Demo.VirtuosoDemoTest.CustomersLinked. Click the "Execute" button.

Press the Execute button and review the results.

Figure 3.21. Remote Table Query



3.4. Web Server

3.4.1. Virtual Directories

The term virtual directory applies to the mechanism to hide the physical location of a Web resource under different path which user agents use to retrieve it. This mechanism in virtuoso is a part of host definition.

This method is useful when one server has to keep many Web sites. Using a redirect mechanism is not a universal way to do this. It is better to define virtual hosts and paths to the directory entries which contain Web pages.

Creating Virtual Directories Programmatically

For an overview of virtual directories, and how to configure them in PL, refer to the Virtual Directories Section .

Setup in Administration Interface

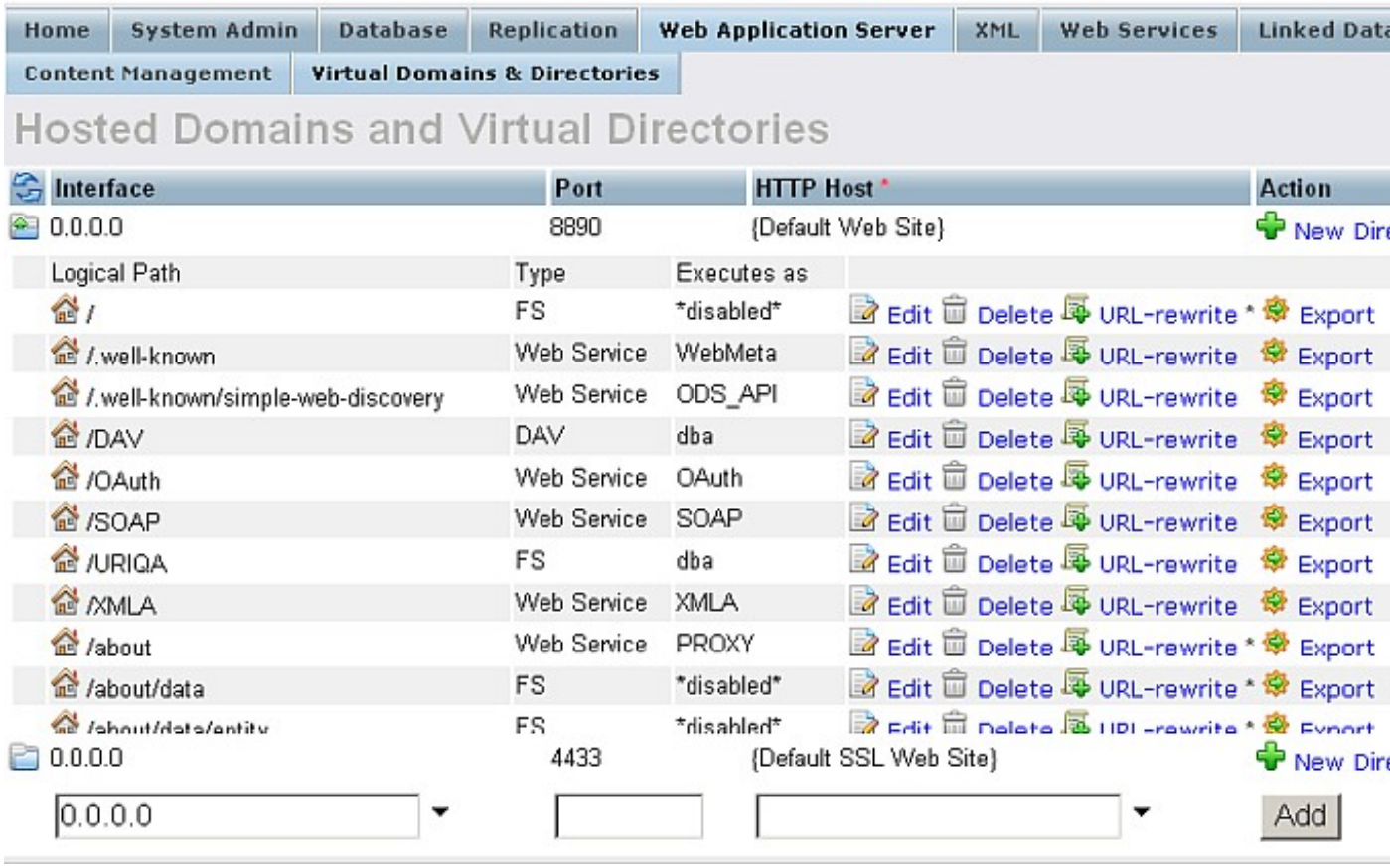
This step by step example will define a virtual directory /help that will point to the directory /departments/support/

1. From the Conductor UI go to Web Application Server/ Virtual Domains & Directories.

   Figure 3.22. Http Hosts and Directories

2. Open the "folder" icon for your {Default Web Site}.

   Figure 3.23. Edit URL mappings

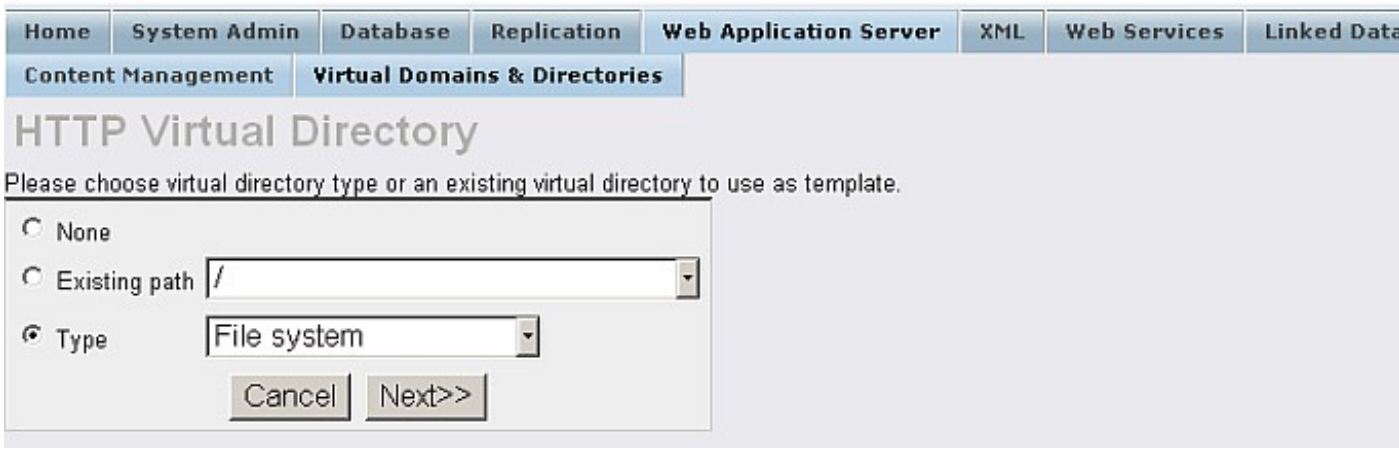


3. Click the link "New Directory" to add a new virtual directory.

Figure 3.24. Add virtual directory

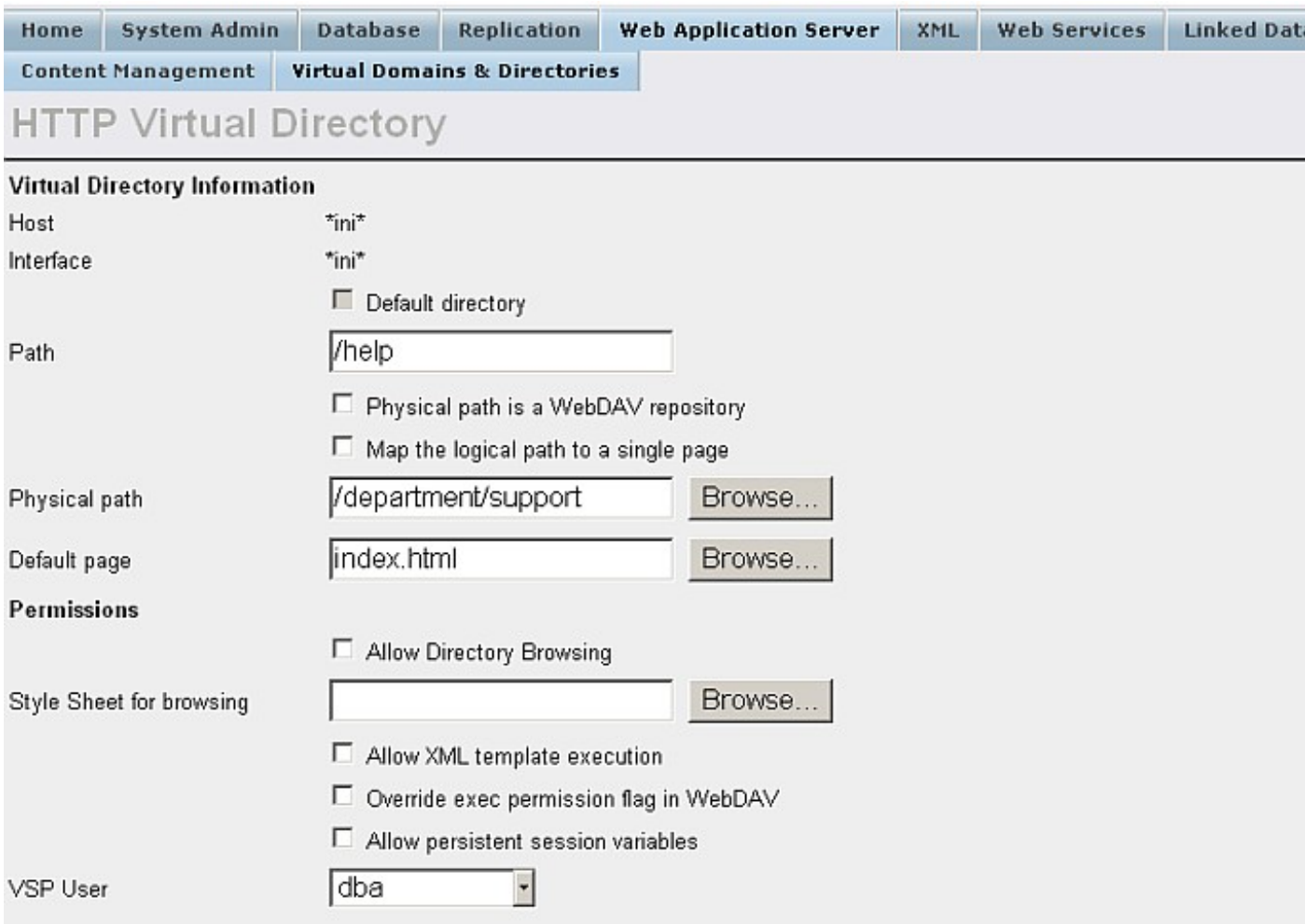
4. Select for "Type" File system, as this mapping example will be from one directory to another, and click "Next".

Figure 3.25. Use File system template



5. Enter details in the form to define the mapping. Most of the fields are optional. In this example, only the logical and physical paths and the default page name are required. Click finally the button "Save Changes".

Figure 3.26. Mapping details



6. The following URLs will then be equivalent:

   http://example.com:8890/help
   http://example.com:8890/departments/support/index.html

3.4.2. Multi Homing

The term Multi Homing refers to the practice of maintaining more than one server on one machine, differentiated by their apparent host name. It is often desirable for companies sharing a web server to have their own domains, with web servers accessible as www.company1.com and www.company2.com, without requiring the user to know any extra path information.

Creating Multiple Homes Programmatically

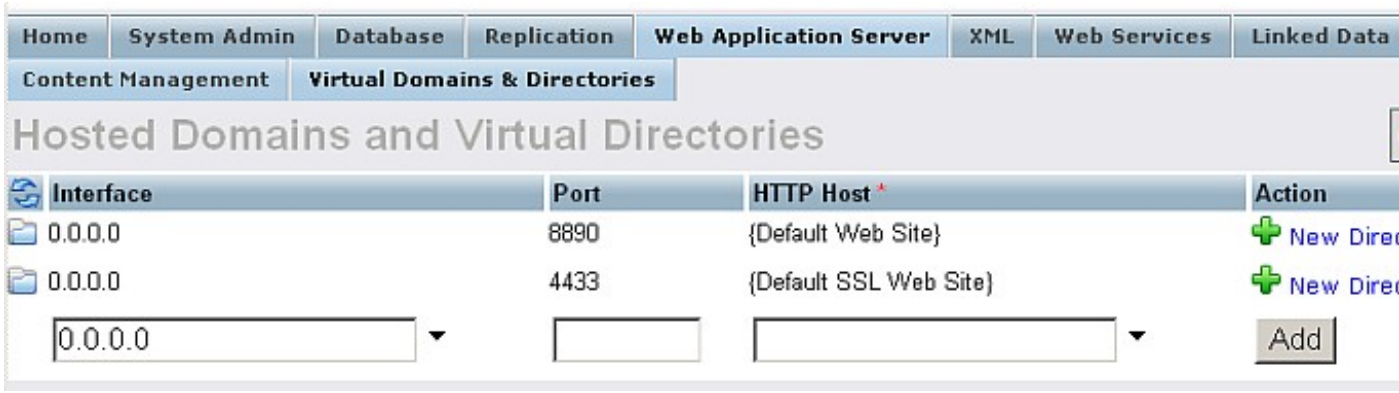
For an overview of Multi Homing, and how to configure it with PL, refer to the Virtual Hosting and Multi Hosting section.

Setup in Administration Interface

This step by step example will define a virtual home for the URL http://www.ahelp.com/ to the server www.a.com and directory /departments/support/

1. Have a domain name allocated in the DNS for the ahelp.com that points to the same IP address of the a.com that is hosting a Virtuoso server.
2. From the Conductor UI go to Web Application Server/ Virtual Domains & Directories.

   Figure 3.27. Http Hosts and Directories.

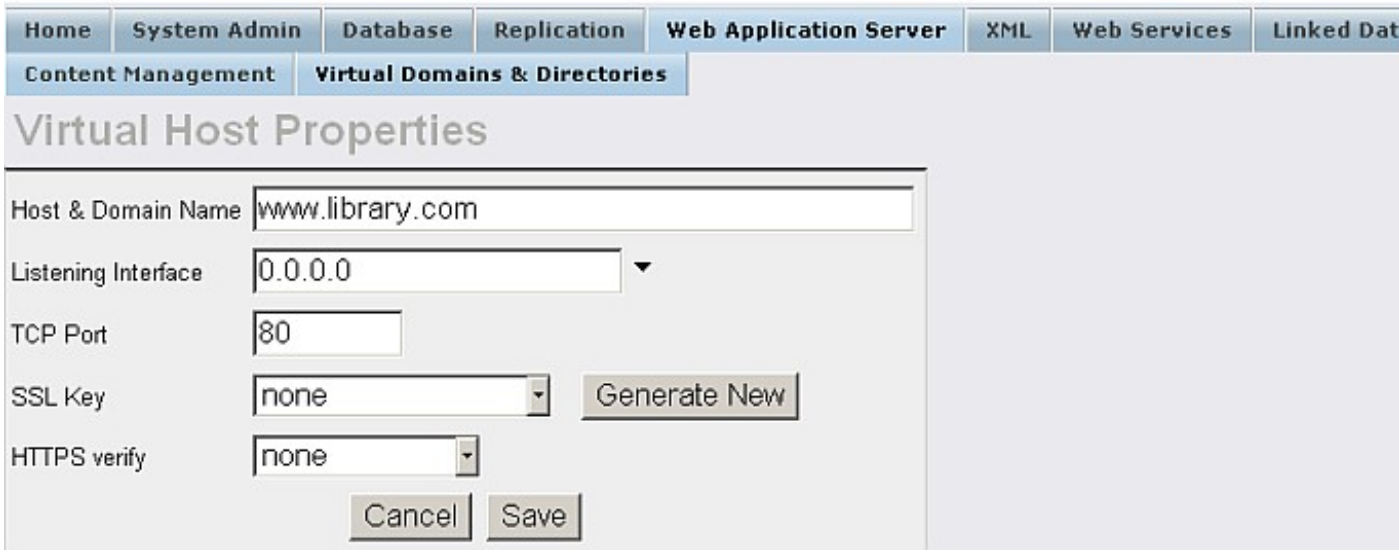


3. To add a new host definition, enter for "Port" 80, enter for "HTTP Host" www.ahelp.com and select the "Add" button.

   Figure 3.28. Add new site

4. Click for the new defined site the "Edit" link in order to define the mapping between the virtual host and the actual listening host domain names.

   Figure 3.29. New site mapping

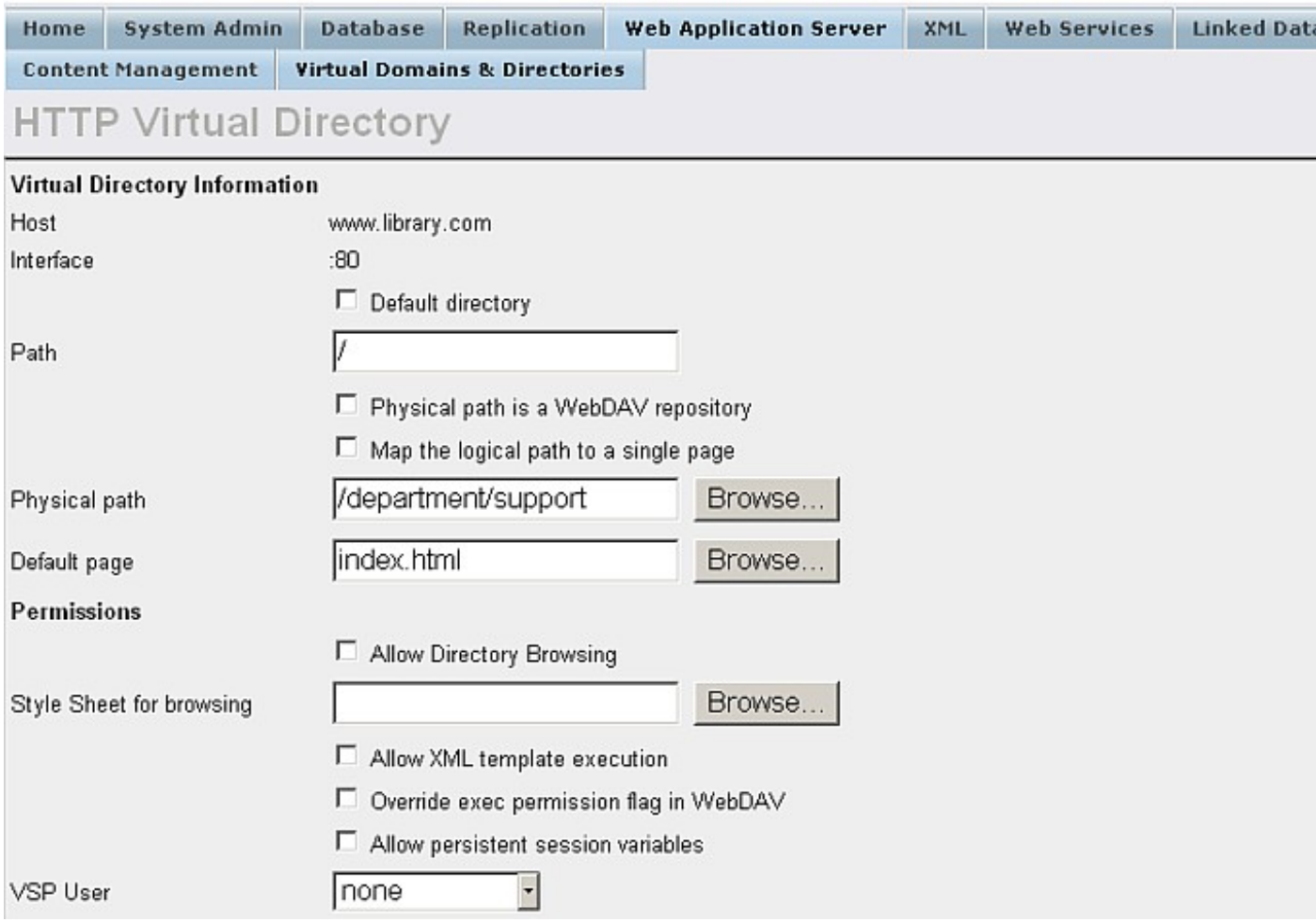


5. Click the "folder" icon for the new defined site and then click the "Edit" link for the Logical Path "/".

Figure 3.30. Set Logical Path

6. Enter details in the form to define the mapping. Most of the fields are optional. In this example, only the logical and physical paths and the default page name are required.

Figure 3.31. Mapping details



7. The following URLs will then be equivalent:

   http://www.ahelp.com/
   http://www.a.com/departments/support/index.html

3.5. WebDAV

WebDAV support enables Virtuoso to act as the Web Content Store for all of your eBusiness data, this includes Text, Graphics and Multimedia files. WebDAV support also enables Virtuoso to play the familiar roles of a FILE & WEB SERVER, hosting entire Web sites within a single database file, or across multiple database files. Virtuoso's WebDAV provides file system like access to its non SQL data repository.

Files contained in the DAV repository are stored in a database table. Free text indexing and replication mechanisms may be applied to the DAV repository as well as the regular database. Standards based WebDAV methods are used to access resources stored in the repository based on Virtuoso's full-featured SQL-92 database engine with performance that matches or exceeds that of current major players in the DB market. Typical WebDAV clients that can access the repository for content management include Network Places on Windows through Explorer, Nautilus on Gnome (Linux and Solaris), and the Mac OS X desktop. Many web development tools also support WebDAV directly.

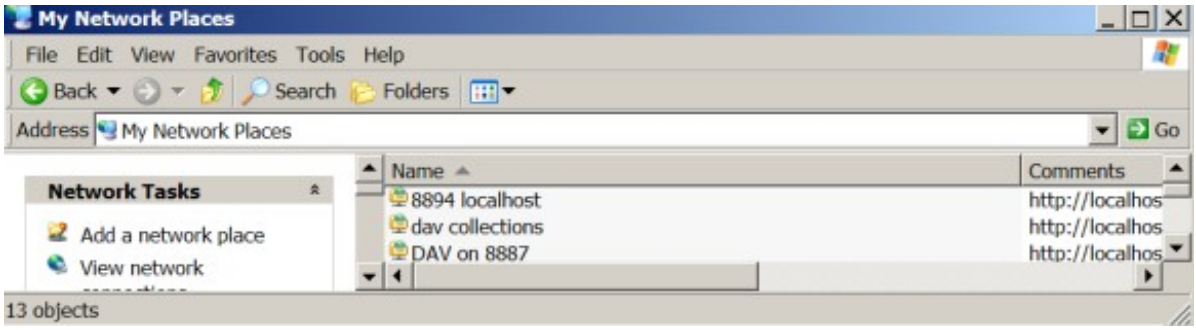
See the WebDAV Administration section, to setup the WebDAV in the Visual Server Administration Interface.

3.5.1. Web Folders

Microsoft Windows has a notion of a Web Folder. This is how Windows connects to a WebDAV store which can then be traversed like any other file system. Files can be copied and moved to and from the DAV using normal drag-and-drop methods.

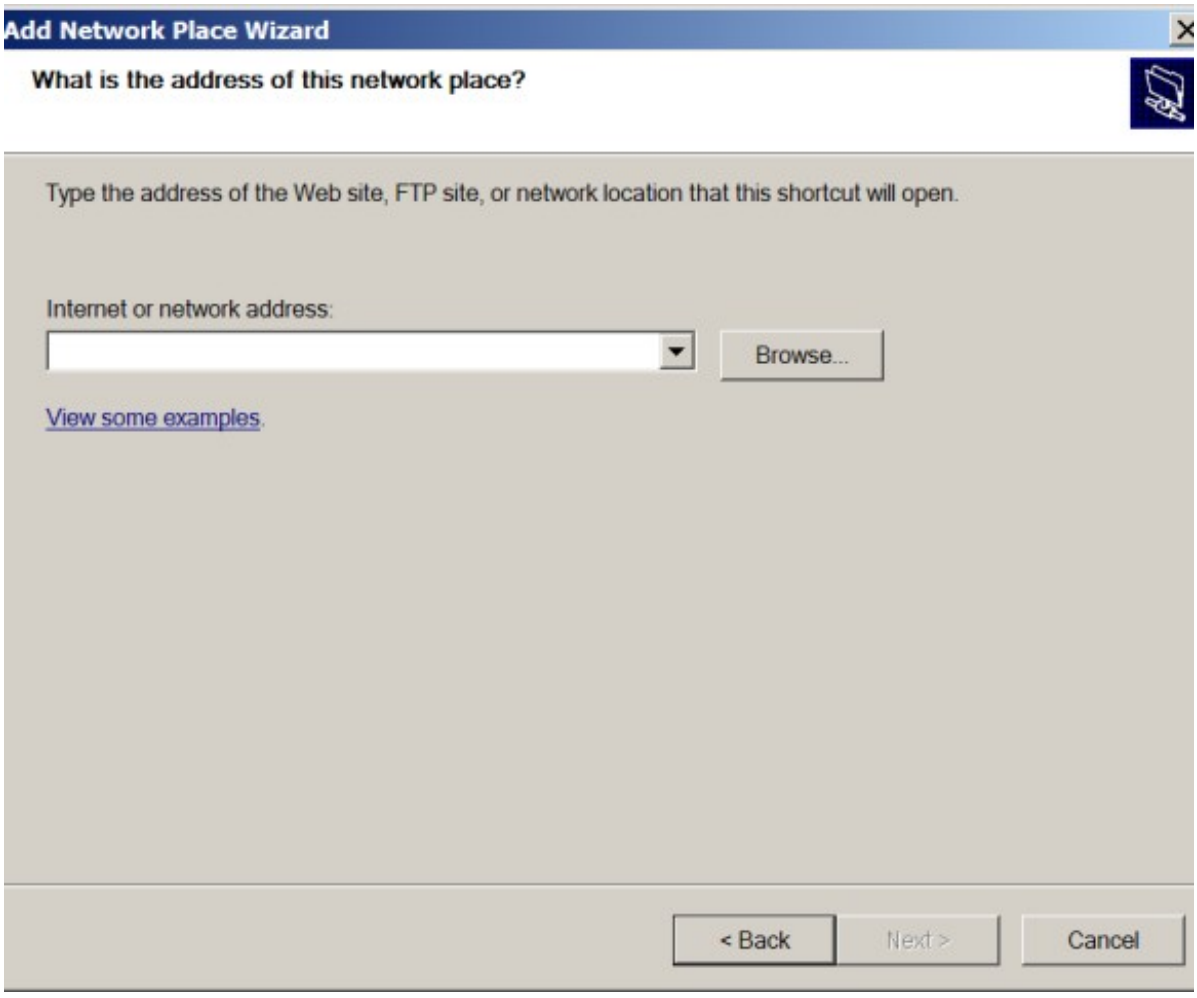
1. To create a Web Folder open the *Windows Explorer* and navigate to *My Network Places* .

Figure 3.32. My Network Places



2. You will find an icon or option to *Add Network Place* which when double-clicked will lead you to the wizard.

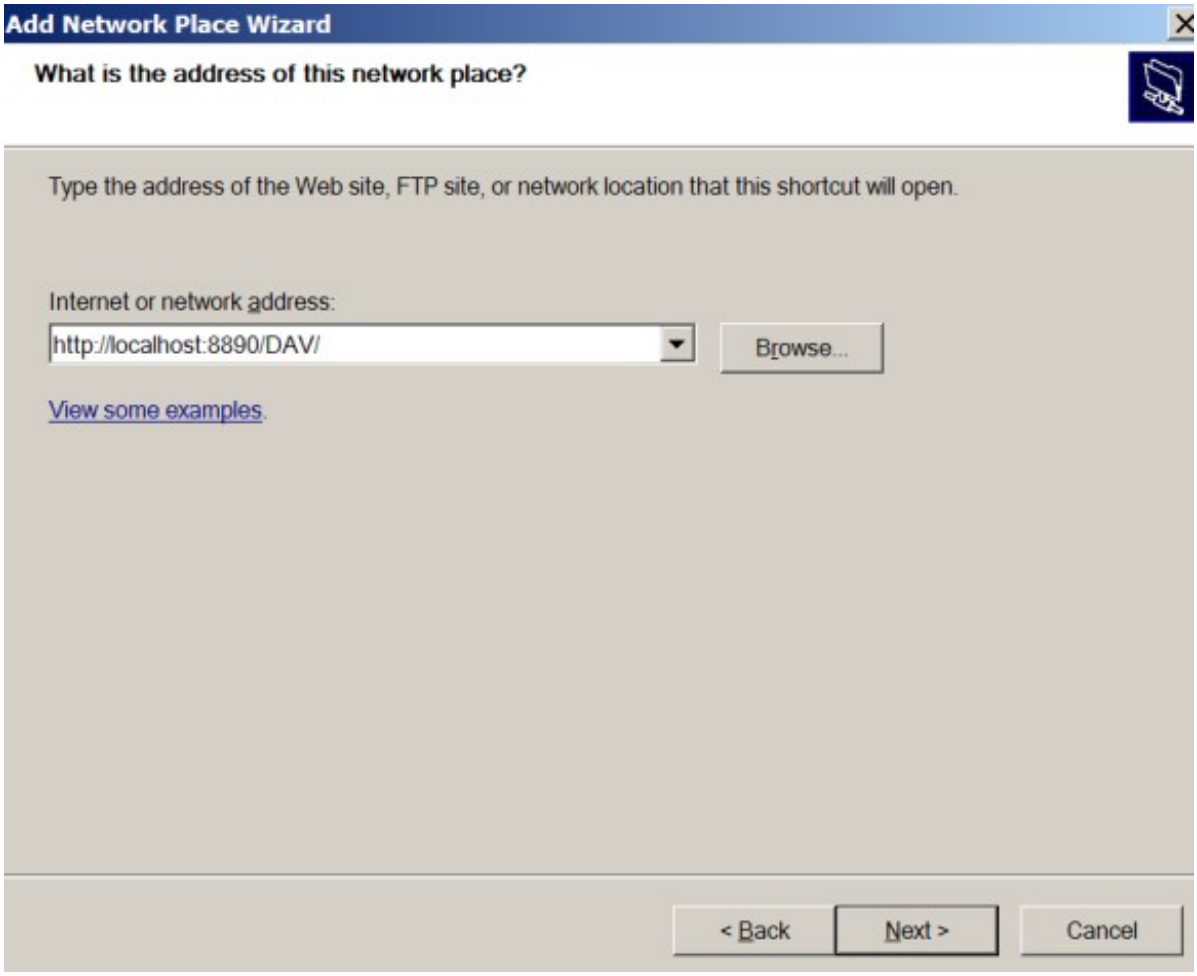
Figure 3.33. Web Folder Wizard



3. The first screen encourages you to type the location of the network place, for connecting to a Virtuoso DAV you will need to know the server location and port number. If you installed Virtuoso on to your local machine taking default options then you would probably be typing:

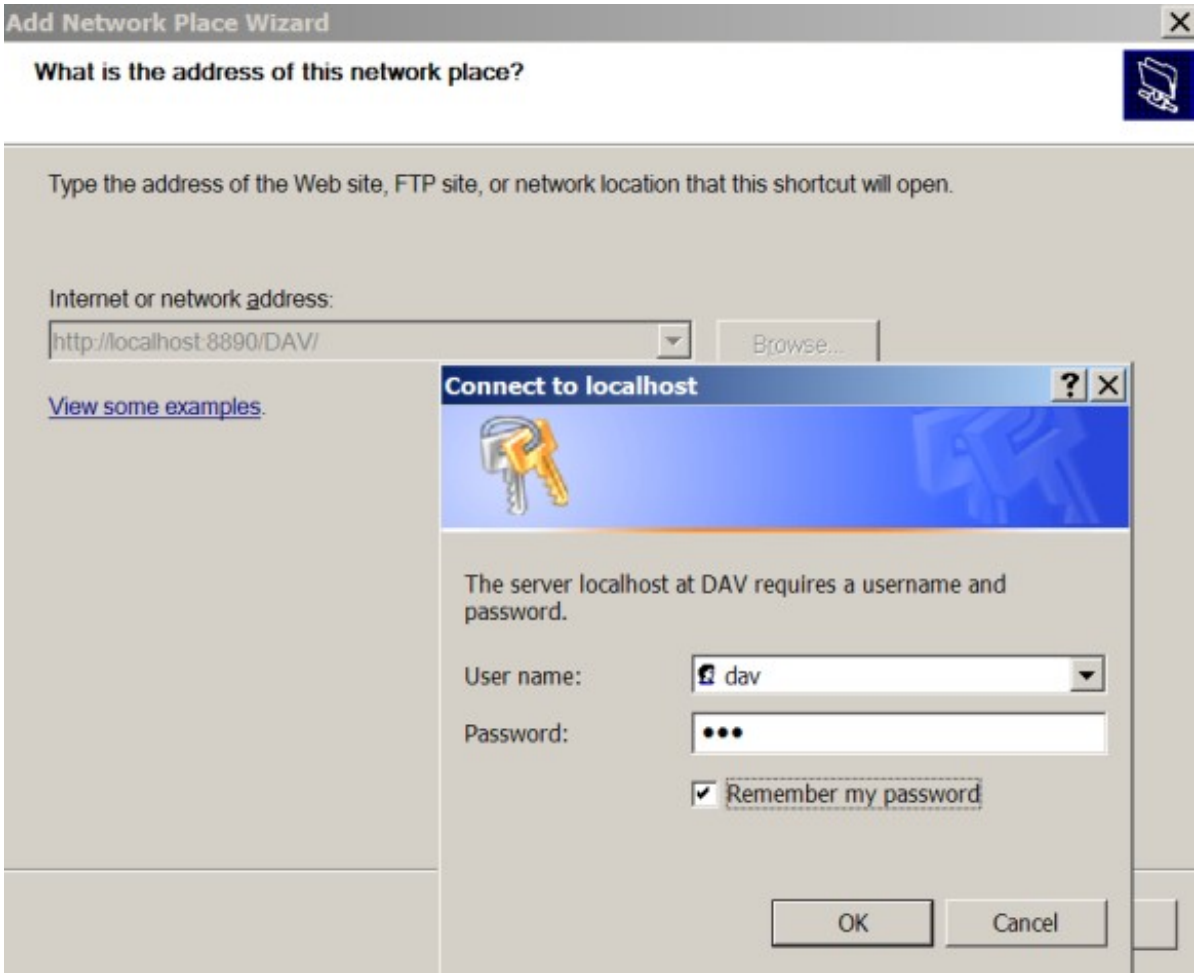
```
http://example.com:8889/DAV/
```

Figure 3.34. WebDAV location



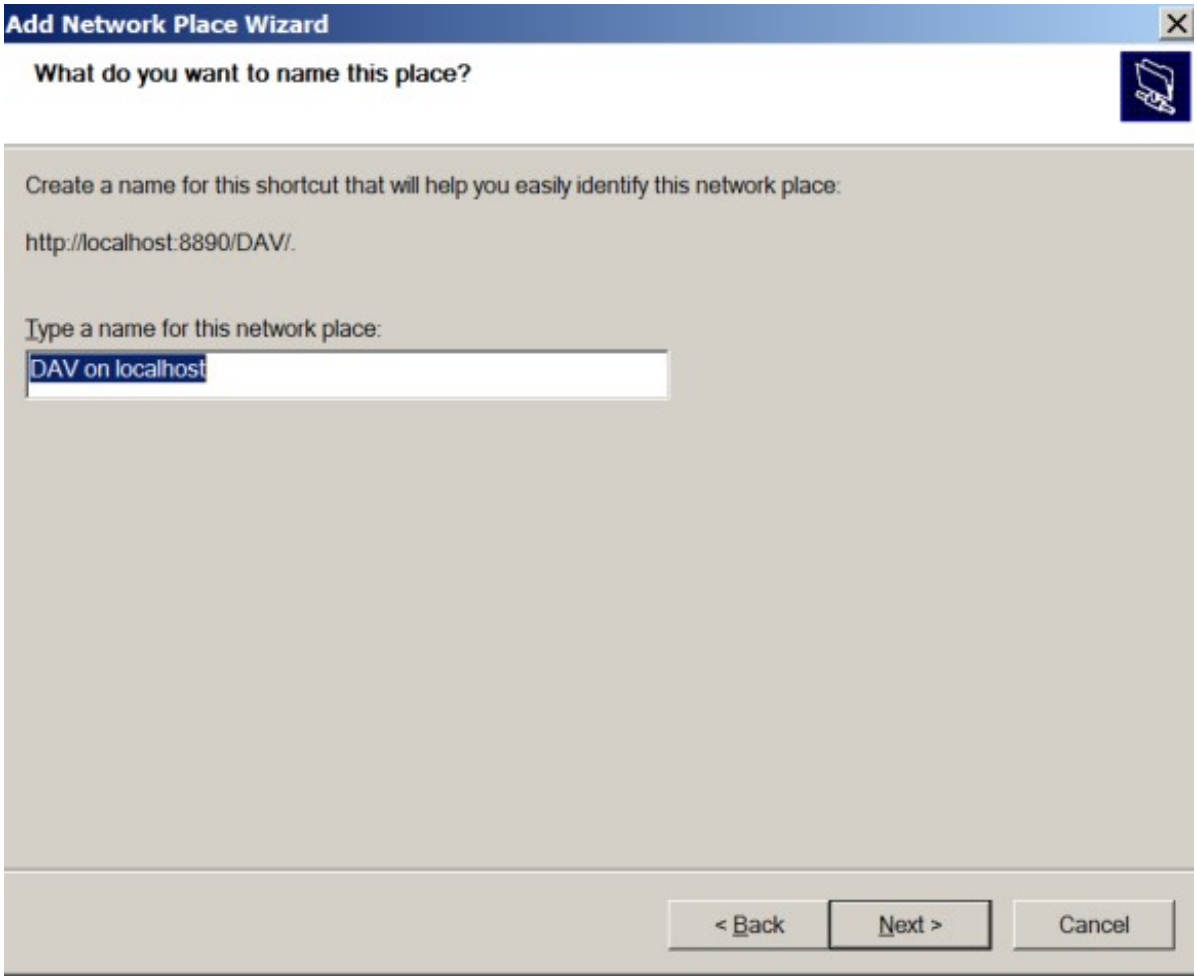
4. You will then be asked for authentication information which will require a username and a password. If defaults are taken then this would simply be dav and dav for both.

Figure 3.35. WebDAV authentication



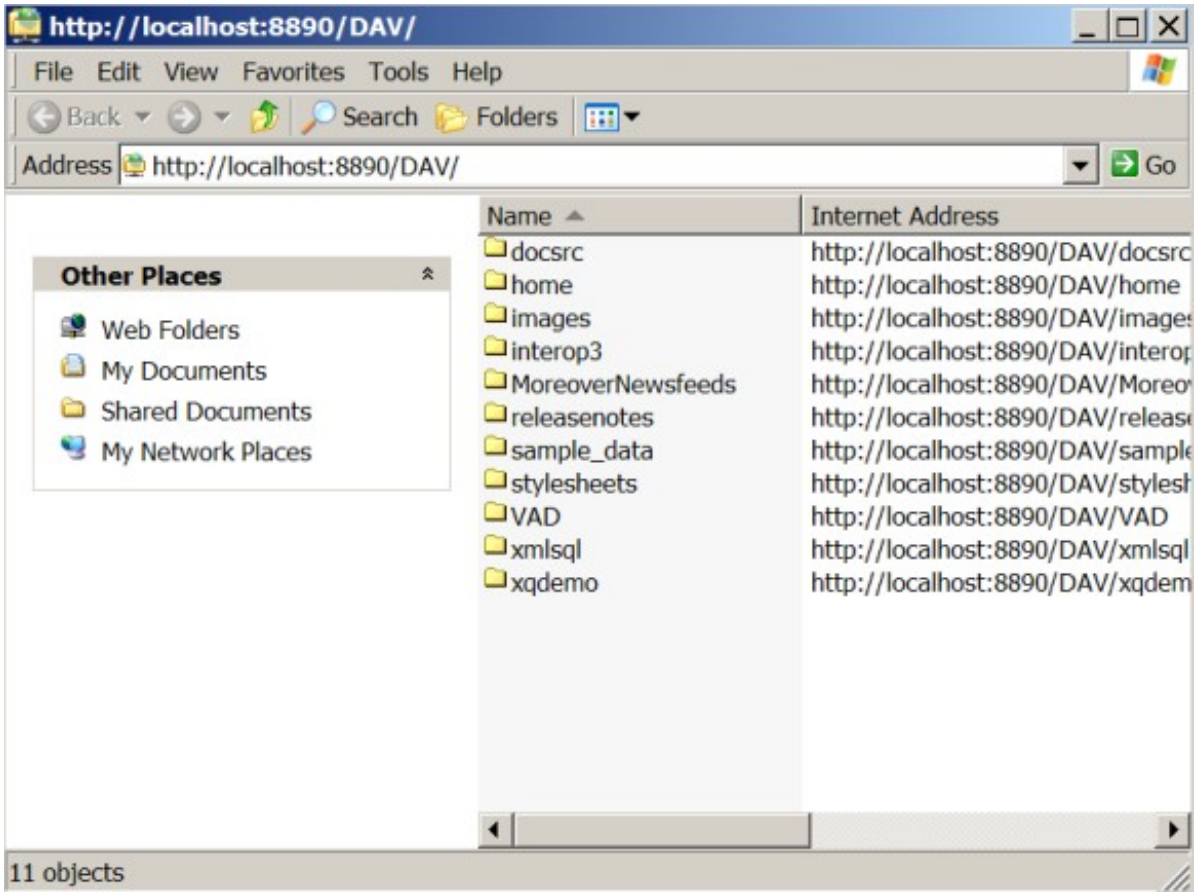
5. To complete the creation of a Web Folder you be asked to supply a name for the network place.

Figure 3.36. Name of your WebDAV Web Folder



6. Once provided Explorer will automatically open a new Window over looking the new location.

Figure 3.37. Files contained in your DAV



7. You can find your new Web Folder again from My Network Places where the link will be saved.

3.6. Web Services

SOAP is a lightweight, extensible, XML-based protocol for information exchange in a decentralized, distributed environment. Primarily, SOAP defines a framework for message structure and a message processing model. SOAP also defines a set of encoding rules for serializing data and a convention for making remote procedure calls. The SOAP extensibility model provides the foundation for a wide range of composable modules and protocols running over a variety of underlying protocols such as HTTP.

By Supporting the Simple Object Access Protocol (SOAP) , Virtuoso enables you to integrate business processes within and across organization boundaries. Virtuoso's SOAP Support implementation enables you to execute Virtuoso Stored Procedures over HTTP. This is a significant component in any B2B development and implementation effort. Development is very rapid and is directly incorporated within the database environment required for keeping B2B processes running accurately.

3.7. Exposing Persistent Stored Modules as Web Services

Virtuoso SQL stored procedures and functions can be exposed as SOAP services very simply from Virtuoso, whether they are native Virtuoso or on remote data sources. This powerful ability means that any database servers already existing within an organization can easily become a component in an eBusiness solution using Virtuoso. All you need is a few simple steps that typically take mere minutes to complete:

- **Choose your stored procedure(s).** The procedures that you want to expose can either be native Virtuoso stored procedures, or remote stored procedures that can be linked in using the Remote Procedures user interface.
- **Choose a virtual directory.** Because SOAP services need to be exposed and accessed via HTTP a Virtuoso virtual directory must be used. Either use the existing SOAP virtual directory or create a new one.
- **Publish procedures to virtual directory.** The user specified as SOAP account on the virtual directory must have execute privileges on the procedures. Use the Publish options on the virtual directory user interface.
- **Test the VSMX output.** Once procedures have been published as SOAP services they are automatically described by WSDL and testable using Virtuoso's VSMX feature.

XML Query Templates provide a direct way to store SQL in an XML file on the Virtuoso server that when executed, i.e. fetched from a web browser, actually returns the results of the query.

The C Interface chapter describes how users can define custom built-in functions, from C or other programming languages, that can be used from within Virtuoso PL. This also means that VSE's can also be published as a Web Service!

3.7.1. Publishing Stored Procedures as Web Services

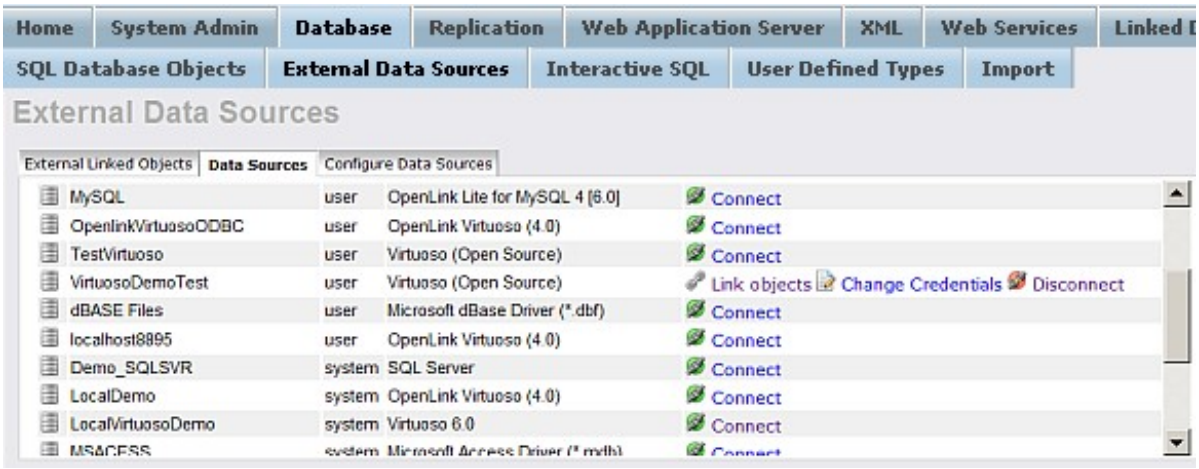
Choosing Stored Procedures to Expose

You can either expose native Virtuoso stored procedures (previously defined or newly created) using the CREATE PROCEDURE statement, or stored procedures from other database types can be linked into Virtuoso using an ODBC datasource.

Virtuoso lists available stored procedures for each catalog in Conductor under: */Database/External Data Sources/External Linked Objects / with checked "Stored Procedures"* .

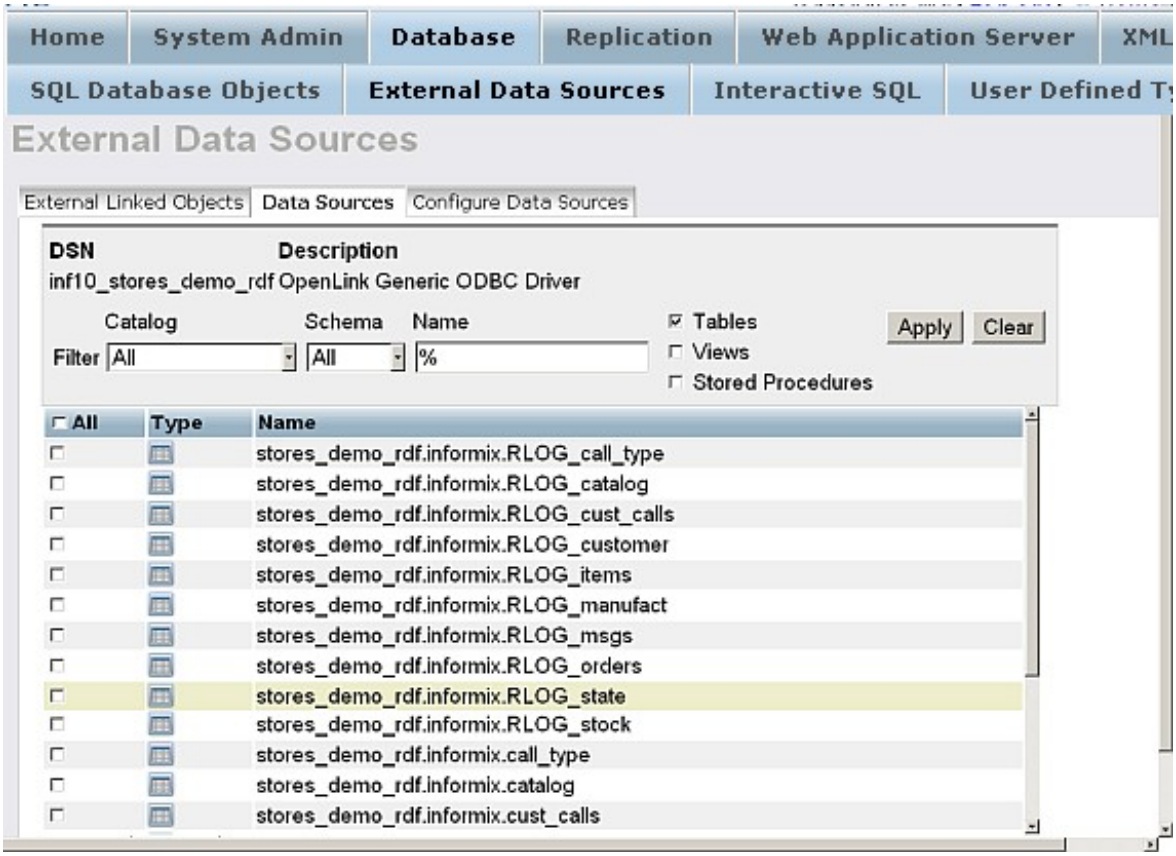
To link a stored procedure from another database system we must first create a valid data source that leads to a connection to that database. Once verified proceed to the Remote Procedures page. Select the "Link objects" link for a data source.

Figure 3.38. Linking Procedures from Remote Data Sources



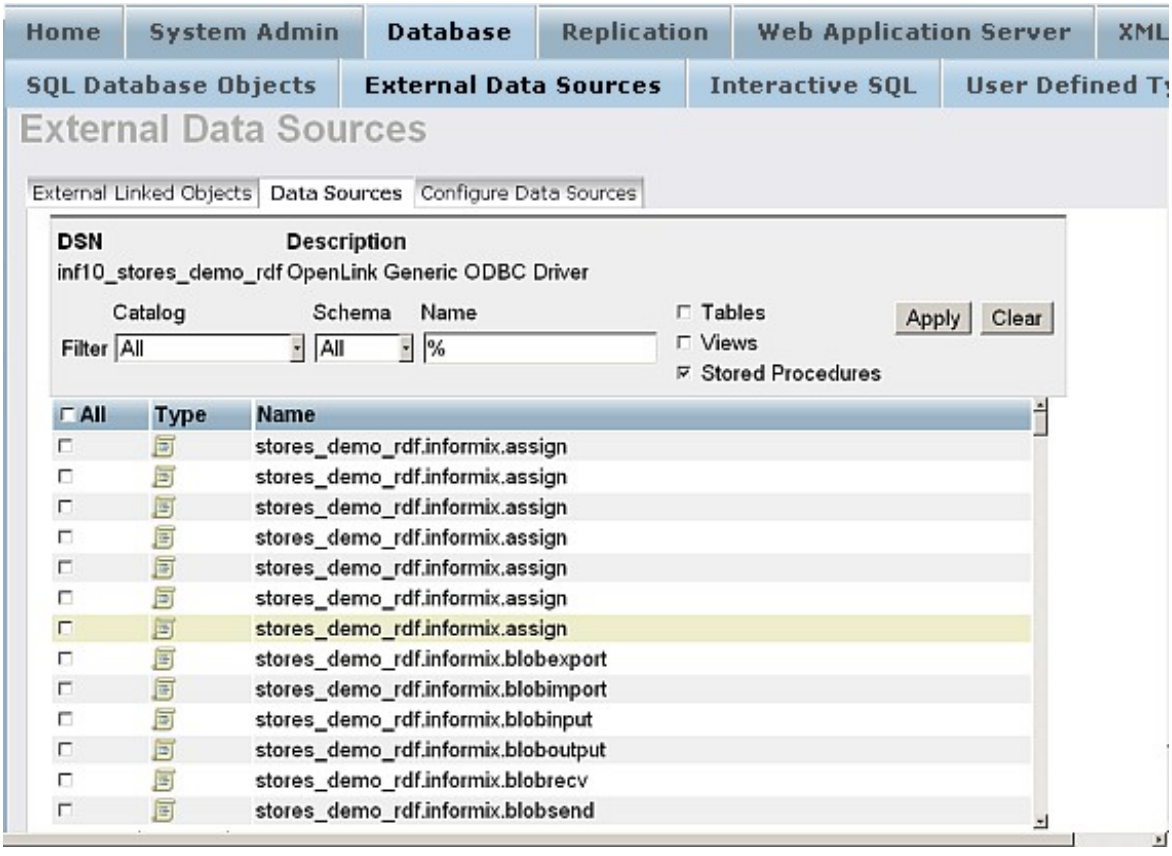
Select the check-box "Store Procedures". Click the "Apply" button. As result will be shown the list of available procedures.

Figure 3.39. Linking Procedures from Remote Datasources



Select the check-boxes for the procedures you want to link and click the "Link" button.

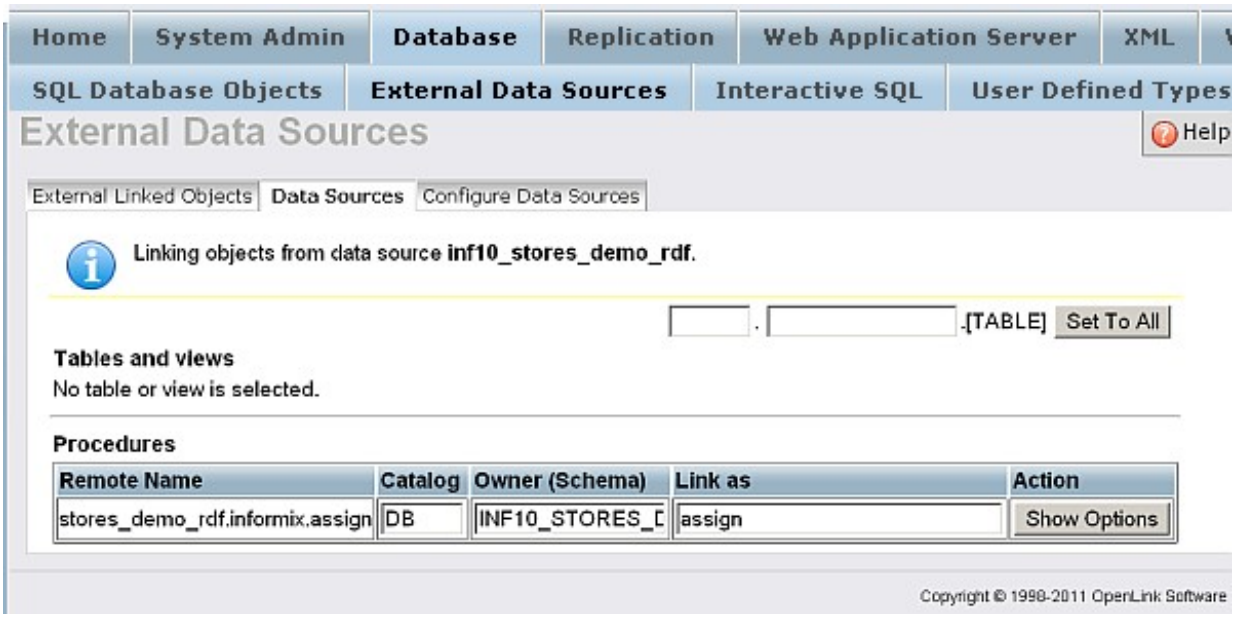
Figure 3.40. Linking Procedures from Remote Datasources



You will be presented with a new page listing the chosen procedures and their data type information. This gives you an opportunity to alter the data type mappings that Virtuoso will use both internally and for any future interactions with the SOAP

server. If you do not want to specify any special type information the details can be left as default.

Figure 3.41. Linking Procedures from Remote Datasources



For each remote procedure you may change how they will be referenced within Virtuoso by making changes to the fields for *Catalog* , *Owner* , *Link as* , and *Description* fields. These fields define how you will find the linked procedure locally to Virtuoso only and do not affect the remote data source.

For each procedure there is an option to *PL Wrapper Requirement* . This option is required if your remote procedure is capable of returning a resultset that you want to process via Virtuoso. Can be *SOAP Execution* , *SQL Execution* or *None* . Also you can specify *Return Type* , *Data Type* , *SOAP Type* .

Once the details are correct press the "Link" button.

See Also:
Linking Remote Procedures

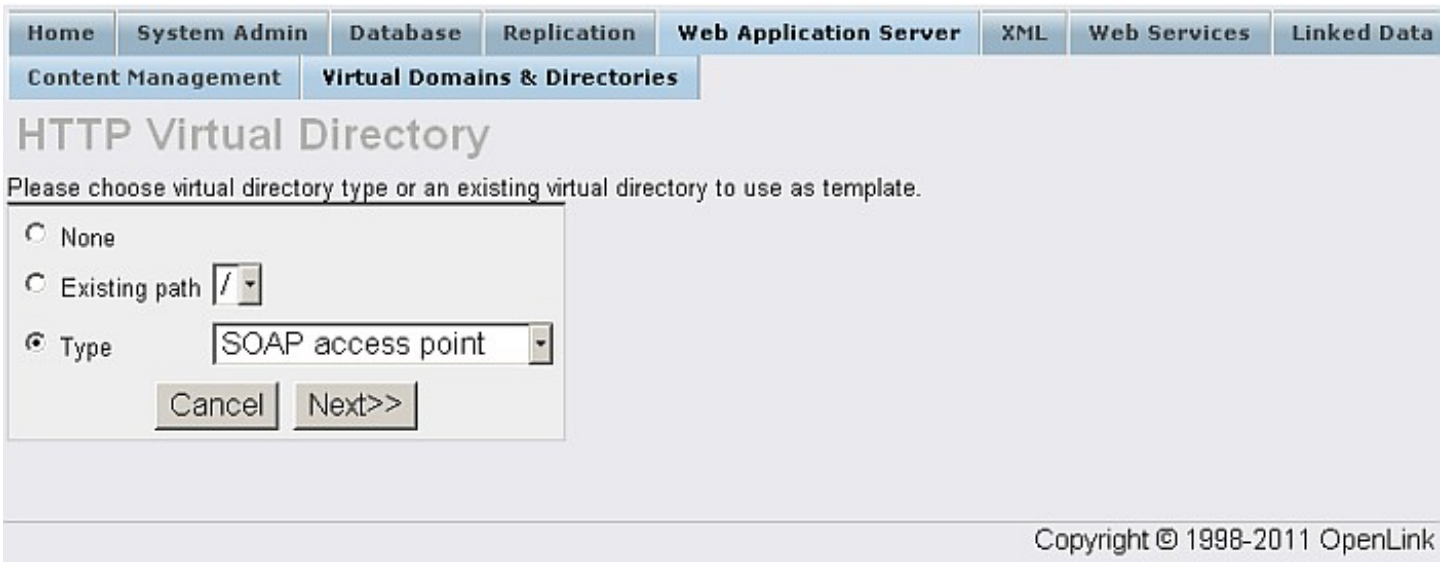
Defining Virtual Directories

Before any procedures native or linked can be exposed as SOAP Services a location in HTTP space must be defined. From Conductor *Web Application Server/Virtual Domains & Directories* you make a new URL Mappings. Click on the *New Directory* link for the {Default Web Site} line to begin defining a new SOAP mapping.

Figure 3.42. Virtual Directories

Select for "Type" from the list the value "SOAP access point" and click the "Next" button.

Figure 3.43. Virtual Directories Mappings

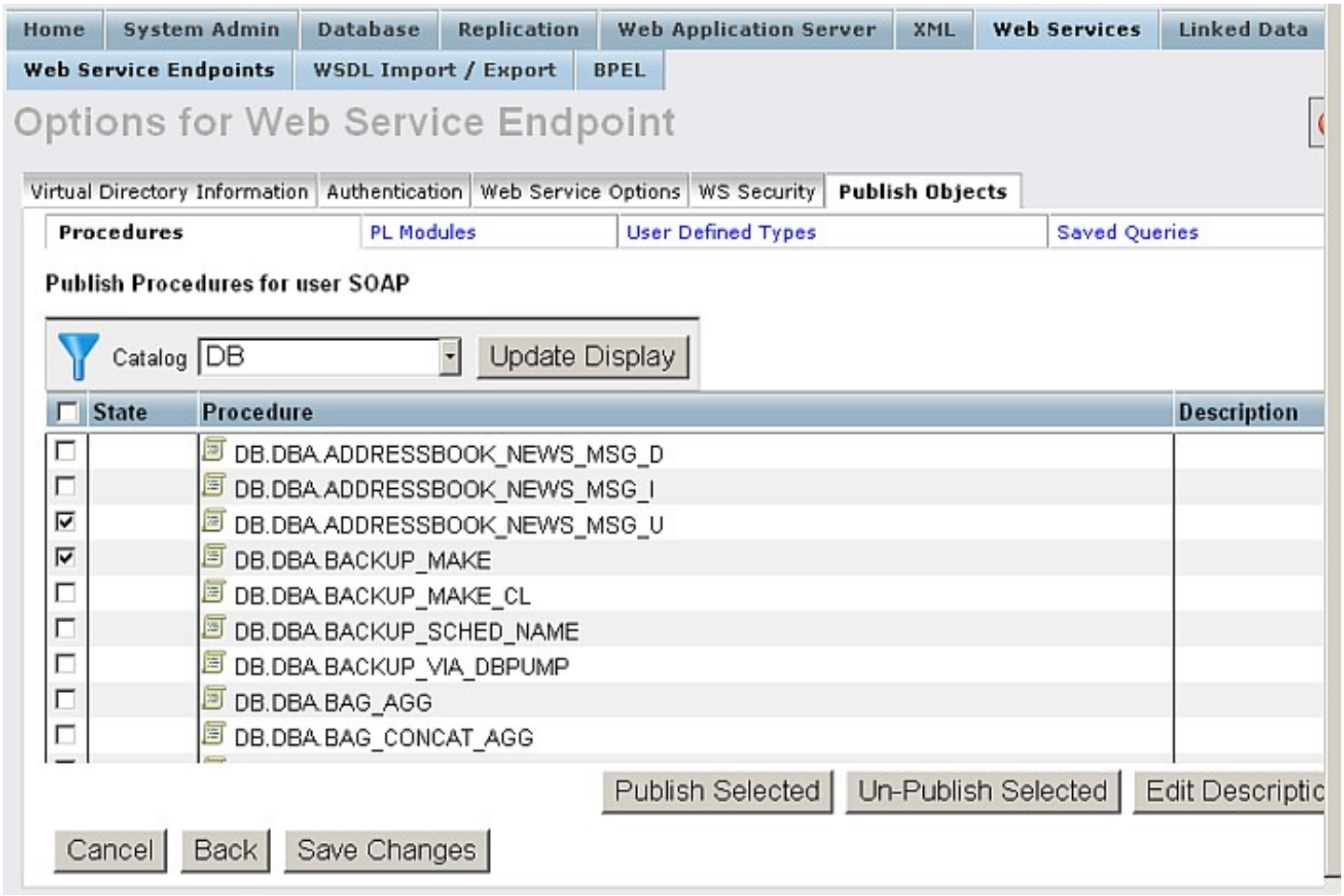


You will then be presented with the following tabs: "Virtual Directory Information", "Authentication", "Web Service Option", "WS Security" and "Publish Objects". Particular options to note are "Virtual Directory Information" and "Publish Objects".

Figure 3.44. Virtual Directories

In *Publish Objects* you can select Virtuoso stored procedures, or remotely linked procedures to be published as SOAP web services. Also you can publish Pl Modules, User Defined Types, or Saved Queries.

Figure 3.45. Publish Objects

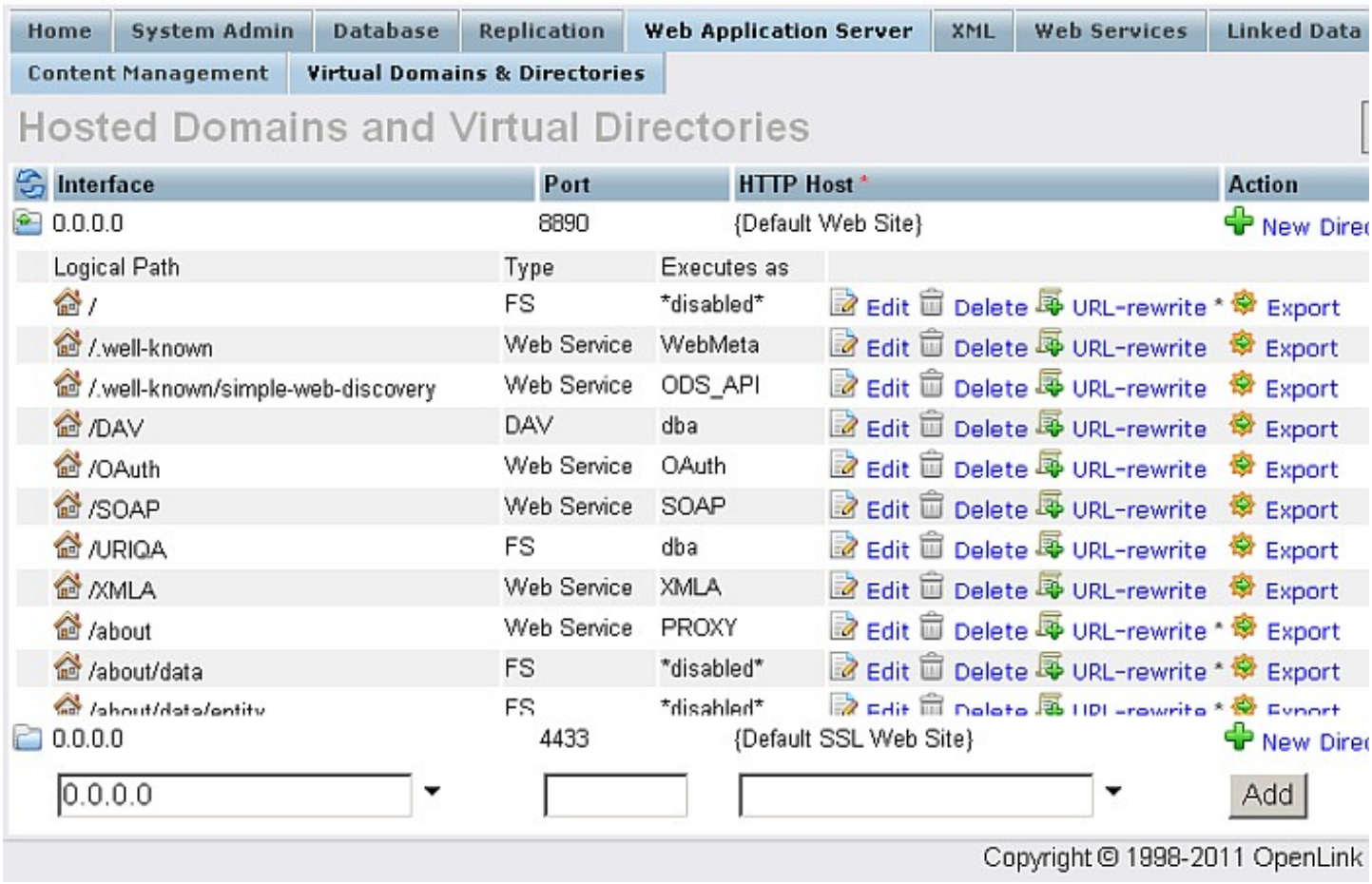


See Also:
Virtual Directories

Publishing Procedures to a Virtual Directory

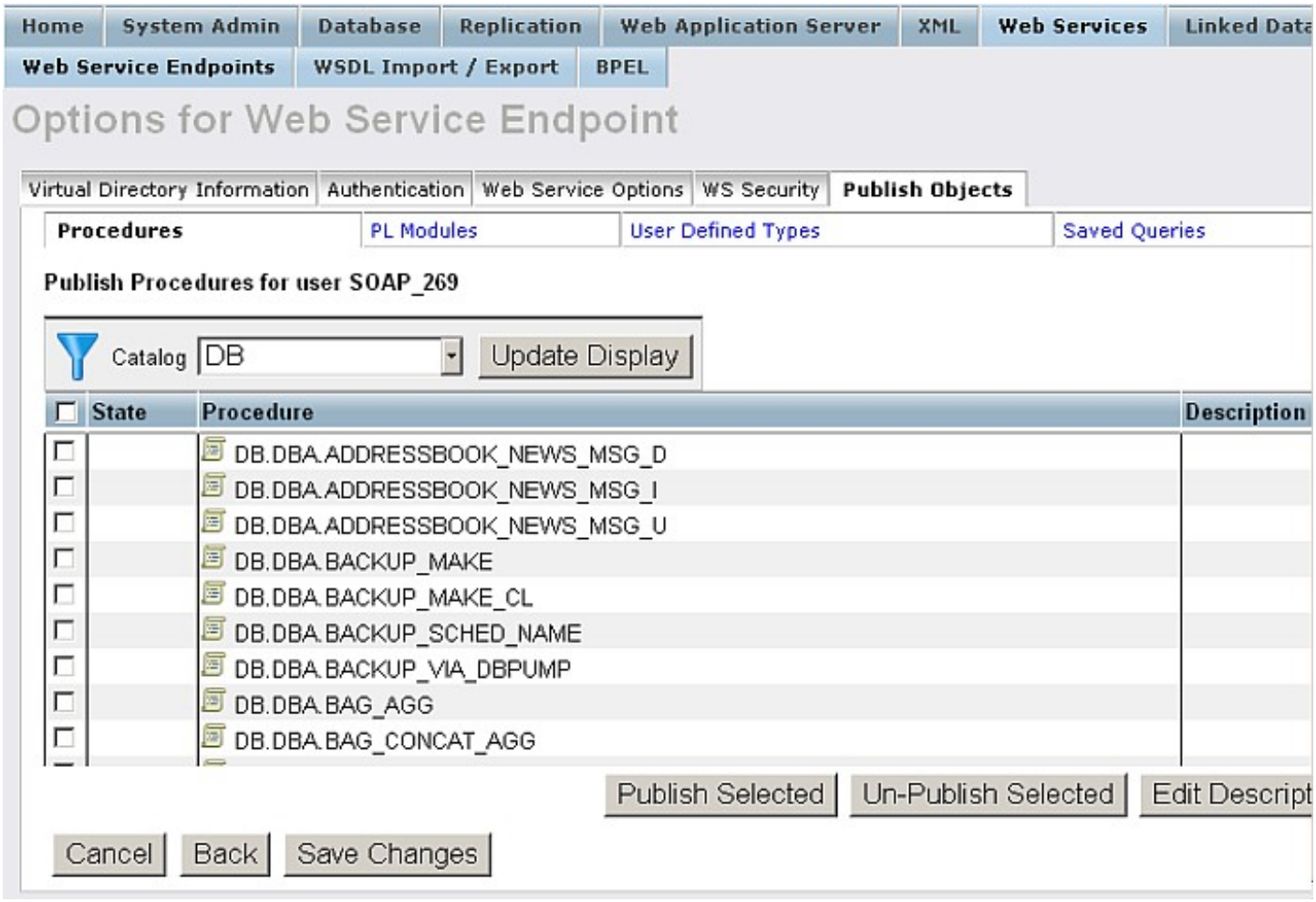
If you already have a virtual directory defined and know what procedures you want to expose as web services you will have to repeat some of the steps in the section above. From Conductor go to *Web Application Server/Virtual Domains & Directories* . Click on the "folder" icon for your {Default Web Site}. You will find the list of previously existing mappings, from which you can select the mapping that you want to edit by pressing on its *Edit* link. Note, the virtual directory should have type "SOAP".

Figure 3.46. Virtual Directories



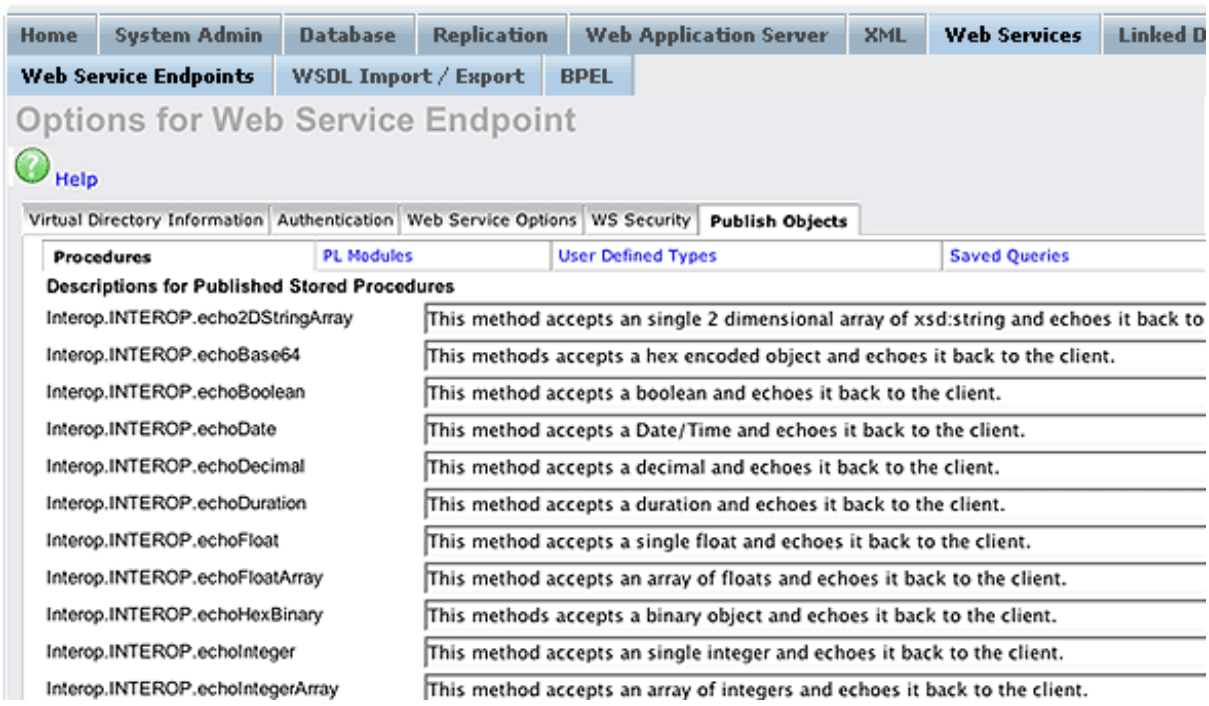
Go to tab "publish Objects" to expose/hide your procedures, Pl Modules, User Defined Types and Saved Queries.

Figure 3.47. Publish Objects



The "Procedures" tab presents the list of available procedures. You can select a catalogue in order to list the procedures you want to publish. When the procedures to be published are selected, you can either click the "Publish Selected" button, or before this to click the "Edit Description" button.

Figure 3.48. Choosing Procedure aPublish



Testing SOAP Services Using VSMX

Virtual directory definitions have a *Logical Path* field, which is reference in URL to find the correct SOAP services. If you connect to Virtuoso on *http://example.com:8890/* , and defined your virtual directory with the logical path of */mysoap* then you will be able to test the following URLs:

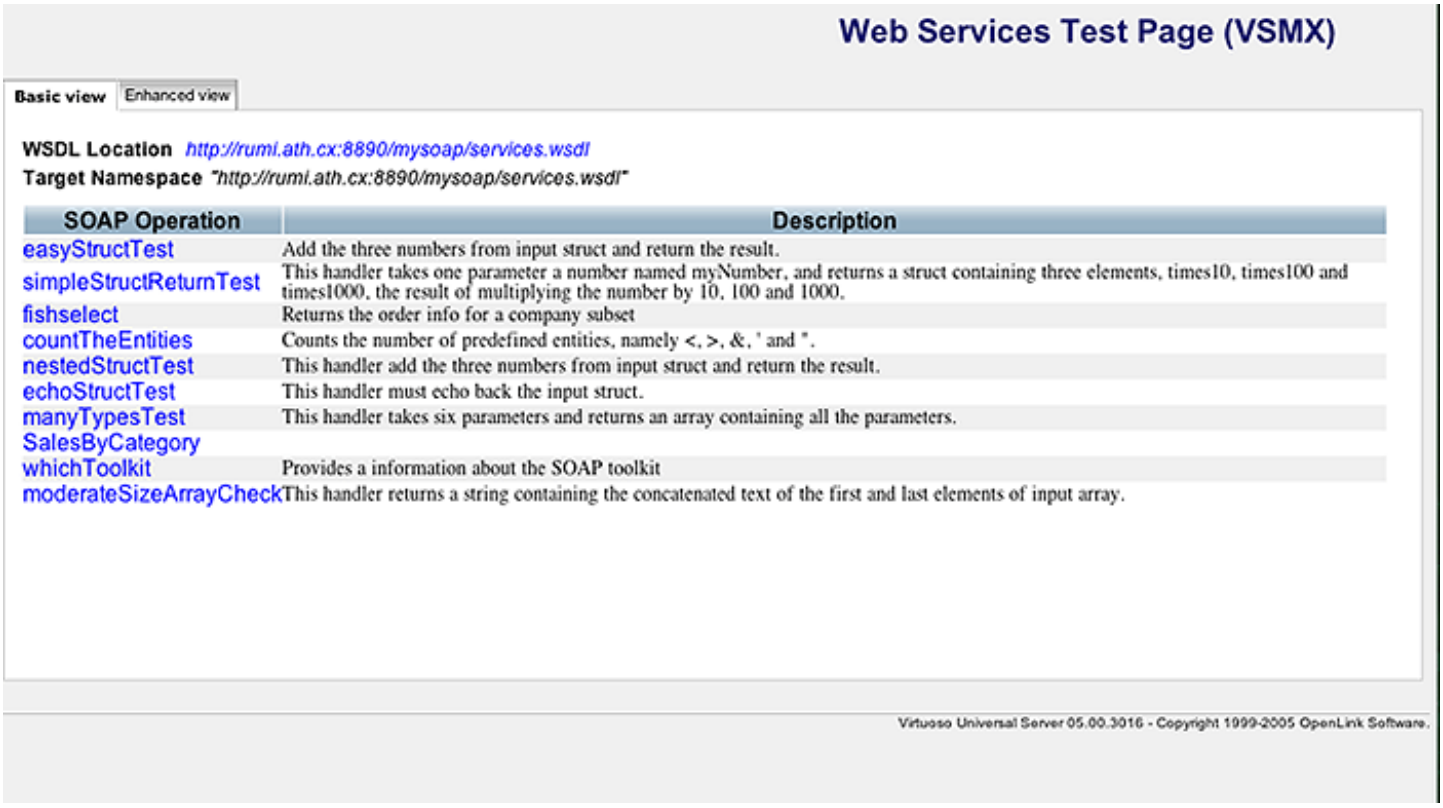
http://example.com:8890/mysoap/services.wsdl

http://example.com:8890/mysoap/services.vsmx

Figure 3.49. Services.wsdl

```
- <definitions targetNamespace="http://rumi.ath.cx:8890/mysoap/services.wsdl" name="SOAP">
  - <types>
    - <schema targetNamespace="http://tempuri.org/">
        <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
      - <complexType name="ArrayOfstring">
        - <complexContent>
          - <restriction base="soapenc:Array">
            - <sequence>
                <element name="item" type="string" minOccurs="0" maxOccurs="unbounded" nillable="true"/>
              </sequence>
              <attributeGroup ref="soapenc:commonAttributes"/>
              <attribute ref="soapenc:arrayType" wsdl:arrayType="string[]"/>
            </restriction>
          </complexContent>
        </complexType>
      - <complexType name="toolkit_info">
        - <all>
            <element name="toolkitDocsUrl" type="string"/>
            <element name="toolkitName" type="string"/>
            <element name="toolkitVersion" type="string"/>
            <element name="toolkitOperatingSystem" type="string"/>
          </all>
        </complexType>
      - <complexType name="ctc">
        - <all>
            <element name="ctLeftAngleBrackets" type="int"/>
            <element name="ctRightAngleBrackets" type="int"/>
            <element name="ctAmpersands" type="int"/>
            <element name="ctApostrophes" type="int"/>
            <element name="ctQuotes" type="int"/>
```

Figure 3.50. Services.vsmx



The WSDL description is a standards-based description of the Web Services available from /mysoap. The VSMX page is a Virtuoso generated test page allowing you to test SOAP services. This feature should improve your development time.

See Also:
VSMX ; SOAP ; WSDL .

3.7.2. XML Query Templates

Virtuoso XML templates allow execution of SQL/XML queries over HTTP to obtain an XML document in response and/or perform some operation in the database using updategrams . XML templates can be executed from within Virtuoso procedure language using the `xml_template()` function. XML templates support two types of action: SQL based or updategram based. SQL query based templates must contain the FOR XML clause used in a SELECT statement and hence cannot update the database. Updates to the database can only occur from an updategram. The XML document returned from calling an XML template can be served either raw, or transformed using XSLT.

XML templates provide quick easy access to results from a SQL query as usual, but now this can be saved to a file. The results are not saved, just the query definition. You can use this feature to rapidly produce dynamic reports that can potentially be rendered in different ways by providing an alternate stylesheet. The report can be refined on the fly by providing parameters for the query. The output is reachable via HTTP directly by providing the URL to the template.

See Also:
The XML Templates Section

XML Templates can also be published just like normal store procedures. This is achieved by using a PL wrapper around the XML template. Then the store procedure is published in the normal way.

Stylesheets transformations with be inactive for published XML templates invoked from SOAP.

See Also:
The Publishing Stored Procedures Section above for a further description of publishing XML Templates.

3.7.3. Publishing VSE's as Web Services

The Virtuoso distribution includes the sample VSE, bif_sample.c. It is thus possible to create a function such as:

```
.....
static caddr_t
bif_hello_world (caddr_t * qst, caddr_t * err_ret, state_slot_t ** args)
{
  return box_dv_short_string ("Hello world.");
}
....
```

Then declare it in the init_func() by adding the following code:

```
...
  bif_define_typed ("hello_world", bif_hello_world, &bt_any);
...
```

The next step is creating a stored procedure that calls this function and you are back to publishing a Virtuoso stored procedure again, as in the above section.

```
create procedure BIF_HELLO_WORLD () { return hello_world (); };
```

See Also:
The C Interface Chapter

3.8. VSMX - Virtuoso Service Module for XML

Every WSDL file generated by Virtuoso is automatically accompanied by a SOAP Operations Test page known as a VSMX file, which has the extension .vsmx. You find this file in the same place as the WSDL file. For a SOAP enabled virtual directory you have the facility to find the WSDL file:

```
http://[host:port]/[SOAP Virtual Directory]/services.wsdl
```

likewise you also get a VSMX file:

```
http://[host:port]/[SOAP Virtual Directory]/services.vsmx
```

You simply point your web browser to this file for the test page. The demo database contains samples that can be found as:

```
http://[host:port]/SOAP/services.vsmx
```

Figure 3.51. VSMX Test Page

See Also:
The VSMX section for more details

3.9. SQL to XML

Virtuoso enables you to develop eBusiness solutions that use XML as both a Data Source and Data Interchange format. Your XML Data documents can take the form of Pure XML Documents, or documents that are transformed from SQL-XML on the fly. By supporting the XPATH query language for XML Data, you are able to use an industry standard query language to query entire XML Documents or portions of XML Documents stored within Virtuoso. Virtuoso's inclusion of an XSLT transformation engine then allow you to transform XML data for other needs. These XML documents are openly accessible to user agents such as Web Browsers via HTTP and/or WebDAV. These XML documents are described as being dynamic because they have varying degrees of sensitivity to changes that occur in the underlying database tables from which the XML data originates. Virtuoso allows you to create two types of XML documents from homogeneous or heterogeneous SQL Data on the fly:

Transient The materialization of the XML Document occurs at the time of file opening, this implies that data from the original SQL database(s) is retrieved and then transformed into XML in one operation. This format of SQL-XML document is highly sensitive to source database(s) changes.

Persistent (Time Synchronized) - The materialization of the XML Document re-occurs at a user configurable interval after initial creation. This is a caching scheme which is less sensitive to changes in the source databases(s) in favor of performance.

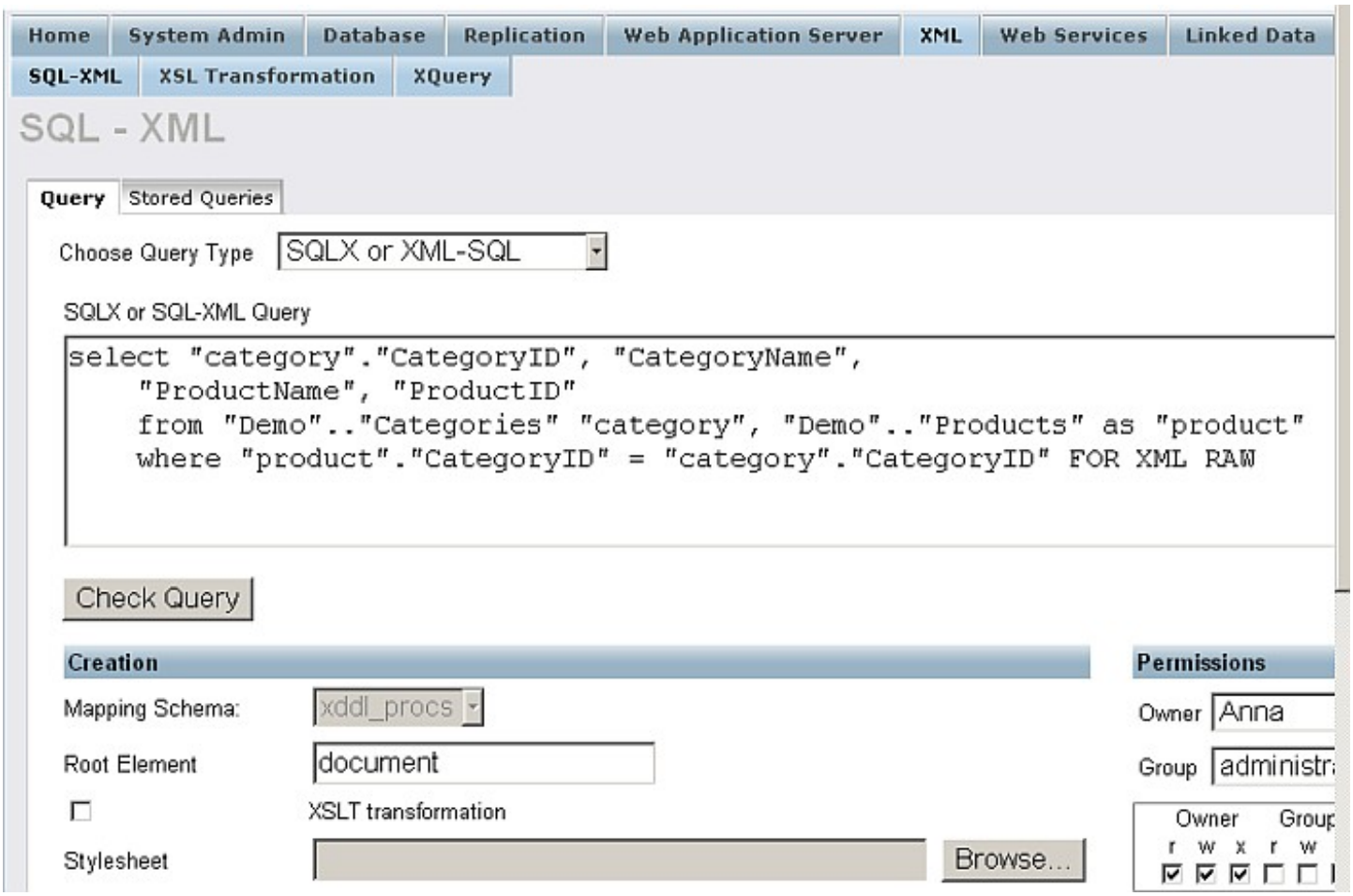
SQL-XML documents may be Valid or Well Formed XML documents, this includes support for both DTDs and XML Schemas which my be external entity references or inlined within the XML Documents prologue in the case of DTDs.

Virtuoso supports an extended SQL syntax that is identical to that implemented by Microsoft SQL Server for the purpose of creating SQL-XML documents. These SQL extensions take the form of a new "FOR XML" clause that includes three main options which control the structure of the resulting XML document tree. These options are *RAW* , *AUTO* and *EXPLICIT* .

Virtuoso's HTML based graphical interface includes a user friendly mechanism for creating dynamic XML documents from SQL data using the "FOR XML" extended SQL syntax. The dynamic XML documents created by this process are typically stored in Virtuoso's WebDAV repository. Documents stored in this repository are accessible by any XML consuming client application via HTTP, Windows Web Folders, or any other WebDAV or HTTP compliant environment. A description of the interface in general can be found in the SQL-XML Statements in the Visual Server Administration Interface section.

From Conductor *XML/SQL_XML* you can execute SQL query with options on how to produce XML structures from the results.

Figure 3.52. SQL to XML



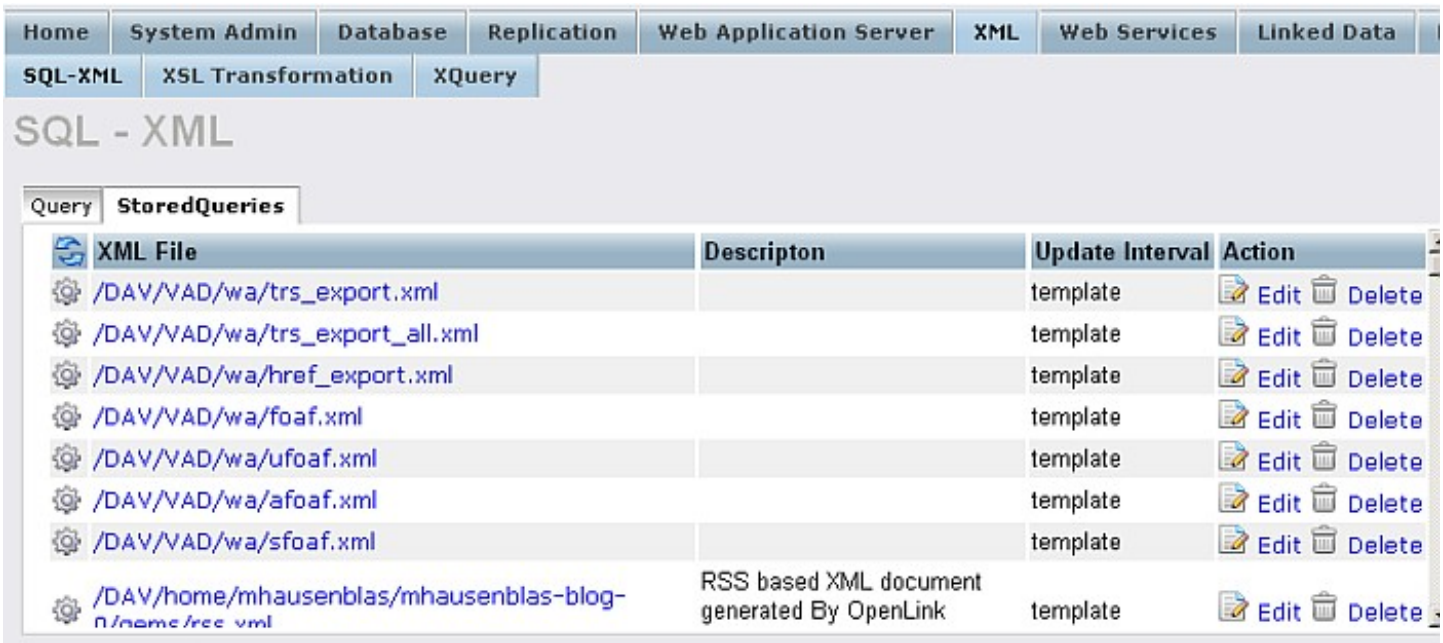
The illustration above depicts the fact that only minor changes to standard SQL are required in order to create powerful dynamic XML documents from SQL. It also illustrates how the entire process of controlling the type and format of the XML documents and their actually WebDAV storage is all achieved without any programming. The XML document extract below is a depiction of the XML document tree produced using the "FOR XML" AUTO option.

Figure 3.53. SQL to XML results

```
- <document>
  - <Customers Customers.CustomerID="ALFKI">
    - <Orders Orders.OrderID="10643">
        <Employees EmpLastName="Suyama"/>
      </Orders>
    - <Orders Orders.OrderID="10692">
        <Employees EmpLastName="Peacock"/>
      </Orders>
    - <Orders Orders.OrderID="10702">
        <Employees EmpLastName="Peacock"/>
      </Orders>
    - <Orders Orders.OrderID="10835">
        <Employees EmpLastName="Davolio"/>
      </Orders>
    - <Orders Orders.OrderID="10952">
        <Employees EmpLastName="Davolio"/>
      </Orders>
    - <Orders Orders.OrderID="11011">
        <Employees EmpLastName="Leverling"/>
      </Orders>
    </Customers>
  - <Customers Customers.CustomerID="ANATR">
    - <Orders Orders.OrderID="10308">
        <Employees EmpLastName="King"/>
      </Orders>
    - <Orders Orders.OrderID="10625">
        <Employees EmpLastName="Leverling"/>
      </Orders>
    - <Orders Orders.OrderID="10759">
        <Employees EmpLastName="Leverling"/>
      </Orders>
    - <Orders Orders.OrderID="10926">
```

The Virtuoso Demo database provides a set of sample tables in the Demo catalogue, and some sample XML views that use them. The "StoredQueries" tab lists saved XML Views as shown below.

Figure 3.54. SQL to XML save views



You can press *Edit* to edit them, or *Delete* to remove them or click on the XML FILE itself to see the results in your default browser, a sample of the output is shown above.

3.9.1. FOR XML Execution Modes

Now we will consider the programmatical approach along side the visual interface approach. We will have one example of each of the modes of FOR XML combined with the `xml_auto()` function to help us display the results simply.

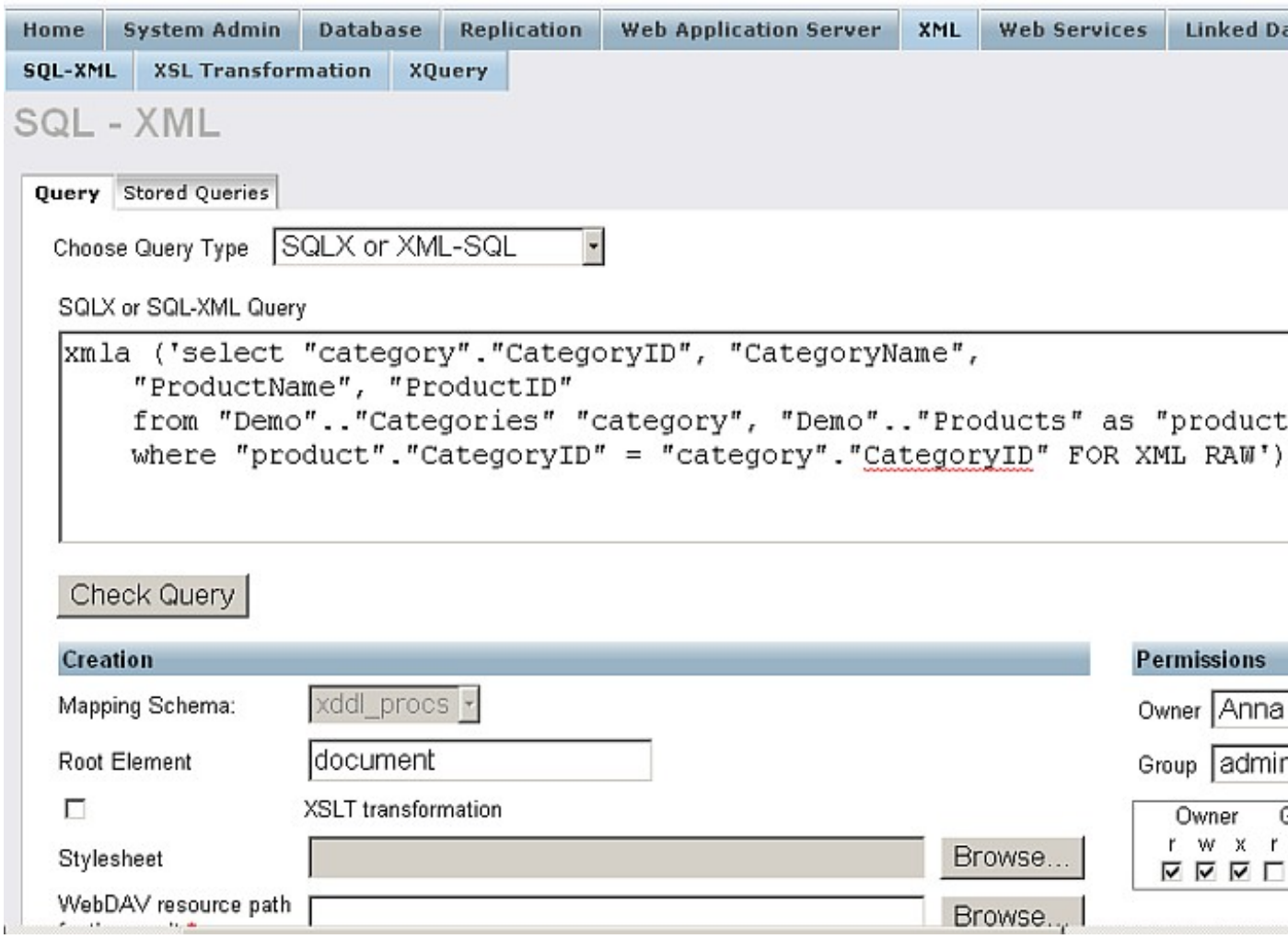
For the programmatical examples to run smoothly using ISQL a number of steps are required to obtain textual output from the xml_auto() function which usually is expected to output directly to an HTTP target. To make the demonstration simpler a utility

procedure will be created that will simply enable us to supply SQL and return XML using the `xml_auto()` function.

```
create procedure xmla (in q varchar)
{
  declare st any;
  st := string_output ();
  xml_auto (q, vector (), st);
  result_names (q);
  result (string_output_string (st));
}
```

- *RAW* mode produces an XML entity from each row of the result set, and does not attempt to construct hierarchies. Each row's data is enclosed in a ROW element and each column is either an attribute or child element.

Figure 3.55. SQL to XML using FOR XML RAW mode



The same SQL statement containing the FOR XML syntax is used in the visual interface shown above, and in the programmatical version shown below. This is because both use the xml_auto() function for generating results. In the visual interface once the settings and query have been confirmed you press the "Execute" button to store the query in the specified DAV location.

```
xmla ('select "category"."CategoryID", "CategoryName",
    "ProductName", "ProductID"
    from "Demo".."Categories" "category", "Demo".."Products" as "product"
    where "product"."CategoryID" = "category"."CategoryID" FOR XML RAW');
```

Note:

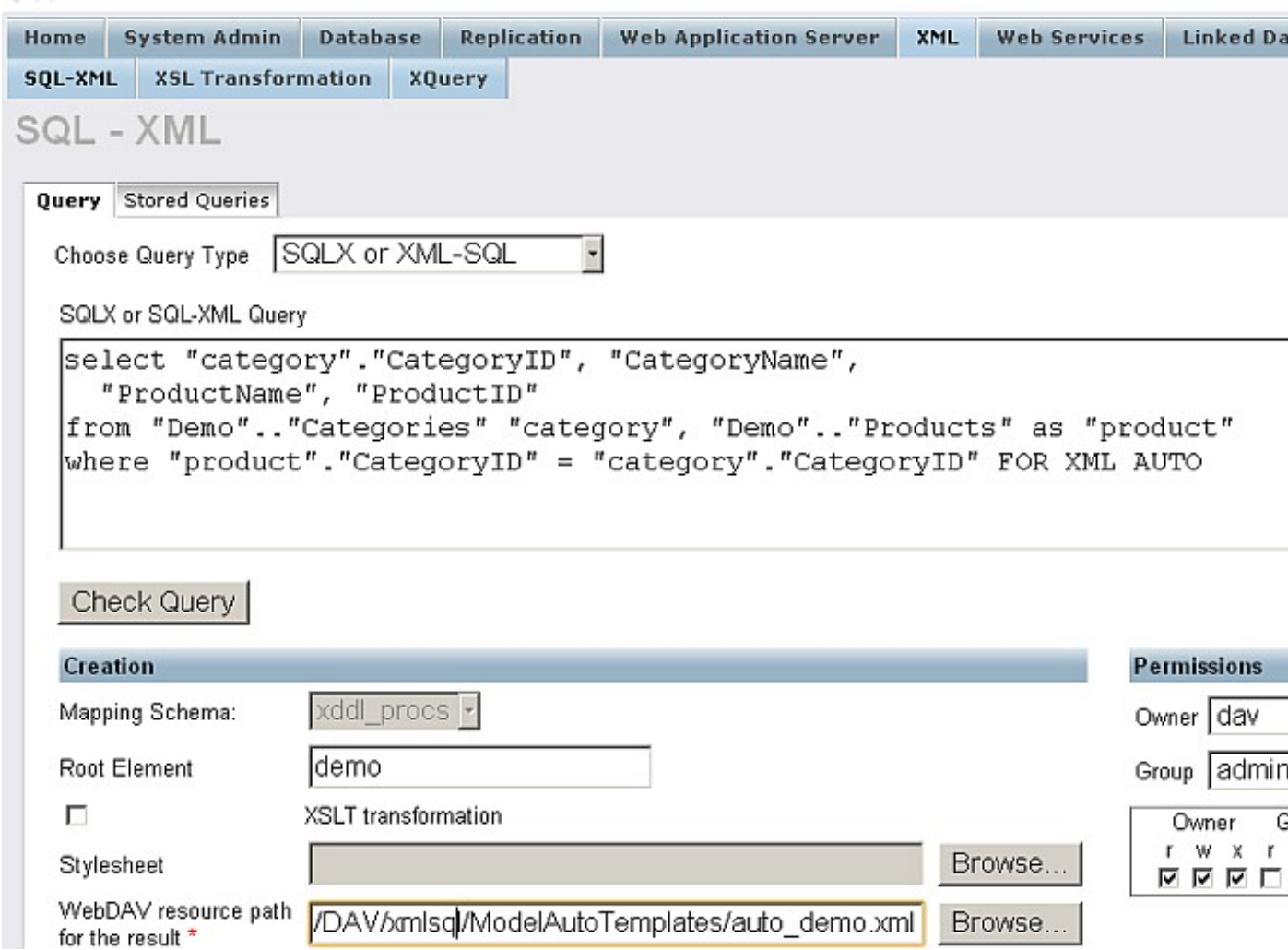
The xmla function is not a standard function but quick utility for quickly rendering a text output for the `xml_auto()` function. The definition is at the top of this section

The resulting XML from either ISQL or the saved links on the visual interface will yield:

```
<ROW CategoryID="1" CategoryName="Beverages" ProductName="Chai" ProductID="1">
</ROW>
<ROW CategoryID="1" CategoryName="Beverages" ProductName="Chang" ProductID="2">
</ROW>
<ROW CategoryID="1" CategoryName="Beverages" ProductName="Guaran Fantastica" ProductID="24">
</ROW>
<ROW CategoryID="1" CategoryName="Beverages" ProductName="Sasquatch Ale" ProductID="34">
</ROW>
<ROW CategoryID="1" CategoryName="Beverages" ProductName="Steeleye Stout" ProductID="35">
</ROW>
<ROW CategoryID="1" CategoryName="Beverages" ProductName="C(te de Blaye" ProductID="38">
</ROW>
<ROW CategoryID="1" CategoryName="Beverages" ProductName="Chartreuse verte" ProductID="39">
</ROW>
<ROW CategoryID="1" CategoryName="Beverages" ProductName="Ipoh Coffee" ProductID="43">
</ROW>
.....
```

- *AUTO* mode. A hierarchy is constructed with one level for each table of the join for which at least one column is selected. The table whose column is first mentioned in the selection will be the topmost element, the next table its child etc. Each level of the tree will consist of one type of element. A parent element will have multiple children if consecutive rows do not differ in the column values coming from the parent element. When a table's column values differ from the previous row, the element and all children thereof are closed and a new element is started, with children filled out from other columns of the result set.

Figure 3.56. SQL to XML using FOR XML AUTO mode



The same SQL statement containing the FOR XML syntax is used in the visual interface shown above, and in the programmatical version shown below. This is because both use the xml_auto() function for generating results. In the visual interface once the settings and query have been confirmed you press the execute button to store the query in the specified DAV location.

```
xmla ('select "category"."CategoryID", "CategoryName",
    "ProductName", "ProductID"
    from "Demo".."Categories" "category", "Demo".."Products" as "product"
    where "product"."CategoryID" = "category"."CategoryID" FOR XML AUTO');
```

Note:

The xmla function is not a standard function but quick utility for quickly rendering a text output for the `xml_auto()` function. The definition is at the top of this section

The resulting XML from either ISQL or the saved links on the visual interface will yield:

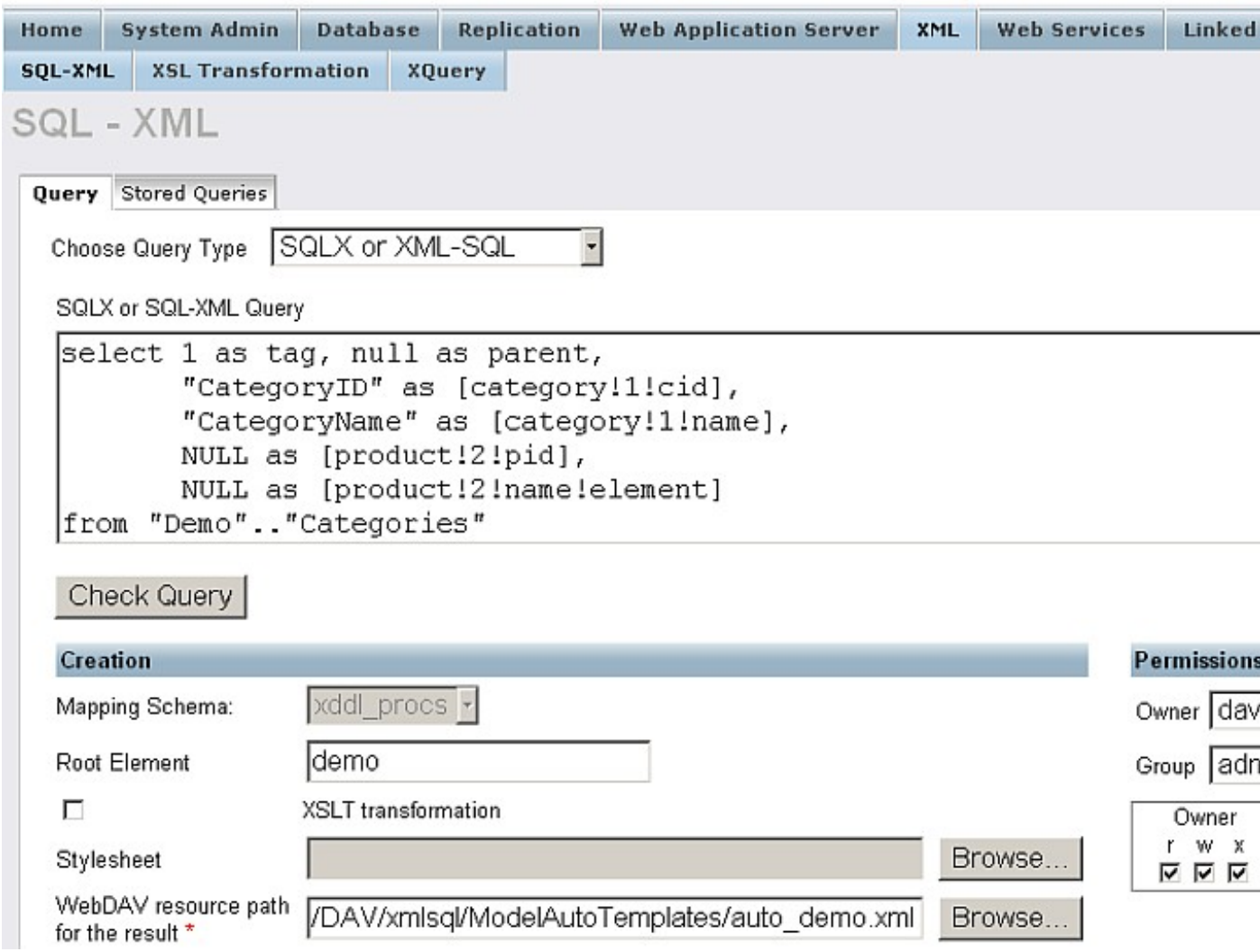
```
<category CategoryID="1" CategoryName="Beverages">
<product ProductName="Chai" ProductID="1">
</product>
<product ProductName="Chang" ProductID="2">
</product>
<product ProductName="Guaranß Fantßstica" ProductID="24">
</product>
<product ProductName="Sasquatch Ale" ProductID="34">
</product>
<product ProductName="Steeleye Stout" ProductID="35">
</product>
<product ProductName="C(te de Blaye" ProductID="38">
</product>
<product ProductName="Chartreuse verte" ProductID="39">
</product>
<product ProductName="Ipoh Coffee" ProductID="43">
</product>
<product ProductName="Laughing Lumberjack Lager" ProductID="67">
</product>
.....
```

Note:

In contrast to the RAW mode AUTO produces results that are more reasonable and intuitive. Only one category element is used for each category which contains all the children of that category.

- *EXPLICIT* mode gives more control on the resulting tree's structure while requiring a more elaborate query structure. In this mode, the query will be a UNION ALL of many joins and each row will specify exactly one element. Which type of element this is and where in the tree it will be placed are determined by the values of the 2 first columns, TAG and PARENT.

Figure 3.57. SQL to XML using FOR XML EXPLICIT mode



The same SQL statement containing the FOR XML syntax is used in the visual interface shown above, and in the programmatical version shown below. This is because both use the xml_auto() function for generating results. In the visual interface once the settings and query have been confirmed you press the execute button to store the query in the specified DAV location.

```
xmla ('
select 1 as tag, null as parent,
       "CategoryID" as [category!1!cid],
       "CategoryName" as [category!1!name],
       NULL as [product!2!pid],
       NULL as [product!2!name!element]
from "Demo".."Categories"
union all
select 2, 1, "category" ."CategoryID", NULL, "ProductID", "ProductName"
    from "Demo".."Categories" "category", "Demo".."Products" as "product"
    where "product"."CategoryID" = "category"."CategoryID"
order by [category!1!cid], 5
FOR XML EXPLICIT');
```

Note:

The xmla function is not a standard function but quick utility for quickly rendering a text output for the `xml_auto()` function. The definition is at the top of this section

The resulting XML from either ISQL or the saved links on the visual interface will yield:

```
<CATEGORY CID="1" NAME="Beverages">
<PRODUCT PID="1">
 <NAME>Chai</NAME></PRODUCT>
<PRODUCT PID="2">
 <NAME>Chang</NAME></PRODUCT>
<PRODUCT PID="24">
```

```
 <NAME>Guaran&#225; Fant&#225;stica</NAME></PRODUCT>
<PRODUCT PID="34">
 <NAME>Sasquatch Ale</NAME></PRODUCT>
<PRODUCT PID="35">
 <NAME>Steeleye Stout</NAME></PRODUCT>
<PRODUCT PID="38">
 <NAME>C&#244;te de Blaye</NAME></PRODUCT>
<PRODUCT PID="39">
 <NAME>Chartreuse verte</NAME></PRODUCT>
<PRODUCT PID="43">
 <NAME>Ipoh Coffee</NAME></PRODUCT>
<PRODUCT PID="67">
 <NAME>Laughing Lumberjack Lager</NAME></PRODUCT>
<PRODUCT PID="70">
 <NAME>Outback Lager</NAME></PRODUCT>
<PRODUCT PID="75">
 <NAME>Rh&#246;nbr&#228;u Klosterbier</NAME></PRODUCT>
<PRODUCT PID="76">
 <NAME>Lakkalik&#246;&#246;ri</NAME></PRODUCT>
</CATEGORY>
<CATEGORY CID="2" NAME="Condiments">
<PRODUCT PID="3">
.....
```

Note:
In contrast again, the EXPLICIT mode produces exactly what we asked for.

For more details about 'FOR XML, refer to Rendering SQL Queries as XML section of the XML Support chapter.

3.9.2. Tables With XML Columns

XML is a new native Virtuoso datatype, based on an extension of LONG VARCHAR for compatibility with ODBC clients, allows direct storage, retrieval and querying of XML stored in a database table. This has always been possible with Virtuoso utilizing `xml_tree_doc()` and friends but now you can easily concentrate on what more important, the data, and not which datatype to convert it to next.

See Also:
XML Column Type

3.10. NNTP

Virtuoso supports the Network News Transfer Protocol used by Internet newsgroup forums. NNTP servers manage the global network of collected newsgroup postings and represent a vast repository of targeted information archives. As an NNTP aggregator, Virtuoso enables integration of multiple news forums around the world. All news content in Virtuoso is dynamically indexed to provide keyword searches, enabling rapid transformation of disparate text data into information. Virtuoso also acts as an NNTP server, enabling creation of new Internet and Intranet News Forums to leverage the global knowledge base into eBusiness Intelligence.

3.10.1. NNTP Server Setup

Enable Server

Before the NNTP server can be used, it has to be enabled to listen on the NNTP port. This change is made in the configuration file.

For more details, refer to Enable NNTP Server section.

Create/Attach News Groups

The definition of news groups is held in system tables.

For more details on inserting news groups by SQL command, refer to Add Groups to NNTP Server section. See also the Conductor News Server and Newsgroups Administration section, to setup the groups in the Visual Server Administration Interface.

Limit Groups

This is an optional step, appropriate if a news group is to be limited for internal use only, or by a group of IP addresses. This is achieved by creating an Access Control List (ACL) in the DB.DBA.NEWS_ACL table. If no ACL is defined, then all groups are public readable and writable.

For table details, refer to NNTP Server Tables section.

3.10.2. Local & Remote Groups

News groups may be Local such that they are the only instance, or a master instance of a group. Remote groups are ones that are replicated from other news servers.

See the Conductor News Server and Newsgroups Administration section, to setup the groups in the Visual Server Administration Interface.

3.10.3. NNTP Client Setup

Virtuoso can make a client connection to an external news server to receive newsgroup postings.

For more details, refer to the NNTP Client section.

3.11. Dynamic Web Pages

Virtuoso provides an extensible array or dynamic methods for creating data driven web pages. Through runtime hosting Virtuoso can directly host, store and drive:

PHP
JSP
ASP.Net
Perl
Python
Ruby
Natively - VSP and VSPX:
Virtuoso Server Pages (VSP) is Virtuoso's specification for creating dynamic database driven Web pages, these files have the extension ".vsp" and are identical in functionality to: ASP, PHP, and JSP pages. A fundamental difference between VSP pages and others (PHP, ASP. and JSP pages) is the fact that VSPs are specialized forms of Virtuoso Stored procedures which implies that data is in-process rather than out-of-process, you do not have to complete a client-server connection in order to actually bind to the data being used in a VSP page. The obvious benefit being significant performance improvements over ASP, PHP, and JSP pages (which all bind to data out-of-process).

Since VSP is essentially Virtuoso PL in a web page you can do anything that PL can from or part of a web page either directly or from interaction with the user. A massive advantage of using VSP is that you do not have to worry about making connections to the database or the overhead of RPC's because the HTTP server is built into Virtuoso. When you write a VSP page the connection is assumed since you are already in Virtuoso! VSP is server script and is therefore executed in the server as it is encountered on the page. For this reason client (JavaScript) and server script cannot directly interact but can supplement each other. You can call JavaScript inside a VSP loop, for example, to manipulate something that already exists on the page but you cannot pass variables by reference from VSP directly to JavaScript or vice versa. Page flow control is managed using FORMs. The state of the page is defined in form fields such as INPUT boxes and TEXTAREA boxes and then passed on to the next form or page using POST.

Virtuoso Server Pages for XML (VSPX) is an XML based framework and state-managed system similar to ASPX. Pages are written in XML to describe templates of data-aware web-widgets. This massively reduces the code-effort and avoids many bugs by providing the functionality for you, of which you specify the data-source, be it XML, SQL or other, and which predefined control you want it represented by. VSPX allows for custom-designed controls too. Since XML is a key factor for VSPX so does XSLT for providing total separation between the data, business logic and the layout on the web page.

See Also:
The Web Application Development Chapter.

3.12. VSP Examples

3.12.1. Simple HTML FORM usage

We will start with a small example that shows a page that constitutes a FORM with data from the user being sent when a submit button is pressed and then examine the elements and attributes that are important to us at this stage.

Consider the following piece of HTML:

```
<HTML>
  <HEAD>
    <TITLE>Simple FORM demo</TITLE>
  </HEAD>
  <BODY>
  <FORM METHOD="POST" ACTION="formdemo_receiver.vsp">
    <P>Test form, type some info and click Submit</P>
    <INPUT TYPE="TEXT" NAME="myInput" />
    <INPUT TYPE="SUBMIT" NAME="submit" VALUE="Submit" />
  </FORM>
  </BODY>
</HTML>
```

All elements contained in the FORM tag are associated with that form. This is how the data and the submit button know what to do next, and which data to send, by the attributes of the FORM tag.

The METHOD attribute of the FORM TAG can be either GET or POST. The GET method allows the form submission to be contained completely in a URL which can be advantageous in that it permits bookmarking in browsers, but it also prevents form data from containing non-ASCII characters and restricts the amount of form data that can be handled. The get method is limited by the maximum length of the URL that the server and browser can process. To be safe, any form whose input might contain non-ASCII characters or more than 100 characters should use METHOD="POST".

With the POST method, the form input is submitted as an HTTP POST request with the form data sent in the body of the request. Most current browsers are unable to bookmark POST requests, but POST does not entail the character encoding and length restrictions imposed by GET.

The ACTION attribute of FORM specifies the URI of the form handler. This will usually be another web page that performs some action based on the data that is sent from the originating form. The URI could point to the same page as the data originated and for pages that perform a well defined small set of functions it usually does. When a page needs to manage multiple states there needs to be some flow control that can determine how the page was reached whether it arrived as a result of someone clicking on the submit button or it is the first time the page has been visited.

Now we will add some VSP to check the values of the parameters in the form. VSP markup is typically contained in <?vsp ... VSP ... ?> blocks.

```
<HTML>
  <HEAD>
    <TITLE>Simple FORM demo</TITLE>
  </HEAD>
  <BODY>
  <P>Last value sent:</P>

  <?vsp
    http(get_keyword('myInput', params, 'no value'));
   ?>

  <FORM METHOD="POST" ACTION="formdemo.vsp">
    <P>Test form, type some info and click Submit</P>
    <INPUT TYPE="TEXT" NAME="myInput" />
    <INPUT TYPE="SUBMIT" NAME="submit" VALUE="Submit" />
  </FORM>
  </BODY>
</HTML>
```

This is the same example as above that now uses the same page for the form handler and displays the parameters each time. The first time you click the button will take you to the same page and will display whatever you typed in the field last time.

The VSP block uses two functions nested. The http function allows you to send data to the HTTP client, the browser. What we send to the browser is the result of the get_keyword function. The get_keyword function has 3 parameters, it searches for the keyword-value pair (keyword=value) where keyword matches the first parameter (in this case 'myInput') in array passed in the second parameter, and returns the value if one is found otherwise will return the 3rd parameter in the function 'no value'. The params array is a special array that contains all page parameters.

Now we will extend this further to add some conditional control so that if a value was entered we can respond directly to it. We will also use a variable this time, which must be declared first.

```
<HTML>
  <HEAD>
    <TITLE>Simple FORM demo</TITLE>
  </HEAD>
  <BODY>

  <?vsp
        declare _myInput varchar;

        _myInput := get_keyword('myInput', params, 'no value');

    if (_myInput <> 'no value')
        {       http('<P>Hello, ');
                http(_myInput);
                http('</P>');
        }
        else
        {       http('<P>Please enter you name</P>');
        }
   ?>

  <FORM METHOD="POST" ACTION="formdemo.vsp">
    <P>Test form, type some info and click Submit</P>
    <INPUT TYPE="TEXT" NAME="myInput" />
    <INPUT TYPE="SUBMIT" NAME="submit" VALUE="Submit" />
  </FORM>
  </BODY>
</HTML>
```

We will now extend this even further to control the whole contents of the page. In this example we will see that VSP and HTML can be interleaved.

```
<HTML>
  <HEAD>
    <TITLE>Simple FORM demo</TITLE>
  </HEAD>
  <BODY>

  <?vsp
        declare _myInput varchar;
        declare Mode varchar;

        _myInput := get_keyword('myInput', params, 'no value');
        Mode := get_keyword('submit', params, '');

    if (Mode = 'submit')
        {
   ?>

    <P>Hello, <?vsp http(_myInput); ?>
        </P>

  <?vsp
        }
          else
        {
   ?>

  <P>Please enter you name</P>
  <FORM METHOD="POST" ACTION="formdemo.vsp">
    <INPUT TYPE="TEXT" NAME="myInput" />
    <INPUT TYPE="SUBMIT" NAME="submit" VALUE="Submit" />
```

```
  </FORM>

  <?vsp
    }
  ?>

  </BODY>
</HTML>
```

3.12.2. Manipulating Database Data in VSP

The following example demonstrates a basic page that has form based flow control, takes input from the user to put into the database and then displays the results. This simple example can be extended to perform more substantial operations by adding a few more inputs, buttons and states.

Things to look at:

get_keyword is used to extract parameters from the form that were posted last time

the current mode is determined by the value of the submit parameter

straight HTML can be inline with VSP flow control which is how the whole page is contained in an *if* condition.

```
<HTML>
<HEAD></HEAD>
<BODY>
<form method="POST" action="simpletest.vsp">

<?vsp
-- assumes that you have a table db..test_table(txt varchar(2000))

-- declare variables for use
declare _mode varchar;
declare _theValue varchar;
declare _stmt varchar;

-- get the current mode and continue accordingly
_mode := get_keyword ('submit', params, 'default');

if (_mode = 'Submit')
{
  -- if a submit was detected then insert the value into the DB
  _theValue := get_keyword('myTxtBox', params, 'no comment');
  _stmt := sprintf('insert into db..test_table(txt) values(\'%s\')', _theValue);
  exec (_stmt, '', '', '', '', '', '');

?>
   <P>Thank you for your submission.</P>
<?vsp
} else {
?>
   <DIV>
   <DIV>Simple test form, enter some text and hit submit.</DIV>
   <DIV><textarea name="other" rows="3" cols="64"></textarea></DIV>
   </DIV>

   <DIV><input type="submit" name="submit" value="Submit"></DIV>

   <HR />
   <H2>Values currently in table</H2>
   <TABLE>
   <?vsp
   for (select txt from db..test_table) do
     http(sprintf('<TR><TD>%s</TD></TR>', txt));
   ?>
   </TABLE>

<?vsp
}
?>
</form>
</BODY>
</HTML>
```

You may wish to offload some of the functionality of the page to a stored procedure and call that from that page. You may do this to improve readability of the page or there may be a series of functions that you repeat such as displaying a particular table in some format.

You could used a procedure as follows:

```
create procedure table_list()
{
   http('<H2>Values currently in table</H2>');
   http('<TABLE>');
   for (select txt from db..test_table) do
      http(sprintf('<TR><TD>%s</TD></TR>', txt));
   http('</TABLE>');
};
```

You could then call this in instead of defining the query and table layout as above.

The aspects of VSP are explained in more detail in the following sections.

3.12.3. Simple Tutorial

The following example prints the result from executing explain:

```
<?vsp
  declare meta, data any;
  exec ('explain (?)', null, null, vector ('select * from sys_users'),  0, meta, data);
  foreach (any row in data) do
  {
     http_value (row[0], 'p');
  }
?>
```

The vsp can be also written like this:

```
<?vsp
  declare meta, data any;
  exec ('explain (?)', null, null, vector ('select * from sys_users'), 0, meta, data);
  for (declare i,l int, i := 0, l := length (data); i < l; i := i + 1)
   {
     http_value (data[i][0], 'p');
   }

?>
```

3.13. Third-Party Runtime Typing, Hosting & User Defined Types

All barriers are broken. If Virtuoso does not readily provide the data type that you require, then make your own. If you want a database trigger to test data against existing externally developed logic, then do that too. Virtuoso has been designed with open-design in mind giving ultimate flexibility. These are the systems current available (linked to the appropriate section of this documentation):

- Runtime Hosting

  support other environments and/or languages in-process with Virtuoso and utilizing Virtuoso storage methods such as DAV for replication and roll-out benefits.

  - CLR & Mono
  - Java & Jakarta JSP
  - PHP
- Plugins

  enable support for other scripting langauges.

  - Perl
  - Python

- Ruby
- Extensibility

  the above features are applications of one or another of these interfaces, which are provided so that you have the potential to enhance Virtuoso further for more custom requirements.

  - Virtuoso Server Extension Interface (VSEI)
  - VSEI Plugins
  - User-Defined Types (UDT)
  - Hosted/Imported Assemblies/Classes
- Web/Service Exposure

  every part of Virtuoso can be view, interacted with or consumed by some third-party via a plethora of interfaces, to name a few:

  - SOAP
  - WSDL
  - Static/Dynamic Web Content
  - XML/XSLT

    XML Storage System

    XML RPC

3.14. Troubleshooting Tips

3.14.1. General Tips

The following sections are some common faults and tips. For a complete list of troubleshooting tips, please visit sources that are listed in the Product Support section of the Appendix .

3.14.2. DBMS Server will not start

If the Virtuoso DBMS server won't start, there could be 3 reasons.

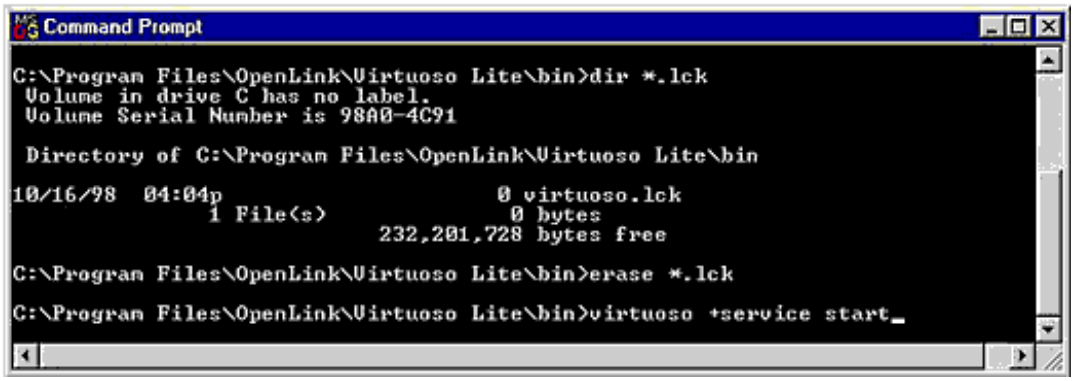
It is already running

It died earlier

It terminated normally earlier, but the virtuoso.lck file was not deleted

When the DBMS server starts up, it creates a file in the bin directory named "virtuoso.lck". If this file is present, a new instance cannot run. If you are certain that the DBMS isn't running, then you can delete the virtuoso.lck file and then start it from the Services icon in the Control Panel, or by using the command "*virtuoso +service start* " in the virtuoso directory. You can check if virtuoso is running from the Task Manager, and you can shut it down using the command "*virtuoso +service stop* ".

Figure 3.58. Command Prompt

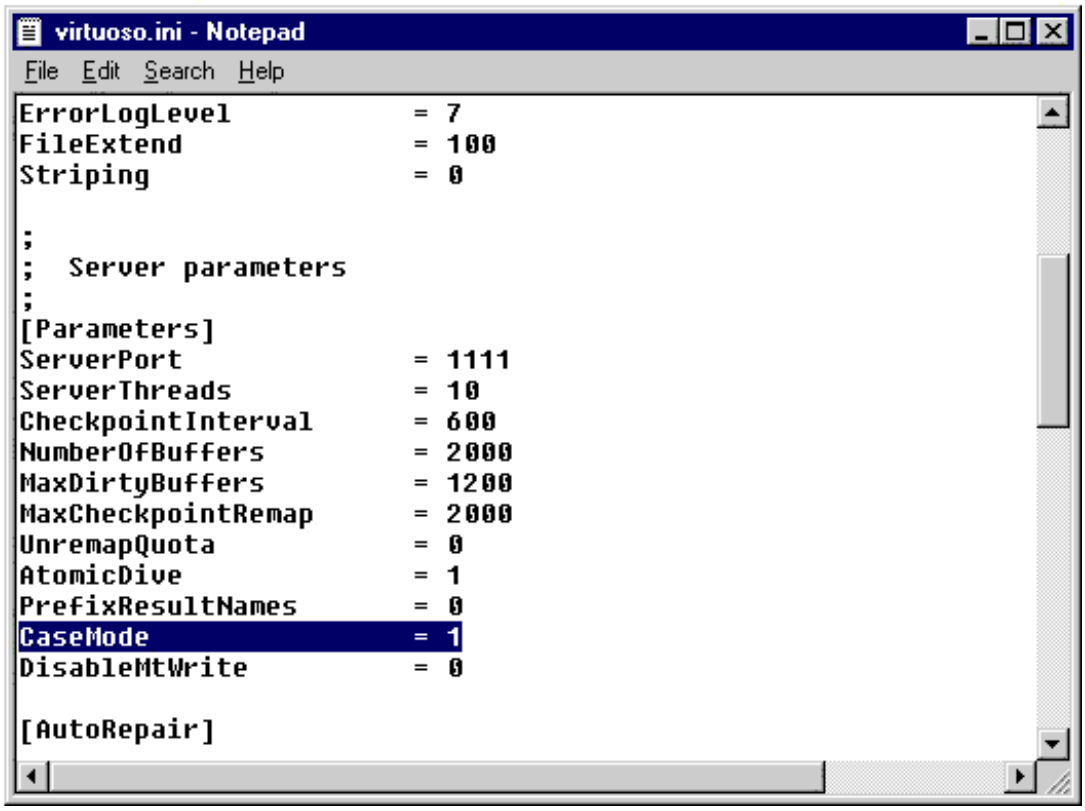


3.14.3. Case Mode

Whenever a database object is referenced, all names (schema, owner, table and column) should preferably be placed in double-quotes, exactly as it appears in the catalog. This way, the object name in the Virtuoso catalog will be correctly referenced. If names are not quoted, then the case of the object name will be determined by the value of the CaseMode property in the virtuoso.ini file:

- The default files supplied with Virtuoso specify a CaseMode of 2, which is a case insensitive mode that preserves the declaration case of identifiers. If there is no supplied ini file, the default value of CaseMode is 1.
- A CaseMode of 1 specifies the upper case mode, which is most commonly used in SQL databases, e.g. Oracle. In the upper case mode, all unquoted identifiers are converted to upper case by the SQL parser. If identifiers are not quoted, the case in which they are entered is irrelevant.
- The identifier quote character is the double quote ("). Quoted identifiers are processed in the case they are written in and are thus case sensitive.
- SQL reserved words are case insensitive in all case modes.
- If CaseMode is 0 or absent, identifiers will be treated as case sensitive in all situations, whether quoted or not.
- If an identifier's name is equal to a SQL reserved word, e.g. TABLE, it must be quoted ("TABLE") in order to be used as an identifier.
- If an identifier contains non-alphanumeric characters, e.g. space, '-' etc. it must be quoted regardless of CaseMode.
- Although CaseMode can be changed at any time it should only be set at database creation. Changing the CaseMode may result in view or procedure code becoming invalid if it relies on specific case conventions.

Figure 3.59. Virtuoso.ini file in notepad



Chapter 4. Sample ODBC & JDBC Applications

Abstract

This chapter applies exclusively to the various commercial releases of Virtuoso. If you are working with the open source version, please refer to the instructions on the web site where you obtained it.

The OpenLink Virtuoso Installation provides sample applications to help you get quick and easy results from you installation. This chapter explains how to use these applications.

A number of sample applications are bundled with your Virtuoso installation for the following purposes:

To simplifying the process of getting Virtuoso up and running

To accelerate the support case creation and resolution process

To demonstrate Virtuoso's unique product features highlighting the benefits it brings to your organization

To demonstrate application programming techniques that can used to aid and assist your ODBC and JDBC programmers

Virtuoso's services are consumed primarily via ODBC and JDBC applications (OLE-DB applications connect to Virtuoso via ODBC Data Providers for OLE-DB), thus separate ODBC & JDBC sample applications (including source code) have been packaged and integrated into the Virtuoso installer. The current list of sample applications include:

- **C++ Demo.** an ODBC based Interactive SQL processor written in C++.
- **ODBC Bench Test.** a 32 Bit C++ program based on the industry standard TPC-A benchmark (we will be extending this program to include the TPC-C and TPC-D benchmarks also). This program helps you compare the performance of Virtuoso against other backend database engines as well as compare the performance of various ODBC Drivers connecting to any ODBC compliant backend database.
- **ODBCTEST.** ODBC based Interactive SQL processor written in 'C' for Linux & UNIX

- **JDBCDemo.** a JDBC sample application that demonstrates Virtuoso's SQL query.
- **ScrollDemo2.** a JDBC 2.0 sample application that demonstrates Virtuoso's support of Scrollable Cursors and its ability to perform scrollable cursor operations across heterogeneous databases.
- **JBench.** a Java and JDBC based adaptations of the industry standard TPC-A and TPC-C benchmarks. This program helps you compare the performance of Virtuoso against other backend database engines, it also helps you to compare the performance of various JDBC Drivers connecting to any JDBC compliant backend database.
- **JTADemo.** a sample based on the TPC-A benchmark as well but implemented as a J2EE application which shows the use of XA distributed transactions as defined in JDBC 3.0 and JTA 1.0 specifications.

4.1. Binary & Source File Locations

4.1.1. ODBC Demonstration Applications

Windows 95/98/NT/2000, Linux & UNIX:

The binary executables of these sample applications reside under the following directory structure:

```
<VIRTUOSO_INSTALLATION_DIRECTORY>\samples\odbc
```

The source code of some of these sample applications, when available, reside under the following directory structure, for example:

```
<VIRTUOSO_INSTALLATION_DIRECTORY>\samples\odbc\cppdemo
```

4.1.2. JDBC Demonstration Applications

Windows 95/98/NT/2000, Linux & UNIX:

The binary executables (Java class files), and sources for these sample applications reside under the following directory structure:

```
<VIRTUOSO_INSTALLATION_DIRECTORY>\samples\jdbc\<JDK_Version>\<Demo_name>
```

4.2. Sample ODBC Applications

4.2.1. Mac OS X

ODBCTEST:

This is a simple 'C' based and ODBC compliant Interactive SQL processor.

1. Open a Terminal session, and start ODBCTEST by executing the following command:

   ```
   /Library/iodbc/bin/odbctest
   ```
2. At the SQL command prompt enter "?" for a list of ODBC DSNs on your machine or enter a valid ODBC Connect String. If you have a DSN named "Marketing" you would enter:

   ```
   DSN=Marketing;UID=username;PWD=password
   ```

   Note: If there is no password, you must include a semicolon at the end:

   ```
   DSN=Marketing;UID=sa;PWD=;
   ```
3. Any valid SQL or ODBC command may be executed through this interface. The following example shows a connection to Microsoft SQL Server 2000, making a simple query against the sample Northwind database:

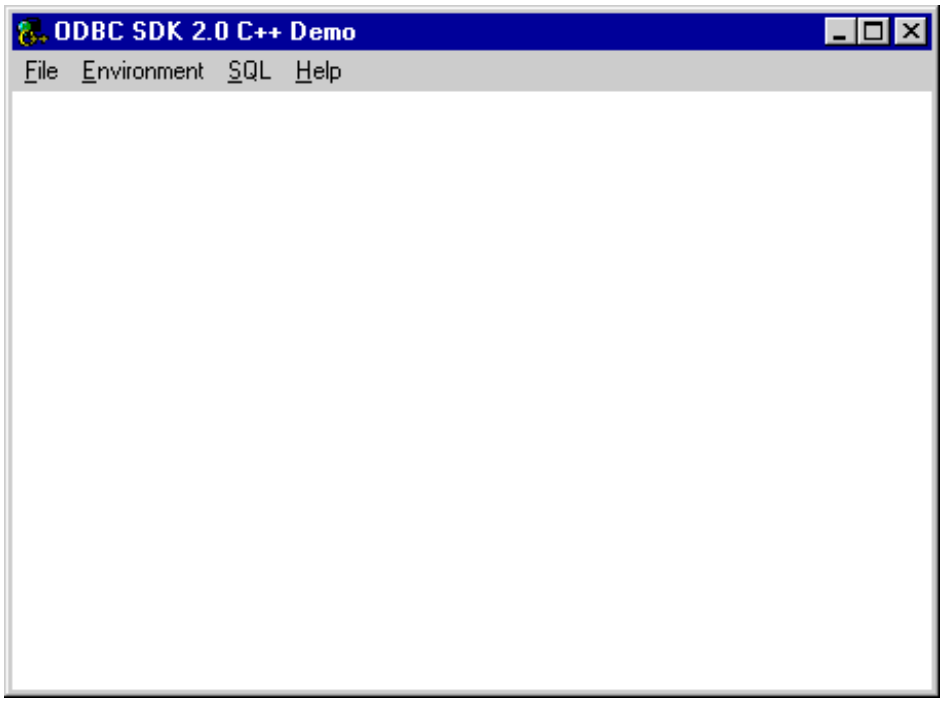
   ```
   [localhost:~] openlink% /Library/iodbc/bin/odbctest
   iODBC Demonstration program
   This program shows an interactive SQL processor

   Enter ODBC connect string (? shows list, or DSN=...): DSN=user_tthib_sql2k

   SQL>select au_lname, au_fname, state from authors where au_id < '333-33-3333'
   au_lname                                |au_fname            |state
   ----------------------------------------+--------------------+-----
   White                                   |Johnson             |CA
   Green                                   |Marjorie            |CA
   Carson                                  |Cheryl              |CA
   O'Leary                                 |Michael             |CA
   Straight                                |Dean                |CA
    5 row(s) fetched.

   SQL>quit
   Again (y/n) ? n

   Have a nice day.
   [localhost:~] openlink%
   ```

4.2.2. Windows 95/98/NT/2000

C++ Demo

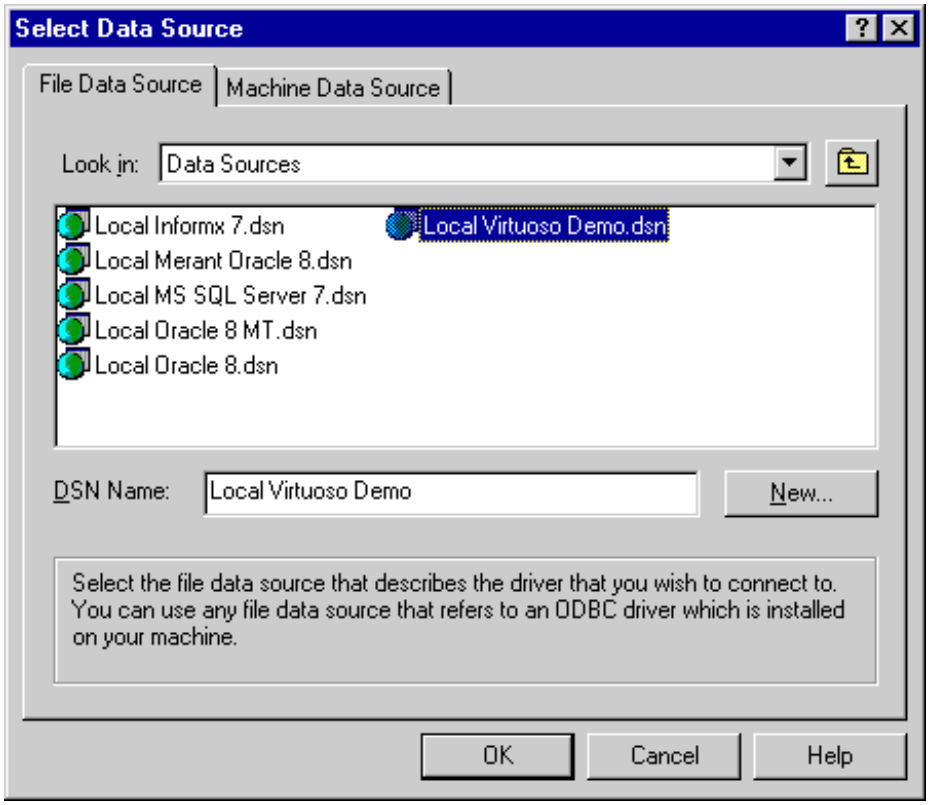
1. Go to the Virtuoso "Start Menu" item, then click on the "C++ Demo 32 Bit" menu item.

Figure 4.1. C++ Demo



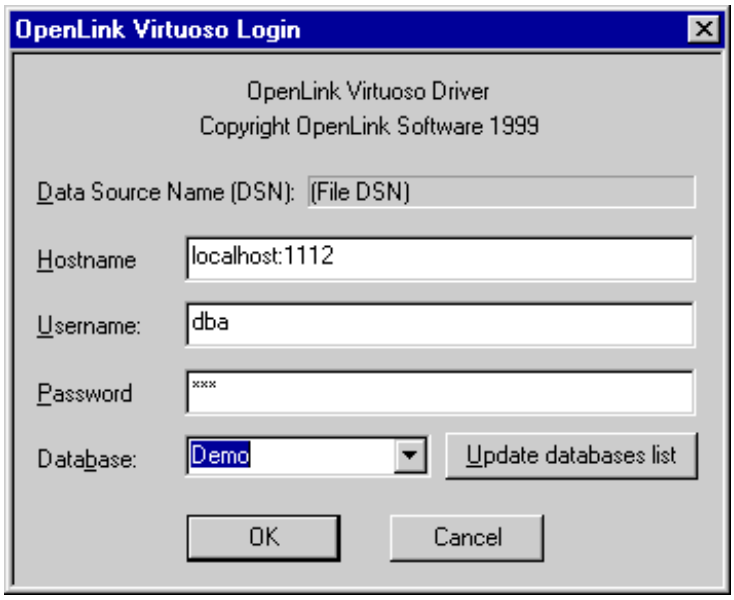
2. Follow the Environment->Open Connection menu path. Selecting the "Open Connection" menu item results in the ODBC Driver Manager presenting you with a list of ODBC DSNs on your machine as depicted by the screen capture below:

Figure 4.2. C++ Demo



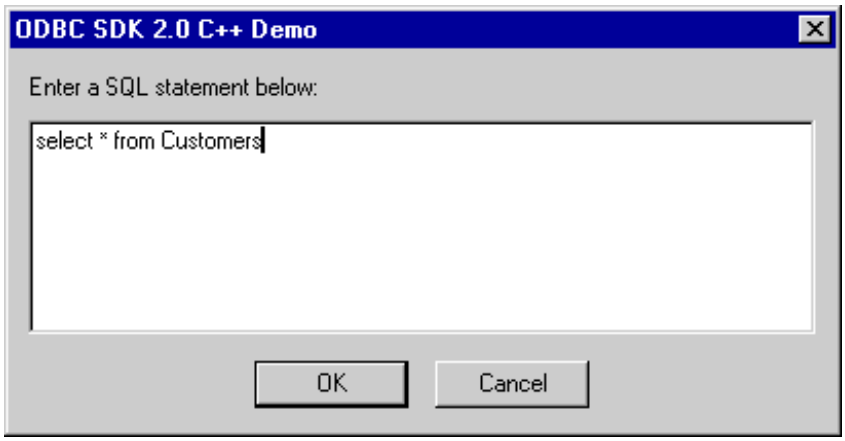
3. Select the ODBC DSN that you want to be connecting to, in this case "Local Virtuoso Demo" has been chosen since this will connect you to a sample Virtuoso database that has been populated with data.
4. You are then presented with a Login Dialog by the Virtuoso driver for ODBC, enter a valid user name and password (default being user: demo and password: demo) into the appropriate fields.

Figure 4.3. C++ Demo



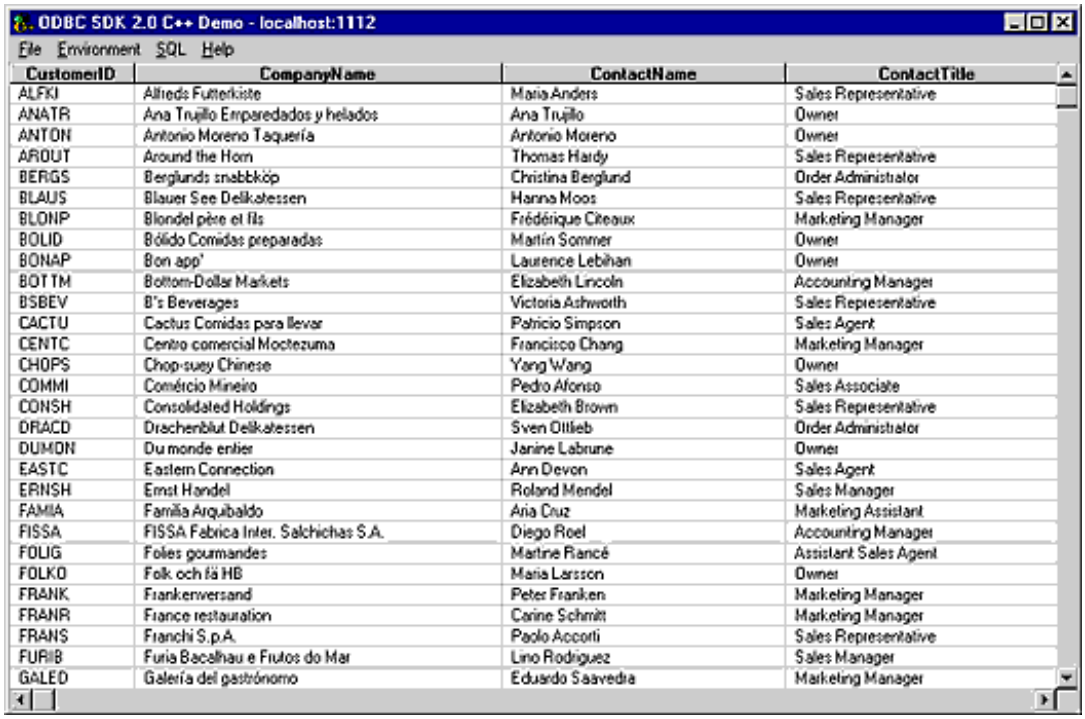
5. At this point you will be connected to the Virtuoso demonstration database, you can now use the SQL-->Execute SQL menu path to open up the Interactive SQL input dialog. Enter a valid SQL statement (see example in screen shot) and then click on the "OK" button.

Figure 4.4. C++ Demo



6. You will be presented with the results of your query.

Figure 4.5. C++ Demo



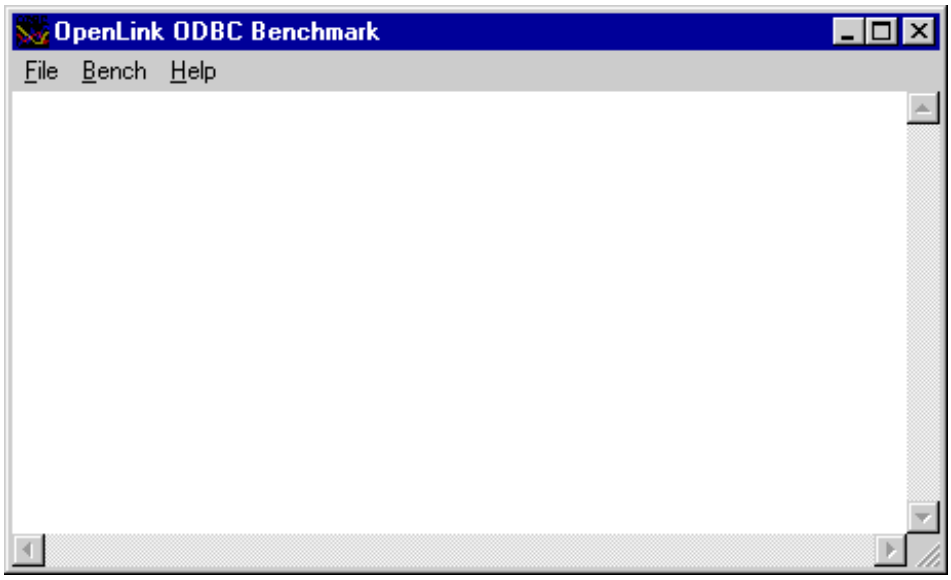
CustomerID	CompanyName	ContactName	ContactTitle
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner
AROUT	Around the Horn	Thomas Hardy	Sales Representative
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative
BLONP	Blondel père et fils	Frédérique Citeaux	Marketing Manager
BOLID	Bólido Comidas preparadas	Martín Sommer	Owner
BONAP	Bon app'	Laurence Lebihan	Owner
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager
BSBEV	B's Beverages	Victoria Ashworth	Sales Representative
CACTU	Cactus Comidas para llevar	Patricio Simpson	Sales Agent
CENTC	Centro comercial Moctezuma	Francisco Chang	Marketing Manager
CHOPS	Chop-suey Chinese	Yang Wang	Owner
COMMI	Comércio Mineiro	Pedro Afonso	Sales Associate
CONSH	Consolidated Holdings	Elizabeth Brown	Sales Representative
DRACD	Drachenblut Delikatessen	Sven Ottlieb	Order Administrator
DUMON	Du monde entier	Janine Labrune	Owner
EASTC	Eastern Connection	Ann Devon	Sales Agent
ERNSH	Ernst Handel	Roland Mendel	Sales Manager
FAMIA	Familia Arquibaldo	Aria Cruz	Marketing Assistant
FISSA	FISSA Fabrica Inter. Salchichas S.A.	Diego Roel	Accounting Manager
FOLIG	Folies gourmandes	Martine Rancé	Assistant Sales Agent
FOLKO	Folk och fä HB	Maria Larsson	Owner
FRANK	Frankenversand	Peter Franken	Marketing Manager
FRANR	France restauration	Carine Schmitt	Marketing Manager
FRANS	Franchi S.p.A.	Paolo Accorti	Sales Representative
FURIB	Furia Bacalhau e Frutos do Mar	Lino Rodriguez	Sales Manager
GALED	Galería del gastrónomo	Eduardo Saavedra	Marketing Manager

7. You exit this demo by following the Environment-->Close Connection menu path.

ODBC Bench Test 32

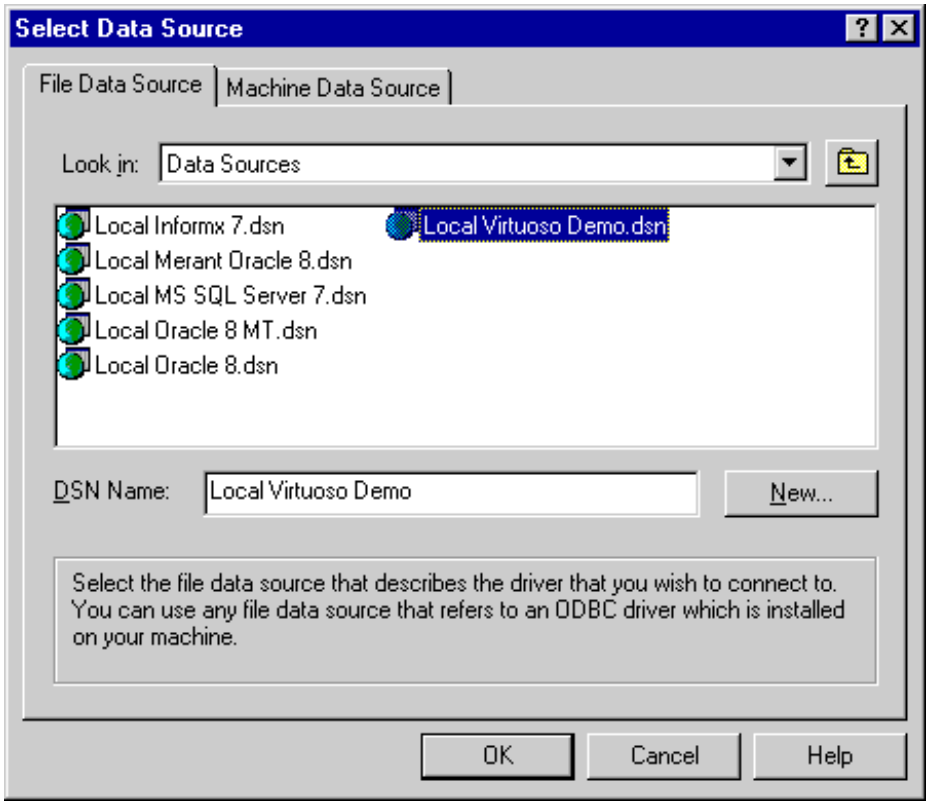
1. Go to the Virtuoso "Start Menu" item, then click on the "ODBC Bench Test 32 Bit" menu item. You will be presented with the "Bench Test" interface.

Figure 4.6. ODBC Bench



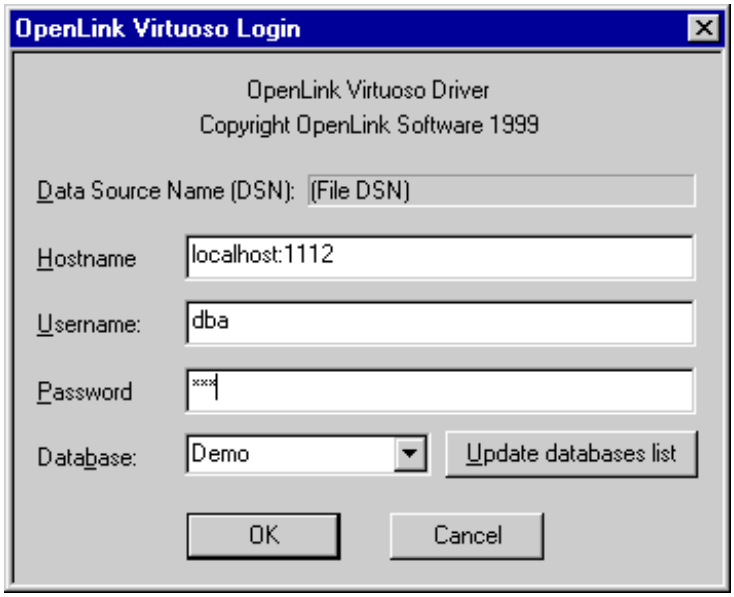
2. Follow the File-Connect menu path which initializes the ODBC Driver Manager which in turn presents you with a list of ODBC DSN's installed on your machine. Select the DSN that you want to benchmark, remember that by benchmarking a DSN you are benchmarking the ODBC Driver that serves the DSN in question and the backend database engine that serves the ODBC Driver. Choose the "Local Virtuoso Demo" DSN if you want to benchmark Virtuoso.

Figure 4.7. ODBC Bench



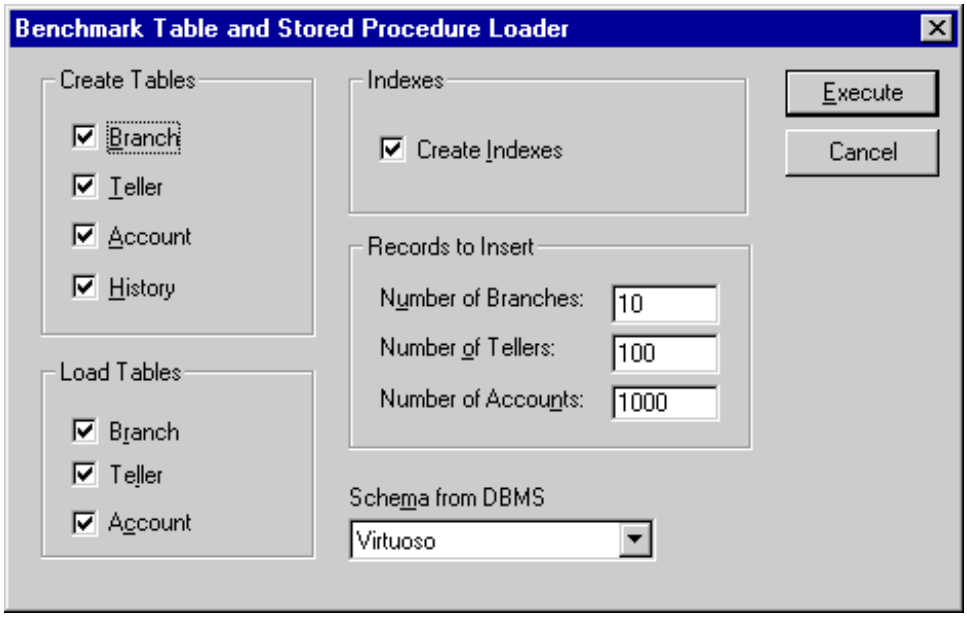
3. You will then be presented with a Login Dialog by the Virtuoso driver for ODBC, enter a valid user name and password (default being user: demo and password: demo for the Demo database) into the appropriate fields.

Figure 4.8. ODBC Bench



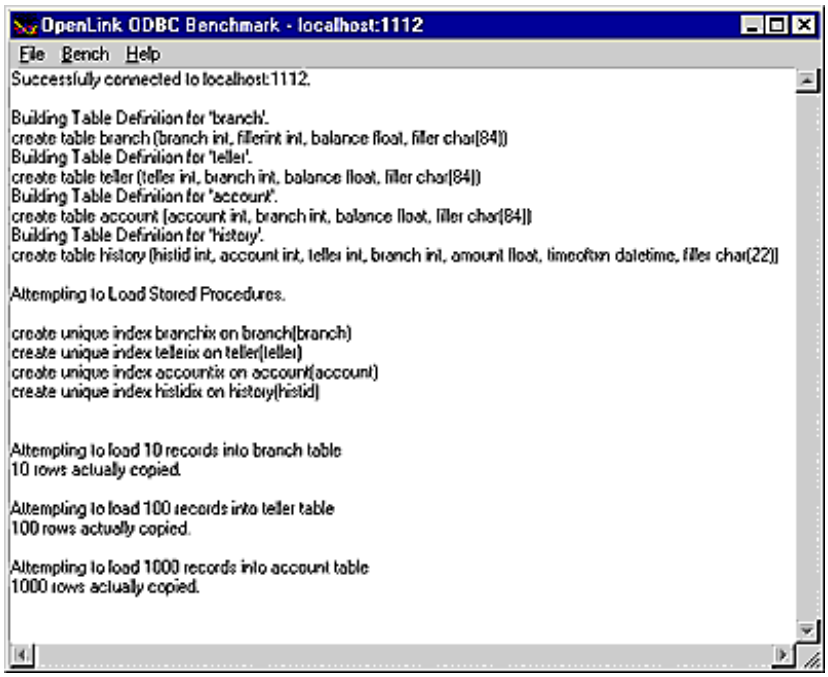
4. Now follow the Bench-->Load Tables menu path and you will be presented with a dialog that enables you to configure key elements of your benchmark. Click the "Execute" button to commence the process of setting up your database for the benchmark tests.

Figure 4.9. ODBC Bench



5. As the process of loading data occurs, all the way up to completion, the benchmark program will provide status information into the benchmark output pane as shown below:

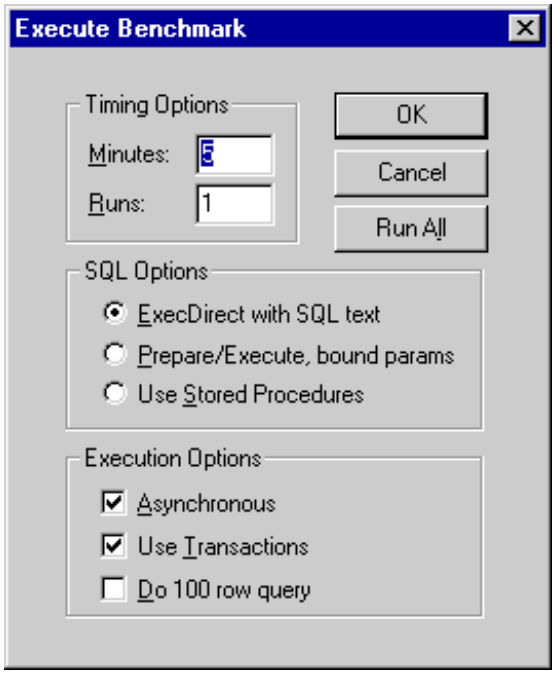
Figure 4.10. ODBC Bench



6. Now that all the benchmark data has been loaded into your database, follow the Bench-->Run Benchmark menu path and then configure your actual benchmark session parameters:

   These benchmark parameters fall into 3 categories, Timing Options, SQL Options, and Execution Options.

Figure 4.11. ODBC Bench



Timing Options: These setting allow you to configure the duration related aspects of this benchmark program

Minutes - this is the duration of each benchmark run

Runs - this controls how many iterations of the benchmarks you actually run (the default is one benchmark iteration with a duration of 5 minutes)

SQL Options: These setting allow you to configure how your benchmark's SQL instructions are actually handled.

ExecDirect with SQL Text - this means that no form of repetitive SQL execution optimization is being applied (SQL statements are prepared and executed repetitively)

Prepare/Execute Bound Params - this means that the Parameter Binding SQL execution optimization is being applied (SQL is prepared once but executed many times without the overhead of re-preparing statements prior to execution)

Use Stored Procedures - this means that the Stored Procedure SQL optimization is being applied (benchmark instructions are stored within database being benchmarked)

Execution Options: These setting allow you to configure the tone of your benchmark, for instance it could have Transaction scoping and a mix of record retrieval queries, or it could simply be input and update intensive with a minimal amount of record retrieval queries (the case when the 100 row query checkbox is unchecked a typical OLTP scenario)

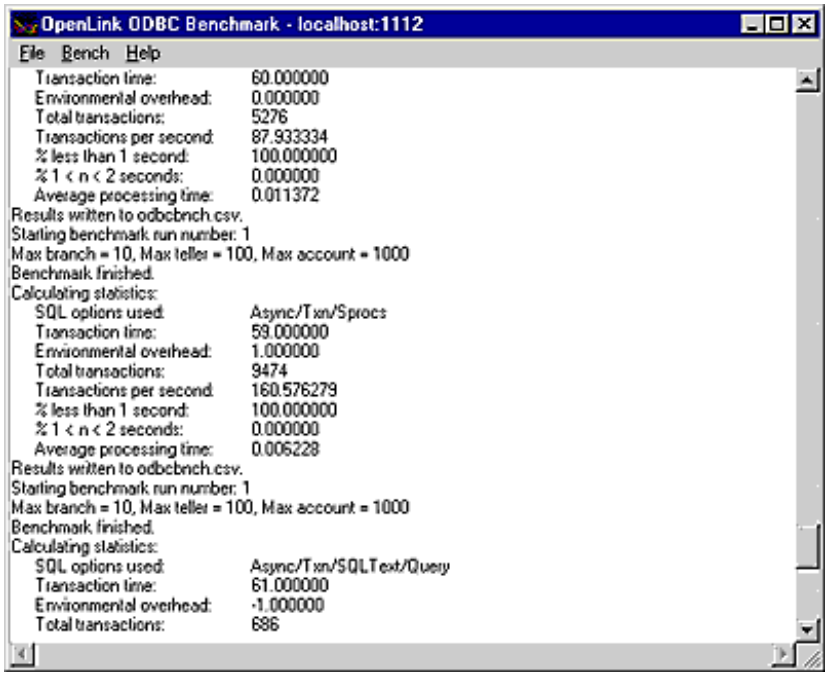
Asynchronous - execute the benchmark instructions asynchronously

Use Transactions - make the benchmark use transaction control (instructions are scoped to transaction blocks)

Do 100 row Query - perform a simulation of a 100 record retrieval as part of the benchmark activity.

1. Click on the "Run All" button if you would like all the different benchmark type combinations to be performed.
2. When benchmark run complete benchmark data is written to the benchmark program's output pane.

Figure 4.12. ODBC Bench



The key pieces of benchmark data that you need to look out for are:

Total Transactions: total number of transactions completed during the benchmark run

Transactions Per Second number of transaction completed per second for the benchmark run

Information from this benchmark is automatically written to an Excel format CSV (the file odbcbnch.csv) which makes it easy for you to graph and pivot data collated from several benchmark runs. A later version of this demo will actually write the benchmark data into an ODBC DSN that you provide thereby offering even more flexibility and accessibility to benchmark data.

4.2.3. Linux & UNIX

ODBCTEST:

This is a simple 'C' based and ODBC compliant Interactive SQL processor.

1. Run the script virtuoso-enterprise.sh to set up your environment:

   ```
   . virtuoso-enterprise.sh
   ```
2. Start ODBCTEST by executing the following command:

   ```
   odbctest
   ```
3. At the SQL command prompt enter "?" for a list of ODBC DSNs on your machine or enter a valid ODBC Connect String. If you have a DSN named "Marketing" you would enter:

   ```
   DSN=Marketing;UID=username;PWD=password
   ```

   Note: If there is no password, you must include a semicolon at the end:

   ```
   DSN=Marketing;UID=sa;PWD=;
   ```
4. Any valid SQL or ODBC command may be executed through this interface. The following example shows a connection to Microsoft SQL Server 2000, making a simple query against the sample Northwind database:

   ```
   [localhost:~] openlink% odbctest
   iODBC Demonstration program
   This program shows an interactive SQL processor

   Enter ODBC connect string (? shows list, or DSN=...): DSN=test_sql2k

   SQL>select au_lname, au_fname, state from authors where au_id < '333-33-3333'
   ```

```
au_lname                                |au_fname            |state
----------------------------------------+--------------------+-----
White                                   |Johnson             |CA
Green                                   |Marjorie            |CA
Carson                                  |Cheryl              |CA
O'Leary                                 |Michael             |CA
Straight                                |Dean                |CA
 5 row(s) fetched.

SQL>quit
Again (y/n) ? n

Have a nice day.
[localhost:~] openlink%
```

4.2.4. MS DTC ODBC Sample Application

The MS DTC demo is located in the

```
<VIRTUOSO_INSTALLATION_DIRECTORY>\samples\odbc\MSDTCdemo1
```

folder. This demo shows usage of distributed transactions driven by MS DTC through the ODBC API.

Setup

The sample works with two instances of Virtuoso server. The running MS DTC service is needed. The servers must be started with MS DTC support (see Virtuoso documentation).

First of all, edit the virt.odbc file. By default this file contains two connection strings of local Virtuoso servers running on port 1111 and port 1112, Each line begins with connection string to appropriate Virtuoso server. Initially this file contains the following lines:

```
1111 dba dba 00.sql
1112 dba dba 01.sql
```

where 1111, 1112 are ports of two Virtuoso servers. "dba dba" - user and password.

Initialization

Start

```
 mtstest.exe +load
```

in the

```
<VIRTUOSO_INSTALLATION_DIRECTORY>\samples\odbc\MSDTCdemo1
```

folder. This must check whether all needed servers are running, create and initialize tables, procedures, etc.

Test

Run the command in the demo's working directory:

```
mtstest.exe   +exec N
mtstest.exe   +exec 0
```

where N - is a number of test iterations. The second command checks logical consistency of the tables.

Description

The demo source is

```
start.c
```

file in the

```
<VIRTUOSO_INSTALLATION_DIRECTORY>\samples\odbc\MSDTCdemo1
```

directory. Several highlights of the most significant parts of code are presented below:

```
ITransactionDispenser *disp;
ITransaction *transaction;

...
HRESULT hr =
    DtcGetTransactionManagerC (0, 0, &IID_ITransactionDispenser, 0, 0, 0,
    &, disp);
```

The code above creates Dispenser object which represents the local instance of MS DTC. If MS DTC is not available

```
DtcGetTransactionManagerC
```

fails. The Dispenser is needed to create distributed transaction objects later.

Begin new distributed transaction:

```
disp->lpVtbl->BeginTransaction (disp, 0, ISOLATIONLEVEL_ISOLATED,
      0, 0, &transaction);
```

Enlist connection in the transaction:

```
SQLSetConnectOption (hdbc, SQL_COPT_SS_ENLIST_IN_DTC,
      (DWORD) transaction);
```

And, finally, commit the transaction:

```
transaction->lpVtbl->Commit (tran, 0, 0, 0);
```

4.2.5. MS DTC OLE DB Sample Application

The MS DTC OLE DB demo is located in the

```
<VIRTUOSO_INSTALLATION_DIRECTORY>\samples\odbc\MSDTCdemo2
```

folder. This demo shows usage of distributed transactions driven by MS DTC through OLE DB.

Setup

The sample works with two instances of Virtuoso server. Running MS DTC service is needed. The servers must be started with MS DTC support (see Virtuoso documentation).

The test needs two Virtuoso server instances running on ports 1111 and 1112

Run

Run the command in the demo's working directory:

```
voledbtest.exe
```

Description

The demo source is

```
voledbtest.cs
```

file in the

```
<VIRTUOSO_INSTALLATION_DIRECTORY>\samples\odbc\MSDTCdemo
```

directory. Several highlights of the most significant parts of code are presented below:

```
[TransactionAttribute(TransactionOption.Required)]
```

It is significant to set this attribute of class to enable automatic transaction initialization.

Create connections to appropriate servers:

```
string strConn = "Provider=VIRTOLEDB;Data Source=" + dsn1
    + ";User Id=dba;Password=dba;Initial Catalog=Demo;Prompt=NoPrompt;";
string strConn2 = "Provider=VIRTOLEDB;Data Source=" + dsn2 +
    ";User Id=dba;Password=dba;Initial Catalog=Demo;Prompt=NoPrompt;";
obj_conn = new OleDbConnection(strConn);
obj_conn2 = new OleDbConnection(strConn2);
```

And, finally, execute the SQL code in the context of distributed transaction:

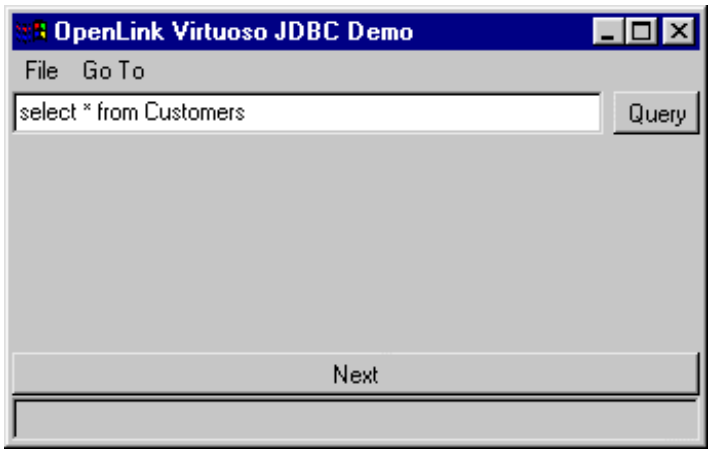
```
OleDbCommand sqlc = new OleDbCommand ("ODBC_BENCHMARK(" + idx + ",1,1,12.00,\'noone\')");
sqlc.Connection = obj_conn;
sqlc.ExecuteNonQuery();
sqlc = new OleDbCommand ("ODBC_BENCHMARK(" + idx + ",1,1,-12.00,\'noone\')");
sqlc.Connection = obj_conn2;
sqlc.ExecuteNonQuery();
```

4.3. Sample JDBC Applications & Applets

4.3.1. JDBCDemo Java Application

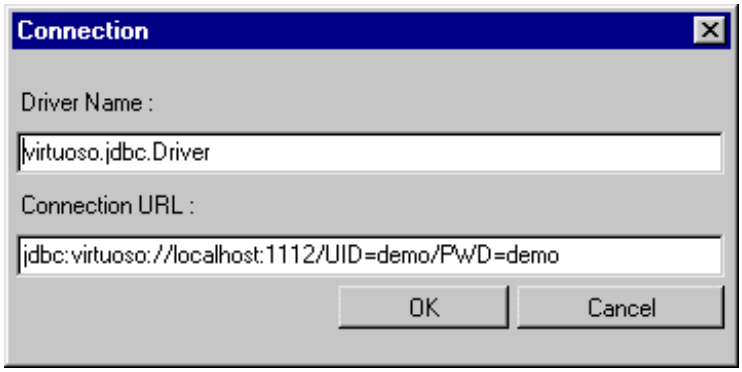
1. Go to the Virtuoso "Start Menu" program group and then follow the JDBC Samples-->JDBCDemo (JDK1.1&1.2) menu path, this will execute a DOS batch program that initializes the Java demo application.

   Figure 4.13. JDBC Demo

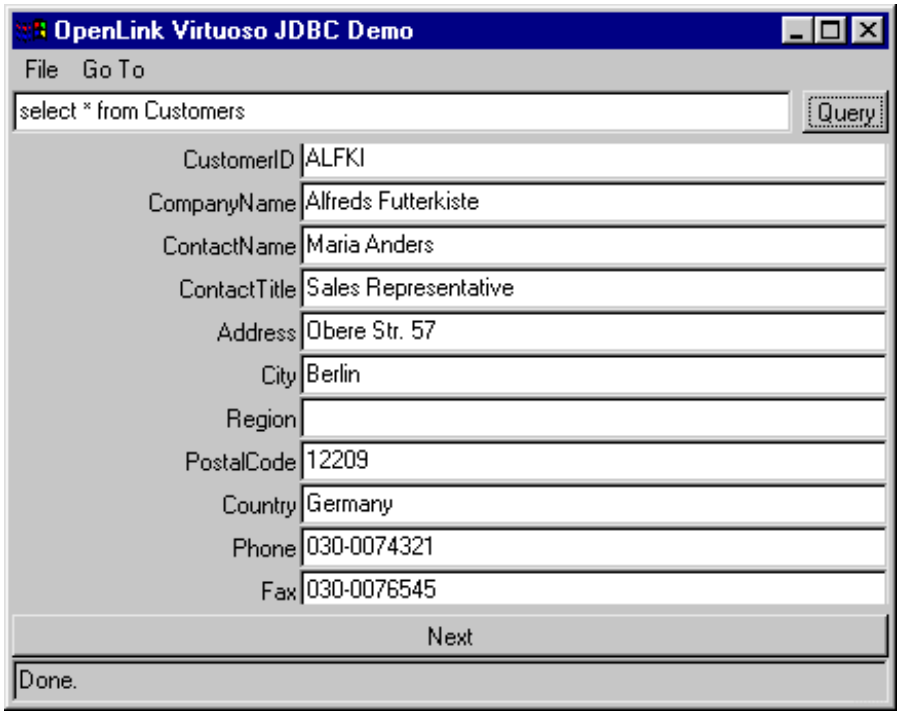
2. Set the JDBC Driver Name and URL settings for your connection to the Virtuoso demonstration database. The "Driver Name" field identifies the Virtuoso Driver. If it is left blank, then it defaults to "virtuoso.jdbc.Driver", which is the Virtuoso Driver for JDBC 1.0. The "Connection URL" field requires a valid Virtuoso JDBC URL.

   Figure 4.14. JDBC Demo



3. Enter a valid SQL statement and then click the "Query" button, the example below uses a fully qualified Virtuoso SQL statement requesting all records from the "Demo" database table "Customers" owned by the Virtuoso user "DBA".

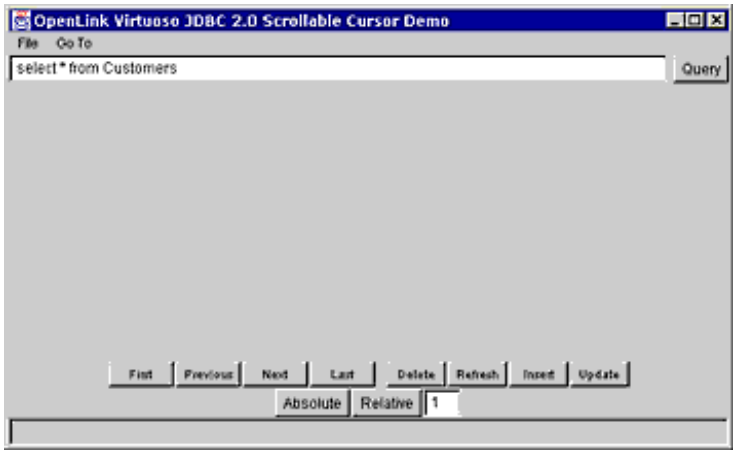
Figure 4.15. JDBC Demo



4.3.2. ScrollDemo2 Java Application

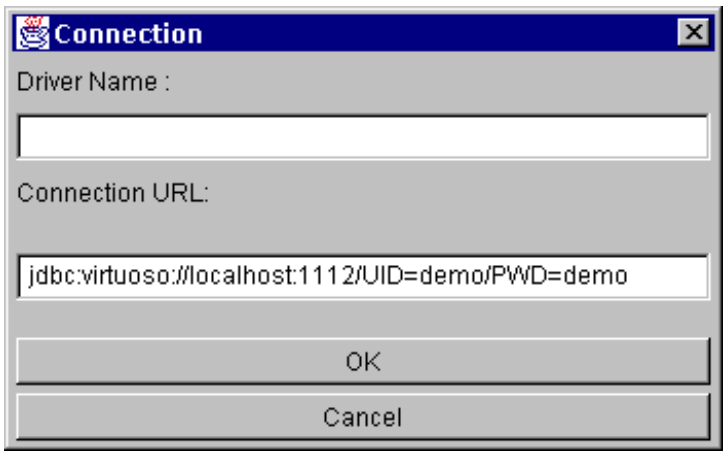
1. Go to the Virtuoso Lite "Start Menu" program group and then follow the JDBC Samples-->ScrollDemo2 (JDK1.2) menu path, this will execute a DOS batch program that initializes the Java demo application.

Figure 4.16. Scroll Demo 2



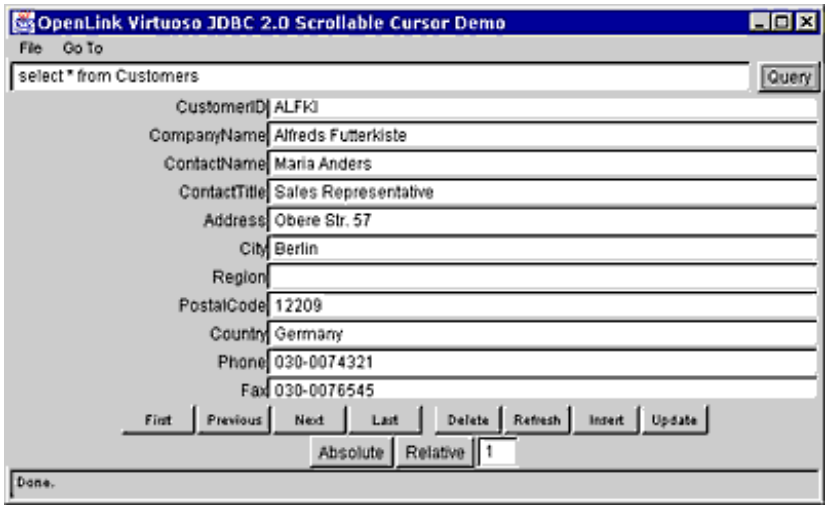
2. Set the JDBC Driver Name and URL settings for your connection to the Virtuoso demonstration database. The "Driver Name" field identifies the Virtuoso Driver. If it is left blank, then it defaults to "virtuoso.jdbc2.Driver", which is the Virtuoso Driver for JDBC 2.0. The "Connection URL" field requires a valid Virtuoso JDBC URL.

Figure 4.17. Scroll Demo 2



3. Enter a valid SQL statement and then click the "Query" button, the example below uses a fully qualified Virtuoso SQL statement requesting all records from the "Demo" database table "Customers" owned by the Virtuoso user "DBA".

Figure 4.18. Scroll Demo 2

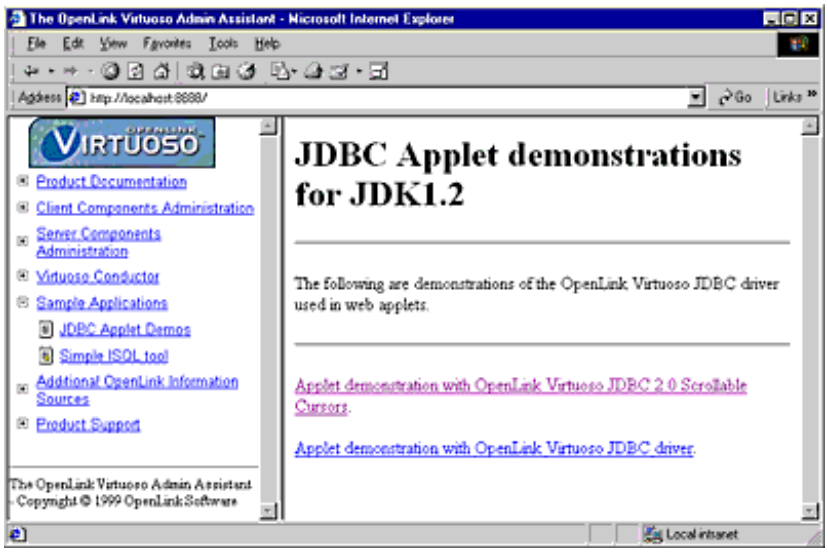


4. You can now use the navigational buttons to Scroll backwards and forwards, each of these navigational buttons highlights Virtuoso's full implementation of the JDBC 2.0 Scrollable Cursors specifications.

4.3.3. ScrollDemo2 Java Applet

1. Start the Virtuoso Admin Assistant and then follow the Sample Applications-->JDBC Applet Demos menu path.

   Figure 4.19. Scroll Demo 2

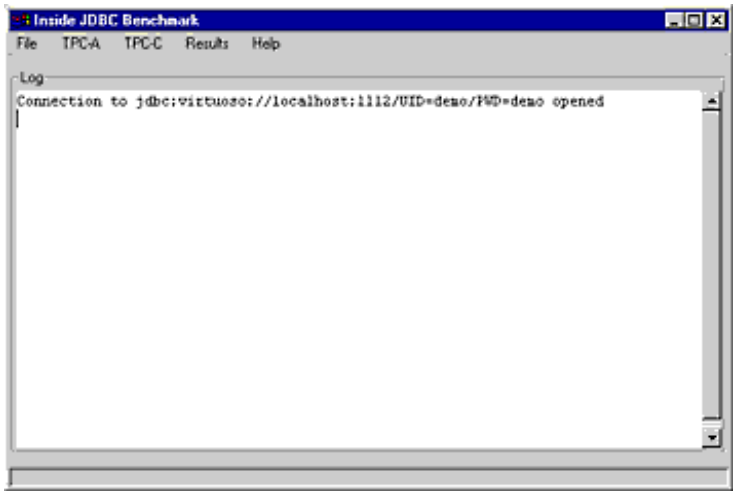


2. Click on the "ScrollDemo2" hyperlink which initializes the ScrollDemo2 applet, if you do not have a Java 1.2 or Java 2.0 compliant browser you will not be able to run this Applet demo. The other way to experience this demo is to run the Application version which uses your operating systems Java Virtual Machine (JVM) instead of a JVM inherently linked to a Web Browser.

4.3.4. JBench Application

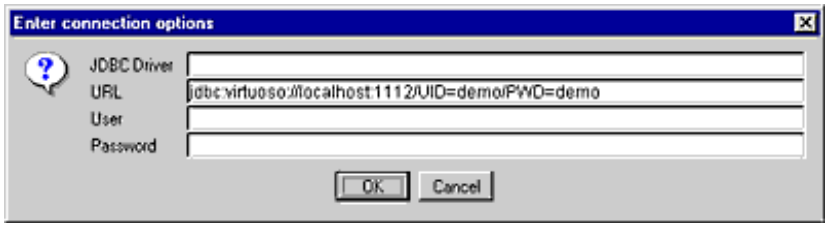
1. Go to the Virtuoso "Start Menu" program group and then follow the JDBC Samples-->Jbench (JDK1.1) or Jbench (JDK1.2) menu path, depending on the JVM you have installed. This will execute a DOS batch program that initializes the JBench application.

   Figure 4.20. JBench



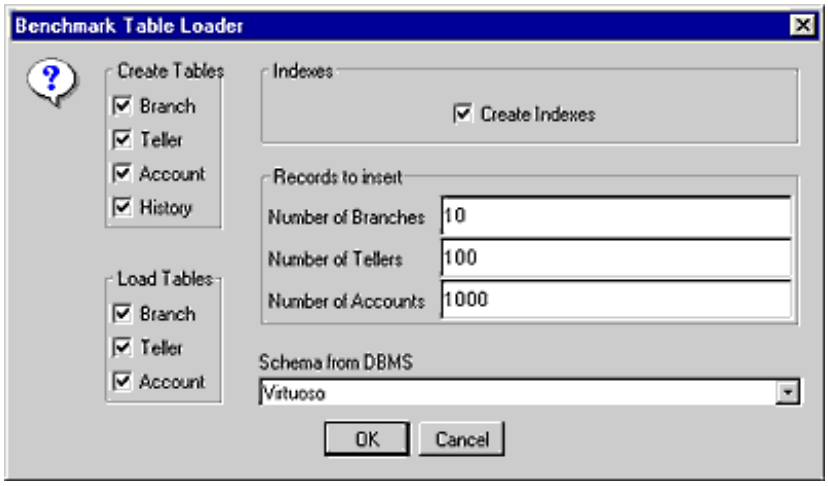
2. The follow the File-->Connect menu path to make your initial connection. You will need to identify your JDBC Driver (by providing appropriate Driver Name values in the JDBC Driver field) and then provide a valid JDBC URL for your specific JDBC Driver (Jbench uses a valid Virtuoso Driver for JDBC URL that points to the demonstration database listening at port 1112 as its default).

   Figure 4.21. JBench



3. Follow the Results-->Table URL menu path, this is how you identify (using a JDBC URL) the location of the "Results" tables into which you will be storing your benchmark data. The default URL is the current JDBC URL (the one used to establish your initial connection), but this can be a totally different database to the one being benchmarked.
4. Follow the Results-->Create menu path, this is how you actually perform the "Results" table creation in the database identified by the URL in the previous step.
5. Follow the TPC-A-->Load Tables menu path to prepare your database for the TPC-A benchmark, select a database schema type that matches the database engine that you are benchmarking. If your database is not listed ANSI should suffice (as long as this is an ANSI SQL compliant database).

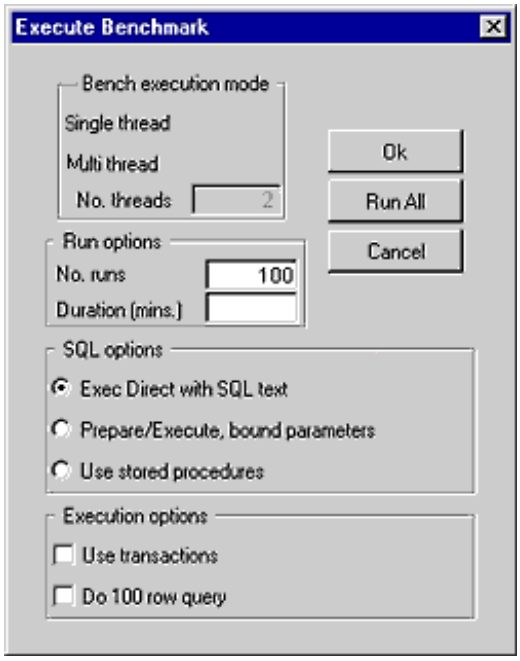
Figure 4.22. JBench



6. Follow the TPC-A-->Load Procedures menu path to Load the Virtuoso TPC-A stored procedures.
7. Now that all the benchmark data and stored procedures have been loaded into your database, follow the TPC-A-->Run Benchmark menu path and then configure your actual benchmark session parameters:

   The benchmark parameters fall into 4 categories, Bench execution mode, Run Options, SQL Options, and Execution Options.

Figure 4.23. JBench



Bench execution mode: These setting allow you to configure the threads used for the benchmark.

Decide on a single or multiple threads test.

No. Threads - this is the number of concurrent threads to be used during the benchmark.

Run Options: These setting allow you to configure the duration related aspects of this benchmark program.

No. runs - this controls how many iterations of the benchmarks you actually run (the default is 100 benchmark iterations).

Duration (mins.) - this is the duration in minutes of each benchmark run.

SQL Options: These setting allow you to configure how your benchmark's SQL instructions are actually handled.

ExecDirect with SQL Text - this means that no form of repetitive SQL execution optimization is being applied (SQL statements are prepared and executed repetitively)

Prepare/Execute Bound Params - this means that the Parameter Binding SQL execution optimization is being applied (SQL is prepared once but executed many times without the overhead of re-preparing statements prior to execution)

Use Stored Procedures - this means that the Stored Procedure SQL optimization is being applied (benchmark instructions are stored within database being benchmarked)

Run All - this implies you want to perform all of the above benchmarks

Execution Options: These setting allow you to configure the tone of your benchmark, for instance it could have Transaction scoping and a mix of record retrieval queries, or it could simply be input and update intensive with a minimal amount of record retrieval queries (the case when the 100 row query checkbox is unchecked a typical OLTP scenario)

Use Transactions - make the benchmark use transaction control (instructions are scoped to transaction blocks)

Do 100 row Query - perform a simulation of a 100 record retrieval as part of the benchmark activity.

8. Run your TPC-A benchmark.
9. Follow the TPC-A-->Cleanup menu path to clean up your database so that you can then run other benchmarks (TPC-C like benchmark).
10. To run the TPC-C benchmark simply follow the appropriate menu path, create the benchmark tables & stored procedures, load the benchmark data and then run the TPC-C benchmark.

4.3.5. JTA Demo Application

The JTA demo is located in the

```
<VIRTUOSO_INSTALLATION_DIRECTORY>\samples\jdbc\JTADemo
```

folder.

Software Setup

1. The sample requires 2 or 3 Virtuoso instances and an instance of a J2EE 1.3.1 server properly set up and running on the same computer or on the network. To build the sample it's also necessary to have a J2SE 1.4 SDK installed.
2. The JAVA_HOME and J2EE_HOME environment variables must be set to the J2SE SDK and J2EE installation path respectively.
3. The virtjdbc3.jar file must be in the classpath of the J2EE server. The setting of the classpath differs across different J2EE servers. For J2EE Reference Implementation (RI) the J2EE_CLASSPATH variable might be set in %J2EE_HOME%\bin\userconfig.bat on Windows or $(J2EE_HOME)/bin/userconfig.sh on Unix.

   ```
   set J2EE_CLASSPATH=C:/Virtuoso/lib/virtjdbc3.jar
   ```

   or

   ```
   J2EE_CLASSPATH=/home/login/virtuoso/lib/virtjdbc3.jar
   export J2EE_CLASSPATH
   ```
4. Add the Virtuoso XA datasources to J2EE server. For J2EE RI this is done like this:

   ```
   j2eeadmin -addJdbcXADatasource jdbc/Virtuoso1 virtuoso.jdbc3.VirtuosoXADataSource dba dba -pro
   j2eeadmin -addJdbcXADatasource jdbc/Virtuoso2 virtuoso.jdbc3.VirtuosoXADataSource dba dba -pro
   j2eeadmin -addJdbcXADatasource jdbc/Virtuoso3 virtuoso.jdbc3.VirtuosoXADataSource dba dba -pro
   ```

   Please note that the newly added datasources take effect only after restart of the J2EE server.

Configuration

The sample folder contains the jtademo.properties file which can be used to configure the sample application. The configuration options are described within the file.

Building and Deployment

The sample folder contains build and deploy scripts (build.sh and deploy.sh for Linux and Unix and build.bat and deploy.bat for Windows).

Use the build script to build the sample application from the source files.

Deploy the application to the J2EE server. The deploy script performs this task for J2EE RI. Other J2EE application servers might provide different ways for application deployment.

Running the Sample

The sample folder contains the run scripts for Linux/Unix and Windows.

The run script accepts the following commands:

create
: Creates and populates the necessary tables.

check
: Verifies data consistency.

drop
: Removes data and tables from the databases.

run
: Executes a number of transactions over database tables.

The first action to do is to initialize the data with the create command. After this the command might be used one or more times. This executes a series of concurrent updates on the distributed data. The check command verifies that this updates left databases in the consistent state.

For instance, for Unix:

```
run.sh create
run.sh run
run.sh check
```

And for Windows:

```
run.bat create
run.bat run
run.bat check
```

Chapter 5. Conceptual Overview

Abstract

Virtuoso provides both a native database capability and a virtual database that integrates remote or third-party ODBC data sources seamlessly with Virtuoso's own. The virtual database (VDB) allows transparent unified queries across all linked data sources. The Virtuoso server supports ANSI SQL92, object extensions, and a significant set of PL extensions. The most important extensions include support for modern Internet standards for communication, data, and document exchange. Virtuoso supports communications protocols such as HTTP, SMTP and NNTP as well as a full suite of XML-based protocols including XML, XSL, SOAP, and WSDL.

5.1. Core Database Engine

5.1.1. Logical Data Model

Virtuoso provides an extended Object Relational model which offers all the flexibility of relational access with inheritance, run time data typing, late binding, identity based access.

Table

A table is a uniquely named entity that has the following characteristics:

- Zero or more columns
- One primary key
- Zero or more keys (indices)
- An optional super table from which this inherits properties
- An optional object ID key, which may or may not be the primary key
- Various SQL table constraints, e.g. CHECK's

A table will then have zero or more rows. The relationship of a table and its rows can be thought of as a class-instance relationship.

Column

A column is always defined in one table and has a name that is unique within that table. A column may appear in more than one table as a result of inheritance but always has one place of definition. i.e. one database wide 'identity'.

A column has the following characteristics:

- Table
- Name inside the table
- database wide ID
- Data type
- Various SQL constraints, e.g. DEFAULT, CHECK etc.

Key

A key or index is the means by which tables manifest themselves in the physical database. A key is always defined with respect to one table but may occur in several as a result of inheritance. Keys have unique names inside the table. A key has the following characteristics:

- A database wide key ID
- One or more 'significant' key parts, which are columns of the defining table or super tables
- Zero or more 'trailing' key parts, columns of the defining table or supertables.
- Whether the key is primary
- Whether the key is unique
- How the key is clustered
- Whether the key is an object ID key

Subtable

A subtable is a table that inherits all columns, the primary key and all other keys from another table, called the super table.

A subtable can define its own columns and keys which add themselves to those of the super table. No primary key can be redefined, though.

The inheritance relationship between tables is manifested by a key-subkey relationship between the tables' primary and other keys.

A table has at most one supertable.

Object ID

A table does not necessarily declare a primary key. Even so, the table must have a primary key - in this case a synthetic record ID which is defined as primary key. The record ID is an autoincrementing column that is normally invisible but, if present, can be accessed by explicit reference. One should not rely on this feature being available, though.

Thus

```
create table nokey (a integer);
```

expands to

```
create table nokey (a integer, _IDN integer identity, primary key (_IDN));
```

The first unique index to be defined will become the primary key if the table is empty at the time of definition.

Thus

```
create unique index a on nokey (a);
```

will change the nokey table to be as if defined by

```
create table nokey (a integer, primary key (a));
```

Having a primary key other than _IDN is always better than the default primary key. Declaring a primary key is therefore always advisable.

5.1.2. Data Types

Virtuoso supports most SQL 92 data types.

CHARACTER & VARCHAR

- CHARACTER
- VARCHAR
- VARCHAR '(' INTNUM ')'

- VARCHAR
- NVARCHAR '(' INTNUM ')'
- CHARACTER '(' INTNUM ')'

The CHAR, CHARACTER and VARCHAR datatypes are implemented as a single string type with dynamic length. The precision that may be specified controls how the column is described by `SQLColumns()` , `SQLDescribeCol()` and so on. If a precision is not specified for a VARCHAR then the default precision will be 0, which means do not check. If a precision is not specified for a CHARACTER then Virtuoso sets the precision to 1. An explicit precision of 0 can be specified to turn off length checking for values stored in the column. If a value other than string or NULL is assigned to the column it is cast to a varchar (using CAST internally) and then stored into the column. If the value is not castable to a varchar then Virtuoso returns an error. Additionally if the column precision is greater than 0 and the value string length is greater than the column precision Virtuoso will also return an error.

The length is stored separately. Space required is 2+length for

A varchar column may contain binary 0 bytes.

A string literal is delimited by single quotes.

ANY

- ANY

The ANY datatype is implemented as a single binary string type with dynamic length. It is reported as a VARCHAR in `SQLColumns()` , `SQLDescribeCol()` and so on. The precision returned by these columns is 24 but has no effect. This type can contain arbitrary binary data, including zeros.

The length is stored separately. The space required is 2+length

NUMERIC & DECIMAL

- NUMERIC
- NUMERIC '(' INTNUM ')'
- NUMERIC '(' INTNUM ',' INTNUM ')'
- DECIMAL
- DECIMAL '(' INTNUM ')'
- DECIMAL '(' INTNUM ',' INTNUM ')'

The various forms of NUMERIC and DECIMAL refer to one variable-precision floating point decimal data type that provides accurate arithmetic and decimal rounding. The default maximum precision and scale are 40 and 20. The precision is the number of decimal digits used in the computation. The scale is the maximum number of decimal digits to the right of the decimal point. Internal calculations are always precise but numbers are truncated to the column's precision and scale when stored. If a value being stored has more digits to the left of the decimal point than allowed in the column, Virtuoso signals an error. If a number being stored has more digits to the right of the decimal point than allowed in a column the decimal part is rounded to the precision of the column.

The space consumption of a number is

```
3 + precision / 2
```

bytes. The precision and scale of a column of this type are returned by functions such as `SQLColumns()` and `SQLDescribeCol()` .

A DECIMAL or NUMERIC with precision <= 9 and scale = 0 is transformed to INTEGER.

Literal numbers outside of the 32 bit signed integer range are of type decimal. Any numeric literals with a decimal point are of type decimal. Literals with an exponent are of type double precision.

INTEGER & SMALLINT

- INT
- INTEGER
- SMALLINT

These types are represented as a 32-bit signed binary integer, described as having a precision of 9 and a scale of 0, although the range is +/- 2**31. Storage space is 2 bytes for SMALLINT and 4 bytes otherwise.

A column declared SMALLINT is described as SQL_SMALLINT. A column declared INTEGER or INT is described as SQL_INTEGER.

Literals composed of of an optional sign and digits are of the integer type if they fit in the 32 bit range.

FLOAT & DOUBLE

- FLOAT
- FLOAT '(' INTNUM ')'
- DOUBLE PRECISION

These types refer to the 64-bit IEEE floating-point number, the C *double* type. This is a fixed-precision binary floating point number is described as having a precision of 15 and a scale of 0. This type is preferable to NUMERIC if decimal rounding is not required since it is precise enough for most uses and more efficient than NUMERIC. The storage requirement is 8 bytes.

Any number literal with an exponent has the double type, e.g. 2e9.

REAL

- REAL

This type is the 32-bit IEEE floating point number corresponding to the C *float* type. The storage requirement is 5 bytes.

LONG VARCHAR & LONG VARBINARY

- LONG VARCHAR
- LONG VARBINARY

These types implement a binary large object (BLOB) type. The length can be up to 2**31 bytes (2GB). If manipulated with the `SQLGetData()` and `SQLPutData()` ODBC functions a BLOB need not fit in the DBMS's or the client's memory. The LONG VARCHAR and LONG VARBINARY types are distinct only because certain ODBC applications gain from being able to distinguish long text from long binary. The types are described as SQL_LONGVARCHAR and SQL_LONGVARBINARY respectively, with a precision of 2GB.

Several long columns may exist on a single row. A long column may not be a key part in an index or primary key.

Data in long columns is stored as a linked list of database pages. Thus, a long column that does not fit in-line on the containing row will require an integer number of 8K database pages. If a long column's value is short enough to fit within the row containing it, the BLOB will be stored on the row and will not take more space than a VARCHAR of the same length. A long column fits on a row if the sum of the lengths of columns, including the long column, is under 4070 bytes.

ORDER BY, GROUP BY and DISTINCT may not reference long data types. Comparison of long data is not allowed unless first converted to the corresponding short type (varchar, nvarchar or varbinary). This conversion is only possible if the value is under 10MB in size. String functions accept long varchars and long nvarchars and convert them to varchar and nvarchar automatically. There is no long literal type per se, the corresponding character or binary type is assignable to a long type.

VARBINARY

- VARBINARY

This type is internally like VARCHAR but is distinct for compatibility with ODBC applications. A VARBINARY column is described as SQL_BINARY to ODBC clients. The storage requirement is the same as for a corresponding VARCHAR column.

VARBINARY and VARCHAR data are never equal even if the content is the same, but they can be cast to each other. VARBINARY data sorts in the unsigned order of the bytes comprising the data.

A varbinary literal is introduced by 0x followed by a hexadecimal representation of the bytes, 2 characters per byte, e.g. 0x0123456789abcdef.

TIMESTAMP; DATE & TIME

- TIMESTAMP
- DATETIME
- TIME
- DATE

All the time- and date-related types are internally represented as a single 'datetime' type consisting of a Julian day, hour, minute, second, 6-digit fraction and timezone. The range of the year is from 0 to over 9999. This type can accommodate all values of any SQL92 time-related type.

Although the internal representation is the same, a column of a time-related type is described as being of the appropriate ODBC type, i.e. SQL_TIMESTAMP for TIMESTAMP and DATETIME and SQL_DATE for DATE and SQL_TIME for TIME.

A DATETIME is described as precision 19, a DATE as precision 10 and a TIME as precision 8.

A column declared a TIMESTAMP is automatically set to the timestamp of the transaction that inserts or updates any column of the table containing it. The timestamp of a transaction is guaranteed to be distinct from that of any other transaction. For compatibility reasons a TIMESTAMP column is described to ODBC clients as a binary of 10 bytes. It is possible to use any date-related functions on TIMESTAMPs and to bind a TIMESTAMP column to a DATE or DATETIME variable (SQL_C_TIMESTAMP type in ODBC). Binding to a binary will also work but the data will then be opaque.

SQL92 provides for types with a timezone. Although the ODBC API does not expose the timezone, it is stored with these types and can be retrieved with the `timezone()` function. The timezone has a precision of minutes from UTC.

The storage requirement for these types is 10 bytes.

There is no date literal per se, but the ODBC shorthand for datetime literals can be used. The datetime/timestamp literal is of the form {dt 'YYYY-MM-DD HH:MM.SS'}. The date literal is of the form {d 'YYYY-MM-DD'}. Dates and datetimes may be compared between themselves but not with other types without explicit casting.

Timezoneless Datetimes

Some traditional relational databases keep all values of DATETIME type as combination of time and timezone data. Other keep time in some specific timezone without paying any attention to timezone at all. In RDF, the incoming triples may contain literals of types like xsd:dateTime with arbitrary values matching ISO 8601, and this standard permits the use of time values with optional timezone. Thus there should be a way of handling both "timezoned" and "timezoneless" datetimes inside one database. Virtuoso server supports this starting from version 07.20.3214.

Important note: The use of timezoneless datetimes may result in subtle errors in data processing. Applications that worked fine with timezoned datetimes may work incorrectly if timezoneless datetimes are used. The related application errors may stay unnoticed during local testing and reveal after worldwide use. To stay on safe side, the use of timezoneless datetimes with pre-07.20.3214 databases remains blocked even after the server executable is upgraded, so old applications will continue to work as before. When developing new applications, please pay attention to the check-list at the end of this section.

Enabling Timezoneless Support

Different applications may require different behavior when input data contain timezoneless values. In some cases it is better to "cast" all of them to timezoned than to upgrade existing code. Virtuoso offers 5 different modes of support. The mode is selected by `TimezonelessDatetimes` parameter in `[Parameters]` section of virtuoso.ini . This should be set before creating the database and the set value is stored in the database. After database is created, an attempt to change the mode by patching virtuoso.ini will have no effect and virtuoso.log will contain a warning about mismatch between virtuoso.ini and the database file.

The possible variants are:

- Never use timezoneless, as it was in old databases. Always set local timezone on parsing strings if no timezone specified. An attempt to set timezoneless by calling function `forget_timezone()` will signal error. Timezoneless values still may come from outside as dezerializations of timezoneless DATETIME values, serialized by other database instances, but not in any other way:

  ```
  TimezonelessDatetimes=0
  ```
- When parsing strings, set timezoneless if ISO format tells so:

  ```
  TimezonelessDatetimes=1
  ```
- Set timezoneless always, exception is when the parsed string contains explicit timezone or when RFC requires the use of GMT or when timezone is set by function `adjust_timezone()` . This is default for new databases if `TimezonelessDatetimes` parameter is missing in virtuoso.ini

  ```
  TimezonelessDatetimes=2
  ```
- Never use timezoneless. Always set local timezone on parsing strings if not timezone specified. An attempt to set timezoneless by calling function `forget_timezone()` will signal error. Timezoneless values still may come from outside as deserializations of timezoneless DATETIME values, serialized by other database instances, but not in any other way. The difference with `TimezonelessDatetimes=0` is that timezones are always printed on cast datetimes to strings etc. so timezoneless-aware clients will get unambiguous data.

  ```
  TimezonelessDatetimes=3
  ```
- On parsing string, set timezone to GMT if no timezone specified. However, timezoneless can be set by calling function `forget_timezone()` . This mode can be convenient for global web services when real "local" timezones of specific users are not known.

  ```
  TimezonelessDatetimes=4
  ```

For new applications, consider the use of `TimezonelessDatetimes=2` as primary variant, `TimezonelessDatetimes=1` as the second best.

Formats of Datetime Strings

Traditional SQL strings are of format "`YYYY-MM-DD hh:mm:ss` " with optional decimal fraction at the end and optional timezone data. Depending on software, the timezone can be specified as "timezone offset", i.e. the difference with GMT in minutes or as "timezone label", i.e. an identifier of timezone in special system dictionary that contains not only an offset in minutes but also information about daylight saving changes of the offset. Virtuoso does not support timezone labels, only numerical timezone offsets. Depending on system, the notation without the timezone data at the end means timezoneless value or, more probably, the value in some "default" timezone, such as local timezone of the server or GMT.

ISO 8601 introduced format "`YYYY-MM-DDThh:mm:ss` ", with "`T` " character between "`date` " and "`time` " parts. It also prescribed an unambiguous difference between timezoneless and timezoned values: absent timezone means timezoneless value.

The timezone offset is written as "`+hh:mm` " or "`-hh:mm` ", the "`+00:00` " is usually shortened to "`Z` ". Oracle Java may use 1 to 4 digits without delimiting ":", in that case 1 or 2 digits mean whole hours whereas 3 or 4 digits mean an 1 or 2 digits of hour and two digits of minutes. For historical reasons, "`-00:00` " notation differs from "`+00:00` " and mean timezoneless, not GMT datetime.

Comparison of Datetimes

ISO 8601 explicitly warns that comparison of timezoned and timezoneless datetime is not always possible. Valid timezones vary from -14:00 to +14:00, the fact that the span can exceed 24 hours may be not obvious. Nevertheless, storing rows in a database table require some unambiguous order; any order is OK as soon as it does not break the rules and common sense, but it should be well-defined. Virtuoso's order for mix of timezoned and timezoneless datetimes is very simple.

1. All timezoned datetimes are sorted in natural chronological order, like if they are converted to GMT first. The value of timezone offset does not matter.
2. All timezoneless datetimes are sorted in natural chronological order, like they are in GMT already.
3. For each GMT calendar day, all timezoned datetimes are placed before all timezoneless datetimes.

Related Functions

- `is_timezoneless()` -- The function returns 1 for timezoneless arguments, zero for timezoned.

```
integer is_timezoneless (in dt datetime)
```

- `timezone()` -- The function returns timezone offset of its first argument, as an integer value in minutes. If the first argument is timezoneless and second argument is missing or zero then the returned value is NULL. If the first argument is timezoneless and second argument is nonzero then the returned value is 0.

```
integer timezone (in dt datetime [, in ignore_tzl integer])
```

- `adjust_timezone()` -- The function returns its first argument with unchanged GMT value but new timezone offset, as it is specified by the second argument. If the first argument is timezoneless and third argument is missing or zero then error 22023 is signaled. If the first argument is timezoneless and third argument is nonzero then no error is signaled and the argument is handled like it is a GMT value.

```
datetime adjust_timezone (in dt datetime, in tz_offset integer [, in ignore_tzl integer])
```

- `dt_set_tz()` -- The function returns its first argument with unchanged GMT value but new timezone offset. Unlike `adjust_timezone()` , if the argument is timezoneless then no error is signaled.

```
datetime dt_set_tz (in dt datetime, in tz_offset integer)
```

- `forget_timezone()` -- The function returns its first argument as a timezoned value. If the first argument is timezoneless then it is returned unchanged. If the first argument is timezoned and second argument is missing or zero then the result is timezoneless value that "looks like" local time notation. If the first argument is timezoned and second argument is nonzero then the value is first made GMT and then it becomes timezoneless.

```
datetime forget_timezone (in dt datetime [, in ignore_timezone integer])
```

- `now()` -- returns the current transaction timestamp:

```
datetime now ()
```

- `rdf_now_impl()` -- returns the timestamp associated with current transaction as a DATETIME . Alias of `now()` :

```
datetime        rdf_now_impl ()
```

- `getdate()` -- returns the current transaction timestamp, alias of `now()` :

```
datetime getdate ();
```

- `get_timestamp()` -- returns the timestamp of the current transaction:

```
datetime get_timestamp ()
```

- `current_timestamp()` -- All these names refer to one function that returns the timestamp of current transaction. It is the datetime of the beginning of current transaction with the fractional part of seconds replaced with serial number of a transaction within the second. If `TimezonelessDatetimes=0` then the time has local timezone offset (as it was set at the time of last server start); otherwise it is timezoneless.

```
datetime current_timestamp ()
```

- `curdatetime()` -- The function returns current datetime, like `now()` , but fractional part of seconds can be adjusted by providing the number of "microseconds" as the argument.

```
datetime curdatetime ([in fraction_microseconds integer])
```

- `curdatetimeoffset()` -- The function is like `curdatetime()` but the returned datetime is in GMT timezone.

```
datetime curdatetimeoffset ([in fraction_microseconds integer])
```

- `curutcdatetime()` -- Refers to function that is similar to `curdatetime()` but the returned datetime is in GMT timezone.

```
datetime curutcdatetime ([in fraction_microseconds integer])
```

- `sysutcdatetime()` -- Refers to function that is similar to `curdatetime()` but the returned datetime is in GMT timezone.

```
datetime sysutcdatetime ([in fraction_microseconds integer])
```

Unicode Support

Virtuoso allows 30-bit Unicode data to be stored and retrieved from database fields. The data are stored internally as UTF-8 encoded strings for storage space optimization. Unicode fields are easily intermixable with other character data as all SQL

functions support wide-string case and convert to the most wide character representation on demand. The native width of the wide character type may differ between platforms. Windows has a 16 bit wide character, whereas some Unixes have a 32 bit wide character type. The native width applies to the Virtuoso NVARCHAR data type when used as SQL data.

There are 3 additional data types to enable storing of Unicode data:

- NCHAR
- NVARCHAR
- LONG NVARCHAR

All the Unicode types are equivalent to their corresponding "narrow" type - CHAR, VARCHAR and LONG VARCHAR - except that instead of storing data as one byte they allow Unicode characters. Their lengths are defined and returned in characters instead of bytes. They collate according to the active wide character collation, if any. By default this is the order of the Unicode serialization values. These types can be used anywhere the narrow character types can be used, except in LIKE conditions.

Unicode literals are introduced by n' and closed with ' (single quote). See Internationalization section on the interpretation of wide literals. This may be either UTF-8 according to some character set.

When there is a need to convert a wide string to a narrow one or vice versa, a character set is used. A character set returns a wide string code for a wide char. For example there can be a definition of the ISO-8859-5 "narrow" character set which describes mapping of non-ASCII character codes to their Unicode equivalents. Virtuoso relies on the fact that the ASCII character codes are represented in Unicode by type-casting and in UTF8 as one-byte tokens with the same value as in ASCII.

When conversion is done on the server-side using cast or some of the SQL built-in functions, the wide characters are converted to narrow using a system-independent server-side character set. In the absence of such a character set, Virtuoso uses the Latin1 character set to project narrow character codes into the Unicode space as equally valued wide-character codes.

When conversion is done client-side - for example, when binding a VARCHAR to a wide buffer - the default client's system character set is used.

Wide-character literals have ANSI SQL92 syntax: N'xxx' (prefixing normal literals with the letter N). These strings process escapes with a values large enough to represent all the Unicode characters.

User Defined Types

Virtuoso supports user-definable data types that can be based on any hosted language or classes such as C#. New types can be further derived producing sub-types. User-defined types can include methods and constructors to create any potentially complicated system to house data as exactly required.

User defined types can be used to defined database table columns.

See Also:

The User Defined Types section.

Built-in SQL Functions and Wide Characters

All the built-in SQL functions that take character attributes and have a character input calculate their output type such that if any attribute is a wide string or a wide BLOB, then the result is a wide string; otherwise, the output character type is narrow.

Functions like `make_string()` that have character result types but that do not have character parameters produce narrow strings. Virtuoso provides equivalent functions for wide output, such as `make_wstring()` .

Client-side changes to support wide characters

Virtuoso' ODBC client implements the SQL...W functions (like `SQLConnectW()`) that take Unicode arguments. This enables faster wide-character processing and allows binding of the SQL_C_WCHAR output type. Since Virtuoso's SQL parser does not allow Unicode data in SQL commands, they should be bound as parameters or should be represented as escapes.

Virtual Database and National Language Support

Attached tables use the default collation of the data source for narrow strings. Virtuoso maps Wide-string columns in remote tables to the appropriate local wide-character type. The data are then passed intact in case of wide-to-wide mapping. When data are converted client-side in the VDB the Server's system character set is used (where available).

Operations Between Large Objects, Varchars and String Outputs

The built-in data types denoting sequences of characters, wide or narrow, long or short, are:

Varchar : a string of 8-bit characters, including 0's, up to 16MB long. These are contiguously stored, so long contents, such as in the megabytes, will be inefficient.

NVARCHAR : A string of wide characters, of 2 or 4 bytes each, depending on the platform. Because of the 16MB limit, the longest strings will be of 4M or 8M characters, depending on the platform. Again long strings are not recommended due to inefficiencies.

Binary : A string of 8-bit bytes, up to 16 MB long, like a varchar but not usable for character functions. There is a distinct binary type only for compatibility with the SQL92 standard and ODBC, where the binary type is treated differently in parameter binding.

Long varchar, long nvarchar : These are long data types, stored persistently as a series of linked pages and accessible to clients in fragments using the `SQLGetData()` and `SQLPutData()` calls. The length limit is 2GB. The wide variant, LONG NVARCHAR, is internally stored as UTF8.

String_output : This is not a database column type but a run-time object that can be used in stored procedures for accumulating a long sequence of 8-bit bytes, including 0's. This type is not contiguously stored, hence it stays efficient for large output and has no built-in size limit; however, it is not automatically paged to disk, so it will consume virtual memory for all its length. This type is useful for buffering output for a next processing step.

Long varbinary : This is a binary BLOB, identical to long varchar but distinct for reasons of compatibility with SQL92 and ODBC, where this can behave differently from long varchar for parameter binding.

XML Entity : This type is a pointer to an element of an XML tree. The XML tree itself may be either memory- or disk-based. In both cases there is a reference-counted set of XML entities for each tree that Virtuoso uses to reference individual elements of the tree. These are used for navigating an XML tree in XPath or XSLT; hence, one entity gives access to it parents, siblings, and so on. This is not properly a string type, but it can be converted to one, producing the XML string value.

All these types have the common trait of representing sequences of characters and hence some common operations and conversions are possible between them.

Storage in Database

The descriptions below apply to insert and update operations for these types:

- Long varchar = x, where x is:

  varchar - The text is stored as is.

  Long varchar - the text is stored as is.

  string output - the contents in the string output are stored as the value, unchanged. The state of the string output is not changed.

  XML entity - The XML tree rooted at the entity is stored as persistent XML (disk-based) if the entity references a persistent XML tree. Note that this may either extract a subtree or copy a tree, depending on whether the entity references the root. If the entity references a memory-based tree, the text of the tree with the element as the topmost (document) element is produced and set as the value of the column.

  Nvarchar - The text is stored as wide, thus the value is internally a long nvarchar although the declared column type is long varchar.

  Long nvarchar - The value is stored as a long nvarchar, as with an nvarchar.
- Long nvarchar = x

  The cases are identical to long varchar. Thus a wide value stays wide and a narrow value stays narrow. Specifically, a string output and XML entity result in a narrow value, although the character combination in the XML entity may be interpreted as wide.
- Long varbinary = x

Identical to long varchar. The binary type is only distinct in column metadata for ODBC clients, where its type conversions may be different.

- Varchar = x, where x is:

  long varchar, string output, XML entity - as with long varchar.

  Nvarchar, Long nvarchar - the text is stored as wide; no information is lost.

- Nvarchar = x, where x is:

  Long varchar, varchar - the string is converted to wide according to the character set effective in the connection.

  Long nvarchar, Nvarchar - The text is stored as is.

'String output' and 'XML entity' are not valid types for a column. These types are only created by evaluating SQL expressions and are converted as specified above if stored as a column value.

Retrieving Column Values

A BLOB column (long varchar, long nvarchar, long varbinary) may return either a long varchar or a long nvarchar BLOB handle. If the actual value is short enough to be inlined, a varchar or nvarchar value can be returned as the column value instead. These are indistinguishable for assignment and as arguments to SQL functions or for returning to a client application. Only specific SQL functions (`isblob()` , `isstring()` , etc.) allow you to determine the difference at run time. One exception is persistent XML entities, which come back as persistent XML entities and are not compatible with string functions but are assignable to various character columns.

An nvarchar column is always nvarchar.

A varchar value is either varchar or nvarchar. If the value stored was a memory-based XML tree entity it comes back as a long varchar. If it was a persistent XML tree, it comes back as an XML entity.

Assignment

PL variables are typed at run time.

A string (varchar, nvarchar, or varbinary) can be freely assigned and passed as parameter. This makes a copy, except for reference (inout) parameters.

A BLOB (long varchar, long nvarchar, long varbinary) is a reference to a disk based structure, unless stored inline. Therefore, passing these as parameters does not take significant time. If these are inline, these are strings of under 4K bytes; hence assigning them is still efficient, although it involves copying.

A string output cannot be assigned between two variables, though it can be passed as a reference (inout) parameter in a PL procedure call. Copying streams has problematic semantics and can be very resource-consuming.

An XML entity can be assigned and passed as parameter without restrictions.

Built-In SQL Functions

All SQL92 string functions will accept varchar, long varchar, nvarchar or long nvarchar arguments. If the argument is long and its actual length is above the maximum length of a varchar, the conversion fails and Virtuoso signals an error. You can interchange long and varchar types as long as the length remains under the varchar maximum of 16MB.

Note:

Varchars or nvarchars stored in columns have a much lower limit due to the 4K row length limit. Intermediate results or values converted from long columns are not affected by this limit.

If Virtuoso converts a value from long varchar to varchar or from long nvarchar to nvarchar when passing the value as an argument to a string function, the value changes in place. This has the effect of replacing the handle with the string. Users normally do not see this, but may detect it with type test functions such as `isblob()` .

Long Strings and Row Length Limit

You can declare string values that might be long and that do not have to be key parts in indices as long varchar. These will automatically be inlined if the row with the data inlined will fit within the 4K limit. Otherwise the long values will be stored as separate LOBs. The difference between varchar and long varchar is distinguishable only with special test functions if the length is under the varchar limit.

A varchar column is sometimes substantially faster on update than a long varchar column, even if the value ends up inlined. If the value is inlined there is no difference in retrieval speed.

Handling Long Data for Input and Output

LOBs of up to 2GB can be handled as streams without demand on memory from ODBC clients using `SQLGetData()` and `SQLPutData()` . All other ways of processing long data will need to make either a contiguous or non-contiguous copy in memory.

To transfer long data between PL procedures and files one can use the `string_to_file()` function, which will accept a handle and will not need to copy the content to memory in order to write it.

To read a large object from a file to a table, you can use the `file_to_string_output()` function to get contents that may be longer than the varchar limit into a string output. This can then be assigned to a BLOB column.

For long file-resident XML data you can use the `xml_persistent()` function with the file:// protocol.

See Also:
The XML Support chapter.

5.1.3. Virtuoso Column Store

Note: This feature only applies to Virtuoso 7.0 and later.

As of version 7, Virtuoso offers a column-wise compressed storage format alongside its traditional row-wise storage format.

In the column-wise storage model, each column of a table or index is stored contiguously, so that values of a single column on consecutive rows are physically adjacent. In this way, adjacent values are of the same type, and if the index is sorted on said value, the consecutive values often form an ascending sequence. This organization allows the use of more powerful compression techniques than could be used for rows where consecutive values belong to different columns, and thus are of disparate data types with values in different ranges.

Furthermore, when queries only access a subset of columns from one table, only those columns actually being accessed need to be read from disk, thereby making better use of I/O throughput and memory. Unreferenced columns will not take space in the memory based cache of the database. Further, the traffic between CPU cache and main memory is reduced when data is more compact, leading to better CPU utilization.

The column-wise format is substantially more compact and offers substantially greater sequential-access performance, as well as greater random-access performance in situations where many rows are accessed together in a join. For single-row random-access, a row-wise format offers higher performance as long as the data is in memory. In practice, for large tables, the higher compression achieved with column-wise storage allows a larger portion of the data to be kept in memory, leading to less frequent I/O and consequently higher performance.

One should not use column-wise storage in cases where columns are frequently updated, especially if a single row is updated per statement. This will give performance substantially worse than row-wise storage. However, bulk inserts and deletes are efficient with column-wise storage.

Creating Column Store Tables and Indices

Any index or primary key, i.e., any table, can be declared to be stored column-wise. A single table can have multiple indices, of which some are stored column-wise and some are not. As with tables stored row-wise, the table row itself is stored following the primary key index entry on the index tree leaf corresponding to the entry. This arrangement is sometimes called a *clustered index* .

One can specify column-wise storage as the default for any new tables or indices by adding ColumnStore = 1 to the [Parameters] section of the virtuoso.ini file. Otherwise, tables and indices are created tow-wise unless the column option is specified, as

described below.

The statement below declares the table xx to be stored column-wise:

```
CREATE TABLE xx ( id    INT,
                  data  VARCHAR,
                  PRIMARY KEY (id) COLUMN
                );
```

This statement adds a column-wise stored index to the table:

```
CREATE COLUMN INDEX xxi
  ON xx (data);
```

The *COLUMN* keyword can come after the column list of the primary key declaration of a table or anywhere between the *CREATE* and *INDEX* keywords of a create index statement.

Note that the *BITMAP* keyword cannot be used together with the *COLUMN* keyword. Column-wise indices will automatically use bitmap compression when appropriate without this being specified. A column-wise index is likely to be more space-efficient than a row-wise bitmap index with the same key parts.

The directives for column compression in *CREATE TABLE (NO COMPRESS, COMPRESS PREFIX)* have no effect on column-wise stored tables. Data is compressed in a manner chosen at run time based on the data itself.

Column Store Transaction Support

All SQL operations work identically for column- or row-wise tables and indices. The locking behavior is also identical, with row-level locking supported on all isolation levels. The behavior of the *READ COMMITTED* isolation is non-locking, showing the pre-image of updated data when reading pages with uncommitted inserts or updates.

Recovery is by roll forward, and checkpoints will only store committed states of the database, even if started when there are uncommitted transactions pending.

Column Space Utilization

The system table *DB.DBA.sys_col_info* holds information about space utilization of column-wise indices.

This table is updated only after the *DB.DBA.sys_index_space_stats* procedure view has been accessed. Thus, one must first make a selection from *DB.DBA.sys_index_space_stats* .

The columns of *sys_col_info* have the following meaning:

- *COI_TABLE*

  - The table in question.
- *COI_INDEX*

  - The index in question.
- *COI_NTH*

  - The ordinal position of the column in question in the key.
- *COI_TYPE*

  - This indicates the type of compression entry the rest of the row concerns. For each column in the key, there is a row with

  coi_type

  set to -1, representing the total of the remaining fields.
- *COI_COLUMN*

- The name of the column concerned.

- *COI_PAGES*

  - This is the number of database pages allocated for storing data of this column.

- *COI_CES*

  - The count of compression entries for the column. A compression entry is logically an array of consecutive values that share a common compression format. Different parts of the same column may have different compression.

- *COI_VALUES*

  - This is the count of values that are stored with the compression format in question.

- *COI_BYTES*

  - The is the number of bytes actually occupied by the compression entries concerned. Pages may not always by full, thus this metric can be used to measure the page fill ratio, i.e.:

```
100 * coi_bytes / (coi_n_pages * 8192.0)
```

To see which columns take the most space, and how full the pages are, as well as the overall effectiveness of compression, one can do:

```
SELECT                                      coi_column         ,
                        coi_pages * 8192  AS  total_bytes        ,
         coi_bytes / (coi_pages * 8192.0)  AS  page_fill          ,
                                           coi_bytes          ,
             1.0 * coi_bytes / coi_values  AS  ce_bytes_per_value ,
            8192.0 * coi_pages / coi_values  AS  bytes_per_value
    FROM sys_col_info
   WHERE coi_type = -1
ORDER BY coi_pages DESC ;
```

Note that issuing a query like:

```
 SELECT TOP 20 *
    FROM sys_index_space_stats
ORDER BY iss_pages DESC;
```

will update the *sys_col_info* table which is initially empty.

The *sys_index_space_stats* view shows the number of pages used for the sparse row-wise index tree top for column-wise indices.

The number of rows shown there for column-wise indices is the number of entries of the sparse index, not the row-count of the index. The space utilization here will be under 1% of the total for a column-wise index.

Below we look at space utilization of the *O* column of the primary key of the *RDF_QUAD* table.

```
SELECT *
  FROM sys_col_info
 WHERE  coi_index = 'DB.DBA.RDF_QUAD'
   AND coi_column = 'O' ;
 coi_table             coi_index           coi_nth           coi_type          coi_column    coi_pages
 VARCHAR NOT NULL      VARCHAR NOT NULL    INTEGER NOT NULL  INTEGER NOT NULL  VARCHAR       INTEGER
 ________________________________________________________________________________________________

 DB.DBA.RDF_QUAD       DB.DBA.RDF_QUAD     2                 -1                O             654663
 DB.DBA.RDF_QUAD       DB.DBA.RDF_QUAD     2                 1                 O             0
 DB.DBA.RDF_QUAD       DB.DBA.RDF_QUAD     2                 3                 O             0
 DB.DBA.RDF_QUAD       DB.DBA.RDF_QUAD     2                 4                 O             0
 DB.DBA.RDF_QUAD       DB.DBA.RDF_QUAD     2                 6                 O             0
 DB.DBA.RDF_QUAD       DB.DBA.RDF_QUAD     2                 8                 O             0
 DB.DBA.RDF_QUAD       DB.DBA.RDF_QUAD     2                 10                O             0
```

The top line is the overall summary across all the compression types.

The lines below give information per-compression-type. The values of *coi_type* mean the following:

- 1 -

  run length

  . The value occurs once, followed by the number of repetitions.
- 3 -

  array

  . Values are stored consecutively without compression. The array elements are 4- or 8-byte depending on range. For variable length types, some compression applies because values differing only in their last byte will only have the last byte stored.
- 4 -

  bitmap

  . For closely-spaced unique ascending values, the bitmap has a start value in full, and a bitmap with the nth bit set if start + nth occurs in the column.
- 6 -

  dictionary

  . For non-ordered, low-cardinality columns, there can be a dictionary with either 4 or 8 bytes per entry, depending on the number of distinct values being encoded. The compression entry is prefixed by an array with the values in full, followed by an array of positions in the dictionary.
- 8 -

  run length with small deltas

  . For repeating, closely-spaced ascending values, the run-length-delta format stores a start value in full, followed by an array of bytes of which 4 bits are a delta to the previous value, and 4 bits are a run length.
- 10 -

  integer delta with large deltas

  . This format stores an initial value followed by stretches of non-ordered values within 64K of the base value. There can be multiple such stretches, each prefixed with a 32-bit delta from the base value. This is useful for closely-spaced medium- cardinality values like dates, or for relatively sparse ascending sequences, e.g., ascending sequences with a step of 1000 or more.

5.1.4. Explicit Vectoring of Procedural Code

Note: This feature only applies to Virtuoso 7.0 and later.

Vectored execution can be explicitly controlled for Virtuoso PL code, either by declaring a whole procedure to be vectored or by executing a block inside a procedure on multiple values at one time. See more detailed description, respectively for:

- Vectored Procedures
- FOR VECTORED Statement
- Limitations on Vectored Code
- Data Types and Vectoring

5.1.5. Locking

Virtuoso offers a dynamic locking strategy that combines the high resolution of row-level locking with the performance of page locking for large transactions.

Isolation Levels

Virtuoso has a full range of isolation options, ranging from *dirty read* to *serializable*. The default isolation is *repeatable read*, which is adequate for most practical applications.

Isolation is set at the connection, i.e. transaction, level. Variously isolated transactions may coexist and each will behave consistently with its semantic.

Repeatable read and *serializable* transactions are susceptible at any time to termination by deadlock, SQL state 40001. Other transactions are susceptible to deadlock if they own locks as a result of insert, update or delete. Deadlocks are resolved in favor of the older of the contending transactions. A transaction's age is the count of reads performed + 2 * the count of rows inserted, deleted or updated.

Any transaction that has modified the database may be rolled back; all transactions maintain a rollback log. This is a memory-based data structure that contains the state of changed rows as they were before the transaction first affected them. This leads to potential transient memory consumption. All transactions that have changed the database also have a roll-forward log, used to recreate the effects of the transaction during roll-forward recovery.

Read Uncommitted

This corresponds to SQL_TXN_READ_UNCOMMITTED. A read is never prevented by locking, nor do read rows stay locked. The data being read may or may not be committed, hence there is no guarantee of transaction integrity.

Read Committed

Historical Read Committed

Starting with release 5.0, Virtuoso has a non-locking, versioned *read committed* transaction mode. This is similar to Oracle's default isolation.

If a locked row is read without FOR UPDATE being specified and another transaction owns the lock, the reading transaction will see the row in the state it had before being modified by the transaction owning the lock. There will be no wait. If a row has been inserted but the insert not committed, the row will not be seen by the *read committed* transaction. If a row has been updated or deleted, the row will be seen as it was before the uncommitted modifying transaction.

If a row is read in *read committed* mode with FOR UPDATE specified or as part of a searched update or delete statement, the *read committed* transaction will wait for a locked row and will set an exclusive lock on the row if the row matches the search criteria. This exclusive lock will be held until the *read committed* transaction terminates.

Hence, if FOR UPDATE is specified, a *read committed* transaction will have repeatable read semantics, otherwise it guarantees no repeatable read but does guarantee that uncommitted data are never seen.

To make this the default mode, set DefaultIsolation in the Parameters section of virtuoso.ini to 2.

Row-by-Row Autocommit

This transaction mode causes all DML statements to commit after every modified row. This is useful for single user situations where one does large batch updates on tables. For example, an update of every row of a multi gigabyte table would be likely to run out of rollback space before completing. In practice, one can end up in a thrashing situation where a large transaction is in progress, is interrupted by a checkpoint which must temporarily roll back the changed pages, then again resume the transaction etc., leading to serious server unavailability. Note that normally the ini parameter TransactionAfterImageLimit places a cap on transaction size, catching situations of this type before they lead to thrashing.

The row by row autocommit mode prevents this from happening by committing each updated, inserted or deleted row as soon as all the indices of the row are updated. This mode will still maintain basic row integrity, i.e. if the row's data is in one index, it will be in all indices.

This mode is good for any batch operations where concurrent updates are not expected or are not an issue. Examples include bulk loading of data, materialization of RDF inferred data etc.

This mode is enabled with the log_enable function. If the bit of 2's is set in the argument, row-by-row autocommit is enabled and the setting will persist until modified with log_enable or the calling connection is disconnected or the calling web request

terminates. Thus, an argument of 2 enables row-by-row autocommit and disables logging. An argument of 3 enables row-by-row autocommit and enables logging. This will cause every updated row to be logged in the transaction log after it is updated, which is not very efficient.

Since transaction-by-transaction recovery is generally not an issue in batch updates, a value of 2 is usually better. If the server is killed during the batch operation, it may simply be restarted and the operation redone. Losing the first half through no logging will not be an issue since the operation will anyway have to be redone.

There is a slight penalty to row-by-row autocommit in comparison with making updates in larger batches but this is under 10%.

Repeatable Read

The transaction will wait for access to exclusively locked rows and will lock all rows it reads. The locking of read rows can be shared or exclusive depending on the FOR UPDATE clause in the SELECT or the SQL_CONCURRENCY statement option. In the case of a select over a range of rows where not all rows match selecting criteria, only matching rows are locked. This mode guarantees that any row locked by the reading transaction can be re-read on the basis of its identity (primary key) and will not have been changed by any other transaction while the locking transaction is in progress. This mode does not prevent another transaction from inserting new rows (phantoms) between rows locked by the original transaction.

Serializable

This mode guarantees that concurrent transactions will look as if the next transaction started only after the previous terminated. This is like `repeatable read` but prevents phantoms. Space found to be empty in one read will remain empty in the next read while the transaction is ongoing.

Serializable isolation is implemented by locking all ranges of rows matching criteria pertaining to the ordering index in a select. The range here includes the last row before the first in the range. An empty range can be locked by locking the row before the range by a special follow lock, which prevents insertions to the right of the locked row. A by-product of this is that serializable locking guarantees that a select count will give the same result repeatedly unless the transaction itself affect the rows counted.

`Serializable` isolation is slower than `repeatable read` and not required by most applications.

All insert, delete and update operations make an exclusive row lock on the rows they operate on, regardless of specified isolation.

Lock Extent

If a transaction is the exclusive owner of locks on a database page and a sufficient percentage of the rows are locked, it makes sense to replace distinct row locks with a single page lock. The LOCK_ESCALATION_PCT parameter controls the threshold for doing this. See the SET statement for details.

If a cursor reads data serially and has a history of locking a high percentage of rows on each page it traverses, it will start setting page level locks as its first choice. It will do this when entering a new page where there are no row-level locks.

Transaction Size

There is no limit in Virtuoso to the transaction size, though the underlying software or hardware may impose limits. Memory consumed by a transaction is proportional to its number of locks held and number of changed rows (insert, update, delete). BKLOBs manipulated by a transaction do not contribute to memory consumption, because they are always disk-based.

5.1.6. Internationalization & Unicode

National strings are best represented as Unicode (NCHAR/LONG NVARCHAR) columns. There is no guarantee that values stored inside narrow (VARCHAR/LONG VARCHAR) columns will get correctly represented. If the client application is also Unicode then no internationalization conversions take place. Unfortunately, most current applications still use narrow characters.

The national character set defines how strings will get converted from narrow to wide characters and back throughout Virtuoso. A character set is an array of 255 (without the zero) Unicode codes describing the location of each character from the narrow character set in the Unicode space. It has a "primary" or "preferred" name and a list of aliases.

Character sets in Virtuoso are kept inside the system table SYS_CHARSETS. Its layout is :

```
CREATE TABLE SYS_CHARSETS (
    CS_NAME varchar,                    -- The "preferred" charset name
    CS_TABLE long nvarchar,             -- the mapping table of length 255 Wide chars
    CS_ALIASES long varchar             -- serialized vector of aliases
);
```

The CS_NAME and CS_ALIASES columns are SELECTable by PUBLIC. To simplify retrieval of all official and unofficial names of character sets, Virtuoso provides the following function:

`charsets_list()`

There are a number of character set definitions preloaded in the SYS_CHARSETS table. Currently these are:

GOST19768-87

IBM437, IBM850, IBM855, IBM866, IBM874

ISO-8859-1, ISO-8859-2, ISO-8859-3, ISO-8859-4, ISO-8859-5, ISO-8859-6, ISO-8859-7, ISO-8859-8, ISO-8859-9, ISO-8859-10, ISO-8859-11, ISO-8859-13, ISO-8859-14, ISO-8859-15

KOI-0, KOI-7, KOI8-A, KOI8-B, KOI8-E, KOI8-F, KOI8-R, KOI8-U

MAC-UKRAINIAN

MIK

WINDOWS-1250, WINDOWS-1251, WINDOWS-1252, WINDOWS-1257

New character sets can be defined using the following function:

`charset_define()`

User-defined character sets can be dropped by deleting the row from the SYS_CHARSETS table and restarting the server.

Virtuoso performs all translations in accordance with a "current charset". This is a connection attribute. It gets its value as follows:

1. If the client supplies a CHARSET ODBC Connect string attribute either from the DSN definition or as an argument to a `SQLDriverConnect()` call, Virtuoso searches for the name in SYS_CHARSETS and, if there is a match, that character set becomes the default.
2. If the database default character set ('Charset' parameter in the 'Parameters' section of virtuoso.ini) is defined, it becomes the default.
3. If neither of these conditions is met, then Virtuoso uses ISO-8859-1 as the default character set; this maps the narrow chars as wide using equality.

At any time the user can explicitly set the character set either with a call to

```
SQLSetConnectAttr (HDBC, SQL_CHARSET (=5002), CharacterSetString, StringLength)
```

or by executing the interactive SQL command:

```
SET CHARSET='<name>|<alias>'
```

The current character set "preferred" name (as a string) is returned by the following system function:

`current_charset()`

Virtuoso has a default character set that gets used if the client does not supply its own and in some special cases, like XML Views and FOR XML AUTO statements.

The HTTP character set can be changed during an HTTP session using:

```
SET HTTP_CHARSET='<name>|<alias>'
```

Example:

```
<?vsp
    set http_charset = 'ISO-CELTIC';
?>
<html><body><h1>Cén chaoi 'bhfuil tú?</h1></body></htm
```

Virtuoso supports the following types of translations from Unicode characters to narrow characters:

- String translation:

  If the Unicode represents a part of the US-ASCII (0-127) character set then its value gets used;
  If the Unicode has a mapping to narrow in the character set then use it;
  If neither of the above then the narrow '?' is returned.
- Command translation:

  If the Unicode represents a part of the US-ASCII (0-127) character set then its value gets used;
  If the Unicode has a mapping to narrow in the character set then use it;
  If neither of the above then the Unicode gets escaped using the form \xNNNN (hexadecimal).
- HTTP/XML translation:

  If the Unicode represents a part of the US-ASCII (0-127) character set then its value gets used after replacing the special symbols (<, >, & etc.) with their entity references;
  If the Unicode has a mapping to narrow in the character set then use it. The narrow char is then checked to see if needs to be escaped;
  If none of the above then the Unicode gets escaped using the form &#DDDDDD; (decimal)

Character Set Use in ODBC/UDBC/CLI Clients

This section describes where a translation is done in the case of an ODBC/UDBC/CLI client. These are described as solution because the Virtuoso CLI is the same as the ODBC/UDBC interface.

For the functions `SQLPrepareW()` , `SQLExecDirectW()` , and `SQLNativeSQLW()` any Unicode arguments will become narrow strings by using the command translation described above.

When doing the bindings

```
SQL_C_WCHAR -> SQL_xxx
```

and

```
SQL_Nxxx -> SQL_C_xxx (except SQL_C_WCHAR)
```

Virtuoso converts Unicode strings to narrow strings using the string translation described above.

Character Set Use in the ODBC/UDBC/CLI Server

The server uses the character set in the CAST operator when converting NCHAR/LONG NVARCHAR to any other type.

Character Set Use in the HTTP Server

The HTTP server appends a

```
charset=xxxx
```

attribute to the

```
Content-Type:
```

HTTP header field when returning the HTTP header to the client. This can be overridden by calling functions such as `http_header()` .

The HTTP server uses the character set mainly to format correctly values using the `http_value()` function or its VSP equivalent <?= ...>. In these cases wide values and XML entities - the result of XML processing function like `xpath_contains()` - get represented using the HTTP/XML translation rules described above. The same rules apply for

results returned by the FOR XML directive, by XML Views, and for WebDAV content.

Character Set Use in the XML Processor

The Virtuoso embedded XML parser correctly processes all encodings defined in the SYS_CHARSETS table and UTF8.

Generation of SQL

The `xpath()` and `xpath_contains()` functions translate their expressions as follows:

Input Processing

Narrow strings are these get translated to Unicode as per the character set and then to UTF-8, which is the internal encoding used by the Virtuoso XML tools.

SQL Views and FOR XML directives take their values from narrow columns by firstly converting them to Unicode based on the database character set and then to UTF-8.

Output Processing

Almost all the XML processors and generators return their values as type DV_XML_ENTITY (__tag() 230). If such a value's character representation is requested either by CAST or by `http_value()` then Virtuoso converts it to narrow characters using the HTTP/XML translation rules given above.

XPath expressions that return string values are returned as NCHAR values to the clients, which then convert them to narrow character if needed.

5.1.7. Creating A Collation

Virtuoso supports collation orders for CHAR and VARCHAR fields that are different from the binary, as per ANSI SQL92. When comparing strings using a collation, Virtuoso compares the "weights" of the characters instead of their codes. This allows programs to make different characters compare as equal (example: case-insensitive comparisons).

A collation can be created by supplying a collation definition text file to the `collation_define()` SQL function. The collation definition file contains a list of the exceptions to the binary collation order. An exception consists of <character code> = <collation weight> pairs. For example a case-insensitive collation can be defined by specifying all the lower case letters to have the same collation weights as the corresponding uppercase ones.

Collation Definition File

The collation definition file should follow the following guidelines:

- Each definition should reside on a separate line.
- The format of the definition is: <CHAR>=<CODE>, where *CHAR* and *CODE* can be either the letters themselves, or their decimal codes. For example: '67=68' is the same as 'C=D' using the ASCII character set. For Unicode collations the codes can exceed the byte boundary.

You can define a new collation using the following function:

```
collation_define ( COLLATION_NAME FILE_PATH ADD_TYPE )
```

Collations System Table

The SYS_COLLATIONS system table holds the data for all defined collations. It has the following structure:

```
CREATE TABLE SYS_COLLATIONS (
    COLL_NAME VARCHAR,
    COLL_TABLE LONG VARBINARY,
    COLL_IS_WIDE INTEGER);
```

COLL_NAME is the fully qualified name of the collation - its identifier.

COLL_TABLE holds the collation table itself. This is 256 bytes or 65536 wide chars.

COLL_IS_WIDE holds the collation's type: 0 for CHAR and 1 for NCHAR. An 8-bit collation cannot be used by anything that requires NCHAR data and vice versa.

A collation can be deleted by deleting its row from SYS_COLLATIONS.

Note

The collation will still be available until the server is restarted, as it's definition is cached in memory.

Collations and Column Data

The collation is a property of the column holding the data. This means that all comparisons including that column will use its collation. SQL functions will strip collation data from the column; for example, if a column "CompanyName" has an assigned collation "Spanish" then the SQL call

```
LEFT (CompanyName, 10);
```

will use the default collation).

Collations can be defined on a per-column basis, at table creation time, and on a per-database basis as a configuration parameter. There is a special form of the CAST operator that allows casting a column to a collation.

A collation identifier has the same form as any other SQL identifier (<qualifier>.<owner>.<name>) and it can be escaped with the same syntax as other identifiers.

Defining a Collation for a Table Column

You may assign a collation to a column at table creation using the following syntax:

```
create table TABLE_NAME (
...
COLLATED   VARCHAR(50) COLLATE Spanish,
COLLATED   CHAR(20) COLLATE DB.DBA.Spanish,
....
)
```

Assigning a collation to a non-character column gives an error.

If the COLLATE is omitted, the default database collation is used.

On database start-up the collation for each table's column is loaded from the SYS_COLLATIONS table and if not found, the COLLATE attribute is ignored until the next restart.

Defining Database-Wide Collations

The database's default collation is defined by the configuration parameter "Collation" in the "Parameters" section of the virtuoso.ini file. This database wide collation is the default system collation used where none other is specified. This setting can only be changed in the virtuoso.ini file and hence requires a Virtuoso server restart. As with all collations, legal values are those contained in the DB.DBA.SYS_COLLATIONS table. The list can be retrieved using `charsets_list(1)`

5.2. Virtual Database (VDB) Engine

- The Need for VDB Engines
- First Generation Virtual Database Products
- VDB Implementation Issues
- VDB Engine Components

5.3. Web & Internet Protocol Support

Virtuoso provides direct access to a number of Internet protocols through built in procedures. These protocols include SMTP, NNTP, POP3, HTTP, XML and many more. Combined with Virtuoso's native database and virtual database capabilities applications can be developed very rapidly from scratch or existing systems can be enhanced with a rich set of tools. An old customer table can be used to create a mail shot; a products table can now generate an XML file which can be converted to plain

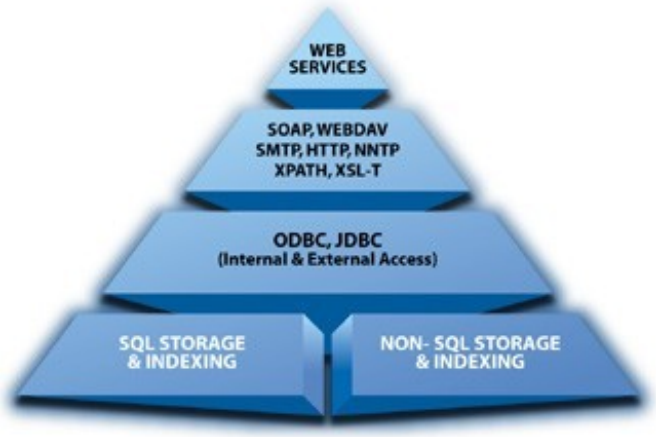
text and or HTML via XSLT which can be emailed to the customers every week automatically using the scheduler, or on request from a VSP page served by the HTTP server. The possibilities are endless and all these abilities are contained within a single server.

5.4. Web Services Protocol Support

The Internet is reaching a stage where automatic processes want to rely on other sites and services presenting their uses in a standard way. This is Web Services. What ever an organization or repository of data contains the it needs to be exploited as fully as possibly. This is exposing existing systems as Web Services. These need to be accessible over well known protocols which combine to provide a service over the web.

SQL provides the means to query data within a database. This is a very well known and supported operation. The query may or may not be able to make use of free text indexes to obtain results faster and more accurately from textual sources. The source could even come from a remote database contacted over ODBC, JDBC or OLEDB - more well supported standards. The results may be shipped to another server for further processing. Further processing could include transformations into various XML's directly or via an XSLT stylesheet. This collaboration of services can be enhanced further with other Web and Internet protocols for the grand objective of making parts of useful information or ability consumable over the Internet as a Web Service.

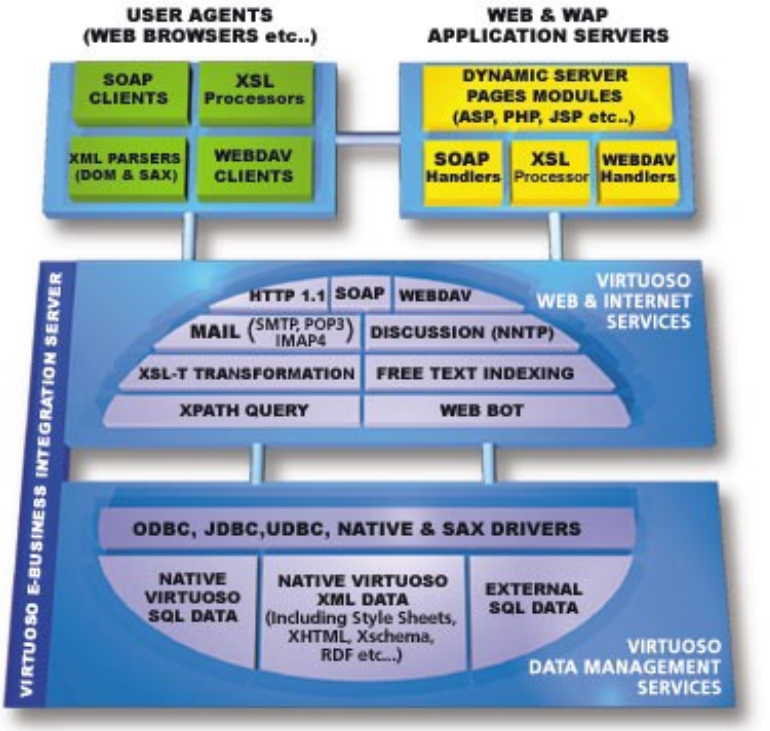
Figure 5.1. Towards Web Services...



5.5. Architecture

The base of Virtuoso is the Virtuoso Data Management. This handles the normal tasks of the database and virtual database collaborating between local native and remote SQL and XML based resources. Web and Internet services and protocols then sit atop of this layer to expose Virtuoso to the Web and its users.

Figure 5.2. OpenLink Virtuoso Product Architecture


USER AGENTS
(WEB BROWSERS etc..)
WEB & WAP
APPLICATION SERVERS
SOAP CLIENTS
XSL Processors
XML PARSERS (DOM & SAX)
WEBDAV CLIENTS
DYNAMIC SERVER PAGES MODULES (ASP, PHP, JSP etc..)
SOAP Handlers
XSL Processor
WEBDAV Handlers
VIRTUOSO E-BUSINESS INTEGRATION SERVER
HTTP 1.1
SOAP
WEBDAV
VIRTUOSO WEB & INTERNET SERVICES
MAIL (SMTP, POP3 IMAP4)
DISCUSSION (NNTP)
XSL-T TRANSFORMATION
FREE TEXT INDEXING
XPATH QUERY
WEB BOT
ODBC, JDBC,UDBC, NATIVE & SAX DRIVERS
NATIVE VIRTUOSO SQL DATA
NATIVE VIRTUOSO XML DATA (Including Style Sheets, XHTML, Xschema, RDF etc...)
EXTERNAL SQL DATA
VIRTUOSO DATA MANAGEMENT SERVICES

Chapter 6. Administration

6.1. Database Server Administration

Abstract

This section explains the use of the server executables, configuration files and general administration procedures. These include backup, server tuning and profiling, installing extension packages etc. See the Access Interfaces chapter for a discussion of the command line SQL interface.

6.1.1. Database

See details for Installation Requirements , Operational Requirements , Operating System Support and Limits .

Server Instance Creation

Multiple Virtuoso server instances may coexist on a single machine. Each database will need to be assigned a unique TCP port number. On the windows platforms (except 95,98,ME) the Virtuoso can be configured for multiple services. For further details,

see the Creating and Deleting Virtuoso Services in the Installation chapter.

To run Virtuoso on a machine, only the Virtuoso server executable and a valid virtuoso.ini file are needed. An empty database file will be created automatically at first startup. None of the other files in the installation are needed for basic operation. Client interfaces may additionally need ODBC drivers or the like to be set up in system configuration files or the Windows registry but the server itself does not depend on these.

Installing Application Packages

Virtuoso comes with optionally installable SQL application packages (VAD packages) for web based admin, on-line documentation, programming tutorials and a BPEL processor. The installer typically asks whether to install these into the demo or the default empty database. Depending on the OS and form of installer, also depending on whether the installation is the commercial or open source one, the package files will be in different locations. To locate them, look for *.vad. The packages are typically made in two variants, where one keeps the installed items in the DAV repository and the other in the file system. They are functionally equivalent in most cases, except for some tutorials that will only work in the file system. Keeping installed resources in DAV has advantages when moving the database or backing up, since the installed items will not have to be treated separately. To install these on a database, login as dba using the isql utility and issue the SQL command:

```
# at the OS command line prompt, assuming the dba password is dba and the server is at the default port 1

$ isql 1111 dba dba

--  At the SQL prompt:
SQL> vad_install ('conductor_dav.vad', 0);
-- The path is relative to the server's working directory.
```

Since the packages are read from and sometimes the extracted contents are written to the file system, the server must have access to the directories in question. Checjk OS permissions and set the DirsAllowed parameter in the ini file to allow the needed access.

After the packages are installed, they can be used by pointing the web browser to their start page. The start pages are listed on the root page of the demo database and at the Getting Started section of the Open Virtuoso web site. The start page for the Conductor web admin interface is /conductor, the docs are at /doc/html, the tutorials at /tutorial, the BPEL admin and demos at /BPELGUI. All these are directly under the default Virtuoso web listener, the port is shown in the messages log and read from the HTTP Server section of the virtuoso.ini file.

Server Licensing

The Virtuoso server requires a valid license file before it will successfully start. The license file is a file always called *virtuoso.lic* and must reside in the working directory of the database instance - where the <database-name>.db file resides for the instance.

Evaluations versions of Virtuoso come without a license file, however the registration procedure takes your email address, which will be used to email a new license file for you evaluation.

Server Logging - Detecting Errors

Virtuoso provides extensive log information for resolving problems. Should any problems arise Virtuoso logs should always be consulted.

The Virtuoso server by default will send log information to two places, the appropriate system log for the operating system and the virtuoso.log file. On Unix based operating systems, including Linux, this information will appear in the system log files. On Windows the Virtuoso log information will appear in the Application Event Log.

Virtuoso logs information in the system logs before the Virtuoso.log file is open. This advantageously will log errors that cannot be placed in the virtuoso.log file, such as when the virtuoso.ini file cannot be located during Virtuoso startup.

The system log feature can be disabled using the "Syslog" parameter in the [Database] section. This is described in more detail in the following section.

Configuring Server Startup Files

Virtuoso Configuration File

The virtuoso.ini file is read from the directory that is current at server startup. This file contains an entry for all user settable options in the server. It is divided into the following sections:

- [Database] Location of database files
- [Parameters] Server tuning, options
- [HTTPServer] Settings in this section control the web server component of the Virtuoso Server.
- [URIQA] Settings in this optional section control URIQA semantic web enabler.
- [Flags] Settings for options that are generally used with `__dbf_set` as temporally valid (such as enable_joins_only), to be set as permanent.
- [SPARQL] Settings in this optional section control SPARQL protocol endpoint default parameters and limits.
- [I18N] encoding settings
- [Replication] The replication section sets the transactional replication parameters for the server.
- [Mono] The Mono section contains settings for Virtuoso with Mono Runtime CLR support.
- [Client] Client default settings
- [AutoRepair] Corrupted database recovery
- [Striping] Multi-file / multi-disk databases
- [VDB] Virtual database functionality related options
- [Ucms] location of UCM files describing multi-byte encodings such as Far East languages
- [Zero Config] ZeroConfig related options
- [Plugins] VSEI plugin modules

Below are the descriptions for each parameter

[Database]

- **DatabaseFile=virtuoso.db.** For a single file database, this is the relative path of the file in the format appropriate to the platform. The path is resolved relative to the directory that is current at server startup. All other paths are interpreted similarly.
- **TransactionFile=virtuoso.trx.** This is the transaction log file. If this parameter is omitted, which should never be the case in practice, the database will run without log, meaning that it cannot recover transactions committed since last checkpoint if it should abnormally terminate. There is always a single log file for one server. The file grows as transactions get committed until a checkpoint is reached at which time the transactions are applied to the database file and the trx file is reclaimed, unless CheckpointAuditTrail is enabled.
- **ErrorLogFile=virtuoso.log.** This file logs database error messages, e.g. 'out of disk'. By viewing this the dba can trace problems and see at which times the server has started, checkpoints have been made, etc.
- **ErrorLogLevel=7.** This controls what events get logged into the database error log. This should always be 7.
- **LockFile=virtuoso.lck.** This optional parameter can be used to manually specify the location of the Virtuoso lock (.lck) file. This can be relative or the full path to the lock file. Virtuoso, by default, creates a file with the same name as the DatabaseFile but with the extension of .lck. This file exists when the Virtuoso server is running to prevent it starting multiple times using the same parameters, and should be automatically removed by the server upon exit. However, not all file systems support file locking, such as NFS, therefore this parameter can be set to keep the lock file on a more appropriate file system.
- **FileExtend=100.** This is the size that the database file automatically grows (in 8k pages) when the current file is not large enough. Default = minimum = 100. The parameter has no effect if striping is set.
- **Striping=0.** A non-zero value will enable the settings in [Striping] to take effect. If this is the case the DatabaseFile parameter is ignored and the files in [striping] are used.
- **LogSegments.** If this is non-zero log segmentation is enabled. This is only used for crash dumps where several files may be needed to accommodate the recovery logs. If non-zero, this will be followed by entries of the form Log1=...
- **Log1=/tmp/log1.trx 100M.** The number in Log<n> is the ordinal number of the log, starting at 1. The entry consists of the file name and allocated size with an optional size modifier. Available modifiers are B for blocks of 4k (default if unspecified, K for kilobytes, M for megabytes, and G for gigabytes. The log is filled up to the first transaction that exceeds the size. The log therefore will be longer than the allocated size. If blobs are involved, this amount can be quite substantial. It is therefore advisable to have some extra space available on the storage device used for recovery logs.
- **crashdump_start_dp, crashdump_end_dp.** These options make it possible to produce a crash dump of a specified range of disk pages. In case the server runs out of disk space during a dump, the error message indicates the page at which the next dump should be started after the user has made more space available. In this case it is important to rename the already produced crash dump or change the Logx= -entries in virtuoso.ini to avoid overwriting said files. If none of

these options are provided, the server will attempt a crash dump of the entire database storage.

- **TempStorage = <TempDatabase_Name>.** The name of a section in the INI file containing temporary database details. If this parameter is omitted or the section does not exist the default values for temporary storage are used.
- **Syslog = 1/0 (default 0).** Virtuoso can writes log worthy messages to the system log (Unix based operating systems including Linux) or the Windows Event Log (Windows operating systems). Messages are written in the system/event log before the virtuoso.log file is opened, therefore errors due to absence of virtuoso.ini log are loggable there. This system/event logging can be enabled using this option, by default it is set to 0 meaning off.

  On Unix/Linux messages are written as "Virtuoso" events.

  On Windows messages are written in the Application event log.

[<TempDatabase_Name>]

This section name must match the TempStorage parameter in the Database section of the Virtuoso INI file to be of any use. Otherwise this section will be ignored.

- **DatabaseFile = <database file name>.tdb.** Name of temporary database file.
- **TransactionFile = <transaction file name>.ttr.** Name of temporary transaction file.
- **FileExtend = NNN.** Increment amount by which the database file will dynamically grow. This setting is identical in use and purpose to parameter in the Database section with the same name.

[Parameters]

- **SingleCPU=0.** This is a Win32 specific option that forces Virtuoso to only run on one CPU in a multiprocessor environment.
- **ServerPort=[<IP Address>]:<port>.** This is the IP Address and port number where the server will start listening. You do not need to specify the listening IP Address but can do in a situation that you want the server to bind to a specific address only.
- **ServerThreads=10.** This is the maximum number of threads used in the server. This should be close to the number of concurrent connections if heavy usage is expected. Different systems have different limitations on threads per process but a value of 100 should work in most places.
- **ServerThreadSize=50000.** Stack size of thread used for reading client messages and accepting connections.(default : 50 000 bytes)
- **MainThreadSize=100000.** Stack size of the main thread (default : 100 000 bytes)
- **ThreadCleanupInterval.** The interval in minutes (default : 0) after which threads in the thread pool should be released.
- **ThreadThreshold.** The maximum number of threads (default : 10) to leave in the thread queue after thread clean-up interval has expired.
- **SchedulerInterval.** Defines the scheduler wake-up interval (in minutes). By default is 0 i.e. the scheduler is disabled.
- **ResourcesCleanupInterval.** The interval in minutes (default : 0) after which allocated resources will be flushed..
- **FutureThreadSize=100000.** Stack size of worker threads. This is the stack size for serving any client SQL statements or HTTP requests. This can be increased if the application uses recursive stored procedures or links in external code needing a large stack. (default 100 000 bytes)
- **TempAllocationPct.** A Percentage that may be greater than 100%. This gives a percentage of the main .db file to which the temp db file may grow before starting to reclaim the oldest persistent hash index. Basically if a particular hash index is reusable (i.e. it references only table columns and the values in these columns have not changed) the engine keeps the hash index defined into the temp db for reuse. This parameter allows some control over the temp db file size.
- **O_DIRECT.** If this is non-zero, the database file(s) will be opened with the O_DIRECT option on platforms where this is supported. This has the effect of doing file system I/O from application buffers directly, bypassing caching by the operating system. This may be useful if a large fraction of RAM is configured as database buffers. If this is on, the file system cache will not grow at the expense of the database process, for example it is less likely to swap out memory that Virtuoso uses for its own database buffers. Mileage will vary according to operating system and version. For large databases where most of system memory is used for database buffers it is advisable to try this.
- **CheckpointInterval=60.** This is the interval in minutes at which Virtuoso will automatically make a database checkpoint. The automatic checkpoint will not be made if there is less than MinAutoCheckpointSize bytes in the current transaction log. A checkpoint interval of 0 means that no periodic automatic checkpoints are made.

  Setting the value to -1 disables also the checkpoints made after a roll forward upon database startup. This setting should be used when backing up the database file(s). This guarantees that even if the server dies and restarts during the copy, no checkpoints that would change these files will take place and thus the backup is clean.

  the checkpoint_interval SQL function may be used to change the checkpoint interval value while the database is running.

- **CheckpointSyncMode=2.** This controls how the file system is synchronized after a checkpoint. Once the checkpoint has issued all write system calls it needs it can do one of the following depending on this setting:

  0. - Continue, leave the OS to flush buffers when it will.

  1. - Initiate the flush but do not wait for it to complete.

  2. - Block until the OS has flushed all writes pertaining to any of the database files.
- **PageMapCheck=0.** This controls the check of page maps:

  0. - disables the check of page maps.

  1. - enables the check of page maps.
- **NumberOfBuffers=2000.** This controls the amount of RAM used by Virtuoso to cache database files. This has a critical performance impact and thus the value should be fairly high for large databases. Exceeding physical memory in this setting will have a significant negative impact. For a database-only server about 65% of available RAM could be configured for database buffers.

  Each buffer caches one 8K page of data and occupies approximately 8700 bytes of memory.
- **MaxCheckpointRemap=2000.** Specifies how many pages Virtuoso is allowed to remap. Remapping means that pages can consume the space of two actual pages until checkpoint. See the Checkpoint & Page Remapping section in the SQL Reference chapter for more information.
- **PrefixResultNames=0.** This setting should always be 0.
- **CaseMode=1.** This controls the case sensitivity of the Virtuoso SQL interpreter. The following values are supported:

  - 0 - SQL is case sensitive and identifiers are stored in the case they are entered in. This is similar to the Progress or Informix default.
  - 1 - SQL is case sensitive and Unquoted identifiers are converted to upper case when read. To use non-upper case identifiers, the identifiers must be quoted with double quotes ("). This is similar to Oracle.
  - 2 - SQL is case-insensitive and identifiers are stored in the case where first seen. Unlike the situation in other modes, two identifiers differing only in case cannot exist. This is similar to the MS SQL Server 7 default behavior.

  Note:

  Once a database is created in one case mode the case mode should not be changed as that may change the interpretation of stored procedures etc.
- **MinAutoCheckpointSize=4000000.** See CheckpointInterval.
- **AutoCheckpointLogSize.** This is the size of transaction log in bytes after which an automatic checkpoint is initiated. If this is non-zero, whenever the transaction log size exceeds this size an automatic checkpoint is started. This will result in approximately like sized logs to be generated. This is useful with the CheckpointAuditTrail option for generating a trail of equal sized consecutive logs.
- **CheckpointAuditTrail=1.** If this is non-zero each checkpoint will start a new log and leave the old transaction log untouched. A 0 value indicates that the transaction log may be reused once the transactions in it have been written in a checkpoint.

  If it is important to keep an audit trail of all transactions in the database's life time then this option should be true. A new log file will be generated in the old log file's directory with a name ending with the date and time of the new log file's creation. Old log files can be manually copied into backup storage or deleted. Only one log file is active at one time. The newest log file may at any time be written to by the server, but that is the only log file Virtuoso has open at each time. Thus reading any logs is safe. Writing or deleting the active log file will result in loss of data, and possibly referential integrity, in the database. See the Back and Recovery chapter for more information on this and related parameters.
- **AllowOSCalls=0.** If non-zero the system SQL function is enabled. This will allow a dba group user to run shell commands through SQL. This poses a potential security risk and hence the setting should normally be 0.
- **MaxStaticCursorRows=5000.** This is the maximum number of rows returned by a static cursor. Default = 5000
- **FreeTextBatchSize=10000000.** This is the amount of text data processed in one batch of the free-text index when doing a batch update or non-incrementally reindexing the data. Default : 10,000,000
- **NullUnspecifiedParams.** When set to 1, if an application prepares a statement with insufficient number of input parameters, the unspecified ones are assumed to be NULL.
- **Collation.** Defines a sorting order according to SYS_COLLATIONS. The name supplied to this parameter must be in `charsets_list(1)`.

- **DirsAllowed=<path> [, <path>].** <path> := <absolute_path> or <relative_path> comma-delimited list of OS directories allowed for file operations such as `file_to_string()` . The server base directory (the directory containing this INI file) must appear in this list to enable File DSNs to work. On Windows, use "\" as the path separator; on Unix-like OS including Linux and macOS 10.x and later, use ":". The Virtuoso ISQL utility can be used to check the Server DirsAllowed params as described below. The following should show the server's working directory and INI file name in the result.

  ```
  SQL> select server_root (), virtuoso_ini_path ();
  ```

  You can also check the relevant INI setting by running the following statement:

  ```
  SQL> select cfg_item_value (virtuoso_ini_path (), 'Parameters',
  'DirsAllowed');
  ```
- **DirsDenied=<path> [, <path>].** <path> := <absolute_path> or <relative_path> OS directories denied for file operations. See Virtuoso ACL's for information on functions that are restricted.
- **VADInstallDir=<path>.** <path> := <absolute_path> or <relative_path> OS directory containig VADs files. When set, enables automatic update of vads on server startup. On Windows use in the path "\".
- **SSLServerPort.** Specifies the port on which the server listens for incoming SSL CLI requests.
- **SSLCertificate.** The SSL certificate to use (same meaning as the SSLCertificate in HTTPServer section)
- **SSLPrivateKey.** The server's private key (same meaning as the SSLCertificate in HTTPServer section)
- **MaxOptimizeLayouts = 1000.** This parameter governs the maximum number of partial or full join orders that the Virtuoso SQL Optimized compiler will calculate per select statement. When MaxOptimizeLayouts has been reached, the best execution plan reached thus far will be used. The default value is 1000, specifying 0 will try all possible orders and guarantee that the best plan is reached.
- **StopCompilerWhenXOverRunTime = 0.** The default value is 0. If non-zero, this specifies that the SQL compiler should stop considering alternative execution plans after the elapsed compilation time exceeds the best run time estimate times the parameter. For example, if this is 2, then compilation stops after using twice the time of the best plan reached thus far. Enabling this option increases performance when processing short running queries that are each executed once. Using this with long running queries or prepared parametrized queries is not useful and may lead to non-optimal plans being selected.
- **TraceOn = option1 [, option2 [, ..]].** This parameter accepts a comma-delimited list of tracing options to activate by default. Enabled trace options will list there respective errors in the virtuoso.log file when encountered. Valid options are:

  user_log
  failed_log
  compile
  ddl_log
  client_sql
  errors
  dsn
  sql_send
  transact
  remote_transact
  exec
  soap

  See Also:

  The functions: `trace_on()` and `trace_off()` which use the same options and can turn logging options on/off while the server is running.

Example 6.1. Using the TraceOn ini file option

```
[Parameters]
....
TraceOn = soap, errors
...
```

This will enable logging of additional information regarding SOAP calls and SQL run-time errors into the virtuoso.log file.

Example 6.2. Using the ThreadCleanupInterval and ResourcesCleanupInterval ini file option

```
[Parameters]
....
ThreadCleanupInterval    = 1
ResourcesCleanupInterval = 1
...
```

Set both to 1 in order to clean up unused threads/resources and reduce memory consumption of the Virtuoso server, which can otherwise be construed as memory leaks.

If an invalid option is set then this error will be listed in the virtuoso.log file upon server startup. Virtuoso will continue to log selected options unless the trace_off() function is called for that item.

- **AllowPasswordEncryption = 1/0.** Determines whether Virtuoso encryption should be accepted from client connections. The default value is 1 - on. If set to 0 then only clear text and digest authentication will be accepted.
- **JavaClasspath.** This parameter is applied to the environment prior to the server's startup. It is valid only for binaries hosted in the Virtuoso Java VM. This has the same format as the Java CLASSPATH environment variable for the platform being used.

  Virtuoso searches for classes in the following order:

  The java_vm_create VSE parameter list

  This "JavaClasspath" in [Parameters] INI section

  The CLASSPATH environment variable

  If none of the above the CLASSPATH is the current directory.
- **JavaVMOption1..N = <opts>.** These can be used for setting Java options for the Java runtime hosted in Virtuoso. These options work as if provided as command line options to the JRE's Java command line.

  More than one line of options can be specified by using consecutively numbered options:

  ```
  JavaVMOption1  = -Ddt1=val1
  JavaVMOption2  = -Ddt2=val2
  ...
  JavaVMOption5  = -Ddt5=val5
  ```
- **PLDebug = 0.** The PLDebug switch controls the type of debugging enabled:

  PLDebug = 0 - default debugging information and test coverage disabled.

  PLDebug = 1 - debugging information enabled.

  PLDebug = 2 - debugging information enabled, test coverage data will be written to file specified in TestCoverage Virtuoso ini file parameter.
- **TestCoverage = cov.xml.**
- **CallstackOnException = 0.** Controls whether Virtuoso will report call stack on errors.

  This parameters takes the following values:

  0 (default) - Call stack reporting disabled.

  1 - Call stack reporting enabled.
- **CompileProceduresOnStartup = 1.** This controls whether Virtuoso will recompile all stored procedures listed in SYS_PROCEDURES and internal PL procedures defined during server startup. By default Virtuoso will recompile all procedures, with this setting set to 0 Virtuoso will defer compilation until procedures' first call. The benefits of this are that Virtuoso may start up up to 2-3 times faster, also the initial memory consumption will be significantly reduced as it does not need to analyze all the long varchar data allocating memory for execution. This setting is a boolean, either 1 or 0. This setting does not apply to attached VDB procedures or modules.

- **FDsPerFile = 1.** Controls the number of file descriptors per file to be obtained from the OS. The default and minimum value is 1. This parameter only effects databases that use striping. Having multiple FDs per file means that as many concurrent I/O operations may simultaneously be pending per file. This allows more flexibility for the OS to schedule the operations, potentially improving file I/O throughput
- **RecursiveFreeTextUsage = 1/0 default 1.** This option controls the behavior of free-text triggers in super-tables. If this option is set to 1 then Virtuoso will scan the hierarchy of tables until a free-text index is used to use when compiling SQL statements involving contains, xcontains or xpath_contains.
- **RecursiveTriggerCalls = 1/0 default 1.** This option controls the behavior of super-table triggers. When this option is set to 1 then triggers in the super-table will be called before its own (sub-table) triggers. This behavior is recursive and will continue up the branch of recursion, hence the triggers in the top most table in the chain will be called first.
- **MaxSortedTopRows = 10000.** The TOP select statement clause caches in memory the rows pertinent to the result. The number of rows allowed to be cached within memory is limited by this parameter.

  Simple example using OFFSET and LIMIT:

  Virtuoso uses a zero index in the OFFSET. Thus in the example below, will be taken position at record 9000 in the result set, and will get the next 1000 rows starting from 9001 record. Note that the MaxSortedTopRows in parameters Virtuoso ini section needs to be increased (default is 10000).

  ```
  select ?name
  ORDER BY ?name
  OFFSET 9000
  LIMIT 1000
  ```
- **DisableUnixSocket = 0/1 default 0.** This parameter is only applicable to Unix servers. Virtuoso clients on the `localhost` of the server can benefit from using Unix Domain sockets to improve connection performance. By default (DisableUnixSocket = 0) Virtuoso will open a Unix Domain listen socket in addition to the TCP listen socket. The name of the UD socket is `/tmp/virt-<tcp-listen-port>` . When a client attempts to connect to the Virtuoso server using the specific address `localhost` it will first try connecting to the UD socket, failing that it will silently revert to the TCP socket. See the Unix Domain Socket Connections section for more details.
- **TransactionAfterImageLimit = N bytes default 50000000.** When the roll-forward log entry of a transaction exceeds this size, the transaction is too large and is marked as uncommittable. This work as upper limit otherwise infinite (transactions). The default is 50Mb . Then also note that transaction roll-back data takes about 2x of roll-forward data. Hence when the transaction roll-forward data which is 50Mb the total transient consumption is closer to 150 Mb.
- **TempSesDir.** Directory for storing temporary data for large object handled in replication and HTTP server. Defaults to server home directory.
- **DbevEnable = 0/1 default 1.** Enable or disable Database Event Hooks .
- **RunAs.** Specifies the OS user name to which the server will switch after opening the listen ports. Has an effect only on the operating systems that support it.
- **MaxMemPoolSize = 200000000.** This parameter specifies the limit of the memory to be used for compiling a SQL statement. If the query compilation requires more memory an error will be signalled. If this is a zero then no limit will be applied. The default is 200000000. i.e. when no parameter is specified, also if negative number or less than 5000000 is given then it would be set to 5000000 bytes.
- **DefaultIsolation.** This specifies the default transaction isolation. This isolation is used unless overridden by a client setting it using the SQL_TXN_ISOLATION or equivalent connection option or a stored procedure locally setting it with the SET statement. The values are as by the SQL_TXN_ constants in ODBC, that is, 1 for read uncommitted, 2 for read committed, 4 for repeatable read and 8 for serializable. If nothing is specified, the default is repeatable read.
- **UseAIO.** This specifies whether to use asynchronous file I/O on supporting Unix systems. A value of 0 means not using it. A value of 1 means using lio_listio for any background write or read ahead if available. A value of 2 means to use the regular blocking read and write but to merge adjacent operations into a single system call when possible.
- **TempDBSize.** Controls the acceptable size of the temp database file. If on startup it's size (in MB) is greater than TempDBSize the file gets deleted and reset. This feature can be turned off by setting TempDBSize to 0. Note that the temp db file serves as an optimization storage only and doesn't have any client data that are not in either the main database files or the corresponding transaction log files.
- **LiteMode = 0/1 (default 0).** Runs server in lite mode. When Lite mode is on:

  - the web services are not initialized i.e. no web server, dav, soap, pop3 etc.
  - the replication is stopped
  - the pl debugging is disabled
  - plugins are disabled
  - rendezvous is disabled
  - the relevant tables to the above are not created
  - the index tree maps is set to 8 if no other setting is given

- memory reserve is not allocated
- affects DisableTcpSocket. So DisableTcpSocket setting is treated as 1 when LiteMode=1, regardless of value in INI file

- **RdfFreeTextRulesSize = 10 or more.** The size of hash to control rdf free text index
- **IndexTreeMaps = 2 -1024 (power of 2).** Size of index tree maps, larger is better for speed but consume memory, in lite is 2 in 'normal' mode is 256 by default.
- **DisableTcpSocket = 1/0.** Default = 0. If set to 1, disables database listener on TCP port; unix socket must be used for data access connections (ODBC, JDBC, ADO.NET, OLE DB). When LiteMode=1, DisableTcpSocket setting in INI file is ignored and treated as if set to 1.
- **ExtentReadThreshold.** Controls speculative read of disk pages. If pages are read in close succession from an extent of 256 consecutive pages, the system may decide to speculatively read the entire extent.

  ExtentReadThreshold parameter gives how many consecutive reads are needed to trigger this.

  When is set to 0, this means that anytime a page is read, the whole extent is read along with it

  When is set to 1, this means that if the first read is at time *t* and the next one at time *t1* and *t1-t* < ini_ExtentReadWindow msec, then 2nd read triggers the speculative read.

  Default is 2.

  Takes effect after the buffer pool is full.
- **ExtentReadWindow.** Controls speculative read of disk pages. If pages are read in close succession from an extent of 256 consecutive pages, the system may decide to speculatively read the entire extent.

  ExtentReadWindow parameter gives the time within which the reads must fall.

  Default is 1000.

  Takes effect after the buffer pool is full.
- **ExtentReadStartupThreshold.** Controls speculative read of disk pages. If pages are read in close succession from an extent of 256 consecutive pages, the system may decide to speculatively read the entire extent.

  ExtentReadStartupThreshold parameter value applies while the server is freshly started and the buffer pool is not yet full. It can be set to preread more aggressively.

  Default is 0.
- **ExtentReadStartupWindow.** Controls speculative read of disk pages. If pages are read in close succession from an extent of 256 consecutive pages, the system may decide to speculatively read the entire extent.

  ExtentReadStartupWindow parameter value applies while the server is freshly started and the buffer pool is not yet full. It can be set to preread more aggressively.

  Default is 40000.
- **ColumnStore.** If 1, all create table and create index statements will create column-store structures by default.

  Note: Only effective with Virtuoso 7.0 and later.
- **AsyncQueueMaxThreads.** Sets the number of threads in a pool that is used for getting extra threads for running queries and for aq_request. Each running statement has at least one thread that is not allocated from this pool plus zero or more threads from this pool.

  See Also:
  Configuring Vectored Execution

  Setting the pool size to the number of cores plus a few is a reasonable default. On platforms with core multithreading, one can count a core thread as a core for purposes of this parameter.

  If one expects to run many slow `aq_requests()` (see `async_queue()` , `aq_request()` , etc.), then the number of threads should be increased by the number of slow threads one expects.

Slow threads are typically I/O bound threads used for web crawling or similar long-latency, low-CPU activity.

Note: Only effective with Virtuoso 7.0 and later.

- **ThreadsPerQuery.** This is maximum number of threads that can be claimed from the thread pool by a single query. A value of one means that no query parallelization will take place, and all queries will run single threaded.

See Also:

Configuring Vectored Execution

The number of cores on the machine is a reasonable default if running large queries.

Note that since every query is served by at least one thread, a single query taking all the extra threads will not prevent other queries from progressing.

Note: Only effective with Virtuoso 7.0 and later.

- **VectorSize.** This the number of simultaneous sets of query variable bindings processed at one time. The default is 10,000, which is good for most cases.

```
SELECT COUNT (*)
FROM t1 a,
     t1 b
WHERE a.row_no + 1 = b.row_no
OPTION (LOOP, ORDER)
```

See Also:

Configuring Vectored Execution

If we are evaluating the query:

with vector size of 10,000, then 10,000 rows of t1 a will be fetched first; 1 will be added to the 10,000 row_no values; and then the corresponding row of t1 b will be fetched for the 10,000 row_no of t1 a. This process will repeat until enough batches of t1 a have been fetched to come to its end.

Note: Only effective with Virtuoso 7.0 and later.

- **AdjustVectorSize.** Using a larger vector size when evaluating large queries with indexed random-access can yield up to a 3x speed-up relative to using the default vector size. However, always using a large vector size will prohibitively increase the overhead of running small queries. For this reason, there is the option to adaptively select the vector size. Set AdjustVectorSize = 1 to enable this feature. The SQL execution engine will increase the vector size when it sees an index lookup that does not get good locality, (e.g., after sorting the keys to look for, too few consecutive lookups fall on the same page). Having more keys to look up increases the chance that consecutive keys should be found on the same page, thus eliminating much of the index lookup cost.

See Also:

Configuring Vectored Execution

Note: Only effective with Virtuoso 7.0 and later.

- **MaxQueryMem.** This controls the maximum amount of memory that can be used across the server process for large vectors, i.e. if the memory in use is near this limit, a query will not switch to large vector size even if it finds it useful. The event counter tc_no_mem_for_longer_batch counts how many times this situation is detected. A size letter of G or M follows the value.

See Also:

Configuring Vectored Execution

Note: Only effective with Virtuoso 7.0 and later.

- **HashJoinSpace.** This controls the maximum amount of memory that can be used across the server process for hash join hash tables. This is followed by a size letter M or G. A single hash join hash table will only claim a percentage of the remaining hash join space, by default 50. This is controlled by the chash_per_query_pct setting, see section on vectored execution tuning. If there is not enough memory, a partitioned hash join will be used, making as many passes over the data as needed so that the hash table will fit within the set limits.

See Also:
Configuring Vectored Execution

Note: Only effective with Virtuoso 7.0 and later.

- **MaxVectorSize.** When AdjustVectorSize is on, this setting gives the maximum vector size. The default is 1,000,000 and the largest allowed value is about 3,500,000.

See Also:
Configuring Vectored Execution

Note: Only effective with Virtuoso 7.0 and later.

- **TimezonelessDatetimes.** Enables Timezoneless Support. Different applications may require different behavior when input data contain timezoneless values. In some cases it is better to "cast" all of them to timezoned than to upgrade existing code. Virtuoso offers 5 different modes of support -- 0, 1, 2, 3 and 4. This should be set before creating the database and the set value is stored in the database. After database is created, an attempt to change the mode by patching virtuoso.ini will have no effect and virtuoso.log will contain a warning about mismatch between virtuoso.ini and the database file.

    - TimezonelessDatetimes=0 -- Never use timezoneless, as it was in old databases. Always set local timezone on parsing strings if no timezone specified. An attempt to set timezoneless by calling function `forget_timezone()` will signal error. Timezoneless values still may come from outside as dezerializations of timezoneless DATETIME values, serialized by other database instances, but not in any other way:
    - TimezonelessDatetimes=1 -- When parsing strings, set timezoneless if ISO format tells so:
    - TimezonelessDatetimes=2 -- Set timezoneless always, exception is when the parsed string contains explicit timezone or when RFC requires the use of GMT or when timezone is set by function `adjust_timezone()` . This is default for new databases if `TimezonelessDatetimes` parameter is missing in virtuoso.ini
    - TimezonelessDatetimes=3 -- Never use timezoneless. Always set local timezone on parsing strings if not timezone specified. An attempt to set timezoneless by calling function `forget_timezone()` will signal error. Timezoneless values still may come from outside as deserializations of timezoneless DATETIME values, serialized by other database instances, but not in any other way. The difference with `TimezonelessDatetimes=0` is that timezones are always printed on cast datetimes to strings etc. so timezoneless-aware clients will get unambiguous data.
    - TimezonelessDatetimes=4 -- On parsing string, set timezone to GMT if no timezone specified. However, timezoneless can be set by calling function `forget_timezone()` . This mode can be convenient for global web services when real "local" timezones of specific users are not known.

    The possible variants are:

    For new applications, consider the use of `TimezonelessDatetimes=2` as primary variant, `TimezonelessDatetimes=1` as the second best.

Note: The default for startup behavior is to always read full extents and the default for the normal behavior is to trigger preread on the third read inside one second.

[HTTPServer]

Settings in this section control the web server component of the Virtuoso Server.

- **ServerPort.** This specifies the initial HTTP listen port for the HTTP server. Can be specified also as ipaddress:port. Once Virtuoso is started it is possible to create multiple listeners using virtual directories.
- **HTTPThreadSize = 120000.** The stack size of the HTTP thread used for reading/processing HTTP client requests and accepting connections. The default is 120,000 bytes. This parameter cannot have value less than the default; if a smaller value is specified the default will be used.
- **ServerThreads.** This specifies the number of concurrently serviced HTTP requests. Its alias is *MaxClientConnections* . If there are more concurrent requests, accepting the connections will be deferred until there is a thread ready to serve each. If an attempt to exceed the number of licensed connections is found. the latter will be used. When the number of threads is low some of the tutorials may perform poorly and appear not to work; this is due to the demonstration licensing. When testing Virtuoso with the demonstration license it can be quite easy to hit the limits unless remote connections and HTTP connections are conserved. You may wish to either wait for previous transactions to finish or restart the server to be sure.

- **HTTPThreadSize = 120000.** The stack size of HTTP thread used for reading/processing HTTP client requests and accepting connections. The default setting, if not supplied, is 120,000 bytes. The default value is the minimum; lesser values will be rounded up.
- **ServerRoot = ../vsp.** This is the file system path of the root directory of files served by the Virtuoso web server. The index.html in that directory will be served for the / URI. If relative, the path is interpreted relative to the server's working directory.
- **ServerIdString = Virtuoso.** String passed as Server: header to HTTP client. This string is not required, in its absence the above default will be assumed. This string can be set to anything required.
- **ClientIdString = Mozilla/4.0 (compatible; Virtuoso).** String passed as User-Agent: header to server by HTTP client. This string is not required, in its absence the above default will be assumed. This string can be set to anything required.
- **Charset = [CHARSET-NAME].** Allows you to set the default server character set. If no default is specified then ISO-8859-1 will be used automatically by the server.
- **EnabledGzipContent = 0.** This sets the default behavior of HTTP transmission. If set to 1 The Virtuoso HTTP server will send GZipped content to user agents. Otherwise content will be sent as is. The function `http_enable_gz()` lets you change the server mode on the fly.
- **MaxKeepAlives = 10.** Connections by HTTP 1.1 clients can remain open after the initial response has been sent. This parameters sets a cap on how many socket descriptors will at most be taken by keep alive connections. Such connections will be dropped by the server ahead of timeout if this number would be exceeded. Thus the maximum number of open sockets for the Virtuoso HTTP server is this number plus the number of threads. A keep alive connection is by definition not associated to any pending processing on any thread.
- **KeepAliveTimeout = 10.** This is a timeout in seconds before Virtuoso closes an idle HTTP 1.1 connection.
- **HTTPProxyEnabled = 0.** Setting this to 1 activates Virtuoso proxy service capabilities. The default value of 0 deactivates the proxy service.

Note:
Ports on which proxying is enabled should not be presented to the outside world under any situation.

- **HTTPProxyServer = proxylocal:3128.** HTTP proxy server name and port
- **HTTPProxyExceptions = localhost:8890, 127.0.0.1:8890.** HTTP proxy exceptions name and port.
- **HTTPLogFile = log.out.** If specified, Virtuoso will produce an HTTP server log file with the date appended to the name given in the parameter and the files extension as ".out". The log file is rotated daily.

  It will contain the following information:

  `logDDMMYYYY.out` :-

  Client IP address

  Date and time of request/response

  Timestamp (milliseconds)

  Request/response line

  An example of which is:

```
127.0.0.1 - - [12/Sep/2006:12:31:17 +0300] "GET /ods/ HTTP/1.1" 200 19453 "" "Mozilla/5.
127.0.0.1 - - [12/Sep/2006:12:31:18 +0300] "GET /ods/default.css HTTP/1.1" 200 41389 "ht
127.0.0.1 - - [12/Sep/2006:12:31:19 +0300] "GET /ods/common.js HTTP/1.1" 200 7481 "http:
```

- **HTTPLogFormat = format string like apache mod_log.** "HTTPLogFormat" INI file param works in conjunction with the "HTTPLogFile" INI file param. In http log we support format string like Apache Module mod_log_config . For example:

```
...
[HTTPServer]
...
HTTPLogFormat = %h %u %t "%r" %s %b "%{Referer}i" "%{User-agent}i" "%{NetId}e"
```

  In this example we have %{User-Agent}i which means to log the HTTP header for user-agent. In similar way can log other input headers, "e" modifier is for environment variable NetId in this case.

  Note that not all escapes from Apache Module mod_log_config are supported.
- **MaxCachedProxyConnections = 10.** When Virtuoso is acting as a proxy or HTTP client, as is the case with the http_get or the SOAP client functions, it caches connections to HTTP 1.1 servers. This is the maximum population of said cache.

- **ProxyConnectionCacheTimeout = 15.** This is a timeout in seconds for dropping idle connections to other HTTP servers. These result from http_get, SOAP client functions and proxying.
- **DavRoot = DAV.** This specifies the root path of DAV resources. If DAV specific HTTP methods are used on Virtuoso, these should only reference resources with paths starting with this. This is the top level DAV collection. Other paths can be declared as managed in DAV using the virtual directory mechanism, but this applies to the default virtual directory.
- **DAVQuotaEnabled = 1/0.** The Virtuoso administrator can enforce a quota on all DAV accounts, apart from the "dav" administration user, that restricts that amount of space a DAV user can consume in their DAV home directory. When this parameter is set to one (1) quotas are enabled, when this parameter is set to zero (0), they are disabled. The default quota value is five Megabytes (5MB) but can be uniquely defined for each user. Every user has a quota except "dav", which is unlimited. Dav quotas are disabled if this parameter is not specified for backwards compatibility.
- **DAVChunkedQuota = 1000000.** Virtuoso send resources to from WebDAV to the requesting client in chunked encoding to save memory if the request is in HTTP/1.1 and the files size is greater then the `DAVChunkedQuota` number of bytes. The default chunked-quota value is approximately one-megabyte.
- **SSLPort=4433.** this is the port on which SSL HTTPS connections will be accepted. Can be specified also as ipaddress:port. If unspecified then the service will be disabled.
- **SSLCertificate=./virtuoso_cert.pem.** the option must point to the file with the server certificate in PEM format.

  If the option begins with 'db:' e.g. 'db:id_server', then certificate and key will be loaded from DBA key space.
- **SSLPrivateKey=./virtuoso_key.pem.** points to the file containing the RSA private key in PEM format.

  the certificate/key pair must be valid (eq. certificate is generated on base of key)

  If the option begins with 'db:' e.g. 'db:id_server', then certificate and key will be loaded from DBA key space. Note: in the case of database stored key, both SSLCertificate and SSLPrivateKey settings MUST be same.
- **X509ClientVerify=0.** Whether the server will require X509 certificates from the browsers.

  X509ClientVerify = 0 - no certificate verification required

  X509ClientVerify = 1 - ask for trusted certificates

  X509ClientVerify = 2 - optionally ask for trusted certificates, if trusted certificate is presented it will be verified

  X509ClientVerify = 3 - optionally accept any certificate including self-signed certificates

  See Also:

  WebID Protocol ODBC Login
- **X509ClientVerifyDepth=1.** Specifies how deep the client certificate verification process will traverse the Issuer chain before giving up. The default value of 0 means that only verification of the client certificate itself being in the server's CA list is performed. Setting this option to -1 will ignore the depth checks
- **X509ClientVerifyCAFile=./calist.pem.** a PEM file of all the X509 certificates of the Certification Authorities (CA) which the server will use for verifying the client's client certificate. A list of these will be sent to the client as a part of the SSL handshake so the client will know which certificate to send.

  Note this option is in effect when X509ClientVerify is 1 or 2. If X509ClientVerify is 3 and X509ClientVerifyCAFile is given, server will send certificated during handshake, thus some clients may restrict user to send any certificate.
- **POP3Port=1234.** Defines the TCP port number on which the Virtuoso POP3 server will listen. The POP3 server will be disabled if this value is 0 or undefined.
- **NewsServerPort=1235.** Defines the TCP port number on which the Virtuoso NNTP server will listen. The NNTP server will be disabled if this value is 0 or undefined.
- **FTPServerPort.** The Virtuoso FTP server can be enabled by supplying this parameter with a value. This value will then be the listening port of the FTP server. The usual port for FTP is port 21.
- **FTPServerMinFreePort = 20000.** The Virtuoso FTP client and server use FTPServerMinFreePort and FTPServerMaxFreePort parameters as lower and upper bounds for active and passive connections. This parameters sets the lower bound.
- **FTPServerMaxFreePort = 30000.** The Virtuoso FTP client and server use FTPServerMinFreePort and FTPServerMacFreePort parameters as lower and upper bounds for active and passive connections. This parameters sets the upper bound.
- **FTPServerLogFile = ftpserver.** If specified Virtuoso will produce an FTP server log file with the date appended to the name given in the parameter and the files extension as ".log". The log file is rotated daily.

  It will contain the following information:

  `ftpserverDDMMYYYY.log`:-

Client Host Name

Authorized User

Time

User Command

Server Response Code

Bytes Transferred

An example of which is:

```
hostname anonymous [22/Oct/2003:15:21:43 +0300] "PASS user@domain.com" 230 0
hostname anonymous [22/Oct/2003:15:23:11 +0300] "LIST" 226 162
hostname dav [22/Oct/2003:15:25:00 +0300] "PASS <hidden>" 230 0
```

- **FTPServerAnonymousLogin = 0.** Allows the FTP server to be accessible via the "anonymous" user login. The anonymous user is not a real user, it has no SQL or DAV login ability. The anonymous user can only access collections or resources that are set to public.
- **DefaultMailServer=localhost:25.** Default SMTP server name and port, this is used when the first parameter of the smtp_send function is omitted or NULL.
- **TempASPXDir.** Allows you to choose what file system directory to be used for temporary storage of ASPX files hosted in DAV.
- **PersistentHostingModules = [1/]0.** When set to "1" prevents Virtuoso from removing the plugin interpreters from the HTTP threads after each request. The default setting is "0".
- **HttpSessionSize = size in bytes, default 10000000.** The size threshold for large objects received by HTTP server. When this limit is exceeded by incoming or outgoing data, this will be stored in a temp file, see 'TempSesDir' parameter in 'Parameters' section.
- **MaintenancePage = file_name.** The name of a HTML page or other static content to be returned to the user agents when server is running in atomic mode. Note that this page should be self-contained, i.e. no image or CSS not JavaScript references can be used.

[Flags]

- **enable_joins_only = 0.** Setting enable_joins_only will cause the optimizer to only consider next plan candidates that are connected by a join to the existing partial plan. In other words, no cartesian products will be considered. This may save some space and time.

[URIQA]

The URIQA section sets parameters of both URIQA HTTP extension and URIQA web service. Section URIQA Semantic Web Enabler contains detailed description of this functionality, including more details about URIQA configuration parameters . This section should stay commented out as long as URIQA is not in use.

- **DefaultHost = canonical name of the server, used by default for metadata retrieval, no default value.** The server name, including domain and port if needed, such as "www.example.com" or "www.example.com:8088".
- **LocalHostNames = comma-delimited list of names of the server, that can be used for retrieval of metadata, no default value.** List of various allowed spellings of the server name, such as "www.example.com, mail.example.com, mail, localhost, localhost.localdomain".
- **LocalHostMasks = comma-delimited list of name masks of the server, no default value.** List of various allowed spellings of the server name in form of pattern strings for SQL "like" operator, not in form of exact strings.
- **Fingerprint = unique fingerprint string of the server or group of identical servers, no default value.** Do not use this without an explicit need, to let server configure itself. Please refer to detailed description before use.
- **DynamicLocal = 1/0 default 0, allow dynamic hostname translation in the IRIs.** If DynamicLocal is on and the host part of the IRI matches the Host header of the HTTP request in context or the DefaultHost if outside of HTTP context, then this is replaced with local: before looking up the IRI ID.

[SPARQL]

The SPARQL section sets parameters and limits for SPARQL query protocol web service service. This section should stay commented out as long as SPARQL is not in use. Section RDF Data Access and Data Management contains detailed description of this functionality.

- **ExternalQuerySource = 1 or 0 .** This controls processing of the "query-uri" parameter of the SPARQL query protocol webservice, means enable 1 or prohibited 0.

- **MinExpiration = 86400 .** Sponger caching parameter in seconds. It will cause sponger to use this value as minimal expiration of the pages, which would help in cases where source document's server do not report expiration or it reports no caching at all.
- **MaxCacheExpiration = 1 .** Cache Expiration time in seconds that overrides Sponger's default cache invalidation.
- **MaxDataSourceSize = 20971520 .** Controls the max size that can be sponged. Default is 20 MB.
- **ExternalXsltSource = 1 or 0 .** This controls processing of the "xslt-uri" parameter of the SPARQL query protocol webservice, means enable 1 or prohibited 0.
- **ResultSetMaxRows = number .** This setting is used to limit the number of the rows in the result. The effective limit will be the lowest of this setting, SPARQL query 'LIMIT' clause value (if present), and SPARQL Endpoint request URI &maxrows parameter value (if present).
- **DefaultGraph = IRI .** IRI of the default graph to be used if no "default-graph-uri" parameter is specified.
- **MaxQueryCostEstimationTime = seconds .** This setting is used to limit the estimate time cost of the query to certain number of seconds, the default is no limit.
- **MaxQueryExecutionTime = seconds .** This setting is used to set the transaction execution timeout to certain limit in number of seconds, the default is no limit.
- **ImmutableGraphs = URI .** IRI of graphs over which the sponger not to be able able to write.
- **PingService = URI .** IRI of notification service to which the sponger results will be send.
- **DefaultQuery = SPARQL Query .** Default SPARQL Query.
- **DeferInferenceRulesInit = 1 .** Defer Loading of inference rules at start up.
- **ShortenLongURIs = 1 .** Shorten extremely long URIs in datasets when loading with the RDF Bulk Loader. Default is 0.

  Note : This parameter is only in the Virtuoso 06.03.3131+ commercial builds, at the time of writing it is not included in the open source 6.1.4 archives but will be in the next 6.1.5 release. A patch to enable this feature is however available from the Virtuso patches page on source forge , which can be applied to a 6.1.4 archive from source forge and the Virtuoso server binary rebuilt.
- **MaxMemInUse = 0 .** Maximum amount of memory that is allowed for temporary storing parts of a SPARQL query result. Default is zero for no limit. ResultSetMaxRows is maximum length of a SPARQL query result. Default is zero for no limit. These two options may be useful for protecting public web service endpoints against DOS attacks, but at the same time it may cause returning incomplete results without reporting errors. When used, it is strongly advised to set the value orders of magnitude larger than the expected size of longest reply. As a rule of thumb, timeout should happen before this limit has reached. Values less than 1000000 bytes are impractical in all cases.

The SPARQL INI can be get as RDF via http://cname/sparql?ini service.

[I18N]

- **XAnyNormalization = 1/2/3/0.** 0: default value. It means not to normalize anything, so, for example, "José" and "Jose" are two distinct words.

  
  See Also:
  How Can I Control the normalization of UNICODE3 accented chars in free-text index?

  1: Any pair of base char and combining char (NSM, non-spacing modifier) is replaced with a single combined char, so if character "é" is written as a sequence of "base" character "e" and a unicode char U+0301 ("combining acute accent"), then the pair will be replaced with a single U+00E9 ("latin small letter e acute").

  2: Any combined char is converted to its (smallest known) base, so, "é" will lose its accent and become plain old ASCII "e".

  3: This is equal to 1+2, and when set, both conversions are performed. As a result, any pair of base char and combining char loses its second char, and any chars with accents will lose accents.

  If the parameter is required at all, the needed value is probably 3, so the fragment of virtuoso.ini should be:

  ```
  [I18N]
  XAnyNormalization=3
  ```
- **WideFileNames = 1/0.** Default is 0. When 1 then file access and directory listing functions may use wide strings as file names. If a file name contains non-ASCII characters then directory listing result set will contain a wide string instead of narrow string with question marks. Note, however, that most of existing applications do not support wide strings for that purposes, so the feature should be used with extreme care.

- **VolumeEncoding.** Encoding identifier. If set, file names are translated from wide strings and default server encoding to the specified encoding before using them in file manipulation BIFs. This is useful when server's filesystem is in some national encoding or in UTF-8. The translation is bi-directional: directory listing items are translated from volume encoding to the default server encoding or wide.
- **VolumeEmergencyEncoding.** Encoding identifier. If set, this encoding is used when the use of VolumeEncoding causes encoding errors. The most popular case is when an UNIX filesystem tree with UTF-8 contains some mounts of legacy storages with KOI or CP encoding. Less popular is UTF-8 volume encoding and UTF-8QR as emergency encoding to recover poorly encoded filenames.
- **VolumeEmergencyEncodingDirs.** List of directories in same syntax as DirsAllowed, default is empty. If set, all file names in the listed directories are supposed to be in VolumeEmergencyEncoding and VolumeEncoding is even not tried. The most popular case is when an UNIX filesystem tree with UTF-8 contains some mounts of legacy storages with KOI or CP encoding.

[Replication]

The replication section sets the transactional replication parameters for the server.

- **ServerEnable=0/1.** A boolean parameter controlling whether a Virtuoso can or cannot act as a transactional replication publisher.
- **ServerName = log1.** This identifies the server instance. The entries in SYS_REPL_ACCOUNTS where SERVER equals this name are considered locally published accounts. This is the value returned by repl_this_server () SQL function.
- **QueueMax = 50000.** This controls how much synchronized transactional subscribers may fall behind before being disconnected. This controls how much memory the server will use to buffer undelivered replication casts. If the queue exceeds this byte amount subscribers are disconnected and must request re-synchronization. The byte count refers to the total length of the replay records being buffered. The actual memory usage is somewhat greater.

[Mono]

- **MONO_ROOT.** A path to the directory where the Mono system assemblies are located. Usually it is a compile time setting, but the MONO_ROOT overrides it. This ini setting overrides the Mono environment variable of the same name.
- **MONO_PATH.** A colon separated list of directories where the assemblies are located to be found by Assembly.Load (equivalent to MS.NET Global assembly cache). This ini setting overrides the Mono environment variable of the same name.
- **MONO_CFG_DIR.** A path where the 'machine.config' file is to be found while running the ASPX code in Mono. This ini setting overrides an environment variable of the same name. This is also usually a Mono compile time default.
- **virtclr.dll.** A fully qualified path and filename of the virtclr.dll virtuoso helper assembly.
- **MONO_TRACE = Off.** Mono debug tracing can be enabled by setting this parameter to On. When tracing is On, Mono debug output with be sent to the Virtuoso debug console.

[Client]

- **SQL_QUERY_TIMEOUT=0.** This sets the initial value of the SQL_QUERY_TIMEOUT statement option in connected clients. The ODBC standard value is 0, meaning indefinite, which is impractical in many applications. This allows overriding the default. The timeout is expressed in seconds. If the client application sets this option in a statement, this default is overridden for the statement in question.
- **SQL_TXN_TIMEOUT=0.** This is an ODBC extension option allowing setting a maximum duration for a transaction. 0 means that there is no maximum.
- **SQL_PREFETCH_ROWS=100.** For a forward only cursor, this option sets the number of rows prefetched at the execute and on subsequent fetch requests. A high value will speed up long selects but will be a disadvantage if only the first few rows are fetched from a cursor that has a large result set.

  This should not be confused with the SQL_ROWSET_SIZE setting for scrollable cursors.
- **SQL_PREFETCH_BYTES=16000.** This option specifies the maximum number of bytes the server will send as prefetched rows on a forward only cursor. If long rows are being prefetched this will cut off the prefetch after this many bytes even if the number of rows is less than SQL_ROWSET_SIZE.
- **SQL_NO_CHAR_C_ESCAPE=0.** This options is 0 by default and can be either 1 or 0. This option controls Virtuoso's interpretation of the backslash in PL text which is normal interpreted as escaping rather than literal.
- **SQL_UTF8_EXECS = 0.** Setting SQL_UTF8_EXECS = 1 enables UTF-8 identifier storage and retrieval, whereas setting SQL_UTF8_EXECS = 0 disables it. The default setting is 0: disabled for backwards compatible. See the Wide Character Identifiers section for more information

- **SQL_BINARY_TIMESTAMP = 1.** When SQL_BINARY_TIMESTAMP is set to 1 Virtuoso will describe all TIMESTAMP columns as SQL_BINARY. If it is set to 0 then Virtuoso will report the TIMESTAMP columns as SQL_TIMESTAMP
- **SQL_NO_SYSTEM_TABLES = 0.** This setting can be used to prevent SQLTables from returning system tables. The default value of this setting is 0, which is allow system tables to be returned in the normal way. Setting this to 1 will prevent system tables from being returned from SQLTables. The client can also issue a "set SQL_NO_SYSTEM_TABLE = 1/0" statement to set this in-connection.

[AutoRepair]

- **BadParentLinks=0.** As a result of an internal error in the database the physical integrity of references may be lost. Enabling this option causes the database to automatically repair such faults without having to go through a crash dump plus restore. A value of 0 should be normally used.

[VDB]

- **ArrayOptimization=0/1.** Boolean parameter which allows the Virtuoso VDB to use Array parameters if the remote data source supports these.
- **UseMTS = 0.** This parameter turns on/off MTS support in Virtuoso. It is applicable to windows multithreaded version of the Virtuoso server only.
- **NumArrayParameters.** Specifies a size of the parameter batch used by the VDB (default = 10)
- **VDBDisconnectTimeout.** The time (in seconds) after which a VDB connection is considered timed-out and closed. Default : 1000
- **VDBOracleCatalogFix=0/1.** This setting can be enabled to improve compatibility with the MS Oracle Driver which has problems with mixed case table names in catalog calls.

  Boolean parameter : allows a special mode for the ORACLE ODBC drivers which do not return correct catalog data for mixed case tables. When this is on and the return from the catalog functions is empty, then a Catalog function is issued to take the same catalog data and the result is filtered on the client side.
- **AttachInAutoCommit.** An boolean parameter controlling whether the VDB Catalog functions called while attaching a table will be called in AutoCommit mode (required by some Sybase drivers).
- **NumArrayParameters=10.**
- **ReconnectOnFailure=0 [1|0].** The default setting of 0 instructs the VDB layer to return underlying DB errors to the client rather than automatically reconnecting without reporting the error, which would be setting 1.
- **KeepConnectionOnFixedThread=1 [0|1|2|4|8].** The default option is 1, this forces the VDB to map a single thread to each ODBC session in the VDB rather than multiple threads. Some ODBC drivers expect that calls concerning a particular connection all take place on one thread. For example, the Oracle 8.xx ODBC driver can produce flaky crashes if this option is not set.

  0 - If a second request comes while a thread is processing the epilogue or previous request for same connection the second request is scheduled for the same thread. This is called burst mode. Once this mode is active all requests on this connection will be processed in the same thread. until there is a break at delay more than 'PrpcBurstTimeoutMsecs' elapses between ending the last and receiving the next in same connection. This break in activity return the thread to the thread pool. The burst mode optimization saves thread switching time and may improve performance by up to 30% in situations involving short requests immediately following each other.

  "ForcedFixedThread" mode (4). This is the same as VDBFixedThread mode, except that the thread enters this state immediately on connection (as opposed to doing that on the first VDB activity). Otherwise the scheduling stays the same as in VDB FixedThread.

  "Forced Burst" mode (8). When a connection is initially made it's added to the select thread for monitoring. When an RPC comes in it causes the select thread to take the connection out of the select thread and assign that connection to designated worker thread. That thread from now on will read the data coming in from that connection directly (not depending on the select thread to detect the incoming data and wake the worker thread up). That mode saves the thread switch done handling each RPC (from the select thread to the worker thread). When the connection terminates the thread is unbound from it and returned to it's default idle state. If an lengthy RPC is being processed the server will switch the thread from burst mode to default mode while running the RPC, so the asynchronous cancellation requests can be processed.

  To disable ever going to burst mode, this option can be set to 2
- **PrpcBurstTimeoutMsecs=100 (milliseconds).** RPC burst mode timeout in milliseconds. (see above)

- **SerializeConnect=0 [1|0].** When enabled causes Virtuoso to wrap SQLAllocConnect/SQLConnect calls sequence in a mutex, thus preventing abnormal program exits when there are a large number of connection requests to the VDB. Certain ODBC drivers have been noted to produce flaky errors when a large number of threads is concurrently inside one of the ODBC connect calls..
- **SkipDMLPrimaryKey=0 [1|0].** This setting controls SQL compilation (not execution) for conditions where rows in a local table are used to determine rows of a remote table to be deleted (e.g. delete from rmt1 where rmc1 = 12 and rmc2 in (select lc1 from lt1)) where the remote table was linked without a Primary Key primary key specified. Normally the primary key is used, even when not in the original select select, to identify the rows to be deleted. When the primary key is false or not available and SkipDMLPrimaryKey = 1 then the original where clause will be used instead.
- **RemotePKNotUnique=0 [1|0].** This option controls the SQL compiler ability to do some optimizations when it knows it will receive 1 row from a remote table (using the applicable WHERE clause). In some cases the remote is not providing any info about a primary key (or is providing wrong data for the primary keys and unique indices of a table) through ODBC API and the Virtuoso SQL optimizer may get fooled into thinking that a given SQL query over a remote table will return no more than 1 row. To avoid that turn on the above parameter to stop the compiler from doing that optimizations.
- **UseGlobalPool=0 [1|0].** This option controls the aggregation point of the VDB connection pools. When it is off (=0, the default) the VDB connections that are no longer needed after transaction end are collected server-side in a per-user connection cache. This (together with the `KeepConnectionOnFixedThread` INI option) ensures conformance even with older ODBC drivers that require that connections must be used only within the OS thread that initiated them. This however may result in somewhat low reuse rate for the pooled connections. That's why when this option is 1 (On) the connections are stored in a server-side per-DSN connection cache. Keep in mind that this may not work with all the ODBC drivers, but it may provide lower ratio between virtuoso client connections and VDB-to-remote DBMS connections, thus requiring smaller number of active connections on the remotes and faster execution of VDB operations.

[Ucms]

- **UcmPath.** String parameter which specifies the path where UCM files are located. If this parameter is not specified, UCM files cannot be loaded.
- **Ucm1, Ucm2,... Ucm99.** Every UcmN parameter specifies one UCM file to load. The value of UcmN is a pair of comma delimited strings. The first string is a name of the UCM file to load, (relative to the path specified in UcmPath), the second string is a name of the encoding as it was used by the server. E.g. Ucm1 = java-Cp933-1.3-P.ucm,Cp933 will load the encoding from the file java-Cp933-1.3-P.ucm and associate it with the identifier 'Cp933'. You can register one encoding with more than one name, if they are delimited by '|' (with no white spaces in the string). e.g. Ucm2 = java-Cp949-1.3-P.ucm,Cp949|Korean will load the encoding from the file java-Cp949-1.3-P.ucm and associate it with two identifiers, 'Cp949' and 'Korean'. See UCM Encodings for more details.

[Zero Config]

- **ServerName.** Name used to advertise the Virtuoso ODBC service details in ZeroConfig. This is the name that will be shown to clients amongst other ZeroConfig datasources.
- **ServerDSN.** An ODBC style connect string to preset the values of the parameters when the ODBC service offered by this server is selected by the Virtuoso ZeroConfig enabled clients.
- **SSLServerName.** Name used to advertise the Virtuoso ODBC SSL encrypted service details in ZeroConfig.
- **SSLServerDSN.** An ODBC style connect string to preset the values of the parameters when the ODBC SSL encrypted service offered by this server is selected by the Virtuoso ZeroConfig enabled clients.

See Also:

The Zero Config section.

[Plugins]

- **LoadPath = /home/virtuoso/hosting.** The directory containing shared objects/libraries for use as Virtuoso VSEI plugins.
- **Load<number> = <module type>, <module name>.** `<number>` is the module load number, required and starting with 1. `<module type>` specifies the type of module that is to be loaded, and hence how Virtuoso is to use it. So far "Hosting" and "attach" types exist. `<module name>` is the file name of the modules shared library or object.

  Example:

```
Load6 = attach, libphp5.so)

Load7 = Hosting, hosting_php.so)
```

"Attach" is used for now for the php library. It can be used to load other libraries in future too. The reason is to load PHP5 functionality into virtuoso namespace, so when actually is loaded the hosting plugin, it can bind to the already available symbols for php5.

See Also:
VSEI Plugins

[Striping]

- **Segment<number> = <size>, <stripe file name> [, <stripe file name> ..].** <number> must be ordered from 1 upwards; The <size> is the size of the segment which is equally divided across all stripes comprising the segment.

This section is only effective when `Striping = 1` is set in the `[Database]` section. When striping is enabled the Virtuoso database spans multiple segments where each segment can have multiple stripes.

Striping can be used to distribute the database across multiple locations of a file system for performance. Segmentation can be used for expansion or dealing with file size limitations. To allow for database growth when a file system is near capacity or near the OS file size limitation, new segments can be added on the same or other file systems.

Striping only has a potential performance benefit when striped across multiple devices. Striping on the same file system is needless and unlikely to alter performance, however, multiple segments do provide convenience and flexibility as described above. Striping across partitions on the same device is likely to reduce performance by causing high unnecessary seek times on the physical disk.

Database segments are pre-allocated as defined. This can reduce the potential for file fragmentation which could also provide some performance benefit.

Virtuoso striping alone does not allow for any fault tolerance. This is best handled at the I/O layer or by the operating system. File system RAID with fault-tolerant striping defined should be used to host the Virtuoso files if striping based protection is desired.

The segments are numbered, their segment <number> must be specified in order starting with segment1.

The <size> is the total size of the segment which is that will be divided equally across all stripes comprising the segment. Its specification can be in gigabytes (g), megabytes (m), kilobytes (k) or in database blocks (b) the default.

Note:
The segment size must be a multiple of the database page size which is currently 8k. Also, the segment size must be divisible by the number of stripe files contained in the segment.

Segments can be added to a database, however once defined they should never be altered. Databases that were created without striping cannot automatically be restarted with striping. You can convert a non-striping database to striping by dumping the contents of the database to a transaction file and replaying it with striping enabled. An on-line backup made with backup_online () can be restored on a database with a different striping configuration as long as the total number of pages is no less than the number of pages of the backed up database.

Striping can be useful for the temporary objects database if large hash join temporary spaces or such are expected. This is enabled by the Striping setting in the Temp Database section of the ini file. The stripes will be declared in the TempStriping section.

See Also:
Rebuilding A Database in the Backup section.

Sample Configuration File ("virtuoso.ini")

Following is the text of the sample virtuoso.ini file that comes with the distribution.

```
;
;  virtuoso.ini
;
;  Configuration file for the OpenLink Virtuoso VDBMS Server
;
;
;  Database setup
;
[Database]
DatabaseFile            = virtuoso.db
```

```
TransactionFile         = virtuoso.trx
ErrorLogFile            = virtuoso.log
ErrorLogLevel           = 7
FileExtend              = 200
Striping                = 0
; LogSegments           = 1
; crashdump_start_dp    = 0
; crashdump_end_dp      = 0
; Log1                  = log1.trx
;
;  Server parameters
;
[Parameters]
ServerPort              = 1111
ServerThreads           = 10
CheckpointInterval      = 60
CheckpointSync          = 2
NumberOfBuffers         = 2000
MaxDirtyBuffers         = 1200
MaxCheckpointRemap      = 2000
PrefixResultNames       = 0
CaseMode                = 1
;MinAutoCheckpointSize  = 4000000
AutoCheckpointLogSize   = 40000000
CheckpointAuditTrail            = 1

[HTTPServer]
ServerPort = 1122
ServerRoot = ../vsp
ServerThreads = 2
MaxKeepAlives = 10
KeepAliveTimeout = 10
MaxCachedProxyConnections = 10
ProxyConnectionCacheTimeout = 15
DavRoot = DAV

[Replication]
ServerName = log1
Server = 1
QueueMax = 50000

[Client]
SQL_QUERY_TIMEOUT       = 0
SQL_TXN_TIMEOUT           = 0
SQL_PREFETCH_ROWS               = 100
SQL_PREFETCH_BYTES      = 16000

[AutoRepair]
BadParentLinks      = 0
BadDTP                          = 0

;[Ucms]
;UcmPath = /ucm
;Ucm1 = java-Cp933-1.3-P.ucm,Cp933
;Ucm2 = java-Cp949-1.3-P.ucm,Cp949|Korean
;Ucm3 = java-ISO2022KR-1.3-P.ucm,ISO2022KR|ISO2022-KR
;
;  Striping setup
;
;  These parameters have only effect when Striping is set to 1 in the
;  [Database] section, in which case the DatabaseFile parameter is ignored.
;
;  With striping, the database spans multiple segments
;  where each segment can have multiple stripes.
;
;  Format of the lines below:
;    Segment<number> = <size>, <stripe file name> [, <stripe file name> .. ]
;
;  <number> must be ordered from 1 up.
;
;  The <size> is the total size of the segment which is equally divided
;  across all stripes comprising  the segment. Its specification can be in
;  gigabytes (g), megabytes (m), kilobytes (k) or in database blocks
```

```
;  (b, the default)
;
;  Note that the segment size must be a multiple of the database page size
;  which is currently 8k. Also, the segment size must be divisible by the
;  number of stripe files constituting the segment.
;
;  The example below creates a 200 meg database striped on two segments
;  with two stripes of 50 meg and one of 100 meg.
;
;  You can always add more segments to the configuration, but once
;  added, do not change the setup.
;
[Striping]
Segment1        = 100M, db-seg1-1.db, db-seg1-2.db
Segment2        = 100M, db-seg2-1.db
```

Configuring Vectored Execution

Note: Only effective with Virtuoso 7.0 and later.

A Virtuoso 7 executable executes all SQL statements in vectored mode, except for positioned updates and deletes in stored procedures. Procedures are executed without vectoring unless they are declared vectored using the vectored keyword. A scalar (non-vectored) procedure can have a vectored block introduced with the for vectored construct. These are discussed in the SQL reference section on vectoring.

Vectored execution is controlled by the following virtuoso.ini settings:

- *VectorSize = 10000*

  : This is the default number of concurrent variable bindings that are generated for a column in a batch of bindings.
- *MaxQueryMem = 1G*

  : This controls the maximum amount of memory that can be used across the server process for large vectors, i.e. if the memory in use is near this limit, a query will not switch to large vector size even if it finds it useful. The event counter tc_no_mem_for_longer_batch counts how many times this situation is detected. A size letter of G or M follows the value.
- *AdjustVectorSize = 1*

  : If non-zero, this enables automatic increasing of vector size whenever the system notices a random access pattern that is not benefiting from vectoring due to too few consecutive hits falling on the same page. The vector size can be increased up to MaxVectorSize if the situation warrants. This is the case if there is enough unprocessed state in the query.
- *MaxVectorSize = 1000000*

  : This is the maximum vector size. This can reach up to 4000000 but values in excess of 1000000 have not been found useful in practice. If the server is running out of memory with multiuser workloads involving long queries, dropping the MaxVectorSize to a lower value is a means of curtailing per query memory consumption.
- *ThreadsPerQuery = 8*

  : This controls the maximum number of parallelizable work units a query will have outstanding at any one time. A value of 8 means that a scan or vector of lookups is maximally divided into 8 units, of which 7 will be allocated to a pool of worker threads and one will be processed by the thread coordinating the query. Each query will always have one thread running, which is the thread allocated for serving the client-server or HTTP request initiating the query. Extra work units are serviced by threads from a parallel execution thread pool. If there are unstarted parallel work units that have not started at the time the coordinated thread finishes its own work unit, it will execute any of its own pending work units locally.
- *AsyncQueueMaxThreads = 16*

  : This is the maximum number of worker threads that can be started for running parallelized work units generated by queries or functions requested with the aq_request () PL function. Thus the number of threads running on a server at any time is the number of client server or HTTP threads that have a running request plus this number. These threads consider the totality of all async execution requests across all async queues and each thread picks a unit of work from the oldest queue that has unstarted work units. The age of the queue is the timestamp of its creation. A queue is created whenever a query operator decides to split its work. This scheduling favors thus old queries over new ones. However any request is guaranteed one thread, that on which its initiating request is served.

For a query processing workload, one should set AsyncQueueMaxThreads to the number of cores minus the expected number of concurrent queries, this would in principle use all cores or core threads. In practice for machines with large numbers of cores, e.g. 32 cores, a lower value may serve in practice better, subject to experimentation. For a web crawling situation where threads are waiting for the network for most of the time, a larger number will do better.

These parameters can be set at run time by a dba group user using the `__dbf_set()` function.

Index Defragmentation

When data is inserted into tables, database pages are split and typically pages end up not being fully utilized. If data is inserted in ascending order of key value, which is often the case, space utilization is more efficient. Still, gaps can be left by updates, deletions and page splitting. Virtuoso has an autocompact feature which will take groups of adjacent dirty pages and see if it can fit the same content on a smaller number of pages. It will rewrite the pages to save space before writing these to disk. This operates automatically. Statistics on autocompaction are shown beside the Read ahead status line of the result set of status ('');. The number of pages affected by autocompaction and the number of resulting pages are shown. The numbers are cumulative since the start of the instance.

Autocompact is non-locking and if pages are busy, they will not be touched. Only relatively old dirty pages, about to be written to disk are considered for compaction, not interfering with the hottest part of the working set.

The automatic compaction is not however effective if pages are updated singly, never making stretches of consecutive dirty pages. Therefore a manual compaction function called DB.DBA.VACUUM () is also offered.

The vacuum stored procedure gets an optional table and index name and will read the index from beginning to end. If neither argument is given, all indices of all tables will be compacted. If only the table is given, all indices of this table will be compacted.

If consecutive leaves can be fit on fewer pages than they now occupy, this will rewrite them and free the pages left over. This does however require transient space since the pages are not really replaced until the next checkpoint, hence a vacuum operation can run out of disk. Using the checkpoint statement to force a checkpoint will free the space.

The effects and need for explicitly vacuuming a database can be assessed with the DB.DBA.SYS_INDEX_SPACE_STATS view.

Running:

```
select * from DB..SYS_INDEX_SPACE_STATS order by ISS_PAGES desc;
```

produces a result set with the most space consuming index on top. ISS_PAGES is the total count of pages. ISS_ROW_BYTES is the byte count of the rows. If dividing the total count of bytes by the count of pages is much below 8172, (8K - 20), chances are that vacuuming the index may save space. Note that blobs are not affected by vacuuming. If the blobs are small enough to fit on the row as normal strings they are already there. Otherwise they occupy the needed number of pages and cannot be made more compact.

Note that querying the SYS_INDEX_SPACE_STATS view will always read through all the allocated pages of the database and may take a while. The operation is not locking. Only the state as of the last checkpoint will be shown, hence it is a good idea to run the checkpoint statement before querying this view.

Examples:

```
DB..VACUUM ();
-- Compact all tables and indices of the Virtuoso instance

Db..VACUUM ('WS.%');
-- compact all tables of the WS. qualifier

DB..VACUUM ('DB.DBA.RDF_QUAD', 'RDF_QUAD_PGOS');
-- compact the rdf_quad_pgos index of the rdf_quad table.
```

Virtuoso has an autocompact feature.

Server Startup Command Line Options

Virtuoso Server

This section presents the command line switches of the Virtuoso server executable. Depending on the model and virtual database middleware the server will have different names, all starting with virtuoso-. All these however have the same options for UNIX systems and slightly different for Windows platform.

on Windows platform are available following server command line options:

```
Usage:
  virtuoso-odbc-t.exe [-clnCbDARf---dSIMKmrd] [+configfile arg] [+licensefile arg]
                  [+no-checkpoint] [+checkpoint-only] [+backup-dump]
                  [+crash-dump] [+crash-dump-data-ini arg]
                  [+restore-crash-dump] [+foreground] [+pwdold arg]
                  [+pwddba arg] [+pwddav arg] [+debug] [+service arg]
                  [+instance arg] [+mode arg] [+dumpkeys arg] [+manual]
                  [+restore-backup arg] [+debug]
  +configfile             specify an alternate configuration file to use,
                         or a directory where virtuoso.ini can be found
  +licensefile            specify an alternate license file to use,
                         or a directory where virtuoso.ini can be found
  +no-checkpoint          do not checkpoint on startup
  +checkpoint-only        exit as soon as checkpoint on startup is complete
  +backup-dump            dump database into the transaction log, then exit
  +crash-dump             dump inconsistent database into the transaction log,
                         then exit
  +crash-dump-data-ini    specify the DB ini to use for reading the data to dump
  +restore-crash-dump     restore from a crash-dump
  +foreground             run in the foreground
  +pwdold                 Old DBA password
  +pwddba                 New DBA password
  +pwddav                 New DAV password
  +debug                  allocate a debugging console
  +service                specify a service action to perform
  +instance               specify a service instance to start/stop/create/delete
  +mode                   specify mode options for server startup (onbalr)
  +dumpkeys               specify key id(s) to dump on crash dump (default : all)
  +manual                 specify when create a service to make it for manual startup
  +restore-backup         restore from online backup
  +debug                  Show additional debugging info

The argument to the +service option can be one of the following options
  start          start a service instance
  stop           stop a service instance
  create         create a service instance
  screate        create a service instance without deleting the existing one
  delete         delete a service instance
  list           list all service instances
```

The below are switches for server for UNIX platforms:

```
Usage:
  virtuoso-iodbc-t [-fclnCbDARwMKr---d] [+foreground] [+configfile arg]
                   [+licensefile arg] [+no-checkpoint] [+checkpoint-only]
                   [+backup-dump] [+crash-dump] [+crash-dump-data-ini arg]
                   [+restore-crash-dump] [+wait] [+mode arg] [+dumpkeys arg]
                   [+restore-backup arg] [+pwdold arg] [+pwddba arg]
                   [+pwddav arg] [+debug]
  +foreground              run in the foreground
  +configfile              use alternate configuration file
  +licensefile             use alternate license file
  +no-checkpoint           do not checkpoint on startup
  +checkpoint-only         exit as soon as checkpoint on startup is complete
  +backup-dump             dump database into the transaction log, then exit
  +crash-dump              dump inconsistent database into the transaction log, then exit
  +log6                    If you're starting with a Virtuoso 5 server, and migrating to a Virtuoso 6 or
                           later server, append +log6 after +backup-dump or +crash-dump.
                           Note: The extra +log6 argument is not needed, and may have unexpected effect,
                           so please leave it off if starting with a Virtuoso 6 server and moving to
                           Virtuoso 6 or later.
```

```
+crash-dump-data-ini   specify the DB ini to use for reading the data to dump
+restore-crash-dump    restore from a crash-dump
+wait                  wait for background initialization to complete
+mode                  specify mode options for server startup (onbalr)
+dumpkeys              specify key id(s) to dump on crash dump (default : all)
+restore-backup        restore from online backup
+pwdold                Old DBA password
+pwddba                New DBA password
+pwddav                New DAV password
+debug                 Show additional debugging info
```

The +*crash-dump* option will make use of the segmented log defined in virtuoso.ini for storing the recovery log. See Crash Recovery and virtuoso.ini below for more information. The other options will not use the segmented log.

The +*restore-crash-dump* option will alter the server startup sequence so that the recovery log produced by +crash-dump will be re-played correctly.

The +*mode* option can be a combination of the following letters:

o - only open the database and define the SYS_KEYS, SYS_COLS, SYS_KEY_PARTS, SYS_CHARSETS AND SYS_COLLATIONS system tables.

n - leaves out the initialization of the system tables.

b - do not process anything except the transaction log and system tables when restoring a crash dump (+restore-crash-dump).

a - leaves out the initialization of the replication, users, compilation or stored and system procedures, as well as the caching of the grants.

l - write only the schema tables in the backup or crash dump.

r - don't do the complete initialization (useful for performing a crash dump).

On Unix platforms the executable will detach itself from the console and run in the background as a daemon unless the +*foreground* switch is specified.

For Windows NT and Windows 2000, the Virtuoso server will normally be installed as a Windows service and can be started from the Control Panel or automatically at system startup.

Ordinarily the Windows service will be a system process that runs in the background. If you want the Virtuoso service on Windows to allocate a debugging console the you can use the +*debug* (-d) switch. This switch is only applicable to starting a service.

Virtuoso on Windows can be run directly from the command line using the +*foreground* (-f) switch. The server will then start in the foreground of the current "cmd" session. If this switch is not used then the executable on Windows will assume that you are attempting to start a Virtuoso service.

Windows services can be created and removed from the system as required. The default installation under Windows will create a service by the name: *OpenLink Virtuoso VDBMS Server* , and optionally another service with the name *OpenLink Virtuoso VDBMS Server [demo]* . The Demo service is a supplied demonstration database that can be installed.

The following options are available to the +*service* switch for configuring Virtuoso services:

The argument to the +service option can be one of the following options *start* start a service instance

stop stop a service instance

create create a service instance

screate create a service instance without deleting the existing one

delete delete a service instance

list list all service instances

They are used with the +*instance* <*name*> where <*name*> is the instance name to configure a particular instance. All instances are listed in the services applet, with their name in square brackets.

+*service list* can be used to obtain the list if services that are registered with Windows.

For each service listed you can start, stop, or delete the service.

+service create can be used to create a new service. In this case you also need to specify other start up options that would be associated with the new service entry. If you were using an alternative configuration file this must be specified using *+configfile* switch.

Note:

Make sure the Services Control Panel is closed, before attempting to modify services from the command line, otherwise locking may occur.

ZeroConfig ("Zero Configuration") Support

The "ZeroConfig" protocol, also known as "Zeroconf" or "Zero Configuration" is a protocol that allows discovery of services on the network that are advertised by their hosts. It also has provisions for automatic discovery of computers and various devices. The main benefit of ZeroConfig is that it does not require DHCP, DNS or directory servers.

ZeroConfig is an open protocol that Apple submitted to the IETF for a standard-creation process.

The Virtuoso server and ODBC driver use the capabilities of ZeroConfig to facilitate DSN (Data Source Name) setup and usage. This is divided in two parts: Server-side and Client-side.

The Virtuoso server (Server-side) is configured via the Virtuoso INI file to advertise its availability on a network with a given name. This allows applications, and in particular the Virtuoso ODBC driver, to receive information about a server, such as its network address, default login, etc, and use it for configuring a data source or directory making a connection.

The Virtuoso ODBC driver (Client side) uses ZeroConfig to locate the desired Virtuoso server during the set-up phase of a data source, and determine available connection options such as:

secure connection options

default database

default user

default password (if public/demo login is required)

default character set

ZeroConfig provides the client with a service name, which must be bound to the IP address/port of a host of the chosen service during DSN configuration. This is used when existing DSN using a ZeroConfig name is used to connect, it will map name with IP address and port before making a connection.

Setting-up the Server for Service Advertising

The Virtuoso server is configured to advertise itself based on the details specified in the `[Zero Config]` section of the Virtuoso INI file. Below is an example of such:

```
...
[Zero Config]
ServerName    = Virtuoso Server
ServerDSN     = UID=demo;PWD=
SSLServerName = Virtuoso Server (via SSL)
SSLServerDSN  = UID=dba;PWD=;ENCRYPT=1
...
```

The ServerName and SSLServerName are human readable strings chosen by the administrator to provide clients with a suitable description of the service being provided.

Note:

If the Virtuoso does not have the SSL listener enabled then the SSL service will not be advertised automatically. The SSL* keys will simply be ignored and do not need to be removed.

The ServerDSN and SSLServerDSN are default connection strings that can be used by clients to make the advertised connection. You only need to to specify default username and password in these strings. The default database can be specified or left to the setting for the username. You cannot specify the server hostname, IP address or port number, these are supplied by Virtuoso automatically.

ZeroConfig service advertising is multicast, hence it is advertised on all available network interfaces.

Using the Windows ODBC Driver with ZeroConfig

Upon DSN set-up the ODBC driver listens for advertising servers, and compiles a roster. This is displayed for the user to choose the desired service to connect to.

If ZeroConfig is used to for data source set-up then the set-up dialog will be initialized based on the details in the connection string configured on the server.

When a DSN is configured based on a ZeroConfig service, the driver will resolve the service name before making the connection to the server. The driver does not store the network address or port number of the Virtuoso server, only the ZeroConfig server name, so if the server's physical address is changed the client DSNs associated with it do not all have to reconfigured; they will resolve to the new address automatically on next use.

Server Status Monitoring

The database status report is divided into 6 sections:

Server
Database
Locks
Clients
Replication
Index Usage

Server

This section shows how many connections are open and how many threads the process has and how many are running at the present time. This also displays the number of requests that have been received but are not yet running on any thread.

Database

```
  File size 203161600, 24800 pages, 259 free.
  7000 buffers, 6987 used, 3884 dirty 8 wired down, repl age 8251 .
  Disk Usage: 14246 reads avg 6 msec, 74% read last  10 s, 14457 writes,
    4 read ahead, batch = 5.
Gate:  2729 2nd in reads, 0 gate write waits, 3372547 in while read 0 busy scrap.
Log = wi.log, 9851835 bytes
14950 pages have been changed since last backup (in checkpoint state)
Current backup timestamp: 0x0000-0x00-0x00
Last backup date: unknown
Clients: 18 connects, max 17 concurrent, 1024 licensed
RPC: 54441 calls, 17 pending, 17 max until now, 0 queued, 53988 burst reads (99%), 0 second
Checkpoint Remap 7646 pages, 0 mapped back. 0 s atomic time.
    DB master 24800 total 259 free 7646 remap 3415 mapped back
   temp  200 total 196 free
```

The status consists of the following items:

File size: The database file size in bytes or 0 if the database consists of statically allocated files. The total number of 8K database pages follows, then the number of free pages. The number of buffers shown the total count of 8K file cache buffers, followed by the number of used buffers and the number of buffers that are dirty at the time. The wired down count is normally zero but can be transiently other if pages are wired down for processing by threads in the server.

Disk Usage: Shows the cumulative total number of reads and writes and the average length of time spent inside the read system call for the database files.

The percentage is the percentage of the real time spent inside read between this status report and the previous status report. This may exceed 100% if several reads are taking place concurrently on different stripes in a multi-file database.

The Gate: Lists concurrent events. The *2nd in read* is the count of concurrent requests for the same page, the *gate write waits* is the count of times a modify operation had to wait for exclusive access to a page being read by another thread, the *in while read* is the count of file cache hits that have taken place while a read system call was in progress on another thread.

Databases. Thus section shows the count of pages, free pages, checkpoint remap and mapped back for the main database and the space for temporary data such as sort results and hash indices. The page count is the total size, the free count is the count of free pages, the checkpoint remap is the count of pages that occupy two pages in checkpoint space instead of one, the mapped back count is the number of pages that will return to their original place in checkpoint space at the next checkpoint. Understanding these is not necessary.

See:
The Disk Configuration section for a discussion of checkpoint remapping.

Locks

The lock section shows various locking statistics accumulated since the server was started. The deadlock count is divided into deadlocks caused by a situation where several transactions read a page and one wants to get write access and all other deadlock situations. The first is called 2r1w deadlock in the report.

The lock section also shows the total number of threads running, i.e. engaged in performing some operation for a SQL or web client. The number of threads waiting is the number of running threads that are presently waiting for a lock. The number of threads in vdb is the number of threads engaged in remote database operations or other 'slow' I/O, such as access to outside HTTP or SOAP services.

All locks currently in effect are listed with the owners (a) and possibly waiting transactions. The transactions are named after their client. A log or replication replay transaction is here named 'INTERNAL'.

Clients

Each connected client is listed with the number of bytes sent and received from the client. The transaction status is either PENDING for OK or BLOWN OFF or DELTA ROLLED BACK for a transaction killed by deadlock or timeout. The locks owned by the transaction are listed following the status. IE means exclusive and IS shared lock.

Replication

This section shows the server in question and a list of replication accounts either provided or received by this server. Accounts where the server name (left column) is the same as the server name are those provided by this.

The columns are server name, account name, last transaction number and status. The status is OFF for a local account or a replicated account where the remote is not available. It is SYNCING if a resync is in progress, IN SYNC if the account is up to date or REMOTE DISCONNECTED if there was a connection to a remote party which subsequently disconnected.

See Also:
RDF Graph Replication

Index Usage

This part of the report summarizes the database's access statistics. The output is a table with a row for each index in the database. Each row is composed of the following columns:

```
Table           The name of the table
Index           The name of the index. Same as the table for primary key.
Touches         The number of touches since startup. Each time the database
                engine looks at an entry of the index is counted as a touch.
                Not all touched entries are selected. For instance if the
                engine scans a table with non-indexed selection criteria it
                will touch each row but might select none.
Reads           The number of disk reads caused by reading this index.
%Miss           The percentage of touches that required a read. This can be
                over 100% since getting one entry may required more than one
                read if the top levels of the index are not in memory.
Locks           The number of times a lock is set on an entry of the index.
Waits           The number of times the engine has to wait for another
                transaction to finish in order to set a lock on this index.
%W              The percentage of waits of all locks set.
```

In interactive SQL

```
SQL> status();
```

Will print out the report.

Re-labeling Server Executable on Win32 Platforms

The Virtuoso Service name can be altered using the key *Win32ServiceName* in the *Parameters* section. The default name is 'OpenLink Virtuoso VDBMS Server'

To change the name of services:

stop the service
delete the service
change the name in file virtuoso.ini
create the service
start the service

Services with old names must be deleted before creating service with the new name, i.e. with the Win32ServiceName setting set to the current name of the service.

The name displayed in the ODBC Administrator, Setup and Configuration dialogs is taken from the driver section in the ODBCINST. This can be directly edited in the registry using the regedt32 Windows utility, or a registry import file can be created which can be applied by simply double-clicking the .reg file. Always exercise extreme caution when making changes to the registry.

Transport Level Security

Encryption

Virtuoso has the ability to encrypt it's CLI network connections using SSL. The server listens on a separate port for SSL CLI connections and handles them just as the normal CLI connections, vut now providing transport level security.

Server-side Support

Server side secure connections utilizes three parameters in the [Parameters] section of the virtuoso.ini:

SSLServerPort - Specifies the port on which the server listens for incoming SSL CLI requests.
SSLCertificate - The SSL certificate to use (same meaning as the SSLCertificate in HTTPServer section)
SSLPrivateKey - The server's private key (same meaning as the SSLCertificate in HTTPServer section)

These parameters should be all set in order to enable the SSL CLI server.

If SSLServerPort is not specified, then the Virtuoso server ignores the other two and does not listen for SSL CLI connections.

If a non-SSL connection is attempted to the SSL server port, the server rejects the connection. If an SSL connection is attempted against the non-SSL port the server rejects the connection.

Client-side Support

The client does not require any SSL-specific files (like Certificates or Private keys) in the SSL connection process.

Native Clients (e.g. ISQL). There is an custom ODBC connect option SQL_ENCRYPT_CONNECTION (=5004) supported by the Virtuoso CLI. It should be set before issuing the SQLConnect call. Values are 'NULL' (no encryption - default), '1' (encryption with no server X509 certificate checking and no X509 certificate sent to the server) and a valid file path to a PKCS#12 certificate file (protected with the same password as the one used to log in. Note that with the iODBC/ODBC clients this connect option is not applicable since the driver managers don't cache or pass through the custom ConnectOptions set before connecting to the data source. The ISQL has an additional option (-E) to do encrypted connects using the encryption option '1') and -X <file> to set the above option to the file supplied.

ODBC & iODBC Driver. The drivers support an additional DSN attribute:

```
ENCRYPT=<string>
```

If this attribute is not specified then it defaults to "No".

It has the same meaning as the SQL_ENCRYPT_CONNECTION options (see above). If this is not specified then it defaults to NULL.

The corresponding iODBC odbc.ini & ODBC Registry DSN attribute name is:

```
"Encrypt"= <string>
```

The Windows Connect & Setup dialogs have an additional wizard page to configure encryption.

X509 Certificate Support

Virtuoso supports X509 certificate validation: server side for both ODBC and HTTPS connections, and client side for the ODBC connections.

Server Side. To enable this option there are three new INI file parameters added the HTTPServer section for the HTTPS, and the Parameters section for ODBC):

X509ClientVerify - Whether the server will require X509 certificates from the browsers. If set to 1 a client should send a X509 certificate which will be validated against the server CA list.

X509ClientVerifyDepth - Specifies how deep the client certificate verification process will traverse the Issuer chain before giving up. The default value of 0 means that only verification of the client certificate itself being in the server's CA list is performed. Setting this option to -1 will ignore the depth checks.

X509ClientVerifyCAFile - a PEM (base64) file of all the X509 certificates of the Certification Authorities (CA) which the server will use for verifying the client's client certificate. A list of these will be sent to the client as a part of the SSL handshake so the client will know which certificate to send.

ODBC Client Side. In order for verification of the server certificate to take place a PKCS#12 file should be supplied to the ODBC client. It will use the CA list in this PKCS#12 to verify the server certificate. It will set the verification depth to -1 (unlimited) while performing such a check.

If the server certificate is not verified correctly it will refuse to connect to the server. When the ENCRYPT parameter is set to "1" (do SSL without X509 validation) the client will return a SQL_SUCCESS_WITH_INFO in SQLConnect/SQLDriverConnect with the Server's certificate subject and the verification result as the server will always send it's X509 certificate to the client as a part of the SSL connect handshake.

If the PKCS#12 file is supplied the ODBC client will try to open it using the login password. In order for the file to be successfully opened it should be encrypted with the same password used for logging in.

Normally when exporting a PKCS#12 file from other programs it will contain only the CAs of the Certificate validation chain. This means that client and server certificates should have common CA in their certificate chains in order to be used for ODBC X509 validation. The client certificate from the PKCS#12 file will not take place in the server certificate validation process.

See Also:

`get_certificate_info()`

File System Access Control Lists

Access Control Lists (ACL) are used to restrict file system access.

These lists are maintained in the Virtuoso INI file under the Parameters section with entries such as:

```
DirsAllowed = <path> [, <path>]
DirsDenied = <path> [, <path>]

<path> := <absolute_path> or <relative_path>
```

Note:

A relative path is relative to the servers current working directory.

The Virtuoso ISQL utility can be used to check the Server DirsAllowed params as follows:

```
SQL> select server_root (), virtuoso_ini_path ();
```

The above should show in the result the server working directory and INI file name.

Also you can check the relevant INI setting by running following statement via ISQL command line utility:

```
SQL> select cfg_item_value (virtuoso_ini_path (), 'Parameters',
'DirsAllowed');
```

See Also:
Virtuoso INI File Configuration

ACL's work in the following way:

All paths are converted from relative to absolute paths.

The path beginning with <http_root> is always allowed.

All DB files are always access denied (.db, db segments, .trx, log segments, .ini specified in INI file etc.)

If a path is not allowed or exists as denied then access to the file is rejected.

If a requested path is allowed and not in denied then access is allowed.

ACL's are inherited. If a directory allows access so does its subdirectories.

The following functions are restricted by file Access Control Lists (ACL) in the virtuoso.ini file:

file_to_string

file_to_string_output

sys_mkdir

sys_dirlist

string_to_file

cfg_write

Note:
the cfg_write function has restrictions against changing file access control lists in ini file

6.1.2. Virtual Database

Linking Remote Tables & Views

The Virtuoso Server supports linking in of tables, views, procedures and other components from a remote ODBC data-source, enabling them to be queried as native objects in Virtuoso; this process is called "attaching" or "linking". The easiest way to link to an external table is to use the Linking Remote Tables Wizard , part of the Visual Server Administration Interface. Alternatively you can attach these objects programmatically, as this section explains; finally you can attach tables manually - see Manually Setting Up A Remote Data Source which is useful for connections to less-capable ODBC data-sources.

ATTACH TABLE Statement

```
ATTACH TABLE <table> [PRIMARY KEY '(' <column> [, ...] ')']
  [AS <local_name>] FROM <dsn> [USER <uid> PASSWORD <pwd>]
  [ON SELECT] [REMOTE AS <literal_table_name>]
```

table
: Adequately qualified table name of the form: identifier | identifier.identifier | identifier.identifier.identifier | identifier..identifier

column
: column to assume primary key

local_name
: fully qualified table name specifying local reference.

dsn
: scalar_exp

user
: scalar_exp

password
: scalar_exp

literal_table_name
: scalar_exp

This SQL statement defines a remote data source, copies the schema information of a given table to the local database and defines the table as a remote table residing on the data source in question.

The table is a designation of the table's name on the remote data source dsn. It may consist of an optional qualifier, optional owner and table names, separated by dots. This must identify exactly one table on the remote dsn. The optional local_name is an optionally qualified table name which will identify the table on the local database. If qualifier or owner are omitted, these default to the current qualifier 'dbname()' and the logged in user, as with CREATE TABLE. If the local_name is not given it defaults to the <current qualifier>.<dsn>.<table name on dsn>. The <dsn> will be the dsn with all alphabetic characters in upper case and all non-alphanumeric characters replaced by underscores. The <table name on dsn> will be the exact name as it is on the remote dsn, case unmodified.

The PRIMARY KEY option is only required for attaching views or tables where the primary key on the remote table cannot be ascertained directly from the remote data source.

If a dsn is not previously defined with vd_remote_data_source or ATTACH TABLE, the USER and PASSWORD clauses have to be given.

The *REMOTE AS* option allows you to provide a string literal for referencing the remote table. This is useful when linking tables from sources that do not support three-part qualification correctly.

Attaching views

A view can be attached as a table if a set of uniquely identifying columns is supplied.

This is done with the PRIMARY KEY option to ATTACH TABLE as follows:

```
attach table T1_VIEW primary key (ROW_NO) from 'somedsn';
```

Note:

Views cannot be attached unless the PRIMARY KEY options is used.

Examples for Linking Remote Tables into OpenLink Virtuoso

- Oracle
- Progress
- Ingres
- IBM Informix
- IBM DB2
- Sybase
- MySQL
- PostgreSQL
- JDBC
- ODBC to ODBC
- Firebird
- Microsoft SQL Server

Linking Remote Procedures

ATTACH PROCEDURE Statement

```
ATTACH (PROCEDURE|FUNCTION) <proc_name> ([<parameter1>[,<parameter2>[...]]])
  [ RETURNS <rettype> ] [AS <local_name>] FROM <dsn>
```

dsn
: scalar_exp

proc_name
: identifier | identifier.identifier | identifier.identifier.identifier | identifier..identifier

parameter1..parameterN
: parameters declaration (as in CREATE PROCEDURE)

local_name
: table

The ATTACH PROCEDURE statement allows you to associate stored procedures from remote datasources with Virtuoso so they can be used as if they were defined directly within Virtuoso. Much like the ATTACH TABLE statement, this SQL statement creates a local alias for a procedure on a given remote data source so it can be considered locally defined. When this alias is called called the procedure at the remote data source will actually be called.

Procedure generated result sets are not supported by the ATTACH PROCEDURE statement. The only portable way to return values from a remote procedure is to use INOUT or OUT parameters. Remote procedure result sets can be used by combination of `rexecute()` and Virtuoso PL, but this is left for the user to implement as required.

The ATTACH PROCEDURE statement is not able to define new connections to remote data sources, the connection should be defined prior using either vd_remote_data_source or by attaching a table or view using the ATTACH TABLE statement with USER/PASSWORD supplied.

Note that when generating pass-through statements to a given remote, any procedure call for an attached procedure is passed through if the current DSN is the same as the remote procedure's DSN.

The *proc_name* is the designation of the procedure's name on the remote data source, DSN. The remote procedure name supplied should always be fully qualified to avoid ambiguity, it may consist of an optional qualifier/catalog, optional owner and finally procedure name, separated by dots. This must identify exactly one procedure on the remote data source.

The optional *local_name* is an optionally qualified procedure name which will identify the procedure on the local Virtuoso database. If the local_name is not given it defaults to the <current qualifier>.<dsn>.<proc name on dsn>. The <dsn> will be the data source name in upper case and with all non-alphanumeric characters replaced by underscores. The <proc name on dsn> will be the exact name as it is on the remote dsn, case unmodified.

If a dsn is not previously defined with vd_remote_data_source or ATTACH TABLE, the ATTACH PROCEDURE will fail.

Example 6.3. Example:

On remote Virtuoso (DSN name : remote_virt):

```
CREATE PROCEDURE RPROC (IN PARAM VARCHAR) returns VARCHAR
{
  return (ucase (PARAM));
}
```

On the local virtuoso (DSN name : local_virt) :

```
vd_remote_data_source ('remote_virt', '', 'demo', 'demopwd');
ATTACH PROCEDURE RPROC (IN PARAM VARCHAR) RETURNS VARCHAR from 'remote_virt';
```

will result in creation of an procedure alias for RPROC in local_virt named DB.REMOTE_VIRT.RPROC

Calling it from the local_virt (using ISQL)

```
select REMOTE_VIRT.RPROC ('MiXeD CaSe') as rproc_result;

rproc_result
---------------
MIXED CASE
1 rows
```

See Also:

Tip

The Virtuoso Visual Server Administration Interface provides a graphical user interface for linking remote stored procedures .

Data Type Mappings

If a statement is passed through to a remote data source, the types returned by SQLDescribeCol are taken directly from the remote prepare and converted to the closest Virtuoso supported type.

If a statement involves both local and remote resources all types are taken from the Virtuoso copy of the data dictionary.

In executing remote selects Virtuoso binds output columns according to the type and precision given by SQLDescribeCol after preparing the remote statement.

When a table is attached from a remote data source the catalog is read and the equivalent entries are created in Virtuoso. Since the types present on different DBMS's vary, the following logic is used to map ODBC types to Virtuoso types.

Table 6.1. Attach Table Type Mappings

SQL Type	Mapped Type
SQL_CHAR	varchar (precision)
SQL_VARCHAR	varchar (precision)
SQL_BIT	smallint
SQL_TINYINT	smallint
SQL_SMALLINT	smallint
SQL_INTEGER	integer
SQL_BIGINT	decimal (20)
SQL_DECIMAL	
SQL_NUMERIC	smallint if precision < 5 and scale = 0 integer if precision < 10 and scale = 0 double precision if precision < 16 decimal (precision, scale) otherwise
SQL_REAL	real
SQL_FLOAT	double precision
SQL_DOUBLE	double precision
SQL_BINARY	varbinary (precision)
SQL_VARBINARY	varbinary (precision)
SQL_LONGVARCHAR	long varchar
SQL_LONGVARBINARY	long varbinary
SQL_DATE	date
SQL_TIME	time
SQL_TIMESTAMP	datetime

Note:
The general case of decimal and numeric both revert to the Virtuoso decimal type, which is a variable precision decimal floating point.

Transaction Model

One transaction on the Virtuoso VDBMS server may contain operations on multiple remote data sources. As a general rule remote connections are in manual commit mode and Virtuoso either commits or rolls back the transaction on each of the remote connections as the main transaction terminates.

ODBC does not support two phase commit. Therefore a transaction that succeeds on one remote party may fail on another.

A transaction involving local tables and tables on one remote data source will always complete correctly since the remote is committed before the local and the local is rolled back if the remote commit fails.

Note that even though the client to Virtuoso connection may be in autocommit mode the continuing connections will typically not be autocommitting.

A remote connection is in autocommit mode if the Virtuoso connection is and the statement is passed through unmodified. In all

other cases remote connections are in manual commit mode.

Virtuoso supports 2PC - Two Phase Commit. See the Distributed Transaction & Two Phase Commit section for more information.

Virtual Database and SQL Functions

Different DBMS's support slightly different sets of SQL built-in functions with slightly differing names. For example, what is called substring on Virtuoso is called substr on Informix. Virtuoso allows declaring equivalences between local user-defined or built-in functions and user defined or built-in functions on remote servers. Knowing that a function exists on a remote database allows passing processing closer to the data, resulting in evident gains in performance.

To declare that substring is called substr on DSN inf10, you can execute:

```
db..vd_pass_through_function ('inf10', 'substring', 'substr');
```

The first argument is the name of the remote database, as used with attach table and related statements. If user defined functions with qualified names are involved, the names should be qualified in the vd_pass_through_function call also. If many qualified or unqualified forms of the name are in use, one should declare the mapping for them all.

To verify whether this declaration takes effect, one can use explain to see the query execution plan, for example:

```
explain ('select substring (str, 1, 2) from inf10.sample_table');
```

The declarations are persistent and can be dropped by using a last argument of NULL for a given function. The declarations are kept at the level of a DSN and not at the level of the type of DBMS because different instances can have different user defined functions defined.

Virtual Database and SQL Optimizer Statistics

If a query can be executed in its entirety on a single remote database, then optimizing this query is exclusively the business of the remote database, as it gets the whole query text. Virtuoso rewrites some things and suggests a join order but these are not binding on the remote database.

If a query involves tables from multiple remote databases or a a mix of local and remote tables, knowing basic SQL statistics on the tables is essential for producing a meaningful query plan. Virtuoso has information on indices existing on remote tables and if the remote table is attached from a known type of DBMS, Virtuoso may read the DBMS specific statistics at attach time.

Note that the statistics of the remote database should be up to date before attaching.

The function sys_db_stat can be used for forcing a refresh of Virtuoso's statistics on remote tables.

```
sys_db_stat (5, 0)
```

will go through all tables, local and remote. For local tables, it will update statistics with a 5 percent sampling rate and for remote tables it will refresh the statistics if the type of the host DBMS is among the supported ones. If the remote DBMS is of an unknown type, Virtuoso will take the count of the remote table and select the 1000 first rows to get a sample of data lengths and column cardinalities. This is not very precise but will be better than nothing.

In order to force a full read of a remote table for statistics gathering, one can use

```
db..sys_stat_analyze ('fully qualified table name', 0, 0);
```

The table name is case sensitive, with all qualifiers, as it appears in SYS_KEYS and other system tables. This will read the whole table.

Statistics on local as well as remote tables are kept in SYS_COL_STAT. One may look at this table to see the effects of remote statistics collection. In special cases, if a special effect is desired or the information is not otherwise available, as in the case of a very large table on an unsupported type of server, it is possible to manually update the contents of the table. Shutting down and restarting Virtuoso will force a reload of the statistics information.

Presently Oracle, Informix and Virtuoso are supported for direct access to the remote database's statistics system tables. It is possible to define hook functions for accessing this same information from any other type of DBMS. Please contact support for

instructions on this.

Distributed Query Optimization

When a query contains mixes of tables from different sources, the compiler must decide on an efficient execution plan that minimizes the number of round trips to remote servers and evaluates query conditions close to the data when possible. Additionally, any normal query optimization considerations such as choice of join order and join type apply. See the section on SQL optimization and optimizer hints for more on this. Additionally, the SQL optimizer uses round trip time statistics for the servers involved in the query.

In the following examples, we use the tables r1..t1, r2..t1 and t1, of which r1..t1 is on a server close by, r2..t1 on a server farther away and t1 on the local server. The column row_no is a unique key and the string1 column is in indexed with 300 distinct values, the column fs1 has 3 distinct values. The tables all have 100000 rows. A round trip to r1 takes 10 ms and a round trip to r2 takes 100 ms.

Consider

```
select * from r1..t1 a, t1 b where a.row_no = b.row_no and a.fs1 = 'value1';
```

The compiler notices that 33333 rows will be selected from r1..t1 based on fs1. It will decide to read these into a hash table, causing one linear scan of r1..t1 with relatively few round trips. Then it will read t1 locally and select the rows for which there is a matching entry in the hash. This is slightly faster than doing 33333 random lookups of t1. If fewer rows were selected from r1..t1, the compiler would do a loop join with the local t1 as the inner loop.

The absolute worst plan would be a loop join with t1 as the outer loop, with 100000 round trips to r1.

Now, if many tables are accessed from the same data source, the compiler tries to bundle these together into one statement. Thus, for:

```
select * from r1..t1 a, r1..t1 b, t1 c where c.string1 = '111' and b.row_no = c.row_no and a.row_no = b.r
```

The compiler will probably do the outer loop for t1, which is expected to select 100000/300 rows. Then it will do a round trip to r1 with the statement.

```
select * from t1 a, t1 b where a.row_no = b.row_no + 1 and a.row_no = ?.
```

This is likely better than doing the remote part as an outer loop, bringing all the approx 100000 results in. 333 round trips selecting 1 row is better than 100000 rows transferred. If the data source were further away, this could be otherwise, hence the importance of the round trip time statistic.

In distributed queries, the compiler will honor the option (order) and the join types e.hg. table option *(hash) insofar the tables are local.

Thus, if we wrote

```
select * from r1..t1 a, t1 b, r1..t1 c where c.string1 = '111' and b.row_no = c.row_no and a.row_no = b.r
```

the compiler could not merge the two tablesfrom r1 into a single query because the order were given and there is an intervening table not located on r1.

Use of Array Parameters

ODBC and other data access API's usually offer a mechanism for executing a single parametrized statement multiple times with a single client-server round trip. This is usually called support of array parameters.

Virtuoso can make use of array parameter support in ODBC drivers when available. Consider the statement:

```
insert into r1..t1 select * from t1;
```

Without array parameters, this would make a round trip to r1 for each row inn t1. With array parameters enabled, with a batch size of 100, this would make only 1000 round trips for 100000 rows, resulting in dramatic increase in performance. Typically, if the remote server is on the same machine, array parameters make such batch operations about 3x faster. If the remote is farther away,

the gain is greater.

Array parameters are used if the remote database and its ODBC driver support them. The feature is globally disabled in the default configuration because some ODBC drivers falsely claim to support array parameters. To enable this feature, the the ArrayOptimization entry in the [VDB] section of the ini file to 1. To set the batch size, use the NumArrayParameters setting. 100 is a reasonable value.

Some ODBC drivers also support array parameters for select statements. To enable using this, you can set the ArrayOptimization setting to 2. This may however not work with some drivers even if DML (insert/update/delete) statements do work with array parameters.

Timestamps & Autoincrement

A transaction timestamp is not the same across the transaction if the transaction has branches in different databases.

The data type and precision of a time stamp will also vary between different types of databases.

Hence timestamp columns coming from tables on different servers are not comparable for equality.

In inserts and updates of remote tables timestamps are assigned by the database where the table in question is physically located.

Identity or autoincrement columns are likewise handled by the database holding the remote table.

Note that MS SQL Server and Virtuoso describe a timestamp column as a binary column in ODBC catalog and meta data calls. Thus remote SQL Server or Virtuoso timestamps will not appear as timestamps at all.

In the case of a Virtuoso remote database the binary timestamp can be cast into a DATETIME data type and it will appear as a meaningful datetime.

VDB Stored Procedures & Functions

These procedures allow you to manually manage remote data sources and their tables.

- vd_remote_data_source()
- vd_remote_table()
- rexecute()
- rnext()
- rmoreresults()
- rclose()
- rstmtexec()

Functions capable of returning a result-set make use of the *results_set* parameter. To prevent them from returning a result-set, the *results_set* parameter should be set to 'null'. If Virtuoso finds an awaiting parameter to contain *results_set* it will fetch the result set regardless of *cursor_handle* parameter.

Unless explicitly granted, only the DBA group is permitted to use the `rexecute()` to maintain security. Caution is required here since any user granted use of `rexecute()` has full control of the remote data source set-up by the DBA, however limited to the overall abilities of the remote user on the remote data source. Users can be granted and denied access to this function using the following commands:

```
GRANT REXECUTE ON '<attached_dsn_name>' TO <user_name>
REVOKE REXECUTE ON '<attached_dsn_name>' FROM <user_name>
```

The following remote catalogue functions help you to obtain information about the remote datasources that you are using. These could be especially useful in Virtuoso PL later on if you are not able to know everything about the remote tables ahead of time for the ATTACH TABLE statement

- sql_data_sources()
- sql_tables()
- sql_columns()
- sql_statistics()

- sql_primary_keys()

Manually Setting Up A Remote Data Source

Defining a remote table involves declaring the table as a local table and then defining the data source if not already defined and associating the new table with the remote data source.

The data source on which a table resides is declared at the table level. This has no connection to the table's qualifier.

Assume a remote ODBC data source named test containing a table xyz declared as follows:

Example 6.4. Example:

```
CREATE TABLE XYZ (
A INTEGER,
     B INTEGER,
     C INTEGER,
     PRIMARY KEY (A));
```

To defined this as a remote table on the data source Virtuoso, first define the table locally, using the above CREATE TABLE statement above.

Then define the data source:

```
DB..vd_remote_data_source ('test', '', 'sa','');
```

And the table:

```
DB..vd_remote_table ('test', 'DB.DBA.XYZ', 'master.dbo.XYZ');
```

This assumes that the remote data source has a login 'sa' with an empty password and no special connection string options. The table names in vd_remote_table have to be fully qualified. We here assume that the Virtuoso table was created by DBA in under the default qualifier DB and the remote XYZ was created by dbo in master.

The vd_remote_table declaration does not affect statements or procedures compiled prior to the declaration.

Additional indices of remote tables may optionally be defined. They do not affect the operation of the SQL compiler. The remote data source makes the choice of index based on the order by clause in the statement passed through.

Caveats

- Never attempt to attach a local table as a remote. The server will hang if it tries to make a remote commit on itself.
- If the schema of the remote table is changed it will need to be re-attached to Virtuoso.
- The Virtuoso server treats dots (.) in the double-quotes escaped names as name element separators. For example : the table name "a.b.c" is treated as "a"."b"."c" . Because of this remote tables with dots in their table name (like tables from MS Text driver) require the dot inside the table name to be replaced with the VDB "non-delimiting-dot" (\x0A) and the vd_attach_table (in dsn varchar, in remote_name varchar, in local_name varchar, in uid varchar, in pwd varchar) to be used instead of ATTACH TABLE statement.

  The statement ATTACH TABLE "datafile.txt" as 'test' from 'text' user 'a' password 'b' should become :

  ```
  vd_attach_table ('text', 'datafile\x0Atxt', 'test', 'a', 'b');
  ```

When Virtuoso interacts with a table or view attached from a remote data source, it must be able to uniquely identify each row of the query. At the attach time Virtuoso will query remote data source for the tables primary keys and indices. These will be used to construct a copy of the table definition in Virtuoso which is then used in reference to the remote data source. At query time this information is used as much as possible. This information may need to be supplemented by calls to SQLStatistics() for further indicies or primary key information, as a last resort Virtuoso will use SQLColAttribute() to determine which columns are SQL_DESC_SEARCHABLE.

6.1.3. Virtuoso User Model

The Virtuoso User Model is designed to support:

Use of an external server for password and user account base maintenance for example an LDAP server, another database server etc. This allows user accounts to be verified against security information stored in some centralized repository, allowing integration into existing security infrastructure .

Single namespace for users and groups for SQL and web service access. In this way the local security info is stored in one place and allows enabling SQL account to work as a web account or a web account as an SQL account. This enforces unique names for users and roles in the database.

Extensibility of user information retrieval and checking.

Extensibility of user account and group attributes.

There is a set of functions for administering the users and groups (roles) of a Virtuoso database. All the user administration functions are restricted to members of the dba group only.

Note:
The terms 'role' and 'group' are identical in their usage in this document. The terms security object or grantee refer collectively to users and groups.

User manipulation functions do not generally return values. In case of error an error condition will be signaled.

Users and roles are in the same namespace and are identified by name. One name will not both designate a user and a group. A user or group may be SQL enabled. A name that identifies a role (group) does not have a password and is not a valid login name.

A user may belong to multiple roles. A role may inherit multiple other roles. One role may be inherited or directly granted to a security object multiple times. The order of granting is not relevant. The effective privileges of a user are those granted to public, those granted to the user plus those granted to a direct or indirect role of the user. The relationship between users and roles is many to many. A role can be granted multiple other roles as well as direct permissions. Cycles in granting roles are not allowed, an error will be signaled if an operation would result in a cycle in the role inheritance graph.

When SQL procedures or queries are executing, the effective privileges are those granted to the owner of the procedure or view. A top level dynamic SQL statement executes with the effective privileges of the logged in user. If a SQL statement executes as a result of an HTTP request, the applicable virtual directory specifies on behalf of which SQL account the SQL is executed. A role cannot be the owner of procedures, views or other executable SQL entities.

Security Objects Management

The following functions allow for creation and deletion of security objects and roles, and for assigning or removing roles from security objects:

```
USER_CREATE()
USER_ROLE_CREATE()
USER_DROP()
USER_ROLE_DROP()
USER_CHANGE_PASSWORD()
USER_SET_QUALIFIER()
USER_GRANT_ROLE()
USER_REVOKE_ROLE()
USER_SET_OPTIONS()
USER_GET_OPTIONS()
```

The security objects and roles data are contained in the system tables described in the User System Tables Section of the Appendix

User Options

- **PASSWORD_MODE.** Function for checking a given password on SQL or DAV login. See below.
- **PASSWORD_MODE_DATA.** Application specific data for the Password Mode hook.
- **LOGIN_QUALIFIER.** Default qualifier for SQL session.

- **SQL_ENABLE.** If set SQL login is granted.
- **DAV_ENABLE.** If set the user account can be user for web authentication.
- **DISABLED.** If set the user account is locked and cannot be used to login as SQL or Web user (depends of SQL_ENABLE and DAV_ENABLE flags). If the account in question is SQL enabled the DBA group can switch the execution identity to it (see set_user_id () function). This is useful when we need an account to execute Web pages (VSP/VSPX) with some execution permissions but we do not want to allow it to login via SQL/ODBC.
- **PRIMARY_GROUP.** This is the primary group of the user. This is no different from other group memberships.
- **GET_PASSWORD.** Function that will retrieve the password. If not defined the password is assumed to be in the SYS_USERS table. This allows for custom encrypted storage of passwords etc. This is simpler to use than the check hook. Note that for security configurations where the server never does know the passwords of user accounts, no digest based authentication schemes can be used, including the HTTP digest authentication, since the digests cannot be computed and checked without knowing the password. Possible users of this feature are DBEV_LOGIN or HTTP login hooks.
- **E-MAIL.** informative: e-mail of that user.
- **FULL_NAME.** informative: full name of the user.
- **HOME.** WebDAV home directory of the account, it is meaningful only if the account is web enabled.
- **PERMISSIONS.** WebDAV default permissions for new WebDAV objects created by the user. This is only meaningful when web access is enabled.

The functions for setting/getting these options will accept any other named values, the above list only being those reserved for Virtuoso so far.

Login Extension PL Hook

```
DB.DBA.USER_FIND (in name varchar, )
```

This is a user-defined PL function hook which, if it exists, will be executed before doing the SQL/ODBC login. In this hook the user can find a user account from some other server and register it in the local database. Or, this can be used to perform some pre-login actions. It is similar to the DBEV_LOGIN, but it does not change any account validation rule, it is purely for pre-processing.

See Also:

The Database Event Hooks chapter.

PL Hooks Examples

Example 6.5. Querying an LDAP Server

```
create procedure DB.DBA.LDAP_SEARCH (inout user_name varchar, in digest varchar)
{
  whenever sqlstate '28000' goto ldap_validation_failure;
  if (lcase(user_name) <> 'dba'))
    {
      ldap_search('foo.bar.com', 0, 'o=organization', '(cn=a*)',
                sprintf('uid=%s,ou=users,o=organization', user_name), pwd_magic_calc(user_name,digest,1))
      user_name := 'dba';
      return 1; -- force validation as dba
    }
  else
    {
      -- bypassing ldap authentication for dba, let validation occur normally
      return -1;
    }

ldap_validation_failure:
  return -1; -- try to validate normally
}

create procedure DB.DBA.DBEV_LOGIN (inout user_name varchar, in digest varchar, in session_random varchar
{
  declare get_pwd varchar;
  get_pwd := user_get_option (user_name, 'PASSWORD_MODE');
  if (get_pwd is not null)
    {
      declare rc integer;
      rc := call (get_pwd) (user_name, digest);
```

```
      return rc;
    }
  return -1;
};

user_create ('test_ldap', 'secret', vector ('PASSWORD_MODE', 'DB.DBA.LDAP_SEARCH'));
```

Example 6.6. USER_FIND PL Hook Example

```
create table
MY_DBA_USERS (M_NAME varchar primary key, M_PASSWORD varchar);

create procedure
DB.DBA.USER_FIND (in name varchar)
{
  -- do nothing for existing users
  if (exists (select 1 from SYS_USERS where U_NAME = name))
    return;
  -- if there is in custom table
  if (exists (select 1 from MY_DBA_USERS where M_NAME = name))
    {
      declare pwd varchar;
      -- get password
      select M_PASSWORD into pwd from from MY_DBA_USERS where M_NAME = name;
      -- create a new SQL user based on external data
      USER_CREATE (name, pwd, NULL);
    }
};
```

SQL Role Semantics

The terms user group and role are used interchangeably. Roles can be nested. There is a many to many relationship between users and roles. There is likewise a similar, acyclic many to many relationship between roles. Each role has a component role list of its granted (parent) roles, recursively, no cycles allowed.

All role grants are listed in the roles system table whether they be explicitly granted or only as a result of granting a group with groups granted to it. The role grant graph has an explicit edge for each role membership, direct or otherwise. The GI_DIRECT flag is true if the grant is direct. Only direct role grants can be revoked.

See Also:

The role system table description can be found in the appendix under System Tables .

The following SQL statements deal with roles. To create a new role (group) object the following statement can be used:

```
CREATE ROLE <NAME>
```

The <NAME> is a name of role to be created. It must be unique in space of all security objects.

Example 6.7. Creating a security role

```
SQL> create role admins;
```

Use the following statement to remove an existing role from the security schema.

```
DROP ROLE <NAME>
```

Example 6.8. Removing a security role

```
SQL> drop role admins;
```

The GRANT and REVOKE statements are used for controlling role membership as follows: To assign a new group, or list of groups (<ROLE>,...) to user <USER> use:

```
GRANT <ROLE> [, <ROLE>] TO <USER> [WITH ADMIN OPTION];
```

If the admin option is specified, the grantee can grant this same privilege further to other grantees.

Roles can be revoked using:

```
REVOKE <ROLE> [, <ROLE>] FROM <USER>;
```

Example 6.9. Granting & revoking security roles

```
SQL> grant admins, users to demo;

SQL> revoke admins from demo;
```

Only the dba group accounts may administer roles.

The dba group is not physically a role. When an empty database is created, it will have the dba account with full privileges. To grant these same full privileges to another user, the dba uses the grant all privileges to <grantee>. statement. This will give the grantee full dba privileges, or in other words, add to the dba group. This may be reversed with the revoke all privileges from <grantee> statement.

The GRANT statement accepts any valid SQL security object in the TO clause. One cannot log in or perform any operations as a role. Roles are exclusively shorthand for specific sets of permissions which are changed together and are needed for multiple users.

6.1.4. VAD - Virtuoso Application Distribution

VAD provides a package distribution framework for installation, management, dependency checking and un-installation of Virtuoso applications. A VAD package contains all required Virtuoso components, which would constitute an application or hosted solution, within a single distributable file. A VAD package cannot contain any system parts independent of Virtuoso thus excluding operating system executables, shared objects, installers or settings.

Virtuoso and VAD provide the following abilities:

List all installed VAD packages.

List all completed operations over VAD packages.

Dependency checking - Check preconditions for installation of a package.

Install a VAD package.

Verification of installed package, compare to distribution state with ability to build of list of locally changed parts of the package.

Check preconditions for un-installation of a package.

Uninstall a VAD package.

Summary of VAD Operations

The following is what the dba needs to know about VAD packages.

A VAD package is installed from a file with the db..vad_install SQL function. The first argument is the file path, which must be in a location that the server process can open, i.e. it is in the DirsAllowed list in the virtuoso.ini file. The second argument is 0, meaning that we are installing from a file.

```
SQL> vad_install ('conductor_dav.vad', 0);
```

is an example. If the package installation fails, the server exits and will have to be restarted. No effects of a failed installation will remain in the database after restart. Contact the supplier of the VAD package for further instructions.

To know what is installed, do:

```
SQL> vad_list_packages ();
```

VAD package installations are not recorded in the transaction log. Thus, if there is a backup followed by archived transaction logs produced if CheckpointAuditTrail is on in virtuoso.ini, the VAD install must be performed before replaying any logs that were made after the VAD installation. The package installation must be just in the right place in the replay sequence. In practice it is simplest to make an incremental backup after installing and packages, see backup_online () or the section on backing up.

For any further information, including how to make VAD packages, see the rest of this chapter.

VAD Package Composition

A VAD package has no developer tie-ins; it is built in a development environment from source code that can be managed and versioned in the developers system of preference.

The VAD package is described by an XML structure called the 'VAD Sticker'. The VAD sticker describes items to deploy, procedures to execute at install and uninstall time and all conditions to be checked to ensure correct installation. The VAD Sticker consists of the following:

- VAD package meta data
    - Names of package, developer, copyright holder etc.
    - Version number of package, build date, build number, build type (e.g. sort of optimization performed).
    - Dependency information: minimal/maximal allowed version numbers of Virtuoso server and depending VAD packages. Every required package may include hint-text that may help the administrator determine (a) why the dependent package is required, and (b) how to obtain the required package.
    - Information regarding known conflicts between packages; conflicting package names and version number, with optional troubleshooting hints.
    - Ability to uninstall, a flag and list of reasons why it may be impossible to uninstall the package.
    - Custom configuration data to be placed in the VAD Registry
- Locations of SQL files containing main and installation code:
    - Pre-install code, used to check application-specific installation preconditions.
    - Application specific table and view definitions.
    - Application specific stored procedure and trigger definitions.
    - Post-install (initialization) code, such as initial contents of tables.
    - Pre-uninstall code, used to check that it is safe to uninstall a package.
    - Post-uninstall code, used for removal of cached resources unusable or meaningless without the package.
- Locations of Resources:
    - Documentation files.
    - Samples data for demonstration or package sanity check.
    - VSP/VSPX pages, related graphics, Java scripts, stylesheets, other web content.
    - XML docs, XSLT sheets, DTDs and Schemas.

Package Versioning

All required packages should be listed in the VAD sticker. Known conflict may be listed in either of the conflicting VAD packages stickers, hence VAD stickers of all installed packages should be checked.

Later versions of a package may be installed replacing earlier versions of the same package. This however can be prohibited by listing either version (or limit) as a known conflict in either VAD package sticker in the usual way. Furthermore, it is possible to prevent re-installation of a package by stating that it conflicts with itself. This provides some security against exploits involving attempts to upgrade, downgrade or re-install a package, in the hope that the administrator may corrupt the existing installation by installing new packages and working through installing their dependencies.

Packages may differ in language and encoding of documentation and resource files, even though the version number remains the same. If a package is sensitive to internationalization issues, the developer should either assign different names to various localizations of the package, or divide it into kernel package for any language-independent parts and set of language-specific packages, with some dependency between them.

Processing of Resources

During creation of a VAD package, the "location" mentioned above may be name of a file in file system or URI or DAV path. Upon package-time, URIs will be resolved and resources under them will be copied into the package. The resulting sticker will thus contain the location of resource within the package, the resource itself, and the target location.

All SQL files have a specific order of loading. Tables, views etc. must be defined before being referenced.

Unsupported Features of VAD

The VAD specification explicitly does not define the following:

Method of development or environment. There are no specific restrictions for the schema or Virtuoso/PL code of the package. The VAD system does not make assumptions on the method of software development.

Method of source code control or versioning. Version numbers used in the sticker have nothing common with tag labels in a developers versioning system. Procedures edited directly within the database using a web interface or CASE tools should be exported to a file for inclusion in a VAD package. If the application developer uses some script to export such code, this script is not usually part of sticker or the resulting package.

Shipping/Deployment the VAD package from vendor to user. VAD provides no methods for downloading dependent packages, or check for package updates etc.

Concurrent running of multiple versions of the same VAD package on a single server. There can be no guarantee that pre- or post-installation checks will provide valid results if more than one VAD is being processed at the same time. VAD does however guarantee that a package installation will be either entirely successful or entirely rolled back.

Installation or maintenance of non-Virtuoso hosted components. Unlike Virtuoso-based packages, these components are usually operating system specific, they may require some complex tuning, and their usage from within Virtuoso applications may even require changes in virtuoso.ini configuration file. VAD packages may contain test calls in pre-installation SQL procedures to check that required external executables are available and provide the functionality required.

Data migration. Some installations may require several days to complete migration/conversion of stored data. Whilst it may be possible to provide a restricted service during such time, VAD contains no tools to simplify such a process, this is left to the administrator or developer. VAD completes its work right after the execution of the post-installation code.

Synchronous installation of a package on all hosts of a distributed system or cluster. VAD has no standardized metadata regarding replication issues, hence package-specific code may be required. Similarly, if a cluster uses "round-robin" or a "director" loading management system and the server should be stopped for VAD installation, the administrator should explicitly inform the cluster manager about this event.

Security

Since VAD packages are run by an administrator as the database DBA user, care must be taken to ensure the package comes from a safe source. Any new package installed may violate the security regulations of the target database and may even inflict damage to files under the web-root of the Virtuoso Server or in directories specified in the "DirsAllowed" parameter of the virtuoso.ini. If the virtuoso.ini parameter "AllowOsCalls" is enabled then the installation procedures of the package may call operating executables. It is the responsibility of the database administrator to control this via the "AllowOsCalls", "SafeExecutables" and "DbaExecutables" parameters of the virtuoso.ini.

VAD packages do not offer any automatic protection against unauthorized modifications. Although ever VAD package contains a checksum, its purpose is to guard against data transfer errors, it may not be sufficient to detect unwanted modification.

Building VAD Packages

Initially, the VAD sticker and resources may reside in the file system, DAV directory and or other locations available through the `DB.DBA.HTTP_URI_GET()` function.

The VAD creation operation parses the VAD sticker's XML description and constructs the VAD file by calling `DB.DBA.VAD_PACK`.

This function reads the VAD sticker identified by the *sticker_uri* which contains the vad:package root element. Then the resources identified in the sticker are retrieved. All resource URIs are interpreted in the context of the *base_uri_of_resources* and are parsed and checked to be syntactically correct. Resources are appended to generated package that will be stored at the *package_uri* . DB.DBA.VAD_PACK returns a human readable log of error and warning messages, it will signal errors if any resource or database objects are unavailable at build time.

By convention, VAD package files have the extension '.vad'.

VAD Utilities

An optional VAD package named VADutils provides various tools for capturing changes made in the database after some point in time. The result of a capture consists of:

- Database object additions whose names match given patterns (e.g. all tables and procedures within a particular catalog/qualifier).
- Resource additions under particular locations.
- Post-install local customizations of selected packages.

The capture results may be useful for the following purposes:

- Archival of changes for replaying later.
- Creating a special package of the changes for applying against a fresh installation of the package.
- Creating a new complete package containing both the original and changes that will be included in the package sticker.

These mechanisms provide good support for centralized development and custom deployment methodology. If a site is localized to contain local links, graphics, custom layout and such, then VAD capabilities offer help to the developer to define the specific overlay of customizations over another VAD package. When the underlying VAD package is updated the local customizations will be overwritten. Being saved in a VAD package, customizations can be reapplied over the updated base package.

VAD Administrator Responsibilities

VAD package installation, upgrade and uninstallation requires a temporary break of service. The package checks may be performed on the fly if it can be guaranteed that the resources being inspected will not be altered by any users. The package check is a read-only process and operates solely within the VAD Registry using read-only functions.

All VAD operations are logged in the server event log. All completed operations are reflected in the DB.DBA.VAD_HISTORY system table.

The optional VADutils package provides some additional administrative tools, mostly for troubleshooting. These include special installation and de-installation functions that can ignore error signals, and provide an interactive editor for the VAD Registry etc.

All operations described below require DBA access to the database.

Check if a VAD package may be installed by calling DB.DBA.VAD_CHECK_INSTALLABILITY .

Checks the presence and correct versions of required packages and of the Virtuoso platform. It does not executes any pre-install Virtuoso/PL code from the package, so there's no guarantee that installation will be successful if the check found no error. If *package_uri* is DAV path, *is_dav=1* , else *is_dav=0* .

Perform VAD Package Installation by calling DB.DBA.VAD_INSTALL .

If *package_uri* is DAV path, *is_dav=1* , else *is_dav=0* .

The administrator performs the following operations when installing:

- Invoke the install procedure from the web user interface or interactive SQL. This will perform the following:
  - Install documentation files.
  - Check for version and prerequisite package compatibility.
  - Disconnect SQL users and terminate web processing.
  - Make a database checkpoint.

    - Run the pre-install SQL script.
    - Load SQL code in the VAD package, in the order specified by the developer.
    - Copy web resources (VSP, VSPX, XSLT, etc.) into their designated places in WebDAV or file system Web root.
    - Run any post-install SQL code.
  - If the installation was successful, the server will come back on-line.
  - If the installation was unsuccessful, e.g., mid-install failure due to running out of disk space, or some other serious unrecoverable database error, the Virtuoso server will exit. The administrator should consult the Virtuoso log file to see what caused the failure. The installation can be completely undone manually by halting the server (if not already stopped), and removing the transaction log file (.trx). Upon Virtuoso restart, the server will continue from the last checkpoint, made prior to install, as if the installation never took place.

The return value of the `DB.DBA.VAD_INSTALL()` function is usually a sum of messages from pre- and post-installation procedures of the package. It should normally contain at least the following:

- any errors and/or warnings encountered.
- created users and catalogs/qualifiers
- root VSP page for accessing the application, if applicable.
- path to installed documentation files.
- performance optimization hints.

The VAD packages should be tested to install on an empty Virtuoso database, after any required VAD packages. Installing a package on an empty server is useful for determining that no other procedures or components were missed. Since the application would normally run on the development machine where the VAD package was built, it can be easy to overlook some components. The completeness of the source archive of the application and its independence from any ad hoc SQL objects is important, this is the only way the package can be reliably versioned, tracked or uninstalled.

Check if a VAD package may be uninstalled by calling `DB.DBA.VAD_CHECK_UNINSTALLABILITY`

Performs a preliminary read-only checks to see whether the package given can be uninstalled. This does not execute any pre-uninstall Virtuoso/PL code from within the package at this stage. Hence, the success of this function does not guarantee that uninstallation will be successful.

Perform VAD Package Uninstallation by calling `DB.DBA.VAD_UNINSTALL`.

The administrator will perform the following operations for the uninstallation process:

- Invoke the uninstall procedure from the web user interface or interactive SQL. This will initiate the following:
    - Check that no other packages are using the package to be uninstalled.
    - disconnect SQL users and terminate web processing.
    - Make a database checkpoint.
    - Run the pre-uninstall SQL script.
    - Remove web resources installed by the package (all VSP , VSPX, XSLT, etc files) in WebDAV or the filesystem under the web root.
    - Drop all SQL procedures and data. This is performed in reverse order to the install.
    - Run any post-uninstall SQL code.
    - Remove documentation files explicitly marked as removable. Usually documentation would not be deleted as part of package uninstallation in case it is needed e.g. if a set of documents is distributed as VAD package)
  - If uninstallation was successful the server will come back on-line.
  - If uninstallation was unsuccessful, the server will exit. Uninstallation could fail due to lack of disk space or some other serious unrecoverable database error. The failed uninstallation attempt can be manually reversed by halting the server (if not already) and deleting the transaction log file (.trx). Upon server restart Virtuoso will continue from the last checkpoint, made prior to uninstallation, as if the uninstallation was never attempted. The administrator should consult the log file for clues to the failure.

Check the state of VAD package installation by calling `DB.DBA.VAD_CHECK`.

This checks to see if the elements of the package are as they are defined in the original distribution. A list of differing elements is returned. Differences revealed may not indicate a corruption, such changes could have been made intentionally by another package, possibly a later version or upgrade that added some columns to tables, and some resources may be customized by the

user post-installation.

This will check for the prior existence of tables, views etc owned by other applications that are not compatible with this application. Any such schema objects found are listed, the installation will not continue. These may be dropped by the DBA to help the installation to succeed. Some such elements may not be part of some other package, hence no package uninstall would be available leading the DBA to drop them with the appropriate SQL commands.

To enable automatic vads updates on server startup, set the VADInstallDir parameter in the [Parameters] ini section with path the folder containg the vads files.

Package Overlap

Each package contains full definitions of all tables and indices. Upon installing the following outcomes can occur:

- If a table already exists with the same primary key as the new definition, additional columns are added to the table. If the primary keys differ, the installation automatically fails. Note that a pre-install SQL script can be defined to explicitly alter tables if consecutive versions of an application use different primary keys.
- Existing indexes are left untouched. New indices are added as specified in the package. If indices should be modified or dropped, the pre-install script is a reasonable place for dropping these.

Thus the same SQL schema can be loaded twice without ill effect.

The post install script should be used to populate tables and such. Inserts should be executed using the insert soft statement so that attempts to insert duplicate are silently ignored without causing the installation to fail. The post install script can perform any application level data format changes.

Packages should define their own distinct catalog or qualifier. They should not overwrite another package unless upgrading a prior version. Sometimes a package will require the use of another package's tables. This should be achieved via grants issued in a pre-install script. A schema element such as a table, view or procedure will always have at most one owner package even though it may be referenced or even modified with additional columns by another package installed later. These elements will only be dropped when the owner package is dropped. Tables created ad-hoc from interactive SQL do not have any owner package.

VAD Sticker

The VAD Sticker contains meta-data and descriptions of resources contained, or to be contained, within a VAD package. Like any XML documents, the target VAD package sticker can be sourced from more than one source file, which can aid maintenance and development.

VAD Sticker DTD

The namespace vad, used below, represents the URI `http://example.com/urn/vad`.

The top level element of a VAD Sticker is <sticker>. It must contain a <caption> element and may contain <dependencies>, <procedures>, <ddls> and <resources> elements.

```
<!--<<top>>-->

<!ENTITY % vad.source_sticker "INCLUDE">
<!ENTITY % vad.package_sticker "IGNORE">
<!ENTITY % vad.ecm.group_content "(dependencies | procedures | ddls | resources | registry)" >
<![%vad.source_sticker;[
  <!ENTITY % vad.ecm.sticker "(caption, (group | %vad.ecm.group_content;)*)">
  <!ELEMENT group ((group | %vad.ecm.group_content;)*) >
  ]]>
<![%vad.package_sticker;[
  <!ENTITY % vad.ecm.sticker "(caption, %vad.ecm.group_content;)">
  ]]>
<!ELEMENT sticker %vad.ecm.sticker; >
<!ATTLIST sticker
  version     NMTOKEN #REQUIRED
  xml:lang    CDATA   #REQUIRED
  >
    <!--<</top>>-->
```

```
    <!--<<caption>>-->

<!ELEMENT caption (name, version)>
<!ELEMENT name ((prop)*)>
<!ATTLIST name
  package NMTOKEN #REQUIRED
  >
<!ELEMENT version ((prop)*)>
<!ATTLIST version
  package NMTOKEN #REQUIRED
  >
<!ELEMENT prop EMPTY>
<!ATTLIST prop
  name NMTOKEN #REQUIRED
  value CDATA #REQUIRED
  >
    <!--<</caption>>-->
```

The caption contains one name and one version element. These elements have a package attribute for keeping requisites used by VAD procedures. Other prop-s are for keeping admin-readable info, but they will not affect the installer's behavior. Typical names of properties here are Vendor, Copyright, Release+Date, Build, Language, Encoding, but any (even non-unique) names are acceptable.

Sticker's elements for dependencies

```
<!--<<dependencies>>-->

<!ELEMENT dependencies ((require | allow | conflict)*) >
<!ATTLIST dependencies>
<!ENTITY % vad.ecm.version_list "((version | versions_earlier | versions_later)*)">
<!ELEMENT require (name, %vad.ecm.version_list;) >
<!ELEMENT allow (name, %vad.ecm.version_list;) >
<!ELEMENT conflict (name, %vad.ecm.version_list;) >
<!ATTLIST require
  group NMTOKEN #IMPLIED
  >
<!ELEMENT versions_earlier ((prop)*)>
<!ATTLIST versions_earlier
  package NMTOKEN #REQUIRED
  >
<!ELEMENT versions_later ((prop)*)>
<!ATTLIST versions_later
  package NMTOKEN #REQUIRED
  >
    <!--<</dependencies>>-->
```

Element dependencies contains an list of packages related to given one. For every version or range of versions of every package, developer may specify whether the given version is required for the package, or allowed but not required, or will cause some sort of troubles.

More precisely, to find information about some particular version of a package, the list of children of dependencies element will be scanned from top to bottom. If the first matching record is in conflict group, not in require or allow, then installation is impossible. From other side, there must be at least one installed package for every require section.

Element require may be labeled with optional group attribute. As an exception from common rule, there must be at least one installed package for every group of require sections with identical name. E.g. If an installation of package B requires either of two interchangeable packages A1 and A2, sticker should contain a pair of nodes in the same group:

```
<require group="G">
  <name package="A1">...</name>
</require>
```

...

```
<require group="G">
  <name package="A2">...</name>
```

```
</require>
```

Note:

There are no methods to specify that exactly one package, either A1 or A2, should be installed. It must be done by placing proper conflict descriptions in stickers of A1 and/or A2, but not in the sticker of B.

Sticker's elements for procedures

```
<!--<<procedures>>-->

<!ELEMENT procedures ((sql)*)>
<!ATTLIST procedures
  uninstallation (supported | prohibited) #REQUIRED
  >
<![%vad.source_sticker;[
  <!ENTITY % vad.sql.include "include CDATA #IMPLIED">
  ]]>
<![%vad.package_sticker;[
  <!ENTITY % vad.sql.include "">
  ]]>
<!ELEMENT sql (#PCDATA)>
<!ATTLIST sql
  purpose (install-check | pre-install | post-install | uninstall-check | pre-uninstall | post-uninstall)
  %vad.sql.include;
  >
    <!--<</procedures>>-->
```

Element procedures contains an list of Virtuoso/PL fragments, and every fragment is tagged by one of four values of the purpose attribute. At every stage of install or uninstall VAD procedure, a whole list of procedures will be scanned from the beginning to the end, and all procedures of appropriate sort will be executed in the same order as they are listed. In source sticker files, include attribute may be used to insert text of some external file instead of having SQL code written inside the element.

Sticker's elements for ddls

```
<!--<<ddls>>-->

<!ELEMENT ddls ((sql)*)>
<!ATTLIST ddls
  >
    <!--<</ddls>>-->
```

Element ddls is very similar to procedures and contains an list of Virtuoso/PL fragments to create schemas etc.

Sticker's elements for resources

```
<!--<<resources>>-->

<!ELEMENT resources ((file | location)*)>
<!ATTLIST resources >
<![%vad.source_sticker;[
  <!ENTITY % vad.file.source_uri "source_uri CDATA #IMPLIED">
  ]]>
<![%vad.package_sticker;[
  <!ENTITY % vad.file.source_uri "">
  ]]>
<!ELEMENT file EMPTY>
<!ATTLIST file
  type (doc | http | dav | code | special) #REQUIRED
  source (http) "http"
  target_uri CDATA #REQUIRED
  makepath (yes | no | abort) "abort"
  overwrite (yes | no | abort | equal | expected) "equal"
  package_id CDATA #IMPLIED
  location IDREF #IMPLIED
  dav_owner CDATA #IMPLIED
```

```
  dav_grp CDATA #IMPLIED
  dav_perm CDATA #IMPLIED
  %vad.file.source_uri;
  >
<!ELEMENT location ((prop)*) >
<!ATTLIST location
  id ID #REQUIRED
  default_target_uri CDATA #REQUIRED
  >
    <!--<</resources>>-->
```

Element resources lists all files to be copied onto target box. For every file, source and target URIs should be specified, and suggested behavior for cases when a directory should be created or file should be overwritten. Target URI may be relative to one of roots: for documentation, web-resources, DAV, SQL code (it's where virtuoso.ini is located) and one of special locations, additionally specified by location elements. (Installer may query administrator to allow changing of locations' roots; in such case, information from location's properties will be shown to the administrator.) By default, the value of *package_id* is a space delimited list of type, location ID (if any) and target URI.

dav_owner - DAV owner for file (used if type="dav", ignored if "filesystem");

dav_grp - DAV group for file (used if type="dav", ignored if "filesystem");

dav_perm - DAV permissions for file (used if type="dav", ignored if "filesystem").

Example 6.10. VAD installable file descriptions

To install files into DAV:

```
<file type="dav" source="http" target_uri="yacutia/yacutia_style.css" dav_owner='dav' dav_grp='administra
<file type="dav" source="http" target_uri="yacutia/yacutia_vdir_style.css"  dav_owner='dav' dav_grp='admi
```

To install files into file system:

```
<file type="http" source="http" target_uri="yacutia/yacutia_style.css" makepath="yes"/>
<file type="http" source="http" target_uri="yacutia/yacutia_vdir_style.css" makepath="yes"/>
```

Sticker's elements for registry

```
<!--<<registry>>-->

      <!ELEMENT registry ((record)*)>
      <!ATTLIST registry >
      <!ELEMENT record ANY>
      <!ATTLIST record
        key CDATA #REQUIRED
        type (STRING | INTEGER | KEY | URL | XML) #REQUIRED
        overwrite (yes | no | abort | equal | expected) "equal"
        >
    <!--<</registry>>-->
```

Element registry lists all branches to be defined in the VAD Registry. Every record element contain data of one record. The first children of record element (either a text or an element) will be serialized and stored as a value of *DB.DBA.VAD_REGISTRY.R_VALUE* cell. To prevent errors, it is recommended to keep comments to the data outside the record element: being in the wrong place inside, they may be stored in the registry instead of actually needed data.

Example 6.11. Sample Stickers

A package that contains only some commonly useful ("exported") functions, one table for internal purposes, a small sample VSP application, and small set of documentation files.

```
<?xml version="1.0" encoding="ASCII" ?>
<!DOCTYPE sticker SYSTEM "vad_sticker.dtd">
<sticker version="1.0.010505A" xml:lang="en-UK">
 <!-- Name and version; common data about the package -->
```

```
 <caption>
  <name package="rdf_lib">
   <prop name="Title" value="RDF Support Library" />
   <prop name="Developer" value="OpenLink Software" />
   <prop name="Copyright" value="(C) 2024 OpenLink Software" />
   <prop name="Download" value="http://example.com/virtuoso/rdf_lib/download" />
  </name>
  <version package="3.14">
   <prop name="Release+Date" value="2003-05-05" />
   <prop name="Build" value="Release, optimized" />
  </version>
 </caption>
 <!-- This package requires no other packages,
but it conflicts with package virtodp of versions
from 1.00 to 2.17, inclusive -->

 <dependencies>
  <allow>
   <name package="virtodp"></name>
   <versions_earlier package="1.00"></versions_earlier>
  </allow>
  <conflict>
   <name package="virtodp">
    <prop name="Title" value="Virtuoso ODP Sample" />
   </name>
   <versions_earlier package="2.17">
    <prop name="Date" value="2001-01-26" />
    <prop name="Comment"
          value="An incompatible version of RDF library is included in some old versions of virtodp " />
   </versions_earlier>
  </conflict>
 </dependencies>
 <!-- There are no installation procedures, other than DDLs -->

 <procedures uninstallation="supported"></procedures>
 <!-- There are some procedures, which may be re-applying and (maybe) reverted automatically -->

 <ddls>
  <sql purpose="pre-install">
   "DB"."DBA"."VAD_CREATE_TABLE" ('DB', 'DBA', 'RDF_SCHEDULED_IMPORTS',
      'ID integer,
           URI varchar,
           CALLBACK varchar,
           VERSION varchar,
           REPORT long varchar,
           primary key (ID)');
  </sql>
  <sql purpose="post-install">
   "DB"."DBA"."VAD_LOAD_RESOURCE" ('rdf_functions');
  </sql>
 </ddls>
 <!-- Resources include... -->

 <resources>
  <!-- ...documentation, ... -->

  <file type="doc" target_uri="rdf_lib/1.1/intro.dxt" />
  <file type="doc" target_uri="rdf_lib/1.1/interface.dxt" />
  <file type="doc" target_uri="rdf_lib/1.1/implementation.dxt" />
  <file type="doc" target_uri="rdf_lib/1.1/sample_app.dxt" />
  <!-- ...the file of commonly-useful functions, ... -->

  <file package_id="rdf_functions"
    type="code" target_uri="rdf_lib/1.1/rdf_lib.sql" />
  <!-- ...pages of the sample application, named rdf_edit, ... -->

  <file type="http" target_uri="rdf_lib/rdf_edit/default.htm" />
  <file type="http" target_uri="rdf_lib/rdf_edit/browse.vsp" />
  <file type="http" target_uri="rdf_lib/rdf_edit/edit.vsp" />
  <!-- ...a DAV resource with sample RDF data, ... -->

  <file type="dav" target_uri="rdf_lib/sample_odp_structure.rdf" />
```

```
  <!-- ...two files of sample application's functions. -->

  <file type="code" target_uri="rdf_lib/1.1/rdf_edit/content_level.sql" />
  <file type="code" target_uri="rdf_lib/1.1/rdf_edit/view_level.sql" />
 </resources>
 <!-- There are no application-specific registry items in this package -->

</sticker>
```

6.1.5. Data Backup & Recovery

Administering a database involves taking backups and having a readiness to recover from backups and subsequent transaction logs.

Backups can be taken in two principal ways:

Using the Virtuoso backup function/procedures.

Copying the database files.

The Virtuoso backup functions can be used from any client directly, such as ISQL. It is possible, and perhaps preferable, to create stored procedures for performing the backup functions and scheduling these with the Virtuoso scheduler.

The actual database file(s) can be copied while the database is running so long as no checkpoint is made during the copy process. Checkpointing can be disabled for this, but make sure it is re-enabled after the backup has been made.

Making a full backup of a large database can take several hours if not days just due to the speed of tapes or local area networks. Full backups must in all cases be done without an intervening checkpoint. This is why frequent full backups are not desirable. To ensure the possibility of full recovery one must have the complete set of transaction logs (audit trail log) since the last backup.

Restarting the database after restoring backed up files will show the state in effect since the last checkpoint preceding the backup. Any transaction log files made after the point of backup can be replayed to bring the state up to the last recorded transaction.

Log Audit Trail

Virtuoso can maintain a transaction audit trail. This is enabled using the CheckpointAuditTrail setting in the virtuoso.ini file. When this setting is non-zero, Virtuoso will begin a new transaction log after each checkpoint. Thus one automatically gets a full, unbroken sequence of transaction logs for the entire age of the database. These logs are named as specified in virtuoso.ini and are suffixed with their creation time.

Transaction logs older than the log that was current at the time of the last backup are superfluous for recovery, since their transactions were checkpointed before the backup started. Transactions of the log current at the time of the backup are NOT in the backed up state since they were not checkpointed, i.e. written into the read-only section of the database containing the data being backed up.

We strongly advise having the CheckpointAuditTrail enabled in any production environment.

It is good practice to have at least two generations of full backup, since the last backup may contain errors that were not known at the time of its making. If such precaution is taken then only transactions logs older than the oldest backup are safe to remove. If we needed to recover from the oldest backup for any reason we would require all audit transaction logs created during and after that backup.

A Virtuoso database can be restored from the last full backup and all Audit Trail transaction files created during and after the backup. You would need to start the database as normal with the backup version of the database file. Once the database has been started, connect using iSQL. You can then use the replay() function to replay the transaction files up to the required point. It is vital that these files are replayed in the correct order.

On-Line Backups

Backup Using Backup_Online()

Virtuoso is capable of performing online backups so that normal database operation does not have to be disrupted in order to take backups. The `backup_online()` can be used to backup the database in the state effective at the last checkpoint to a series of

backup files.

The database storage is divided into a checkpoint space that is a read only image from the time of the last checkpoint and thus can be safely backed up anytime between checkpoints and the commit space where updates subsequent to the last checkpoint data are stored. Additionally, the database records what pages have changed since the last checkpoint every time new checkpoint is made. This change tracking makes it possible to make incremental backups. The first time the backup_online function is called, it saves a compressed copy of the then current checkpoint state into one or more files. The next time it is called, it will write the changes that have come into the checkpoint space since the last time backup_online was called. It is possible to erase the change tracking data with the backup_context_clear function. The next call to backup_online will then make a full backup. Files generated by one or more calls to backup_online without intervening backup_context_clear form a series with distinct serial numbers and will be restored together. In order to restore such files, the administrator must delete the previous database files and start the server with a special flag and indicate the location of the backup files. This will bring the database to the state corresponding to the state as of the checkpoint immediately preceding the last call to backup_online, i.e. the one that wrote the newest of the backup files being restored. To restore onwards from this state, the administrator must replay transaction logs, starting with the log that was current when the last call to backup_online was made. In order to preserve all such logs, one must run with the CheckpointAuditTrail ini parameter set to 1.

A database checkpoint cannot be performed while an online backup is in progress.

Example 6.12. Performing an Online Backup

```
SQL> backup_context_clear ();
SQL> checkpoint;
SQL> backup_online ('virt-inc_dump_#', 150);
```

The `backup_online()` procedure differs from the the CheckPointAuditTrail mainly because it can be started from any point in the database. Unless CheckPointAuditTrail was enabled when the database was created, the database file at a particular state and all transaction logs created by the AuditTrail since that state would be required to restore the database. Only the backup set files would be required to restore from `backup_online()` . The `backup_online()` also makes a compressed backup, making it far more suitable for large databases.

The last optional parameter allows to point the directory(ies) where the backup files must be stored. See `backup_online()` description for details.

Restoring From an Online Backup Series

To restore from a backup series the administrator must first shutdown the Virtuoso database server and move all database files (e.g. virtuoso.db and virtuoso.trx) out of the database directory. It is recommended that copies by taken rather than deleting them entirely. Then the command:

```
<virtuoso exe> +restore-backup <FILE_PREFIX>

-- for example:
virtuoso-iodbc-t +restore-backup dump-20011010_#
```

must be issued in the directory where the "*.bp" were stored. The database will then be restored. The expression <virtuoso exe> above must be replaced with the path and filename to the Virtuoso server executable used on your system (e.g. ..\virtuoso-odbc-t.exe).

Each file in the series has a header containing a unique identifier, for the backup set and the sequence number of the file in the backup set . If an identifier in any file in the backup sequence differs from the identifier contained in the first file, the restoration process will stop and report an error, which is written to the Virtuoso log file.

At times the backup or restoration commands may return errors. Use the following list to help diagnose and resolve them:

- **Timestamp [%lx] is wrong in file %s.** The unique identifier in header of the file differs from the identifier of the first file. It is possible that the file was renamed or corrupt or belongs to another backup set.
- **Number of file %s differs from internal number [%ld].** The sequence number of the file does not correspond to the internal sequence number of file. This could be caused by the file being renamed or corrupt.

- **Prefix is wrong in file %s.** The prefix of the file does not correspond to internal information. Possible reason: file was renamed or corrupt or belongs to another backup set.
- **Could not begin online-backup.** Read error. Possible reason: Virtuoso database file was corrupt.
- **Seek/Read failure on stripe %s/database.** Read error. Possible reason: Virtuoso database file was truncated or hardware error.
- **Read of page %ld failed.** Read error. Possible reason: Virtuoso database file was truncated or hardware error.
- **Backup file writing error.** Write error. Possible reason: disk is malfunctioning or full.

Example 6.13. Restoring an Online Backup

Following the online backup example above:

```
SQL> backup_context_clear ();
SQL> checkpoint;
SQL> backup_online ('virt-inc_dump_#', 150);
```

The following command could be used to restore the database from the backup files created:

```
virtuoso-iodbc-t -c <db-ini-file> +restore-backup virt-inc_dump_#
```

or:

```
virtuoso-odbc-t.exe -c <db-ini-file> +restore-backup virt-inc_dump_#
```

Other Backup Methods

A possible way of making a full backup of a large databases is first to turn off any automatic checkpoints and make a compressed copy of the files. After the back up is complete, checkpointing should be re-enabled. The files should be compressed to make efficient use of space, and should be copied to a disk separate from the location of the database, and preferably to an external backup medium such as tape.

Manual Backup

For a large database it is best to turn off any automatic checkpoints and copy the database files to external storage. Checkpoints should be turned off by issuing the command:

```
checkpoint_interval (-1);
```

at the SQL prompt. Checkpoints can be re-enabled in a post-backup script by:

```
checkpoint_interval (<n>);
```

which sets the automatic checkpoint interval to <n> minutes. The backup will be unusable if there are checkpoints made while it is in progress. Thus it is important to guarantee that checkpoints do not occur. The only safe way of doing this is the above, since it is in principle possible to have a server crash during the backup and a roll forward following restart, all while the backup is in progress. If this happens the backup will be readable and consistent with the state of the last checkpoint if and only if there are no checkpoints between its start and completion. Setting the interval to -1 will guarantee that the server, when starting after recovery will not make a checkpoint.

The dba must make sure that clients do not issue checkpoint or shutdown statements while a backup is in progress.

The presence or absence of checkpoints at a given point in time can be ascertained from the virtuoso.log event log file.

Off-Line Backups

When Virtuoso is not running a complete and clean backup can be taken by making a copy of the database file and transaction file(s) created after the last checkpoint.

To get an up to the minute copy of a running database one can copy the database file and the associated log, i.e. the file specified in TransactionFile in the database's configuration file. When started, the log will roll forward and restore the database to the state following the last logged transaction.

Database Recovery

Rebuilding A Database

The process of rebuilding a database consists of dumping its contents into a large log file, or log files, and doing a roll forward from an empty database with that log.

The general steps to rebuild a database are as follows:

Important
It is recommended that you take a backup copy of the database file(s) prior to performing this procedure.

- Shut down the running server, making a checkpoint. This is done with the SHUTDOWN command from interactive SQL.
- Make sure there is a log file specified in virtuoso.ini.
- Start the server process virtuoso with the -b command line option: e.g. % ./virtuoso -b (+backup-dump)

  This will write the contents of the database into the log file specified in virtuoso.ini and exit when complete.
- Take a backup of the old database file.
- Change the DatabaseFile setting in virtuoso.ini to a non-existing database file or delete the old database file.
- Start the server with the +restore-crash-dump option. The option is essential.

  Important: When restore on v6 you also must give it +log6 flag i.e.: +restore-crash-dump +log6 .

  See Also:
  How can I move a Virtuoso database from one hardware and/or software platform to another?
- The server will build a new database file from the log and once completed it will, by default, perform a checkpoint of the transactions to the database file and start listening at the specified port. Virtuoso can be started without a checkpoint using the -n (+no-checkpoint) option.
- You may then connect to the database with interactive SQL and if necessary make a checkpoint. The checkpoint will write freeze the state following roll forward and delete the log used for the rebuild.
- The database is now ready for normal use.

It may sometimes be useful to rebuild a database as above to save space. Virtuoso does not relinquish space in the DB file back to the file system as records are removed, however, Virtuoso does reuse pages that are made available from a deletion of records. The steps above will build a new compact database file. You would ordinarily not have to worry about this.

Diagnosing and Recovering a Damaged Database File

It is possible to recover data from a damaged Virtuoso database by a procedure similar to rebuilding a database as described above. A database file may be corrupt if the database repeatedly crashes during a specific operation.

To determine whether a database is corrupt, you may use the backup to a null file command in isql, for Unix platforms:

```
SQL> backup '/dev/null';
```

For windows platforms you can use:

```
SQL> backup 'NUL';
```

This command will read through the database checking its integrity. If the server crashes before completing the backup process, then the database is indeed corrupt and needs to be recovered. No other activity should take place while the command is executing.

To recover the database, follow the procedure for rebuilding it, except use the -D 'capital D' or +crash-dump switch instead of -b. This will construct a log file which you can replay to make a new database. The database will contain the transactions that were committed as of the last successful checkpoint. If the database altogether fails to open it may be the case that the schema is damaged.

It is possible that the database to be recovered is too large to fit in a single log file. The crash dump feature therefore allows segmenting the recovery log into a number of files. See the virtuoso.ini configuration file documentation for details. It is possible

to make a crash dump in several pieces if there is not enough total disk space to hold the dump on the system where the database is running.

If the recovery log is split over several files it is necessary to set the transaction file in the ini to point to the first of these files, delete the database file(s) and start the server with the +restore-crash-dump option. When the server comes online, one can connect to it with isql and use the replay () function for replaying the remaining logs, one by one, in their original order.

For example,assuming the virtuoso.ini fragment:

```
Log1    = rec-1.log 100M
Log2  = rec-2.log 100M
```

we would make the dump with

```
virtuoso +crash-dump
```

and once the server has been started with +restore-crash-dump, with the ini setting TransactionFile set to rec1.log, replay the remaining log with the isql commands:

```
SQL> replay ('rec-2.log');
Done.
SQL> checkpoint;
```

Note:

If the recovery is interrupted it can be restarted at the last checkpoint made during the recovery. Note that a mid recovery checkpoint may take a very long time, e.g. 1 hour for a 10GB database, since it is possible that the delta since the previous recovery checkpoint comprise almost all the database.

Crash Recovery When The Normal Crash Recovery Fails

When the schema tables (e.g. DB.DBA.SYS_COLS, DB.DBA.SYS_KEYS) have corrupt rows the normal crash dump/crash restore procedure will not be possible because the server relies on the schema tables to know the key layouts for reading the data rows of other tables upon crash dump.

In such situations there is a special procedure to be followed to save as much data as possible from the corrupt database. The general steps are:

dump the intact schema table rows, and read them into a fresh database

read the schema from the fresh database in the normal way

proceed with dumping the rest of the tables from the corrupt database

Thus the transaction log produced from the corrupt database, when replayed on the new database file (the one holding the schema tables data) makes the closest approximation to the corrupt database's data. However, this will not produce a workable database by itself - it may possibly deny inserting of data into tables with IDENTITY columns and will lose all the data within the Virtuoso registry (accessible from `registry_get()` /`registry_set()` functions).

Because of the very nature of the crash-restore process described here and because of the fact that data is lost in the database schema, the server will not attempt to dump tables whose schema description is lost. So care should be taken when reading the data from the database.

This restoration procedure in no way replaces the regular database backup procedures, it merely tries to save whatever reasonable data there may be left from the database file.

The recovery sequence is as follows:

1. Do a crash dump of the schema tables (using the '`+crash-dump +mode oa +dumpkeys schema`' virtuoso command line options).
2. Create a new INI file to describe the layout of the new database you'll use to temporarily fill up the restored data.
3. Move the transaction log file(s) produced in step 1 to the location required by the new INI file.
4. Replay the transaction log from step 1 on an empty database using the new INI file. You will now have the schema tables readable in the new database (and nothing else):

```
Virtuoso options : -c <your new ini file> +restore-crash-dump -f
```

5. Make a crashdump of the data in the non-schema tables of the old database while having read the schema tables from the new database:

```
+crash-dump -c <your new ini file> +crash-dump-data-ini <your old ini file> +mode o -f
```

6. Move the transaction log file(s) produced in step 1 to the location required by the new INI file.
7. Replay the transaction log from the previous step into the new db file using:

```
-c <your new ini file> +restore-crash-dump
```

8. Do a normal crash dump of the new database:

```
-c <your new ini file> +crash-dump
```

9. Move away (backup) the original (old) database files and put the transaction log produced by the above step into the location specified in the original INI file. You can also delete the rest of the DB files of the new database at that point.
10. Replay the transaction log to make the old database afresh.

To automate the above procedure, a sample Unix script follows that automates it somewhat. This script expects the crashed database virtuoso.db.save and an appropriate INI file (no striping, no log segmentation, transaction log file name virtuoso.trx) in the current directory and creates the restored database. It also expects the virtuoso-iodbc-t executable to be in the operating system path. Also, make sure that you have a suitable `virtuoso.lic` license file in the current directory.

```
#!/bin/sh

rm -rf xmemdump.txt virtuoso.trx virtuoso.tdb virtuoso.log virtuoso.db virtuoso.lck core.* new.lck new.lo
cp -f virtuoso.db.save virtuoso.db
cat virtuoso.ini | sed 's/virtuoso\./new./g' > new.ini

virtuoso-iodbc-t -f +crash-dump +mode oa +dumpkeys schema

ls -la *.trx
mv virtuoso.trx new.trx

virtuoso-iodbc-t -c new -f -R
virtuoso-iodbc-t -c new +crash-dump +crash-dump-data-ini virtuoso.ini +mode o -f

ls -la *.trx
mv virtuoso.trx new.trx

virtuoso-iodbc-t -c new -R -f

virtuoso-iodbc-t -c new +crash-dump -f

rm -f virtuoso.trx virtuoso.tdb virtuoso.log virtuoso.db virtuoso.lck
ls -la *.trx
mv new.trx virtuoso.trx

virtuoso-iodbc-t -R -f
```

Crash Recovery Across Virtuoso VDBMS Server Versions

If the database was created with a version prior to the one being used for rebuilding, the system tables may be different. The creation here refers to the first time the database was made, a crash recovery does not count as a fresh start here.

If this is or may be the case, the first log must be rolled forward into the empty database BEFORE the new and possibly incompatible system tables are created. This is done by setting the TransactionFile parameter to the first of the recovery logs and starting the server with the -R or +restore-crash-dump switch. For good practice one should also specify the no checkpoint switch, so that the log will in no case be damaged after the initial step of the roll forward. After this initial step the system tables will be compatible and the dba can proceed to replay the remaining recovery logs with the replay function.

Backup and Restore individual table(s) and individual index(s) on a new fresh db

This section describes how to get a part of db tables and restore on a new db.

Additionally, part of the steps from below can be used to backup separate table and recover on same db.

Note: Only effective with Virtuoso 6.0 and later.

Basic steps:

1. Create a function for dumping the key:

```
create procedure bkp_key (in f any, in tb_name varchar, in key_name varchar)
{
  backup_prepare (f);
  backup_index (tb_name, key_name);
  backup_flush ();
  backup_close ();
}
;
```

2. On the source db execute:

```
bkp_key ('mylog.txn', 'DB.DBA.T1', 'T1');
```

   This will dump in a "mylog.txn" file the T1 table's primary key.
3. On source db stop server and do:

```
virtuoso +backup-dump +foreground +mode l   ## ( lower case L )
```

   This will dump the schema tables only into the trx file.
4. On the target db make sure there is no db file and place the trx file produced by previous step. Execute:

```
virtuoso +restore-crash-dump +foreground ;
```

   This will create a new db with same db schema as on the source db.
5. Start the target and do:

```
replay ('mylog.txn') ;
```

   This will insert the PK data into the table from the source db dump.
6. If the table in question has other indexes must drop them and re-create them, since they are empty as in previous step we have been restoring only the PK.

Note : following the steps from above can be dumped each index and then replay. Also the steps may be combined for multiple tables and keys in the backup procedure - just needs to be added the corresponding calls to the backup_index() function.

6.1.6. Performance diagnostics

This section provides a checklist for improving performance of a Virtuoso server.

If something does not work fast, this is mostly for the following reasons:

- Not enough memory
- Not the right indices, missing statistics
- Too much locking or too many threads on the same data
- Bad disk layout, for example not striped or not enough file descriptors.

Determining which is the case is simple. The result set returned by status (') has most of the information. Do this twice with some 10 seconds between the samples and see the second result set.

Memory

If there is not enough memory, there will be frequent disk access. This is seen from the buffers and disk usage lines.

The very simplest test for this is looking at the CPU % of the process in top. If there is constant load and the percent is low then the server is IO bound.

If all memory is not in use, then memory cannot be the problem. This is seen from the buffers line. If the used number is under 80% of the total or if the replace age is several times larger than the count of buffers, then things are OK. If the replace age is 0 then no buffers have ever been replaced and all that ever was read is still in memory.

If the replace age is less than or close to the buffer count times 4, then cache replacement is frequent and adding buffers is advised.

Adding more than 60-70% of system ram as buffers is not useful. The setting is NumberOfBuffers in the ini file, count 9K per buffer.

The disk access is summarized on the disk usage line. First is the number of reads since start then the average latency in N ms. If %r is high, then a lot of the time between the previous status and this is taken by disk. This can be over 100. One thread that is waiting for disk all the time counts for 100. If the percent is high then adding more disks and striping over them will be useful. Even with a single database file, adding file descriptors (FDsPerFile setting in the ini) may be useful. If the average read latency is 0 or close, then the data is cached by the OS. If it is high, then adding disks and striping may reduce it.

The read ahead line will tell if there are sequential reads. These are faster than random ones and can efficiently use striped disks.

If the workload is random access, then a high number of read ahead means that one might not have the required indices, thereby causing full table scans. More on this in the query plans section below.

Swapping

Swapping is always bad. If swapping occurs, then one has too many buffers and should decrease the number of buffers. Use an OS tool like top to see the size of the database process and its virtual memory use. Having a resident size smaller than logical size is not always bad since some code or data in the process may simply be unused but having, after running in a steady state, i.e. all buffers used, a resident size less than the amount of memory allocated for buffers is always bad. Before steady state, i.e. during cache warm-up, the resident size is normally less than the buffer pool's size.

To see the count of major page faults. i.e. ones that read the disk, do:

```
getrusage ()[]4];
```

through interactive SQL. The result is the count of major faults since starting the process. The count should not vary between samples, at least not more than a few units per minute.

This function is not available under Windows. Use the task manager instead for tracking this.

There can be an actual memory leak, specially with plugins or hosting. See the growth of the virtual size throughout the run, reaching full buffer utilization and doing a couple of checkpoints. Past this, the process should not grow. You may also see if the ThreadCleanupInterval or ResourcesCleanupInterval ini parameters have an effect. If in spite of all the process grows there can be a genuine leak. It is normal for the size to fluctuate somewhat due to varying amounts of uncommitted data, threads, connections and the like.

Locking

The lock status section indicates the count of deadlocks and waits. If the 2r1w number is high then it means that the application should read for update. It gets shared locks and cannot change them to exclusive. Adding the for update option to selects in the right place will fix this, also setting DefaultIsolation in the ini to 2 for read committed will be good. If the lock wait count increases fast, then locking may downgrade performance. The threads line below shows how many threads have some task to do. The waiting is the number of these threads that are waiting for a lock at the time of the status. The vdb number is the number of threads waiting for network io, either from virtual database or any web protocol.

Take a few samples and if these show few threads waiting and the waits or deadlocks counts do not climb much locking will not be a problem. If these numbers are high, see the sys_l_stat view for details.

```
select sum (waits), sum (wait_msecs) from sys_l_stat;
```

for the totals. The first number is the count of times a thread waited for a lock, the second is the sum of the real times spent waiting.

```
select top 10 * from sys_l_stat order by waits desc;
```

shows the keys on which most waits take place. See also the deadlocks and the wait_msecs columns. Numbers are cumulative.

For details on disk, see:

```
select top 10 * from sys_d_stat order by reads desc;
```

These views are explained in more detail in the performance meters section.

If there is a multi-user application doing random access for read or write, it is an idea to partition the data so that they do not hit the same page all the time. For example, to allow for parallel insert of orders without contention, the TPC C schema prefixes the order number with a warehouse and district number, so that simultaneous inserts will most often not hit the same page.

Even when there is no locking, there is still some serialization for database page access. This is shown in the various wait columns of SYS_L_STAT. If the sum of the waits for a key is over 1% of the touches for the key, as given in SYS_D_STAT, there is contention and a performance penalty of maybe 10%.

Such things can be improved by altering the schema design and configuration parameters do not usually help. If there is disk access, having more memory always helps because then locks will be in effect for less time.

Query Plans

To know if there is a bad query plan, see the explain output of the query.

Unless a full table scan is intended, a full table scan is pretty much always bad.

In the explain () result set, a table access is marked by a From table name. Below is mentioned the driving index, the conditions on the index and the conditions that are not indexable.

Joins are listed one after the other, the outermost first. The join order can be read from the explain output going from top to bottom.

If a table is read on a non primary key and columns not covered by the driving index are accessed, there is a second key mentioned with a full match of the primary key columns as condition.

A full table scan looks like this:

```
explain (select count (*) from t1');

...
from DB.DBA.T1 by STR2 8.4e+04 rows
Key STR2 ASC ()

 Local Code
  0: $30 "count" := artm $30 "count" + <constant (1)>
...
```

There are no conditions mentioned.

A lookup with index looks like this.

```
explain ('select fi2 from t1 where row_no = 11');
...
from DB.DBA.T1 by T1 Unique
Key T1 ASC ($26 ".FI2")
 inlined <col=1694 ROW_NO = <constant (11)>>
...
```

The condition is shown on the line below the key.

A lookup with full scan testing every row looks like this:

```
...
from DB.DBA.T1 by T1 2.5e+04 rows
Key T1 ASC ($26 ".FI2")

row specs: <col=1699 FI2 > <constant (11)>>
...
```

The condition is shown after the heading row specs. The whole key mentioned in the key will be read and the entries tested. if both indexed and row tests exist, the indexed are done first as one would think.

If your query has full table scans, consider adding an index.

If the index choice is not the right one, consider the following possibilities:

- Run statistics. This is not always necessary because the database takes dynamic samples but it can help in cases.
- specify the desired index explicitly with table option.
- Use literal constants instead of parameters, specially if the query is long running and values of the columns being compared are not evenly distributed.
- To exclude index usage for a column, write col + 0 =xx.

Hash joins can make full table scans. This is OK if the table scanned is small.

A hash join looks like this:

```
explain ('select count (*) from t1 a, t1 b where a.row_no = b.row_no + 1');
...

{
Fork
{
Fork
{
from DB.DBA.T1 by STR2 8.4e+04 rows
Key STR2 ASC ($26 "B.ROW_NO")

 Local Code
  0: $30 "temp" := artm $26 "B.ROW_NO" + <constant (1)>
  4: BReturn 0

Current of: <$28 "<DB.DBA.T1 B>" spec 5>
Sort (HASH) ($30 "temp") -> ($26 "B.ROW_NO")

}
from DB.DBA.T1 by STR2 8.4e+04 rows
Key STR2 ASC ($37 "A.ROW_NO")

Current of: <$39 "<DB.DBA.T1 A>" spec 5>
Hash source ($37 "A.ROW_NO") -> ($26 "B.ROW_NO")

After code:
  0: $44 "count" := artm $44 "count" + <constant (1)>
  4: BReturn 0
}
Select ($44 "count", <$39 "<DB.DBA.T1 A>" spec 5>, <$28 "<DB.DBA.T1 B>" spec 5>)
}
```

First t1 is read from start to end and a hash is filled with row_no + 1. Then t1 is read from start to end a second time. The hash source is the hash lookup. A hash join, if there is no index or if the whole table or a large part thereof is traversed is better than a loop join because it replaces random access with sequential. The complexity is close to O(n + n) instead of O (n * log n). Plus sequential read makes better use of read ahead.

The corresponding loop join looks like this:

```
explain ('select count (*) from t1 a, t1 b where a.row_no = b.row_no + 1 option (loop)');

{
Fork
{
from DB.DBA.T1 by STR2 8.4e+04 rows
Key STR2 ASC ($26 "B.ROW_NO")

Current of: <$28 "<DB.DBA.T1 B>" spec 5>

Precode:
  0: $29 "temp" := artm $26 "B.ROW_NO" + <constant (1)>
```

```
  4: BReturn 0
from DB.DBA.T1 by T1 Unique
Key T1 ASC ()
 inlined <col=1694 ROW_NO = $29 "temp">
 Local Code
  0: $35 "count" := artm $35 "count" + <constant (1)>
  4: BReturn 0

Current of: <$31 "<DB.DBA.T1 A>" spec 4>
}
Select ($35 "count", <$31 "<DB.DBA.T1 A>" spec 4>, <$28 "<DB.DBA.T1 B>" spec 5>)
}
```

To prevent hash joins, use table option or option at the end of the select as seen above. A hash join is very bad if a whole table is read for filling a hash and then only a small number of entries are fetched from the hash. However, if there is no index, then even this is better than a loop join.

Checkpoint Duration

A checkpoint can take a very long time in certain special conditions. Normally a log checkpoint is about a minute if flushing several G worth of buffers to disk. The flushing is mostly done online, after which there is an atomic time of a few seconds. After this operation resumes. Applications do not notice the checkpoint except as a temporary increase in response delays.

If spikes in response time are to be avoided, then making frequent checkpoints is better than making infrequent ones.

If there are long running transactions with many locks or uncommitted changes, then the checkpoint interval should anyhow be longer than several times the expected duration of such a transaction. If this is not so, the checkpoint can fall in the middle of the transaction and will internally have to rollback and again reestablish the uncommitted state so as to write a clean image on the new checkpoint. Doing this many times in the life of a transaction is very inefficient. In generally do not make long transactions with locking. Preferentially use read committed for anything long.

A checkpoint's atomic time can be prohibitively long under the following circumstances:

- There are transactions with a lot of locks and uncommitted state at the time of the checkpoint. Specially a transaction which has updated pages paged out to disk is bad. In general one should not do transactions that update a large part of the disk cache, let alone exceed its size. Use the row autocommit mode (log_enable (2)) for doing batch updates. Batch updates most often do not require isolation anyway. If they do, then break the job into smaller transactions.
- The MaxCheckpointRemap parameter in the ini is too small and the database exceeds the size of disk cache either in the Virtuoso process or the OS. This may increase the count of used pages by the value of this parameter but will make for faster checkpoints. Set this to up to 25% of the database page count.

6.1.7. Performance Tuning

I/O

Optimizing Disk I/O

Virtuoso allows splitting a database over several files that may be on different devices. By allocating database fragments onto independent disks I/O performance in both random and sequential database operations can be greatly enhanced.

The basic unit of a database is the segment. A segment consists of an integer number of 8K pages. A segment may consist of one or more files called stripes. If a segment has multiple stripes these will be of identical length and the segment size will be an integer multiple of the stripe size.

The size limit on individual database files is platform dependent, but 64 bit file offsets are used where available. For large databases use of multiple disks and segments is recommended for reasons of parallelism even though a single database file can get very large. A database can in principle grow up to 32TB (32-bit page number with 8KB per page).

When a segment is striped each logically consecutive page resides in a different file, thus for a segment of 2 stripes the first stripe will contain all even numbered pages and the second all the odd numbered pages. The stripes of a segment should always be located on independent disks.

In serving multiple clients that do random access to tables the server can perform multiple disk operations in parallel, taking advantage of the independent devices. Striping guarantees a statistically uniform access frequency to all devices holding stripes of a segment.

The random access advantages of striping are available without any specific configuring besides that of the stripes themselves.

Configuring I/O queues

Striping is also useful for a single client doing long sequential read operations. The server can detect the serial nature of an operation, for example a count of all rows in a table and can intelligently prefetch rows.

If the table is located in a striped segment then the server will read all relevant disks in parallel if these disks are allocated to different I/O queues.

All stripes of different segments on one device should form an I/O queue. The idea is that the database stripes that benefit from being sequentially read form a separate queue. All queues are then read and written independently, each on a different thread. This means that a thread will be allocated per queue. If no queues are declared all database files, even if located on different disks share one queue.

A queue is declared in the striping section by specifying a stripe id after the path of the file, separated by an equal sign.

```
[Striping]
Segment1 = 200M, disk1/db-seg1-1.db = iq1, disk2/db-seg1-2.db = iq2
Segment2 = 200M, disk1/db-seg2-1.db = iq1, (disk2/db-seg2-2.db = iq2
```

In the above example the first stripes of the segments form one queue and the second stripes form another. This makes sense because now all database files on /disk1 are in iq1 and all on /disk2 are on iq2.

This configuration could have resulted from originally planning a 200 MB database split on 2 disks and subsequently expanding that by another 200 MB.

The I/O queue identifier can be an arbitrary string. As many background I/O threads will be made as there are distinct I/O queues.

Striping and using I/O queues can multiply sequential read rates by a factor almost equal to the number of independent disks. On the other hand assigning stripes on one disk to different queues can have a very detrimental effect. The rule is that all that is physically accessed sequentially will be on the same queue.

Schema Design Considerations

Data Organization

One should keep the following in mind when designing a schema for maximum efficiency.

Index Usage

A select from a table using a non-primary key will need to retrieve the main row if there are search criteria on columns appearing on the main row or output columns that have to be fetched from the main row. Operations are noticeably faster if they can be completed without fetching the main row if the driving key is a non-primary key. This is the case when search criteria and output columns are on the secondary key parts or primary key parts. Note that all secondary keys contain a copy of the primary key. For this purpose it may be useful to add trailing key parts to a secondary key. Indeed, a secondary key can hold all the columns of a row as trailing key parts. This slows insert and update but makes reference to the main row unnecessary when selecting using the secondary key.

A sequential read on the primary key is always fastest. A sequential search with few hits can be faster on a secondary key if the criterion can be evaluated without reference to the main row. This is because a short key can have more entries per page.

Space Consumption

Each column takes the space 'naturally' required by its value. No field lengths are preallocated. Space consumption for columns is the following:

Table 6.2. Data type Space Consumption

Data	Bytes
Integer below 128	1
Smallint	2
long	4
float	4
timestamp	10
double	8
string	2 + characters
NULL	data length for fixed length column, as value of 0 length for variable length column.
BLOB	88 on row + n x 8K (see note below)

If a BLOB fits in the remaining free bytes on a row after non-LOBs are stored, it is stored inline and consumes only 3 bytes + BLOB length.

Each index entry has an overhead of 4 bytes. This applies to the primary key as well as any other keys. The length of the concatenation of the key parts is added to this. For the primary key the length of all columns are summed. For any other key the lengths of the key parts plus any primary key parts not on the secondary key are summed. The maximum length of a row is 4076 bytes.

In light of these points primary keys should generally be short.

Page Allocation

For data inserted in random order pages tend to be 3/4 full. For data inserted in ascending order pages will be about 90% full due to a different splitting point for a history of rising inserts.

Efficient Use of SQL - SQL Execution profiling

Virtuoso offers an execution profiling mechanism that keeps track of the relative time consumption and response times of different SQL statements.

Profiling can be turned on or off with the prof_enable function. When profiling is on, the real time between the start and end of each SQL statement execute call is logged on the server. When prof_enable is called for the second time the statistics gathered between the last call to prof_enable and this call are dumped to the virtprof.out file in the server's working directory.

Profiling is off by default. Profiling can be turned on with the statement:

```
prof_enable (1);
```

The virtprof.out file will be generated when prof_enable is called for the second time, e.g.

```
prof_enable (0);
```

will write the file and turn profiling back off.

Below is a sample profile output file:

```
Query Profile (msec)
Real 183685, client wait 2099294, avg conc 11.428772 n_execs 26148 avg exec  80

99 % under 1 s
99 % under 2 s
99 % under 5 s
100 % under 10 s
100 % under 30 s

23 stmts compiled 26 msec, 99 % prepared reused.

 %  total n-times n-errors
49 % 1041791  7952     53     new_order
```

```
28 % 602789    8374      490    delivery_1
12 % 259833    8203      296    payment
5  % 123162    821       35     slevel
2  % 54182     785       0      ostat (?, ?, ?, ?)
0  % 11614     4         0      checkpoint
0  % 2790      2         1      select no_d_id, count (*) from new_orde
0  % 2457      3         1      select count (*) from new_order
0  % 662       2         0      status ()
0  % 11        1         1      set autocommit on
0  % 3         1         0      select * from district
```

This file was produced by profiling the TPC C sample application for 3 minutes. The numbers have the following meanings:

The real time is the real time interval of the measurement, that is the space in time between the prof_enable call that started the profiling and the call that wrote the report. The client wait time is the time cumulatively spent inside the execute call server side, only calls completely processed between profiling start and end are counted. The average concurrency is the exec time divided by real time and indicates how many requests were on the average concurrently pending during the measurement interval. The count of executes and their average duration is also shown.

The next section shows the percentage of executes that finished under 1, 2, 5, 10 and 30 seconds of real time.

The compilation section indicates how many statements were compiled during the interval. These will be SQLPrepare calls or SQLExecDirect calls where the text of the statement was not previously known to the server. The real time spent compiling the statements is added up. The percentage of prepared statement reuses, that is, executes not involving compiling over all executes is shown. This is an important performance metric, considering that it is always better to use prepared statements with parameters than statements with parameters as literals.

The next section shows individual statements executed during the measurement interval sorted by descending cumulative real time between start and completion of the execute call.

The table shows the percentage of real time spent during the calls to the statement as a percentage of the total real time spent in all execute calls. Note that these real times can be higher than the measurement interval since real times on all threads are added up.

The second column shows the total execute real time, the third column shows the count of executes of the statement during the measurement interval. The fourth column is the count of executes that resulted in an error. The error count can be used for instance to spot statements that often produce deadlocks.

Statements are distinguished for profiling purposes using the 40 first characters of their text. Two distinct statements that do not differ in their first 40 characters will be considered the same for profiling. If a statement is a procedure call then only the name of the procedure will be considered, not possibly differing literal parameters.

The profiler will automatically write the report after having 10000 distinct statements in the measurement interval. This is done so as to have a maximum on the profiling memory consumption for applications that always compile a new statement with different literals, resulting in a potentially infinitely long list of statements each executed once. It is obvious that such a profile will not be very useful.

It cannot be overemphasized that if an application does any sort of repetitive processing then this should be done with prepared statements and parameters, for both reasons of performance and profilability.

Note:

Note that measurements are made with a one millisecond precision. Percentages are rounded to 2 digits. Timing of fast operations, under a few milliseconds, will be imprecise as a result of the 1 ms resolution. Also the cumulative compilation time may be substantially off, since the compilation may often take less than 1 ms at a time. Also note that the precision may also vary between platforms.

Reading a Query profile

A query plan is essentially a pipeline of operations that can be read from top to bottom. The operator above produces the input for the operator below. Nesting is indicated by braces. Operators enclosed in braces must completely process their input before continuing with operators after the section in braces. A case in point is aggregation which is demoted by for { ... } ... . The section in braces generates the rows to be aggregated and one it is fully evaluated the execution proceeds with the operators after the fork. These will typically read the temporary table filled by the operators inside the fork.

Query variables are demoted with <$v...>. A reference to a query variable set by an operator that is not immediately the preceding one is demoted by <r $...> via ...>. The position of the variable in the query state follows the $ sign.

After this is given the symbolic name of the variable, e.g. a column name or the name of the function of which this is the return value.

If this is a reference to a query variable assigned earlier in the plan with one or more operators between the reference and the assignment, the reference contains a via .... section which lists the operators that are between the reference and the assignment.

The operators are identified by a unique number. Internally this is the location in the query state where the operator keeps the mapping between its rows of input and output.

With vectored execution, an operator may receive any number of rows of input which it may reorder at its convenience. For this it must record for each row of output which row of input this corresponds to. Thus when referencing a query variable set earlier in the plan one must take into account the reordering and addition/dropping of rows in affected by the operators between the reference and the initial assignment. This is the meaning of the via section in the <r $xx > notation.

The last part of the query variable reference is a two letter indication of the type. If the first letter is followed by n this means that the variable is non-null. The type letters are a for any, r for IRI, i for integer, d for double, f for float, t for datetime, N for wide string, s for string.

If the plan is a profile, i.e. it is annotated with actual times and cardinalities from running the query, each operator is preceded by a passage that indicates its percentage of total CPU time of the plan, its fanout and the number of rows of input. The fanout is the number of rows of Output divided by the number of rows of input.

The principal operators in a plan are:

- *Cluster outer seq start*

  -- Queries and subqueries begin with this operator. It assigns numbers to the sets of parameter bindings the query gets as input. An optional section also begins with this. An optional section is the table or derived table on the right hand side of a left outer join.
- *From*

  -- Accessing an index
- *Sort*

  -- Group by, order by, distinct or hash join build side
- *group by read*

  -- Accessing the results of a group by filled in by a previous sort operator.
- *Top oby*

  -- Accessing a previously filled top k order by temporary space
- *Select*

  -- At the end of a plan, send the rows of input to the client. If Subq select return the rows of input as rows of output of the enclosing subquery.
- *Subquery*

  -- Indicates a SQL derived table, i.e. select in a from clause. The input is the output of the previous node.
- *Cluster ocation fragment*

  -- A set of operators partitioned and shipped to a partition in a cluster. There are single part fragments and distributed fragments (DFG). A DFG is a query fragment which begins with a stage operator. The stage is a partitioning exchange operator which routes its rows of input to the next operator in the appropriate partition based on the partitioningh columns of the row of input.
- *Outer seq end*

  -- This demotes the end of an optional section, i.e. the table/derived table on the right side of left outer join. This has the effect of making aligned vectors of all the vector query variables assigned in the optional section. When producing rows

for which there was no match in the optional section, these will be set to null. References to these variables downstream in the plan will be to the shadowing variables, listed in the shadow clause following this.

Some operators such as index access (From) have a section called vector param casts. This is used when the vector that serves as parameter is not aligned or not of the type required by the operator. For example, a vector of column values in an index lookup might not be of the type of the column but might be of a type castable to this. In any case, the para,key values for index lookup are made into solid aligned, materialized vectors. After this there is a sort step that orders the input according to key order, so that matches are generated from left to right.

In the event of a cluster partitioned operation, i.e. stage node or non-DFG location fragment (QF), the first table source (From) after the stage or fragment serves to both align and cast all inputs needed downstream in the DFG or QF. The operator first partitions the rows of input. Each partition then gets solid, aligned vectors of input values which are directly suitable for input to the first index lookup (From).

We consider an example: The setup is a cluster of 4 server processes with each 8 slices of data. There is one table, ct_c, which is joined to itself:

```
profile ('SELECT COUNT (*) from ct_c a, ct_c b WHERE a.k2 = b.k1 option (loop, order)');
```

This returns a result set in which is found the below. This is also in the ql_plan column of the sys_query_log view:

```
{
-- In the below excerpt some lines are omitted. Comments are inline.

time   1.3e-06% fanout         1          1 rows in

s# 98 cluster outer seq start, set no <V $40 set_ctr in>
save ctx:()

-- the section inside the fork is executed to the end
-- before the section after the fork is started.
s# 161 Fork 42
{
wait time   0.00012% of exec real time, fanout         0

-- The below introduces the 2 stage DFG which does most of the work.
s# 108   { Cluster location fragment 51 0 unordered
   Params: ()
Output: ()

-- First partitioning step. Each of the 32 slices gets the
-- identical plan to execute, this is a flood for all is this
-- first step does not specify a partitioning key value.

time     8e-05% fanout         0          0 rows in
s# 115 Stage 1: Params: (<V $112 set_ctr in>)

-- This is the initial table scan. The operator is executed
-- 32 times, once for each slice of data.

time      0.33% fanout   3.1e+06         32 rows in
s# 121 from DB.DBA.ct_c by ct_c    1.7e+08 rows
Key ct_c  ASC  (<V $28 a.k2 in>)
 [copies params]
vector param casts: <V $40 set_ctr in>-> <V $112 set_ctr in>

-- This is the second partitioning step, each of the rows of
-- output of the previous is sent to the slice corresponding to K2.

time        31% fanout       0.9      2e+08 rows in
s# 131 Stage 2: Params: (<v $134 q_a.k2 S136 in>, <V $138 set_ctr in>)

-- This is the 2nd index lookup, by a.k2 = b.k1 There are two
-- parameters given in the vector param cast section, the first
-- is a.k2 and the second is the set number. This is constant
-- since the query is run on a single row of input but if the
-- query were run on multiple rows of input this would specify
-- which row of input the a.k2 comes from so that this would be
-- aggregated in the right place. Many aggregates can be produced
```

```
-- in a single invocation with multiple rows of input.

time        68% fanout         0      2e+08 rows in
s# 144 from DB.DBA.ct_c by ct_c  unq         1 rows
Key ct_c  ASC  ()
 inlined  k1 = <v $134 q_a.k2 S136 in>
vector param casts: <V $28 a.k2 in>-> <v $134 q_a.k2 S136 in>, <r $112 set_ctr via  S121>-> <V $138 set_c

-- This is the aggregation.

After code:
      0:  sum <V $43 count i> 1 set no <r $138 set_ctr via  S144>
      5: BReturn 0

}
}

-- The below returns the aggregate to the client. The combining of
-- aggregates from different slices is implicit in the for operator
-- whenever this ends with a cluster location fragment.

time   5.4e-07% fanout         0          1 rows in
s# 164 Select (<V $43 count i>)
  set no: <V $40 set_ctr in>
}
```

Tuning Vectored Execution and Hash Joins

Note : This section applies only to versions 7.00 and up.

Query evaluation performance is significantly affected by parallelization, the vector size and the use of hash joins and fast in-memory hash tables for group by and distinct.

This section explains the configuration parameters and event counters that allow optimizing these factors.

The SQL function *cl_sys_stat* (in name varchar, in clr int := 0) allows reading and optionally resetting these counters. In the case of a cluster, the value returned is the sum of the values of the metric gathered from all server processes, for a single server this is the local value. In a cluster, to get individual counter values, use sys_stat instead when connected to the process of interest.

To do TPC H at scale 100G on a 32 thread machine, the virtuoso.ini should have the following settings. Only settings at non-default values are mentioned:

```
[Parameters]
ServerThreads              = 100
NumberOfBuffers            = 5000000

;  The working set of the 100G database when stored column-wise is 38G,
;  so configure this much for database buffers:
; 5000000 / 128             = 39G

;  Read committed
DefaultIsolation           = 2

;  SQL optimizer work space
MaxMemPoolSize             = 100000000

;  Use large vectors when appropriate
AdjustVectorSize           = 1

;  Split queries in up to 32 independently evaluable fragments,
;  so up to 32 threads per query.
ThreadsPerQuery            = 32

;  Thread pool has 32 worker threads divided across all queries
;  in addition to the client thread taken by each query.
AsyncQueueMaxThreads       = 32

;  All queries can collectively take up to 30G without space
;  saving measures being applied. Space saving measures are
;  running on small vector size when a large size would be
```

```
;  faster and the use of partitioned (multiple pass) hash join.
MaxQueryMem               = 30G

;  Of the MaxQueryMem, hash join hash tables can take up to 30G,
;  i.e. all of it. A single hash table will only take half of the
;  remaining space, though. So if 2 queries that both need a 30G
;  hash table run at the same time, the first will do 2 passes,
;  taking a hash table of 15G at a time, This leaves 20G space.
;  The second will have 15G left, of which it will take half,
;  7.5G. This will require 4 passes over the data.
HashJoinSpace             = 30G
```

To analyze the performance of a query workload:

1. Turn on query logging in sys_query_log. This view contains most metrics and the full text of the query plan with the per operator timings and cardinalities:

   ```
   SQL> prof_enable (1);
   ```
2. Print long result columns without truncating:

   ```
   SQL> set blobs on;
   ```
3. Run the query.
4. Use the profile function for a convenient overview of query execution. For example:

   ```
   SQL> profile ('SELECT COUNT (*) FROM orders, lineitem WHERE l_orderkey = o_orderkey');
   ```

   See summary of execution time, CPU%, compilation time and IO
5. Read the relevant event counters, resetting the count for the next query. For example:

   ```
   SQL> select cl_sys_stat ('tc_qp_thread', clr => 1);
   ```

   The relevant counters are:

   - *tc_qp_thread* -- How many threads were started for query parallelization. This is not the number of parallel threads as not all of these threads needed to be running at the same time.
   - *tc_part_hash_join* -- If a hash join is partitioned, i.e. needs to make multiple passes on over the data, this is the count of passes. This is incremented by 2 if the hash join does 2 passes and not incremented if the hash join goes in a single pass. Normally this should stay at 0 or the hash join space (HashJoinSpace in init, see above) should be increased.
   - *tc_no_mem_for_longer_batch* -- This is the count of times the execution engine did not switch to large vectors because there was not enough space. This should normally be 0, if this is not so, increase MaxQueryMem in the ini file.
   - *tc_slow_temp_insert* -- If a distinct or group by temporary space grows over the available query memory, a another data structure will be used so that the hash table can be paged out to disk. This is tens of times less efficient than the memory only structure. This counter is the count of rows inserted into a page-able group by or distinct hash table. This should be 0, if not, increase MaxQueryMem.
   - *mp_max_large_in_use* -- This is the maximum amount of query memory that has been allocated to date. Reset this before the query of interest, run and read the counter. This is the peak simultaneous memory use by the query.
   - *mp_large_in_use* -- This is the current amount of query memory in use. Do not reset this.
   - *c_max_large_vec* -- This is the MaxQueryMem init setting in bytes. This can be altered at run time with `__dbf_set` .

High Cardinality Group By And Distinct

There are multiple implementations of group by and distinct used in different circumstances by different versions of Virtuoso. Versions of Virtuoso 7 prior to 7.5 have a memory based cuckoo hash based group by and hash join. 7.5 and onwards have a linear hash based implementation of same. Additionally, all version 7's have a pageable, i.e. potentially disk based implementation of group by that may get used if there is no space in memory.

Partitioned Group By

Versions 7.5 and onwards have a choice of a partitioned or re-aggregated parallel group by. The group by with re-aggregation produces an intermediate result on each thread and then adds these up after the completion of each thread into a final result. This is efficient for low-cardinality cases (few distinct values of grouping columns) but inefficient for high cardinality. In the high cardinality case, it is better to use a partitioned group by operator. In the partitioned case, a hash is computed from the grouping columns and this decides which thread gets to do the aggregation. There is information exchange between threads (or processes in a scale-out setting) but there is no need for re-aggregation. This is significantly more performant with many groups. The peak memory utilization is the same.

The `c_setp_partition_threshold` setting in the *Flags* section of virtuoso.ini defines when to use a partitioned group by. The default threshold is 100000. If the cost model estimates more distinct values than this threshold, a partitioned group by will be produced. In a query plan with explain of profile, a partitioned group by is present when there is a stage <n> operator in front of the sort operator for the group by. In a cluster plan, the partitioned group by may be colocated with the table right before it, so the stage operator will be before this.

Ordered Group By

For version 7.5 onwards, a special case for ordered group by is recognized. In an ordered group by, the system takes advantage of physical data ordering to limit the memory footprint of the group by: If it is known that grouping keys will not reoccur past a certain window, then all grouping keys that are processed can be emitted as results in a continuous stream so that only a window of groups need be kept at a time.

This is recognized in a query plan by the word "streaming" occurring on the line of the sort operator for the group by.

Ordered group by is rare to non-existent with RDF, where data orderings with an application meaning do not occur, due to having no multipart keys. With a SQL schema grouping on a set of columns including the primary ordering column of the outermost scan in a plan, an ordered group by will typically be generated. An ordered group by is usually better than a partitioned or re-aggregated group by. The exception may be cases where the outermost ordering column has low cardinality, or a very biased value distribution e.g. the P in the RDF PSOG.

Memory for Group By

In Virtuoso 7 prior to 7.5, the memory for a group by is `(3 + n_keys + n_aggregates) * 8` bytes per group plus the natural length of any dates or strings in the keys or aggregates. In 7.5 onwards, the memory is 8 bytes less per entry.

Slow Group By

The use of the disk-based group by is easily 100x less efficient than the vectored memory based hash tables. Falling back to this can be considered an error in configuration and can be detected with the `tc_slow_temp_insert` counter readable with `sys_stat` . If this increases then there is not enough memory for group by/distinct as used by the application. The monitor may print a message about this in the server messages log.

The maximum size of a memory based group by hash table is by default 1/10 of the MaxQueryMem setting in Parameters in the virtuoso.ini file. If a single hash table exceeds this or if the hash table plus the amount of large query memory blocks (e.g. vectors) allocated across the process exceeds MaxQueryMem, then the switch to the slow pageable hash table takes place. This can be avoided by explicitly setting a larger MaxQueryMem or by changing the `cha_max_gby_bytes` setting in the Flags section of the ini file. We note that for parallel query plans the first condition to trigger is the condition on `mp_large_in_use` + the hash table memory exceeding MaxQueryMem. A query will have more than 10 threads and the hash tables for memory based hashes are included in `mp_large_in_use` .

To track the general memory consumption in large blocks, use the `mp_large_in_use` and `mp_max_large_in_use` `meters` accessible with `sys_stat` and `__dbf_set` . Large blocks are for example vectors, group by's and hash join build sides.

To interactively try different memory settings, one can modify `c_max_large_vec` with `__dbf_set` . This is the number of bytes of outstanding large blocks after which pageable group by's will be used and vector sizes will no longer be adjusted upward.

Tuning Parameters for Multiuser Workloads

This section describes parameters and event counters that affect memory and parallelism and are specially relevant for high concurrency situations. Different switches and event counters are described. Setting these requires dba privileges. The event counters are accessed with *sys_stat* and set with `__dbf_set` . For example:

```
select sys_stat ('tc_qp_thread');

or

__dbf_set ('dc_batch_sz', 10000);
```

Some of the parameters have corresponding INI file settings. All of these can be set in an INI file in the [Flags] section with a stanza like:

```
dc_batch_sz = 10000
```

First, make sure that the *ServerThreads* in either [Parameters] ini section (if application with connected clients) or in [HTTPServer] ini section (if over web protocols), is larger than the number of actual connections.

Next, the workload should be profiled to see its memory utilization and intrinsic use of parallelism. Splitting a query into independently executable parallel fragments increases its memory utilization, Switching to larger vector sizes also increases a query's memory utilization while providing increased locality of data access. Together these techniques can result in a 100+ fold difference in the transient memory consumption of a query. Use of hash joins may also increase the transient memory consumption since in a hash join one half of the join must be materialized.

With relatively small numbers of concurrent queries, e.g. 10, these techniques are nearly always beneficial. With hundreds of queries the memory overhead may become a limiting factor.

The *MaxQueryMem* INI file setting controls how much memory is kept for query execution. This amount of space is maintained allocated. Transient memory use may exceed this but then the memory above this amount is unmapped when no longer needed. Mapping and unmapping memory takes time. Concurrent mmap and munmap calls on different threads will serialize and bring down the CPU utilization of a process even when there is enough parallelism for full platform utiliization.

The *mp_mmap_clocks* counter is the cumulative amount of real time in CPU clocks spent in mmap or munmap systtem calls. This can increase faster than real time if multiple threads are involved.

If this value grows fast, e.g. at over 10% of real time, the monitor prints a message in the server event log. This is an indication that memory utilization may have to be tuned.

For analyzing the behavior of a workload as concerns memory, do the following:

- Run the workload once for warmup.
- Clear the following counters:
  - `mp_max_large_in_use` -- Highest to date amount of concurrently used memory
  - `tc_qp_thread` -- count of times a query made a parallel executive fragment. Not increased if running single threaded.
  - `tc_adjust_batch_sz` -- count of times vector size was switched to larger
  - `mp_mmap_clocks` -- cumulative time inside mmap or munmap.

  For example:

  ```
  __dbf_set ('mp_max_large_in_use', 0);
  ```
- Run the workload on a single thread with default settings.
- Observe the values.

Example Scenario

1. Clear the counters.
2. Set `enable_dyn_batch_sz` to 0, causing queries to only use the initial vector size, `dc_batch_sz` or `VectorSize` in the INI file.
3. Set `enable_qp` to 1, causing queries to run single threaded.

4. Run the workload on a single thread. The difference in elapsed real time will show how much benefit the workload has from intra-query parallelization and long vector sizes. If the workload used any of the memory consuming techniques the peak memory in the second case will be lower. How much lower is entirely workload dependent.

    - *Note* : One may additionally try the workload on a single thread with `hash_join_enable` set to 0 and 2, respectively. 0 means no hash join plans are made, 2 means that hash join is used when appropriate for either SQL or SPARQL queries. The peak memory utilization and run times may be significantly affected.
5. Having completed these metrics, one may move to the multi-user case. Note that `MaxQueryMem` , (`c_max_large_vec` setting with `__dbf_set`) should be set to a reasonable value, e.g. the peak consumption with the chosen settings times the number of parallel sessions.
6. Expected Results -- Experience demonstrates for example that with a 128 concurrent users `setting enable_qp` to 1 and `enable_dyn_batch_sz` to 0 increased the throughput of the sample workload by a factor of 2.5. The workload derived, even in single user mode, little benefit from dynamic vector size or multithreading, under 30%. In the case at hand, the difference in performance was mostly accounted for by `mmap` , see the `mp_mmap_clocks` counter.

    - If more concurrent queries are than CPU cores are expected, there is little point in intra-query parallelism, controlled by `ThreadsPerQuery` in the INI file or `enable_qp` in `__dbf_set` .
    - If `mp_mmap_clocks` continues to grow fast during the execution one may increase `c_max_large_vec` . This will cause more `mmap` 's to be kept in reserve, thus in principle decreasing the frequency of `mmap` and `munmap` system calls.
    - If running with hash join enabled, there is a possibility of partitioned hash joins where the query executes in multiple passes in order to build smaller hash tables. This is given by `tc_part_hash_join` , which is increased by 1 for each non-first pass over a hash join. If the counter increases the `HashJoinSpace` setting in the INI file should be increased. With `__dbf_set` this is `chash_space_avail` .
7. Notes:

    - The results from above may not happen with a single user but happen all the time with multiple users. The monitor will print a warning message about this in the message log.
    - One may also try a different default vector size, specially if very pressed on memory. The default of 10000 values is generally a good small value but smaller may be possible, however not under 1000.

Query Logging

As of version 7.00 Virtuoso offers optional server side query logging. This records a large number of execution statistics as well as the full query text and execution plan with per-operator timing and cardinality information.

This feature is enabled in the Parameters section of virtuoso.ini:

```
; virtuoso.ini
...
[Parameters]
QueryLog = filename
```

At run time, this may be enabled or disabled with prof_enable (), overriding the specification of the ini file. The default file name for the query log is virtuoso.qrl in the server's working directory, if not otherwise specified in the ini file.

The file is in binary format and is not conveniently readable as such. It is most easily accessed via the DB.DBA.SYS_QUERY_LOG system view. This view has parameters for specifying a non-default file path for the query log file as well as a datetime range for selecting the entries of interest.

For example:

```
SELECT *
  FROM sys_query_log
 WHERE qrl_file = 'other.qrl'
   AND qrl_start_dt = cast ('2011-10-1' as datetime)
   AND qrl_end_dt = cast ('2011-10-2' as datetime)
```

This will select entries from the file other.qrl that are between the given dates. Note that the qrl* columns are not actual result column of the view but are considered as parameters, hence the use of = instead of a range condition.

All statements executed over a SQL client connection (ODBC/JDBC etc) are logged, as well as any statements executed on an SPARQL end point. DAV and web services requests are not logged unless they perform an exec function call. CPU usage for

committing transactions or for background data organization such as autocompact or automatic checkpoints are not logged.

A select statement is logged at the time it produces its last row of output, not at the time this row of output is fetched by a client.

Some of the columns are in units of clocks, whose meaning varies from system to system. On Intel this corresponds to the value returned by the RDTSC instruction. All values are intervals. The relation of this to real time may vary in function of automatic variation of clock frequency.

The columns of SYS_QUERY_LOG are as follows:

- *ql_id*

  -- Serial number of the log entry. If the server was started many times with the same file these will not be unique. Combine with

  ql_start_dt

  for unique identification.
- *ql_start_dt*

  -- datetime of the start of the query
- *ql_rt_msec*

  -- real time elapsed in milliseconds between the start and the logging of the query
- *ql_rt_clocks*

  -- Clock cycles of real time spent running the query, not including time between consecutive fetches from a client if the query was a cursor fetched in multiple chunks by a client. This is the number of clocks during which there was at least one thread running on behalf of the query. The average CPU% of the query is given by:

  ```
  100 * ql_thread_clocks / ql_rt_clocks
  ```
- *ql_client_ip*

  -- Requesting client IP as dotted decimal.
- *ql_user*

  -- User account on behalf of which the query was executed.
- *ql_sqlstate*

  -- SQL state if query terminated with error, NULL otherwise.
- *ql_error*

  -- Error message if query terminated with error, NULL otherwise.
- *ql_swap*

  -- Cumulative count of major page faults since startup of this Virtuoso process.
- *ql_user_cpu*

  -- Cumulative user CPU in milliseconds for this server process.
- *ql_sys_cpu*

  -- Cumulative system CPU in milliseconds for this server process.
- *ql_text*

  -- Source text of the query
- *ql_params*

  -- NULL.
- *ql_plan_hash*

-- Hash number calculated from the execution plan, ignoring literals. Can be used for grouping executions of the same query with differing literals together. If difference of literals causes a different plan, this number will be different.

- *ql_c_clocks*

  -- CPU clocks of real time used for query compilation. This will be 0 if the query is separately prepared of if the query compilation comes from a cache of recently compiled queries. This is likely if the query is parametrized and executed multiple times.
- *ql_c_msec*

  -- Real time used for query compilation in milliseconds.
- *ql_c_disk*

  -- Count of disk reads done on behalf of the query compilation, this stands for index sampling initiated by the compilation and does not include any speculative read possibly triggered by the sampling.
- *ql_c_disk_wait*

  -- Count of clocks the compilation was blocked waiting for disk.
- *ql_c_cl_wait*

  -- Count of clocks the compilation was waiting for information from cluster peers. Such waiting indicates sampling done on remote partitions. If the run time of the query is small, this may be a significant factor in query execution real time.
- *ql_cl_messages*

  -- Count of distinct cluster messages sent on behalf of the compilation. These are all related to sampling. Many samples can be combined into one message in some situations. Samples are also cached on the requesting server so repeatedly compiling the same statement will send the messages only the first time in unless the cache has timed out in the meantime.
- *ql_c_rnd_rows*

  -- Count of rows retrieved as part of compile time sampling.

The below columns correspond directly to the fields returned by db_activity (). These are summed over all the threads in all the hosts that have done something on behalf of the logged query.

- *ql_rnd_rows*

  -- Count of random row lookups. Each sequential lookup begins with one random one for each partition concerned.
- *ql_seq_rows*

  -- Sequential rows fetched, each non-first row which is selected counts as one. A row that is looked at but does not satisfy applicable query criteria does not count.
- *ql_same_seg*

  -- Count of times the next random lookup in a vectored lookup falls on the same column-wise segment as the previous.
- *ql_same_page*

  -- Ibid, for the next lookup falling on the same row-wise leaf page.
- *ql_same_parent*

  -- Ibid, for the case of the next lookup falling on a sibling page of the row-wise leaf page of the previous lookup.

For column-wise indices, all the three above counters can be non-zero since these consist of multi-row segments each under a row on a row-wise leaf page. For a row-wise index the same seg counter is always 0.

- *ql_thread_clocks*

  -- Sum of clocks spent on any thread for the logged query. Time spent waiting for other threads, for disk or for replies from cluster peers is not counted, thus only running cycles count. These are added up across all hosts in a cluster.
- *ql_disk_wait_clocks*

-- Total clocks any thread spends waiting for disk on behalf of the query. If two threads wait for the same page which is fetched once the wait is counted double. This is not the same as the total read time of the pages since read ahead can fetch pages before they are needed, thus involving no wait.

- *ql_cl_wait_clocks*

-- Total clocks a thread running on behalf of the query spends waiting for a cluster reply. This may be zero if the coordinating thread has work until any cluster replies arrive, in which case there will be no wait. If this is high then the workload is bound by interconnect or is unevenly distributed across a cluster.

- *ql_pg_wait_clocks*

-- Count get page wait.

- *ql_disk_reads*

-- Count disc reads.

- *ql_spec_disk_reads*

-- Count of speculative disk reads triggered on behalf of the query. Any read ahead or any upgrading of a single page read into a read of a whole extent counts towards this, only allocated pages that are read are counted but merging near-adjacent reads may cause actually more disk IO to take place.

- *ql_messages*

-- Count of distinct cluster messages sent on behalf of the query. Any message is counted once. Client-server messages are not counted.

- *ql_message_bytes*

-- Total bytes sent in all cluster messages sent on behalf of the query.

- *ql_qp_threads*

-- Count of times an extra thread is created for parallelizing work on the query in question. The count may be high since consecutively launched threads are counted, this is not a maximum degree of concurrency.

- *ql_vec_bytes*

-- reserved

- *ql_vec_bytes_max*

-- reserved

- *ql_lock_waits*

-- Count of times a thread has waited for a lock on behalf of the query.

- *ql_lock_wait_msec*

-- Total milliseconds any thread has spent waiting for a lock on behalf of the query. This may be longer than real time since many threads may wait at the same time.

- *ql_plan*

-- Text representation of the execution plan, annotated with CPU time percentages and fanouts for the different operators. Fanout is the count of output rows divided by the count of input rows.

- *ql_node_stat*

-- reserved

- *ql_c_memory*

-- Count of bytes allocated for compiling the query. This is the peak size of the memory pool for query compilation.

- *ql_rows_affected*

-- Count of inserted/updated/deleted rows. If the query was a select with a top and an order by, this is is the count of produced rows before the top restriction was applied.

Meters & System Views

DB.DBA.SYS_K_STAT, DB.DBA.SYS_D_STAT, DB.DBA.SYS_L_STAT view

These views provide statistics on the database engine

```
create view SYS_K_STAT as
  select KEY_TABLE, name_part (KEY_NAME, 2) as index_name,
        key_stat (KEY_TABLE, name_part (KEY_NAME, 2), 'n_landings') as landed,
        key_stat (KEY_TABLE, name_part (KEY_NAME, 2), 'total_last_page_hits') as consec,
        key_stat (KEY_TABLE, name_part (KEY_NAME, 2), 'page_end_inserts') as right_edge,
        key_stat (KEY_TABLE, name_part (KEY_NAME, 2), 'page_end_inserts') as lock_esc
        from SYS_KEYS;

create view SYS_L_STAT as
  select KEY_TABLE, name_part (KEY_NAME, 2) as index_name,
        key_stat (KEY_TABLE, name_part (KEY_NAME, 2), 'lock_set') as locks,
        key_stat (KEY_TABLE, name_part (KEY_NAME, 2), 'lock_waits') as waits,
        (key_stat (KEY_TABLE, name_part (KEY_NAME, 2), 'lock_waits') * 100)
          / (key_stat (KEY_TABLE, name_part (KEY_NAME, 2), 'lock_set') + 1) as wait_pct,
        key_stat (KEY_TABLE, name_part (KEY_NAME, 2), 'deadlocks') as deadlocks,
        key_stat (KEY_TABLE, name_part (KEY_NAME, 2), 'lock_escalations') as lock_esc
        from SYS_KEYS;

create view sys_d_stat as
  select KEY_TABLE, name_part (KEY_NAME, 2) as index_name,
        key_stat (KEY_TABLE, name_part (KEY_NAME, 2), 'touches') as touches,
        key_stat (KEY_TABLE, name_part (KEY_NAME, 2), 'reads') as reads,
        (key_stat (KEY_TABLE, name_part (KEY_NAME, 2), 'reads') * 100)

 > / (key_stat (KEY_TABLE, name_part (KEY_NAME, 2), 'touches') + 1) as read_pct
        from SYS_KEYS;
```

These views offer detailed statistics on index access locality, lock contention and disk usage.

'reset' specified as the stat name will reset all counts for the key in question.

SYS_K_STAT - Key statistics

- KEY_TABLE The fully qualified table name, e.g. DB.DBA.SYS_PROCEDURES
- INDEX_NAME The name of the index. This will be equal to the table name for the table's primary key.
- LANDED The count of random accesses, including inserts. Any insert or select, whether empty, single line or multi- line counts as one. Updates and deletes do not count, as they imply a select in the same or previous statement.
- CONSEC The number of times a random access falls on the same page as the previous random access. This is always less than LANDED. For repetitive access to the same place or an ascending insert, this will be near LANDED. For a totally random access pattern this will be near 0.
- RIGHT_EDGE The number of times an insert has added a row to the right edge of the page where the insert was made.
- LOCK_ESC The count of lock escalations, see SYS_L_STAT.

SYS_L_STAT

- KEY_TABLE The fully qualified table name, e.g. DB.DBA.SYS_PROCEDURES
- INDEX_NAME The name of the index. This will be equal to the table name for the table's primary key.
- LOCKS The number of times a lock has been set on the index. Making a new row or page lock counts as one. Entering a row or page lock either after a wait or without wait (for a shared lock) counts as one.
- WAITS The number of times a cursor reading this index waited for a lock. Note that this can be higher than the number of locks set, e.g. a 'read committed' cursor may wait for a lock but will never make one.
- WAIT_PCT The percentage of lock set events that involved a wait.
- DEADLOCKS The number of times a deadlock was detected when trying to wait for a lock on this index. Note that one deadlock event may involve locks on several indices. Each deadlock detection counts as one.
- LOCK_ESC The number of times the set of row locks on a page of this index where escalated into one page lock or a page lock was set initially. This is always less than LOCKS. This value will be near LOCKS when there are many sequential selects which switch to page lock mode. This happens when a cursor has performed over 2 lock escalations and the page being entered has no locks, i.e. the lock can be set over the entire page.

SYS_D_STAT

- KEY_TABLE The fully qualified table name, e.g. DB.DBA.SYS_PROCEDURES
- INDEX_NAME The name of the index. This will be equal to the table name for the table's primary key.
- TOUCHES The number of times a row is located on the index. Every row retrieved by a select or inserted counts as one. All rows scanned by an select count or other aggregate counts as one.
- READS The number of times a disk read was caused by a read operation on this index. This may theoretically be higher than TOUCHES, since several levels of the index tree may have to be read to get to a leaf.
- READ_PCT The percentage of READS in TOUCHES.

Example 6.14. Examples:

```
select index_name, locks, waits, wait_pct, deadlocks
    from sys_l_stat order by 2 desc;

Get lock data, indices in descending order of lock count.

select index_name, touches, reads, read_pct
    from sys_d_stat order by 3 desc;

Get disk read counts, index with most reads first.

select index_name, (consec * 100) / (landed + 1)
    from sys_k_stat where landed > 1000  order by 2;
```

Get the percentage of consecutive page access on indices with over 1000 accesses so far, most randomly accessed first.

status SQL function - status ();

This function returns a summary of the database status as a result set. The result set has one varchar column, which has consecutive lines of text. The lines can be up to several hundred characters.

The contents of the status summary are described in the Administrator's Guide .

Virtuoso db file usage detailed info

All data in a virtuoso database are logically stored as database key rows. Thus the primary key for a table holds the entire row (including the dependent part) and the secondary keys just hold their respective key parts. So the space that the table occupies is the sum of the space occupied by it's primary key and all the secondary keys.

The main physical unit of allocation in a virtuoso db file is the database page (about 8k in virtuoso 3.x). So the server needs to map the key rows and outline blobs to database pages.

Virtuoso will store as many rows in a db page as it can, so usually one DB page will contain more than 1 row of a given key. No page contains rows from more than one key. However blobs (when not inlined on the row) will be placed in consecutive DB pages (up to their size). In addition to the blob and key pages the Virtuoso DB will hold a number of pages containing internal data. So the sum of the pages occupied by the key rows and the blobs is leas then the amount of occupied pages (as reported by the `status()` BIF).

To provide detailed information about the space consumption of each key there's a system view:

```
DB.DBA.SYS_INDEX_SPACE_STATS
    ISS_KEY_TABLE       varchar -- name of the table
    ISS_KEY_NAME        varchar -- name of the key
    ISS_KEY_ID          integer -- id of the key (corresponding to KEY_ID from DB.DBA.SYS_KEYS)
    ISS_NROWS           integer -- number of rows in the table
    ISS_ROW_BYTES       integer -- sum of the byte lengths of all the rows in the table
    ISS_BLOB_PAGES      integer -- sum of the blob pages occupied by the outline blobs on all the rows of
    ISS_ROW_PAGES       integer -- sum of all the db pages containing rows of this key
    ISS_PAGES           integer -- = ISS_BLOB_PAGES + ISS_ROW_PAGES (for convenience).
```

Each select on that view causes the server to go over all the db pages in the db file (similarly to how the crash dump operates) and collect the statistics above. The pages are traversed 1 time per select, but still on large database files that may take some time.

Transaction Metrics, Diagnostics and Optimization

Bad design and implementation of transactions affects applications in the following ways:

- Performance is lost by having to needlessly retry transactions that are aborted by deadlock.
- Concurrency is lost by having rows stay locked for too long.
- Memory is transiently consumed, adversely affecting working set, by keeping data structures for too many simultaneous locks, rollback records and uncommitted roll forward logs.

The following rules should be observed when writing transactions:

- Do not lock needlessly. For example, any report transaction that reads the data once can always be done as read committed instead of repeatable read without affecting semantics. Even if some data is read multiple times, the repeatable read semantic is typically not relevant for reports.
- Lock for what is needed. If you mean to update later, do the initial read with exclusive locks. See the for update clause in select, for example.
- Lock in constant order. If you must lock different resources in one transaction, lock them always in the same order. When updating stock for an order, update the quantity on hand in increasing order of item number, for instance.
- Lock the item with the least contention first. For example, update the detail before updating the summary. Update the quantity in stock for the ordered items before updating the orders count of the whole warehouse.
- Keep transactions short. Use stored procedures. Use the explicit commit work statement.
- For each transaction in a stored procedure, make sure that if it is deadlocked, the deadlocked transaction gets retried. For example, have a "declare exit handler for sqlstate 40001" for every transaction context. Make sure that a deadlocking transaction is never retried endlessly. Two mutually deadlocking transactions can keep retrying and again deadlocking each other endlessly. To avoid this, have a maximum count of retries and a random delay before restarting. The restart delay should be between 0 and the expected duration of the transaction.
- Always break batch updates into multiple transactions. Update a few thousand or tens of thousands of rows per transaction, never more than that. Failing to do this makes for prohibitive cost of retry with deadlocks and can cause swapping by keeping tens or hundreds of megabytes in rollback state, locks and other transaction temporary structures. This happens if one inserts, updates, deletes several million rows in a single transaction. If this is really needed and concurrency is no issue, use the atomic mode, effectively making the server single user for the transaction, thus having no locking or rollback. See the use of the

  __atomic()

  function in the Cluster Administration API .

Programming Virtuoso/PL

The isolation level is set in Virtuoso/PL with the

```
set isolation := level;
```

statement, where level is one of 'serializable', 'repeatable', 'committed', 'uncommitted'. Example :

```
set isolation = 'serializable';
```

The standard SQL syntax is also supported :

```
SET TRANSACTION ISOLATION LEVEL <isolation_level>
isolation level : READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE
```

Example :

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
```

The effect is for the rest of the procedure and any procedure called from this procedure. The effect stops when the procedure having executed the set isolation statement returns.

Transaction Parallelism

From Virtuoso 7.1 onwards, data manipulation statements can be multithreaded. The virtuoso.ini `[Flags]` settings `enable_mt_txn` and `enable_mt_transact` control transaction parallelism. These are 1 for enable and 0 for disable. Defaults are version dependent, off in 7.1 and 7.2 and on in subsequent. The `sys_stat()` and `__dbf_set()` functions can be used for querying and altering these.

DML statements, e.g. insert, delete, update, are automatically multithreaded according to `enable_qp` (`ThreadsPerQuery` in ini [Parameters] when `enable_mt_txn` is on).

RDF load (`DB.DBA.ttlp()` and related functions) when in transactional mode (log_enable 0 or 1) are also multithreaded when `enable_mt_txn` is on. Otherwise these are multithreaded when in non-transactional mode (`log_enable` 2 or 3) and single threaded in transactional mode.

The `enable_mt_transact` setting controls the parallelism of a commit or rollback. Specially for column store deletes, the commit may take a long time, as this is the point where the physical deletion takes place. To accelerate this, `enable_mt_transact` should be on. This is however not worthwhile for short row store transactions, as the overhead of making threads is greater than the time it takes to finalize a transaction om a page. Again, large, scattered row store deletes may justify use of `enable_mt_transact` .

Sample Deadlock Handler

The text for a deadlock handler is

```
  declare retry_count int;
  retry_count := 0;
retry:
  {
    declare exit handler for sqlstate '40001'
      {
        rollback work;
        ...
        delay (rnd (2.5));  --- if 2.5 seconds is the expected duration of
the transaction.
        ...

        retry_count := retry_count + 1;
        if (retry_count > 5)
          signal ("xxxxx", "Too many deadlock retries in xxxxx.");
        goto retry;
      }
   -- do the operations. The actual working code here.

    commit work;
  }
```

An exclusive read is done with

```
select s_quantity from stock where s_i_id = 111 for update;
```

ODBC

For the Virtuoso ODBC driver the isolation is set by :

- connection option (in either SQLSetConnectAttr () or SQLSetConnectOption ())

  ```
  rc = SQLSetConnectOption (hdbc, SQL_TXN_ISOLATION, SQL_TXN_READ_COMMITTED);
  ```

  or

  ```
  rc = SQLSetConnectAttr (hdbc, SQL_TXN_ISOLATION, SQL_TXN_READ_COMMITTED, NULL);
  ```

  Constants are : SQL_TXN_READ_UNCOMMITTED, SQL_TXN_READ_COMMITTED, SQL_TXN_REPEATABLE_READ, SQL_TXN_SERIALIZABLE
- ODBC setup dialog option : In Windows there is a drop-down combo box to set the default transaction isolation level for a connection.

- connection string element : You may specify the default transaction isolation level for a given connection in it's connect string (passed to the SQLDriverConnect () ODBC API). Example :

```
SQLDriverConnect (hdbc, hwnd,
    "DSN=MyDSN;IsolationLevel=Repeatable Read;UID=dba;PWD=dbapwd", SQL_NTS,
    NULL, 0,
    NULL,
    SQL_DRIVER_NOPROMPT).
```

  The possible options for the connection string are : "Read Uncommitted", "Read Committed", "Repeatable Read" and "Serializable"

JDBC

In the Virtuoso JDBC driver the isolation is set by the java.sql.Connection.setTransactionIsolation() JDBC API.

```
conn.setTransactionIsolation (java.sql.Connection.TRANSACTION_SERIALIZABLE)
```

The constants are described in the Java Docs

.Net

In the VirtuosoClient.NET provider the isolation is set by the System.Data.IDbConnection.BeginTransaction Method (IsolationLevel) function.

```
System.Data.IDBTransaction trx = conn.BeginTransaction (System.Data.IsolationLevel.ReadCommitted)
```

The constants are described here

Metrics and Diagnostics

Metrics are presented at the server and the table level.

The first metric to check is the output of status ('');

The paragraph titled transaction status contains the following:

- Count of deadlocks since server startup. There is a total number of deadlocks and the number of 2r1w deadlocks. The latter is a special case where two transactions both hold a shared lock on a resource and one tries to convert the lock to exclusive. This situation can kill the transaction attempting to write. Such deadlocks are essentially always needless. These are avoided by reading for update when first reading the resource.
- Count of waits since server startup. This is incremented every time some operation waits for a lock, except if this wait leads into a deadlock. If the number of deadlocks is high, let's say over 5% of the number of waits, transactions are likely badly designed and deadlock too often, either because of not locking for write at the get go (2r1w) or because of locking resources in varying order.
- Count of threads running. This is the count of all threads that are somehow occupied, whether running or waiting. This count minus the count of waiting minus the count of threads in vdb is the count of threads that in principle could be on CPU.
- Count of threads waiting. This is the count of threads that are waiting for a lock. If this is a high percentage of the count of threads running, say over 30%, resources are likely locked inefficiently, keeping too many locked.
- Count of threads in vdb. This is the count of threads that are at the time waiting for I/O either from a remote database or any sort of network operation, including access to web services on other serbers, access to web pages on other hosts etc.

The system view db.dba.sys_l_stat is used for locating bottlenecks.

The columns are:

- index - The index being locked. Note that when reading on non-primary key, the lock is set on the index first, only then on the pk, that is if the pk is accessed at all. For all updates however, the pk will always be accessed. *locks - The count of times a lock was set on this index.
- waits - The number of times there was a wait on this index. There can be more waits than locks because a read committed cursor can wait but will not lock, thus all waits do not result in locks.
- wait_pct - The percentage of times setting a lock waited. (100 * waits) / locks

- deadlocks - The number of times a deadlock was signalled when attempting to wait for a lock on this index.
- lock_esc - The number of times a set of row locks was converted into a page lock on this index.
- wait_msecs - The total amount of real time spent by some thread waiting for a lock on this index. This may be greater than elapsed time because many threads can wait at the same time.

All counts and times are cumulative from server startup.

The interpretation is as follows:

If deadlocks is high in relation to waits or locks, i.e. over 5%, there are many deadlocks and the transaction profiles may have to be adjusted. The table where deadlocks is incremented is the table where the deadlock was detected but the deadlock may involve any number of tables. So, if A and B are locked in the order A, B half of the time and B, a the rest of the time, then the deadlocks of the tables of A and B will be about the same, half of the deadlocks being detected when locking A, the other half when locking B.

If waits is high in relation to locks, for example 20%, then there is probably needless contention. Things are kept locked needlessly. Use read committed or make shorter transactions or lock items with the less contention first.

Because transaction duration varies, the place with the highest count of waits is not necessarily the place with the heaviest contention if the waits are short. Use wait_msecs in addition to waits for determining where the waiting takes place.

To get a general picture, use the Conductor's Statistics page or simply do:

```
select top 5 *
  from sys_l_statt
 order by wait_msecs desc;
```

to get a quick view of where time is spent. You can also sort by waits desc or locks desc.

SQL Issues

It is possible to get bad locking behavior if the SQL compiler decides to make linear scans of tables or indices and the isolation is greater than read committed. The presence of a linear scan on an index with locking is often indicated by having a large number of lock escalations. If lock_esc is near locks then a large part of the activity is likely sequential reads.

The general remedy is to do any long report type transactions as read committed unless there are necessary reasons to do otherwise.

To see how a specific query is compiled, one can use the explain () function. To change how a query is compiled, one can use the table option or option SQL clauses.

Observation of Dynamics

Deadlocks and contention do not occur uniformly across time. The occurrences will sharply increase after a certain application dependent load threshold is exceeded.

Also, deadlocks will occur in batches. Several transactions will first wait for each other and then retry maybe several times, maybe only one succeeding at every round. In such worst cases, there will be many more deadlocks than successfully completed transactions. Optimize locking order and make the transactions smaller.

Looking at how counts change, specially if they change in bursts is useful.

Tracing and Debugging

The most detailed picture of a system's behavior, including deadlocks nd other exceptions cn be obtained with profiling. If the application is in C, Java or some other compiled language, one can use the language's test coverage facility to see execution counts for various branches of the code.

For client applications, using the Virtuoso `trace()` function can be useful for seeing which statements signal errors. Also the profiling report produced by `prof_enable ()` can give useful hints on execution times and error frequencies. See Profiling and `prof_enable ()`.

For PL applications, Virtuoso provides a profiler and test coverage facility. This is activated by setting PLDebug = 2 in the Parameters section of the ini file and starting the server. The functions cov_store () and cov_report are used for obtaining a report of the code execution profile thus far. See the documentation on "Branch Coverage" for this. The execution counts of the lines for exception handler code will show the frequency of exceptions. If in a linear block of code a specific line has an execution count lower than that of the line above it, then this means that the line with the higher count has signalled as many exceptions as the difference of the higher and lower count indicates.

The times indicated in the flat and call graph profile reports for PL procedures are totals across all threads and include time spent waiting for locks. Thus a procedure that consumes almost no CPU can appear high on the list if it waits for locks, specially if this is so on multiple threads concurrently. The times are real times measured on threads separately and can thus exceed any elapsed real tiime.

Single stepping is not generally useful for debugging locking since locking issues are timing sensitive.

Client Level Resource Accounting

Starting with version 6, Virtuoso keeps track of the count of basic database operations performed on behalf of each connected client. The resource use statistics are incremented as work on the connection proceeds. The db_activity () function can be called to return the accumulated operation counts and to optionally reset these.

The db_activity built-in function has one optional argument. The possible values are:

- 0 - (default) - Return human readable string and reset the counts.
- 1 - return an array of numbers and reset the counts.
- 2 - return a human readable string and leave the counts.
- 3 - return an array of numbers and leave the counts.

The human readable string is of the form:

```
22.56MR rnd  1.102GR seq     10P disk  1.341GB /  102.7K messages
```

The postfixes K, M, G, T mean 10^3 to 10^15, except when applied to bytes, where these mean consecutive powers of 1024.

The numbers, left to right are the count of random row lookups, sequential row lookups, disk page reads, cluster inter-node traffic in bytes and cluster inter-node message count as an integer number of messages. If the configuration is a single server, the two last are 0.

The random and sequential lookup counts are incremented regardless of whether the row was found or not or whether it matched search conditions.

If the information is retrieved as an array, the array contains integer numbers representing these plus some more metrics.

Index - Meaning

- 0 - Random lookups
- 1 - sequential lookups
- 3 - lock waits
- 4 - total msec spent in lock wait on some thread. In a cluster situation, this may be more than elapsed real time.
- 5 - Disk reads
- 6 - Speculative disk reads. These are also counted in disk reads. A speculative read is a prefetch made on the basis of a read history of a disk extent.
- 7 - Cluster inter node message count
- 8 - Total bytes sent in cluster inter node traffic. A message is counted once, when sent.

If the thread calling db_activity is a web server thread, the totals are automatically reset when beginning the processing of the current web request.

6.2. HTML based Administration Console (Conductor) Guide

Abstract

This section describes how to administer the Virtuoso server from the Conductor interface. It primarily allow users to administer the server while giving access to many of the features that Virtuoso has to offer providing many conceptual demonstrations and introductions.

Most of the pages from the administration interface are provided to help administrators quickly and easily tune the Virtuoso server or navigate the various interfaces and features that Virtuoso has to offer.

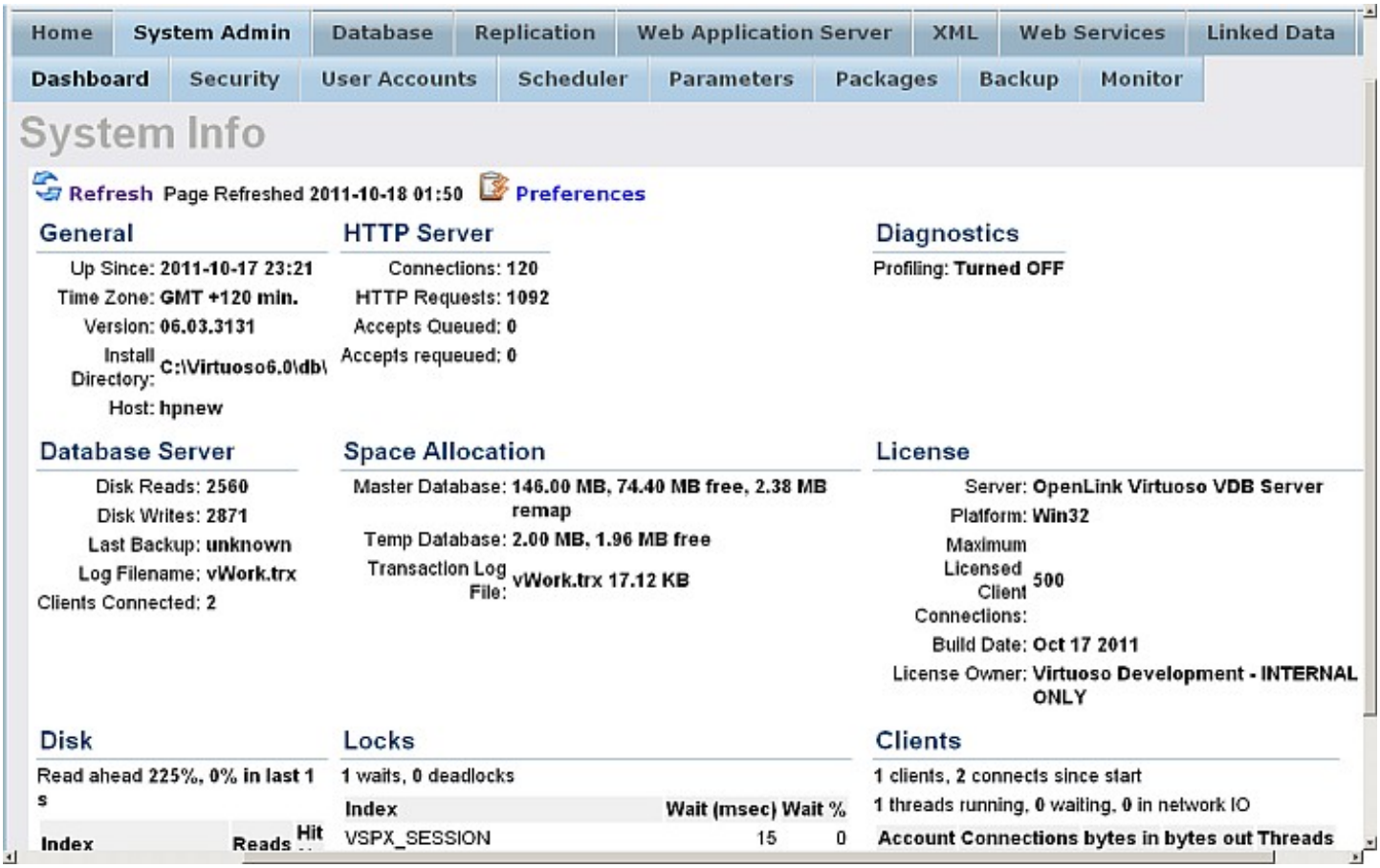
6.2.1. Virtuoso Conductor Administration

The Main Navigation Bar provides different tabs that allow you to administrate your Virtuoso server or use one of the provided samples.

Figure 6.1. Navigation

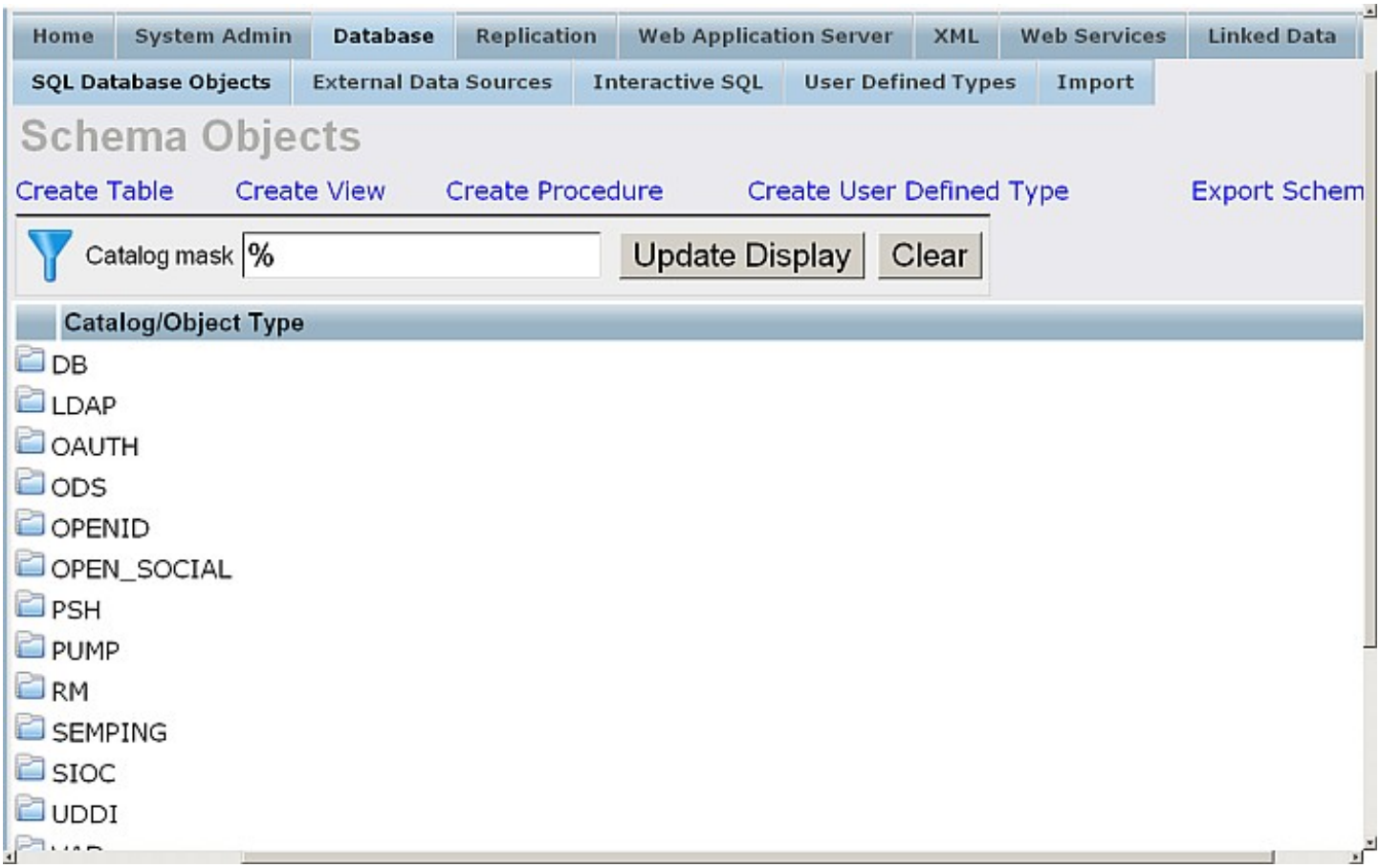
From "System Admin" you can view and change the Conductor Dashboard, manage user accounts; scheduler; Virtuoso Server parameter and Access Control Settings. You can also install /upgrade /uninstall Virtuoso packages and Monitor Virtuoso Server Statistics.

Figure 6.2. System Administration



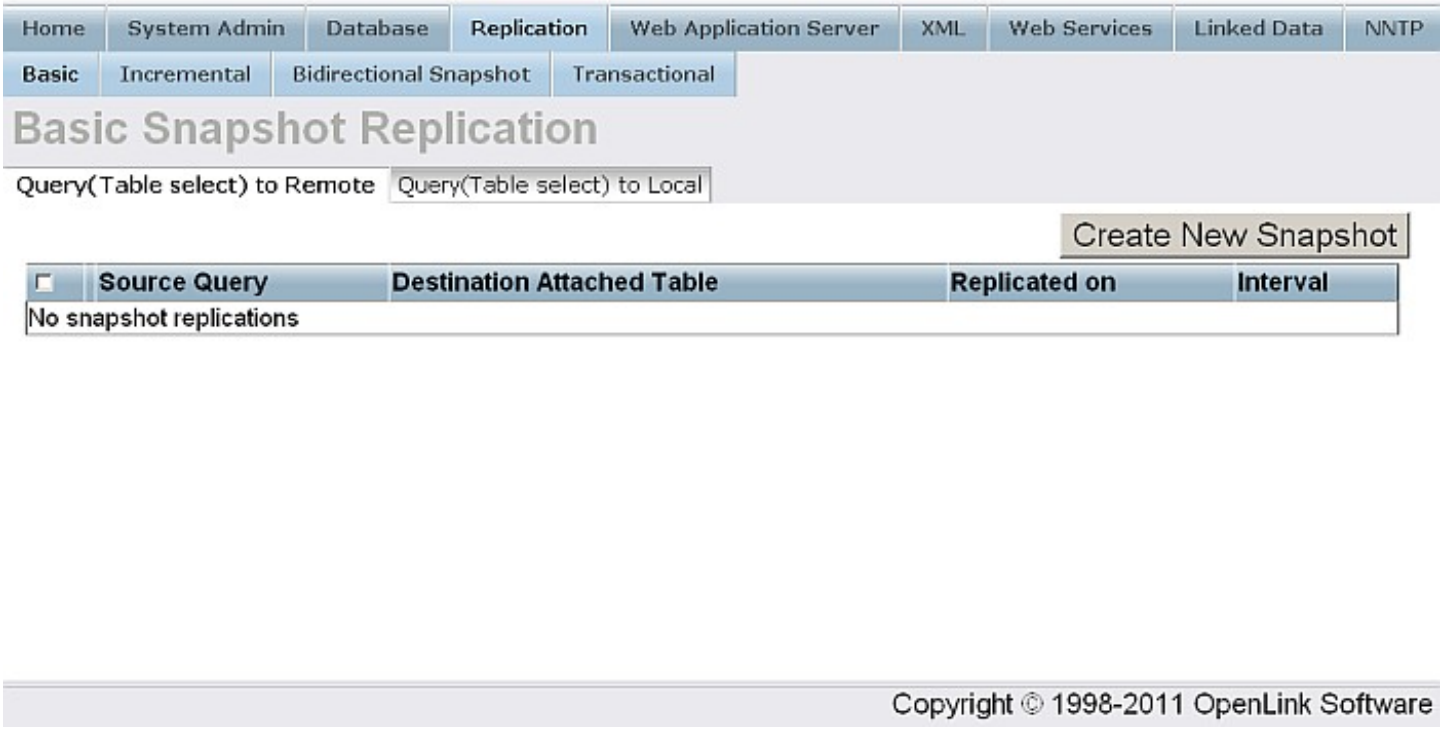
From "Database" you can administrate your database, manage the Virtuoso Relational Database System, administrate views, tables, stored procedures, trigger definitions, user define types, backups.

Figure 6.3. Database Administration



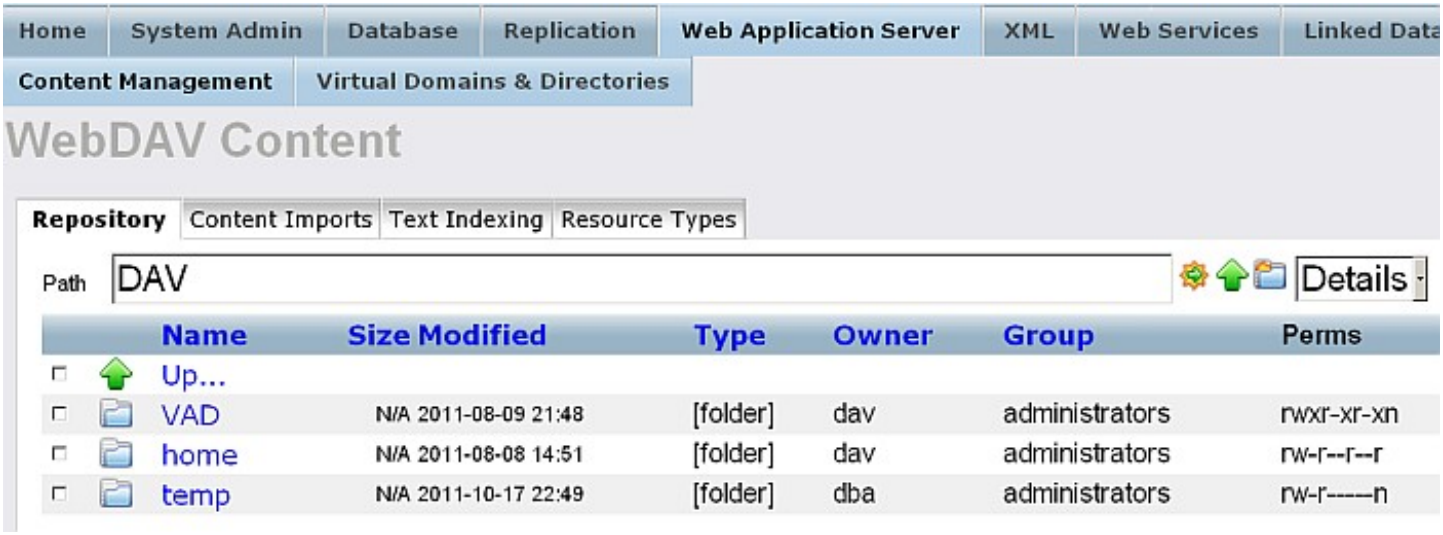
From "Replication" you can create Snapshot Replications to copy sections of the Database to remote locations or use Transactional Replication to keep Virtuoso Servers in sync over a definable interval.

Figure 6.4. Replication



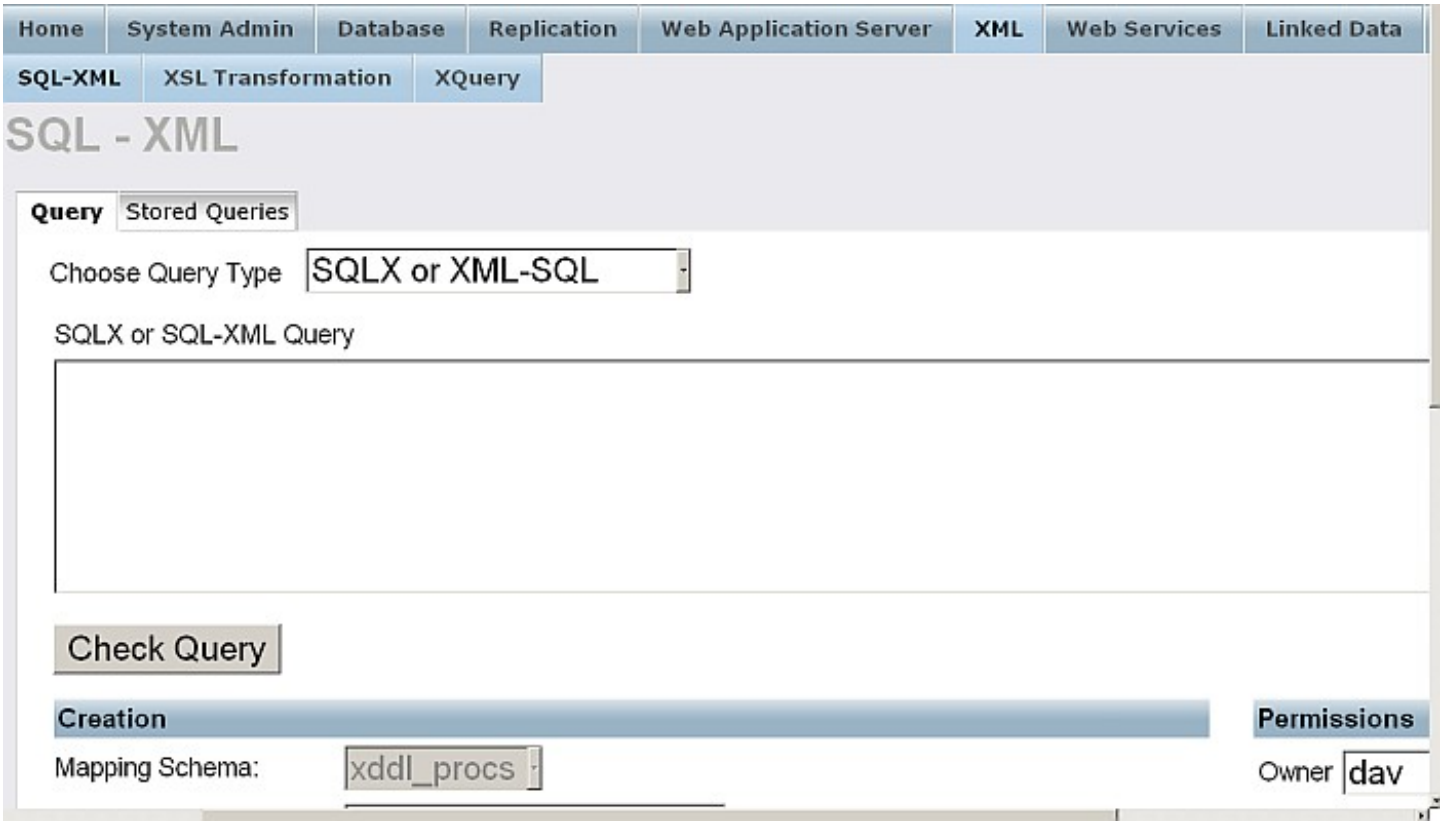
From "Web Application Server" you can configure Virtuoso's WebDAV, and HTTP Server functionality, which includes management of Virtual Domains & Directories.

Figure 6.5. Web Application Server Administration



From "XML" you can query Relational and XML Data using SQL, XQUERY, XPATH, and FREE TEXT.

Figure 6.6. Query Tools

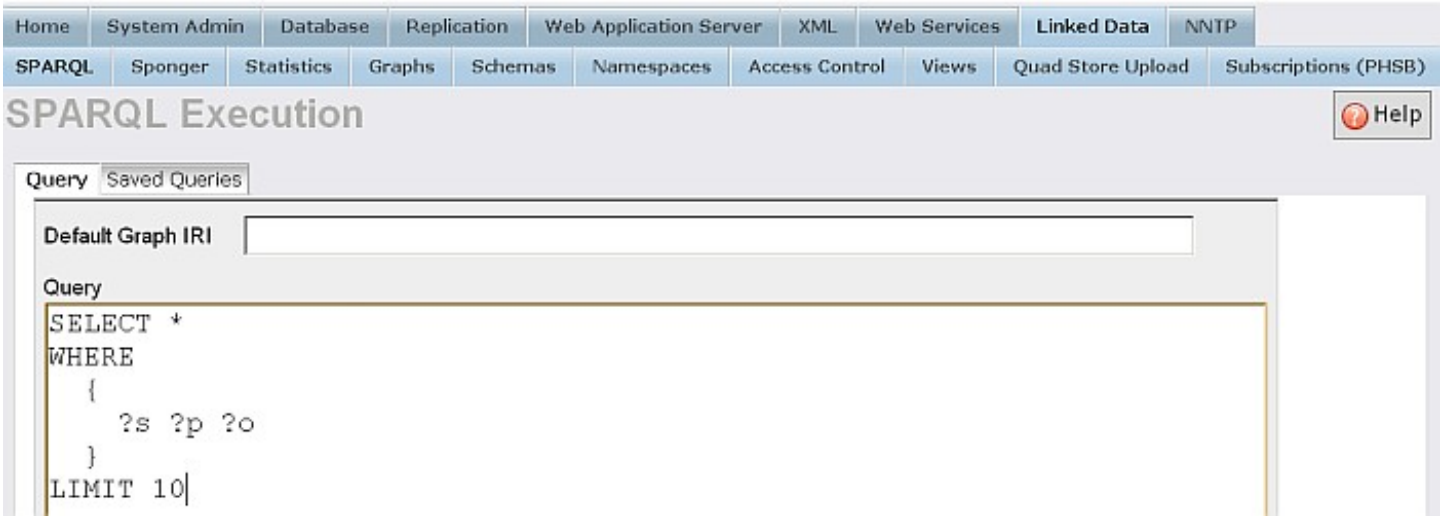


From "Web Services" you can add/ edit/ remove Web Services Endpoints, perform WSDL Import/Export, manage your BPEL processes.

Figure 6.7. Web Services

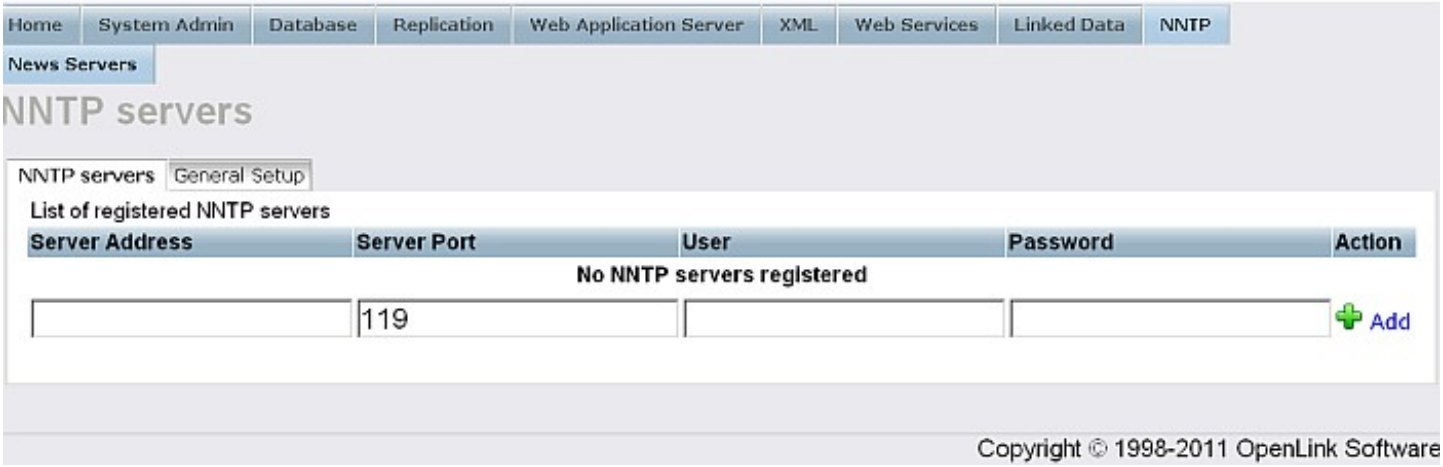
From "Linked Data" tab you can execute/save/load SPARQL queries, add/edit RDF Mapping, make statistics, manage graphs, import schemas and define namespaces, generated Linked Data Views, upload to the Quad Store:

Figure 6.8. RDF



Virtuoso's NNTP support which includes linking third-party NNTP servers into Virtuoso and controlling access to these servers.

Figure 6.9. NNTP Administration



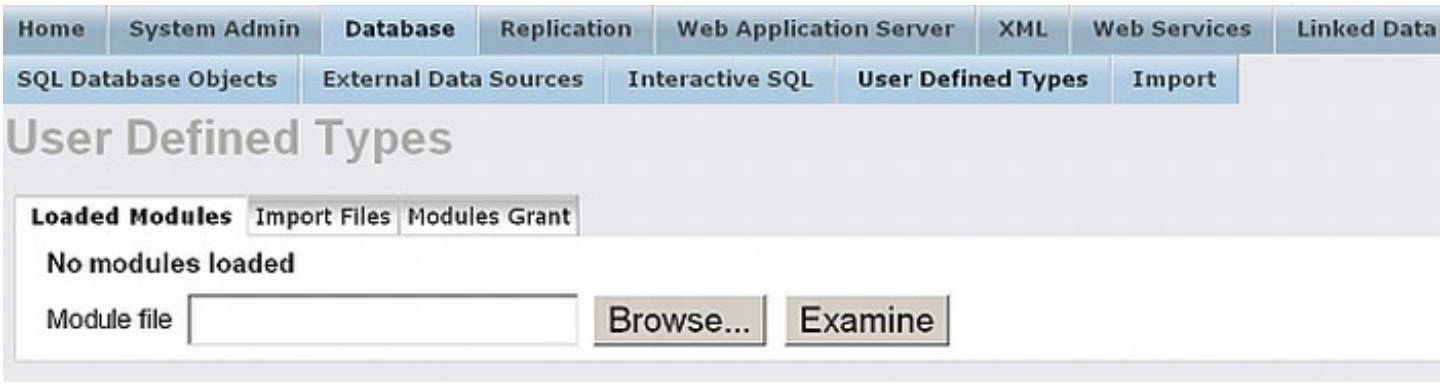
6.2.2. Runtime Hosting

The runtime hosting interfaces require Mono/CLR and Java extension servers.

Loaded Modules

You can view Loaded Modules list and each Module you can "Unload". Also you can browse for Module and examine its content.

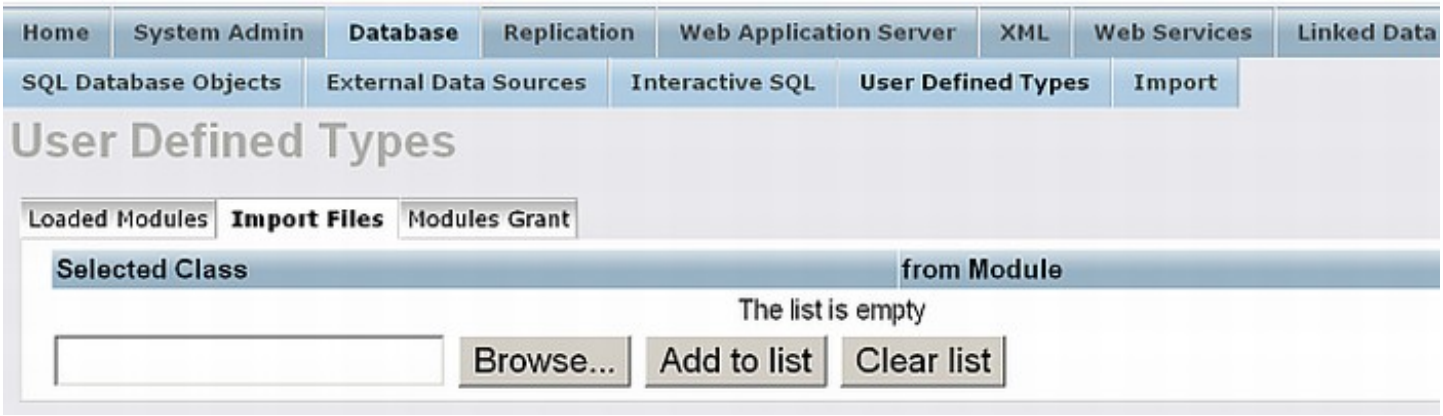
Figure 6.10. Loaded Modules



Import Files

To import files click the "Browse" button and select the class you want to import. Then click the "Add to list" button.

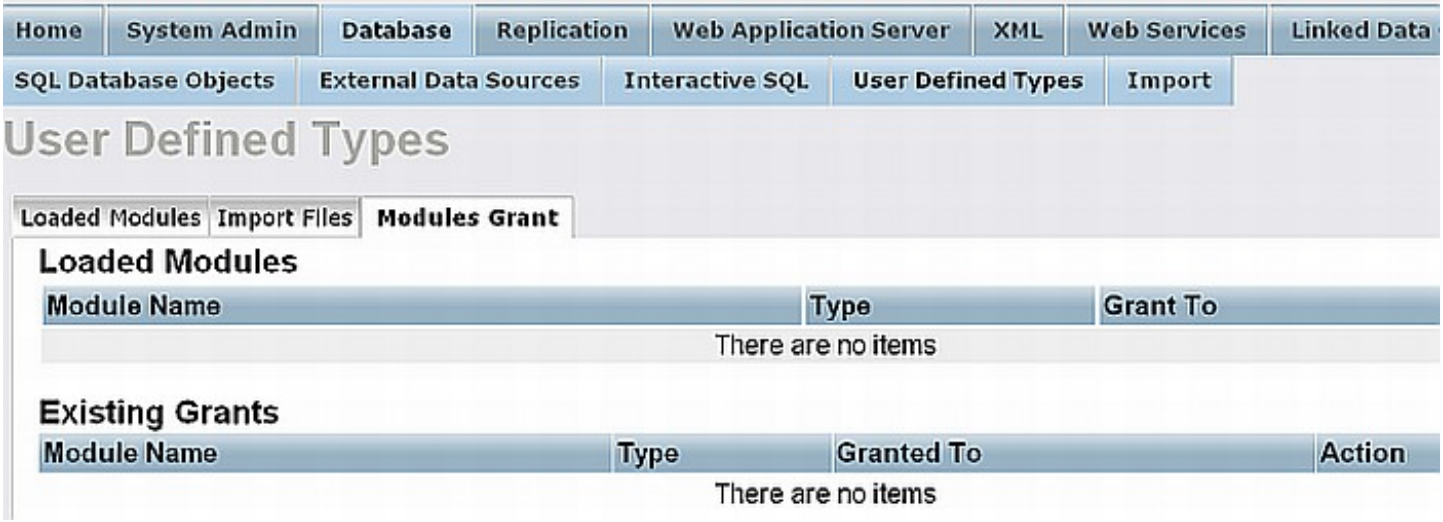
Figure 6.11. Import Files



Modules Grants

In order to change grants to a particular Module, select the desired grant value from the "Grant to" list and click the "Add" button.

Figure 6.12. Modules Grants



6.2.3. Web Services

Content Crawler

Some of the supported features are:

- *Import Targets* : Virtuoso can be set up to retrieve content from external web sites and host it in its own WebDAV repository via this page.

Figure 6.13. Web Robot - Crawl Job

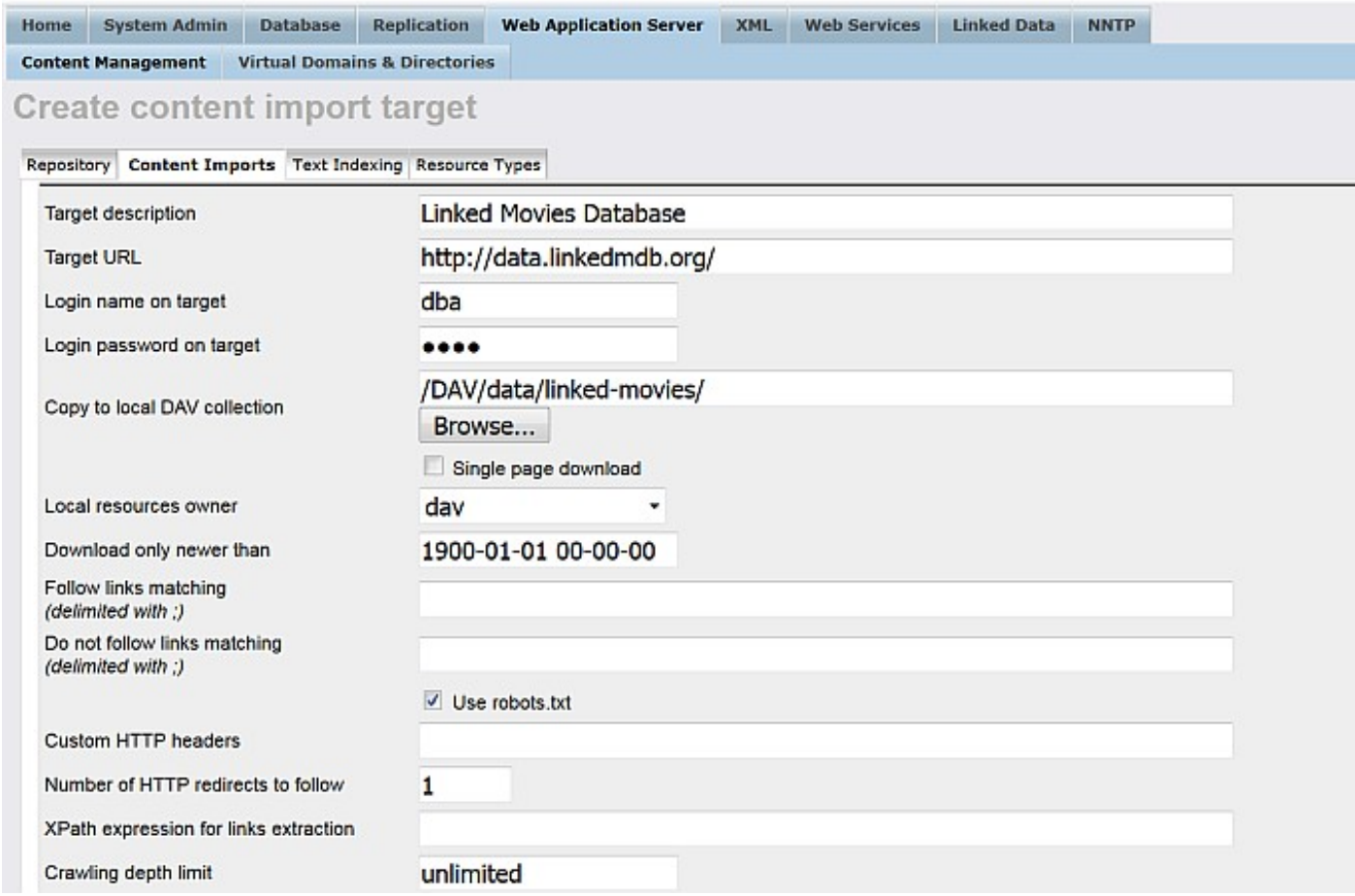
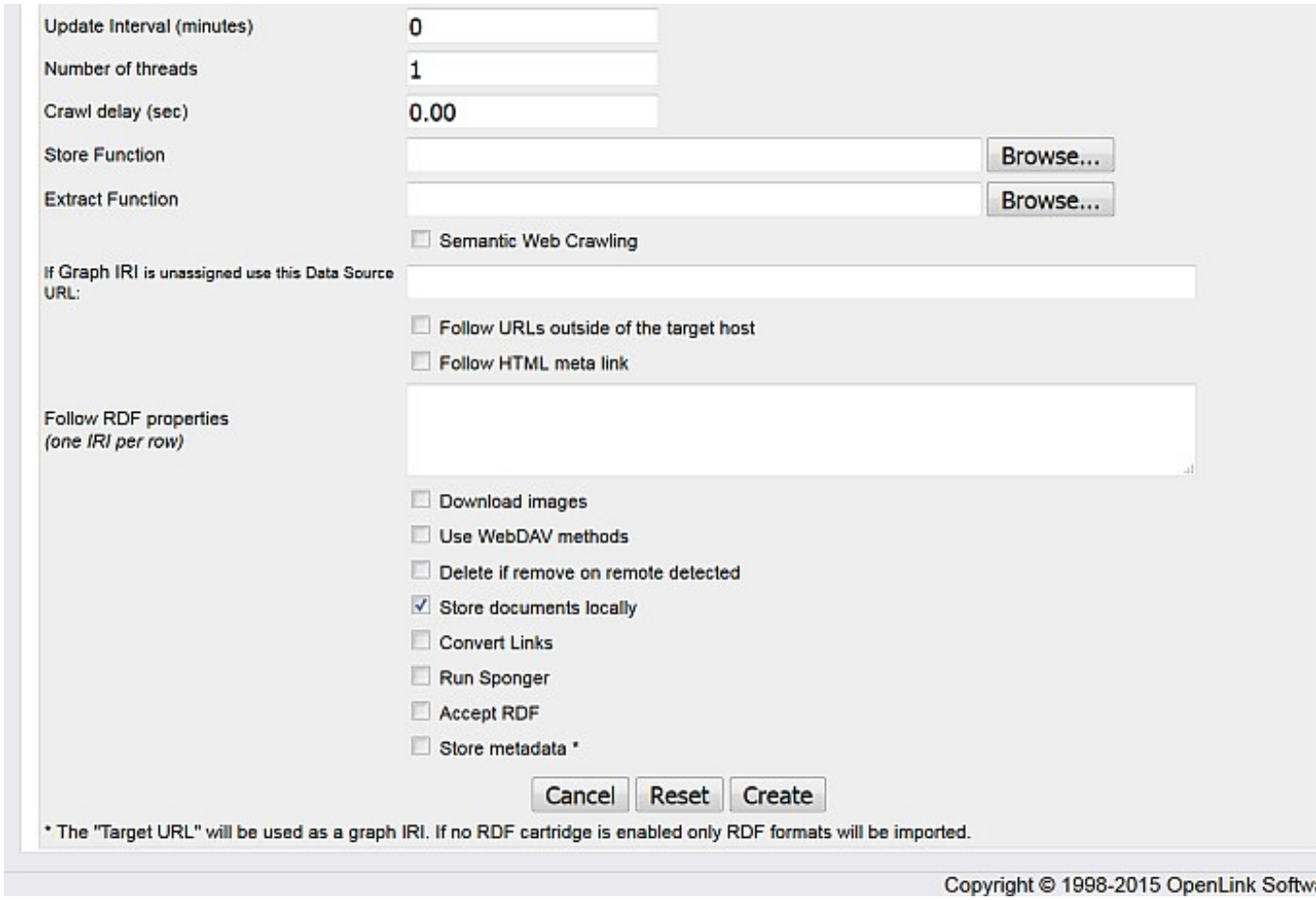


Figure 6.14. Web Robot - Crawl Job



- *Target description* lets you provide a friendly name for the target that you are defining.
- *Target URL* is the url of the web site that you are trying to retrieve content from. Only the hostname should be provided here, along with the protocol. For example http://www.myhost.com.
- *Login name on target* is the username for accessing the remote server, if required.
- *Login password on target* is the password for the login name above.
- *Copy to local DAV collection* this *mandatory* value will serve as the DAV root location if content is stored locally.
- *Single page download* radio button specifies whether Virtuoso will retrieved linked content also.
- *Local Resources Owner* The DAV user that will be the owner of the content that will copied to DAV.
- The *Download only newer than* field allows you to specify a datetime value to prevent Virtuoso downloading files that are older than that datetime.
- Use the *Follow links matching (delimited with ;)* field to limit the content that is downloaded by specify pattern matching criteria.
- *Do not follow links matching (delimited with ;)* allows you limit content by specifying what files not to download.

- *Use robots.txt* check-box is checked by default.
- *Custom HTTP headers* can be used to specigy custom HTTP headers.
- *Number of HTTP redirects to follow* is set by default to 1.
- *XPath expression for links extraction* allows to be entered custom XPath expression.
- *Crawling depth limit* allows to limit crawling depth. By default is set to "unlimited".
- *Update Interval (minutes)* is on what interval the updated of the crawled data should be performed.
- *Number of threads* allows setting crawling threads.
- *Crawl delay (sec)* allows to be specified a delay. By default is set to "0.00".
- *Store Function* allows to be used a specific stored function for ex. in Semantic Sitemaps crawling.

See Also:

Example of Stored Functions.

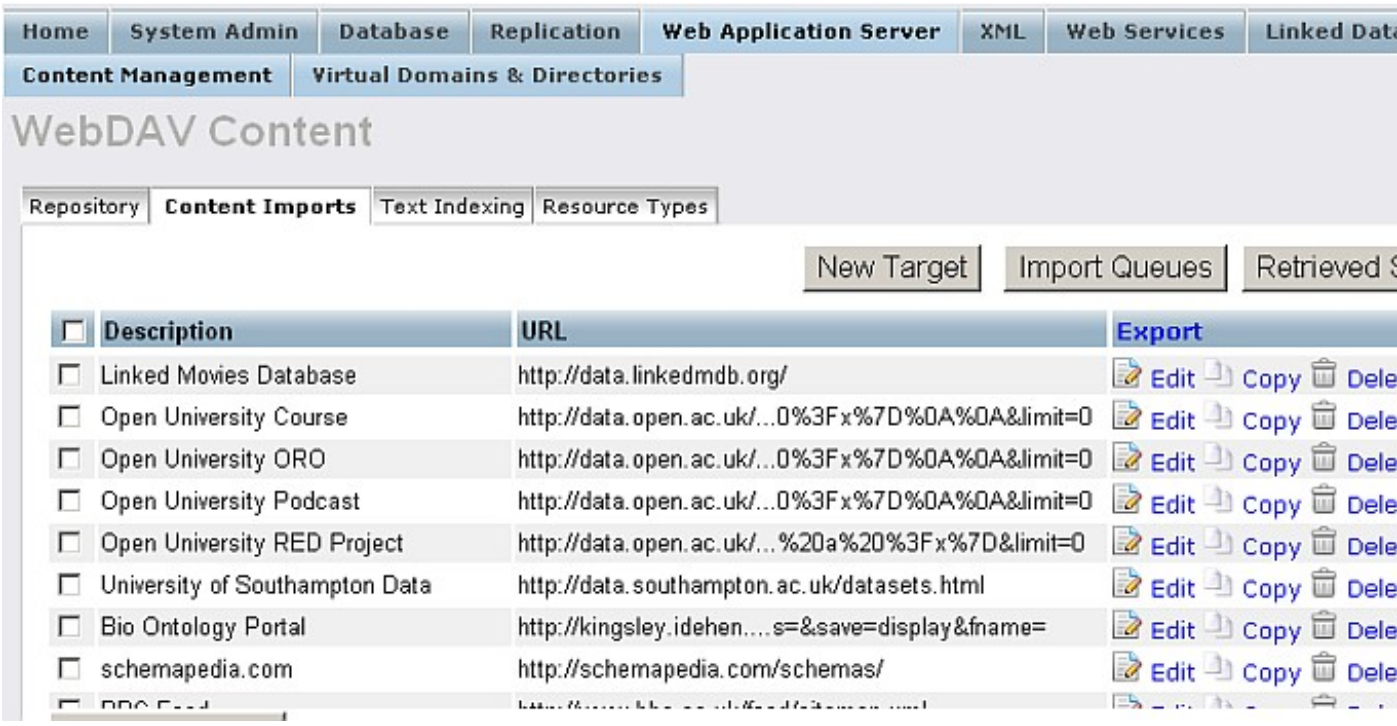
- *Extract Function* allows to be used a specific extract function for ex. in Semantic Sitemaps crawling. If left empty, will be used a system Store function.

See Also:

Example of Stored Functions.

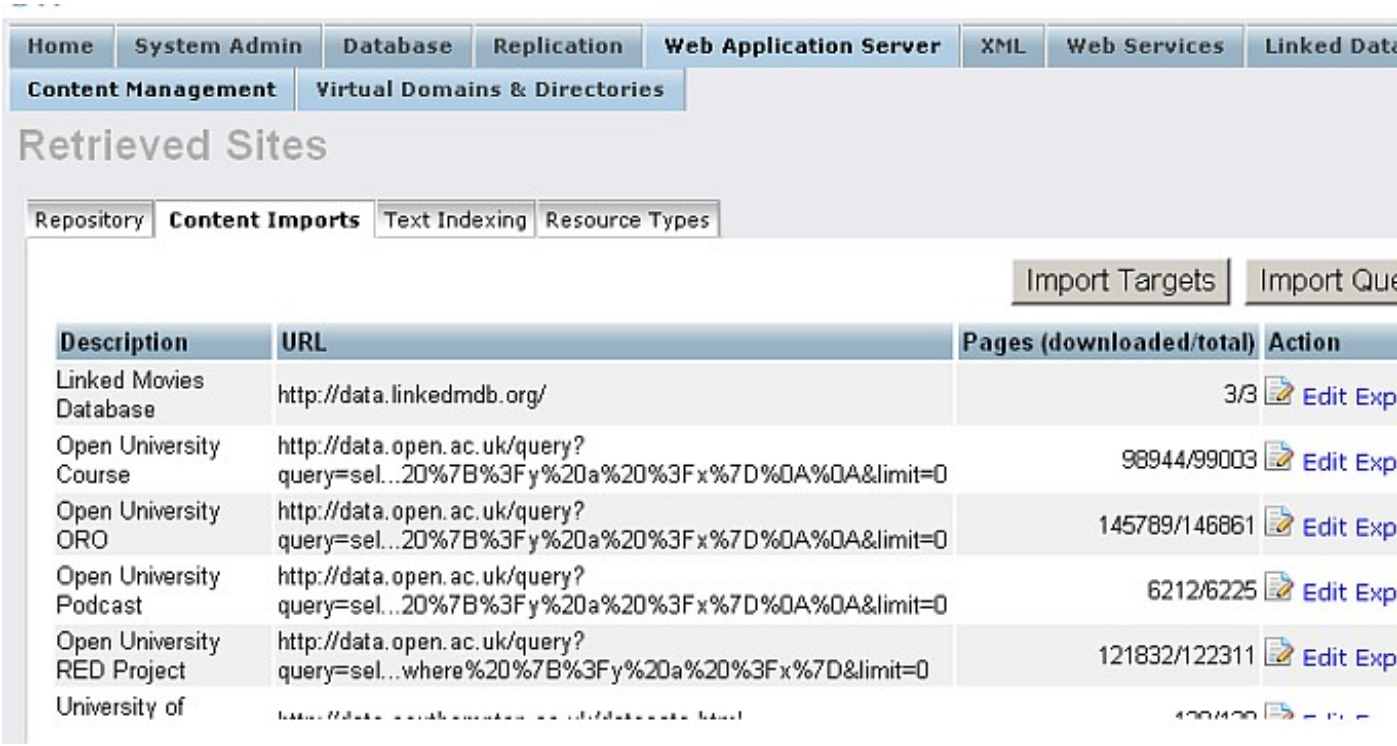
- *Semantic Web Crawling* : hatch to retrieve Semantic Sitemaps. If left empty, will be used a system Store function.
- *If Graph IRI is unassigned use this Data Source URL:* use to specify a custom graph URI for data storage.
- *Follow URLs outside of the target host* is check-box to specify either to follow URIS outside of the target host.
- *Follow HTML meta link* is check-box to specifiy either to follow HTML meta links.
- *Follow RDF properties (one IRI per row)* .
- *Download Images* radio buttons to allow Virtuoso to pull down image type also. You may want to prevent this if you are more interested in the textual content rather than bandwidth draining images.
- *Use WebDAV methods* can be checked if the host is known to support WebDAV methods. This would enable better copying of sites that support DAV.
- *Delete if remove on remote detected* can be switched on so that when Virtuoso synchronizes its content with that on the remote host it will check for files that have been removed on the remote and remove them from the local copy also.
- *Store metadata** when checked offers to be stored respectively metadata from FOAF, RDF, RSS/RDF and GRDLL data depending on which check-box is checked.
- When all details have been completed press the *Add* (or Update if updating) button to submit the web robot task to the queue.
- *Import Queues* : This page shows you a list of web copy targets that have been enlisted with the Virtuoso Server, and a list of web robot update schedules. Several options are available for each item listed: Start, Update, Schedule, Reset, and Stop. You can configure the scheduled update interval by pressing the Schedule link and entering a value in minutes. Once that is done you can start the schedule by pressing the Start link. You make a manual update of the content by pressing the Update link. You can stop the scheduled updates taking place by pressing the Stop link. To reset the details of the web copy item press the Reset link.

Figure 6.15. Web Robot - Queues



- *Retrieved Sites* : You can view a list of the links retrieved from the web copy from this page. You are also able to remove some of the content from this page by following the Edit link.

Figure 6.16. Web Robot - Retrieved Links



- *Export* : You can export content from the WebDAV repository. Note that you can only export content that has been retrieved using Virtuoso's Web Robot.

  When you click the "Export" link for a retrieved collection, you will be presented with a form for selecting the export target location. Choose the export method: either File System or DAV by clicking the "External WebDAV Server URL" check-box. This lets you indicate to the remote target where to store the exported content. Then type the target URL to an existing location on the server. Finally press the *Export* button to export. A confirmation will be supplied once the operation is complete.

Figure 6.17. Web Robot - Exporting Content

Note:

If is not checked the "External WebDAV Server URL" check-box, i.e. you are selecting the filesystem method, then you are restricted to Virtuoso targets. However WebDAV methods can be applied to any WebDAV server. WebDAV methods assume that the target is publicly available for writing.

- *Populate the RDF Quad Store* : Virtuoso's built-in Content Crawler to can be used to populate its RDF Quad Store, as a one-time run or on a scheduled basis.

  Transforming data sources into RDF "on the fly" is sufficient for many use cases, but there are times when the volume or sheer nature of a data source makes batch-loading necessary.

For example, Freebase offers RDF representations of its data, but it doesn't publish RDF dumps; even if it did, such dumps would usually be outdated by the time they were loaded.

One practical solution takes the form of a scheduled crawl of specific resources of interest.

Set Up the Content Crawler to Gather RDF

The Virtuoso Conductor can be used to set up various Content Crawler Jobs:

Setting up a Content Crawler Job to Add RDF Data to the Quad Store

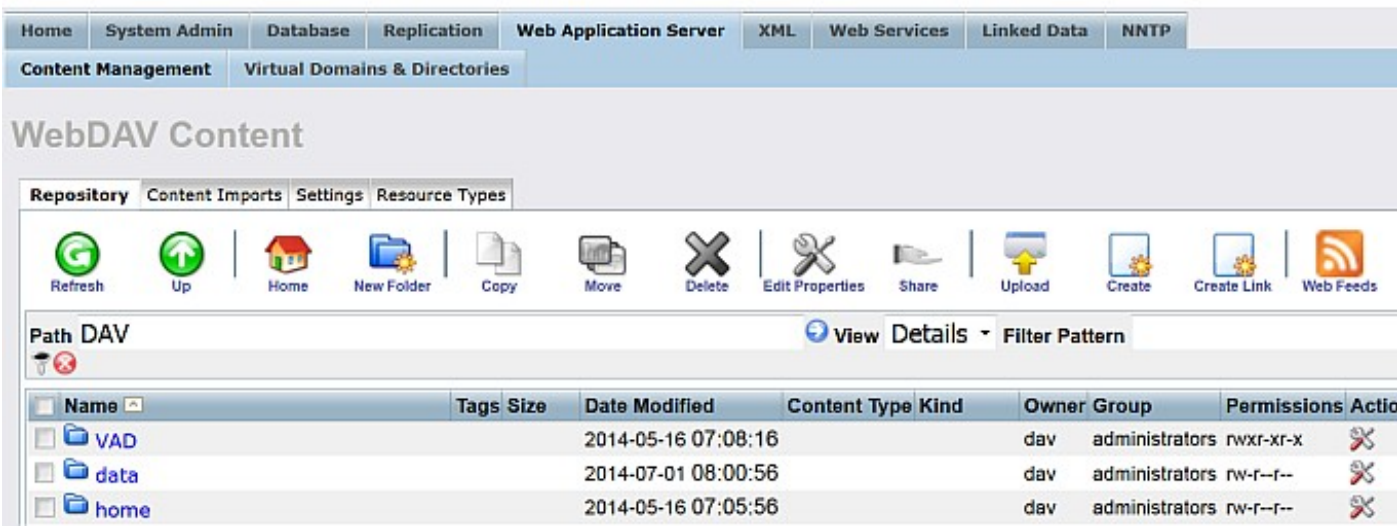
See details how to use Virtuoso Crawler for including the Sponger options so you crawl non-RDF but get RDF and this to the Quad Store.

Setting up a Content Crawler Job to Retrieve Sitemaps (when the source includes RDFa)

The following section describes how to set up a crawler job for getting content of a basic Sitemap where the source includes RDFa.

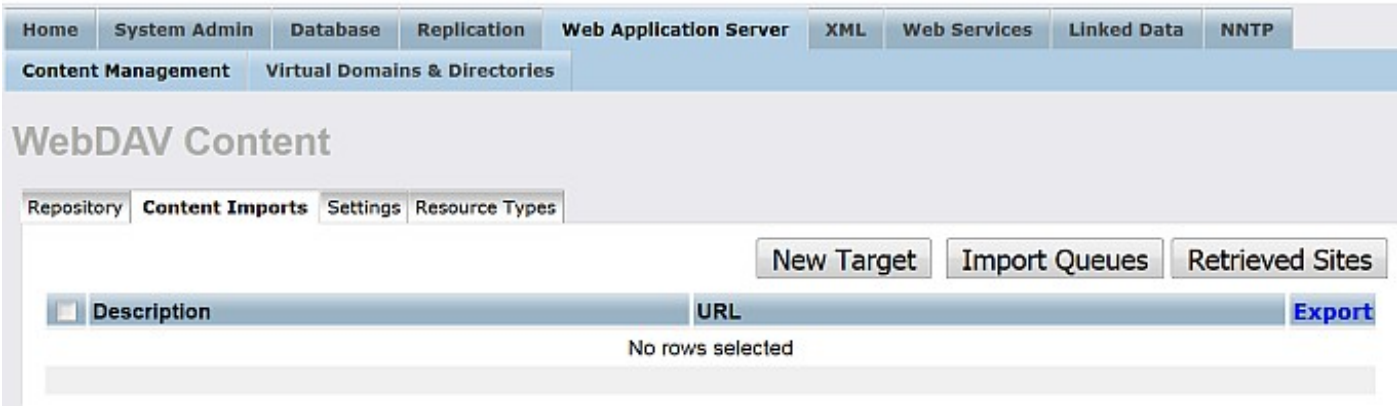
1. From the Virtuoso Conductor User Interface i.e. http://cname:port/conductor, login as the "dba" user.
2. Go to the "Web Application Server" tab.

Figure 6.18. Setting up a Content Crawler Job to Retrieve Sitemaps



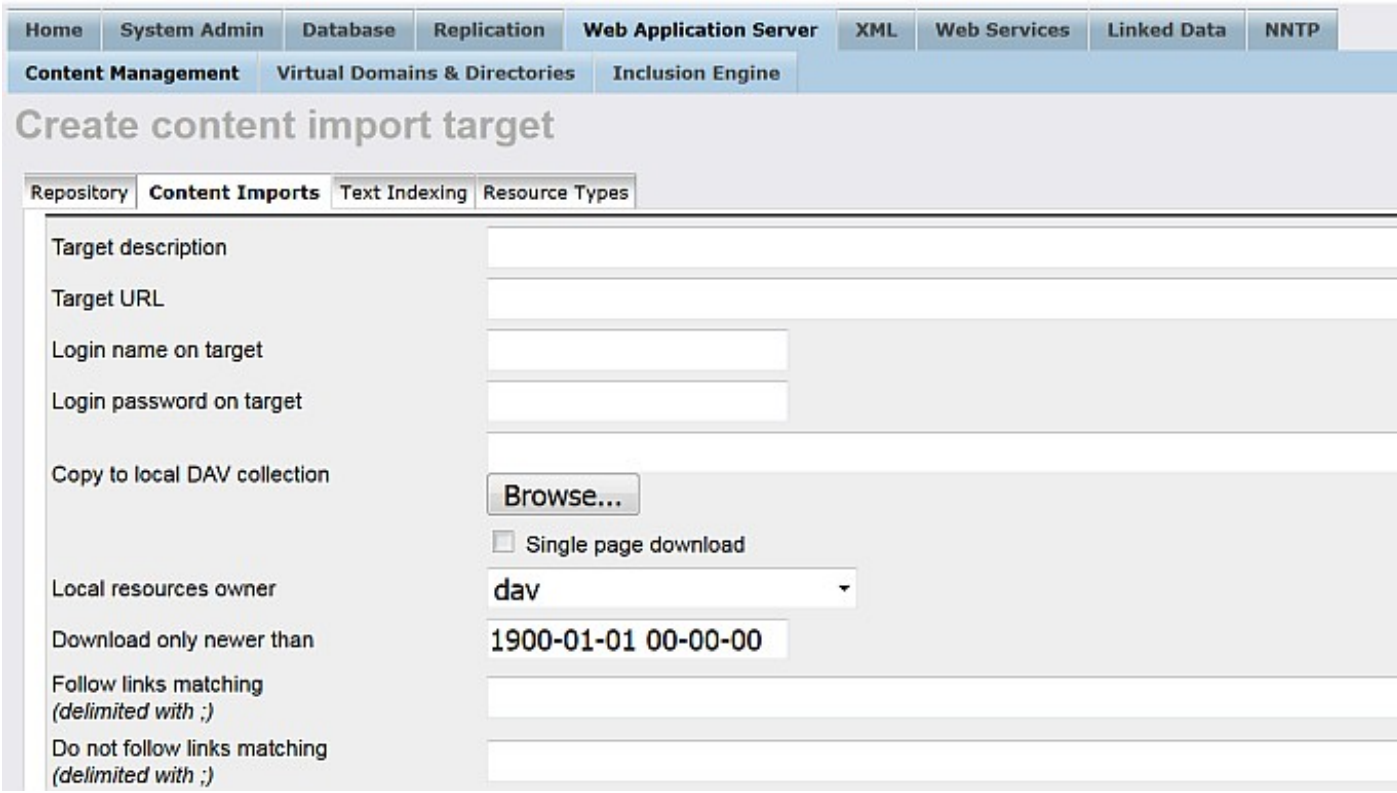
3. Go to the "Content Imports" tab.

Figure 6.19. Setting up a Content Crawler Job to Retrieve Sitemaps



4. Click on the "New Target" button.

Figure 6.20. Setting up a Content Crawler Job to Retrieve Sitemaps



5. In the form displayed:

   - Enter a name of choice in the "Target description" text box:

     ```
     Virtuoso Sample Example
     ```

   - Enter the URL of the site to be crawled in the "Target URL" text box:

     ```
     http://virtuoso.openlinksw.com/sitemap.xml
     ```

   - Enter the location in the Virtuoso WebDAV repository the crawled should stored, in the "Copy to local DAV collection" text box, for example, if user demo is available, then:

     ```
     /DAV/home/demo/VirtSample/
     ```

   - Choose the "Local resources owner" for the collection from the list box available, for ex: user demo.
   - Select the "Accept RDF" check box.
   - Optionally you can select "Convert Links" to make all HREFs in the local stored content relative.

Figure 6.21. Setting up a Content Crawler Job to Retrieve Sitemaps

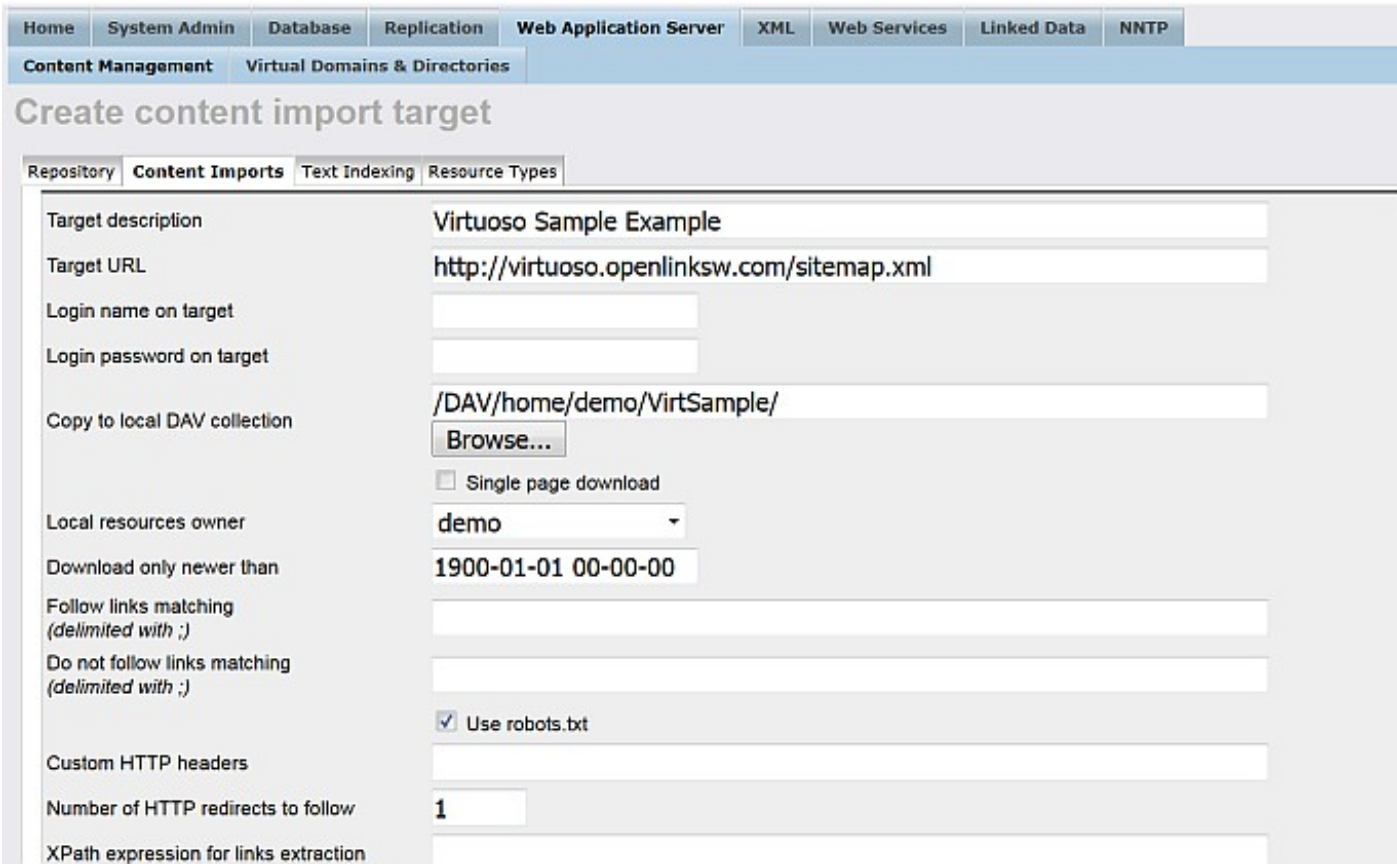
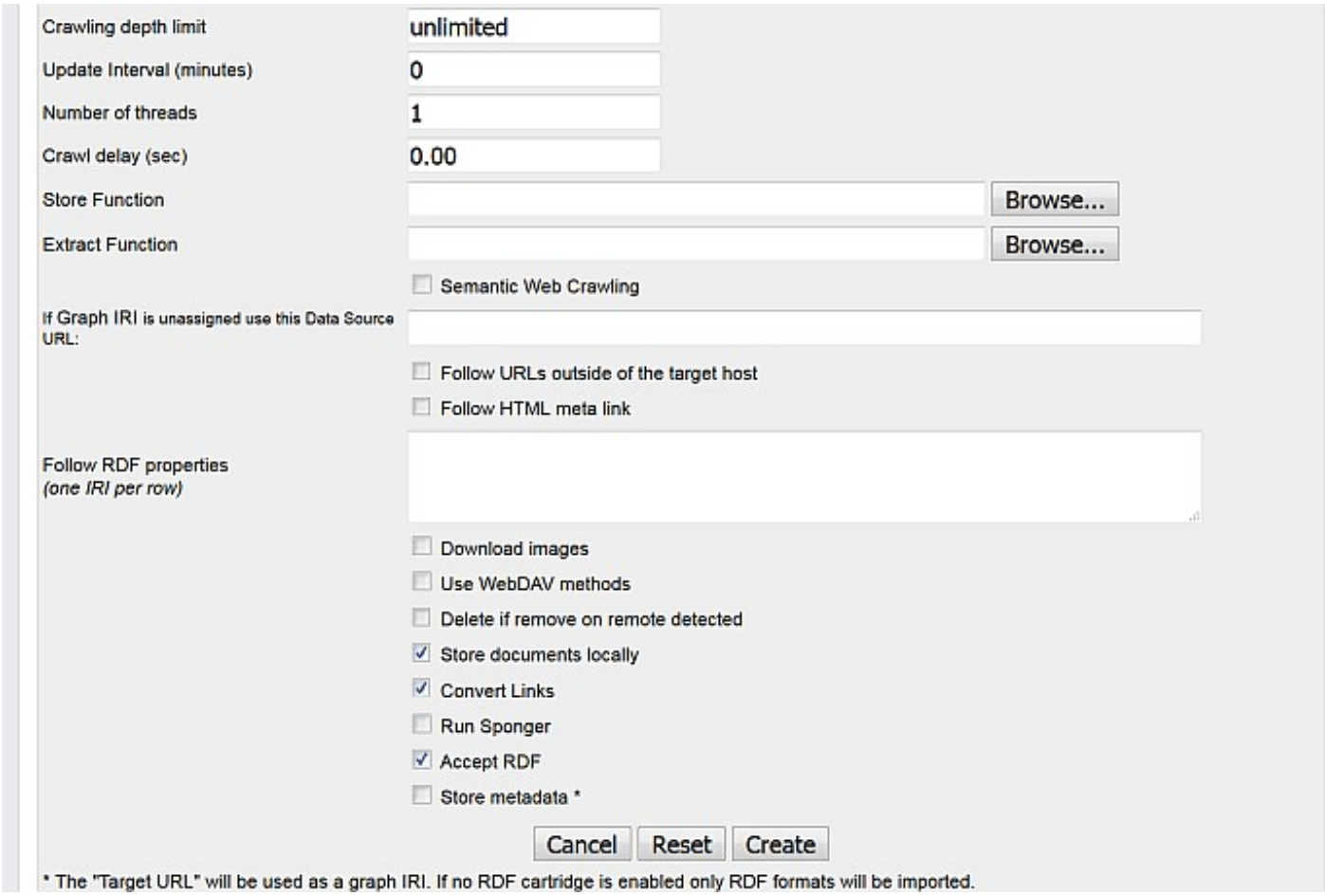
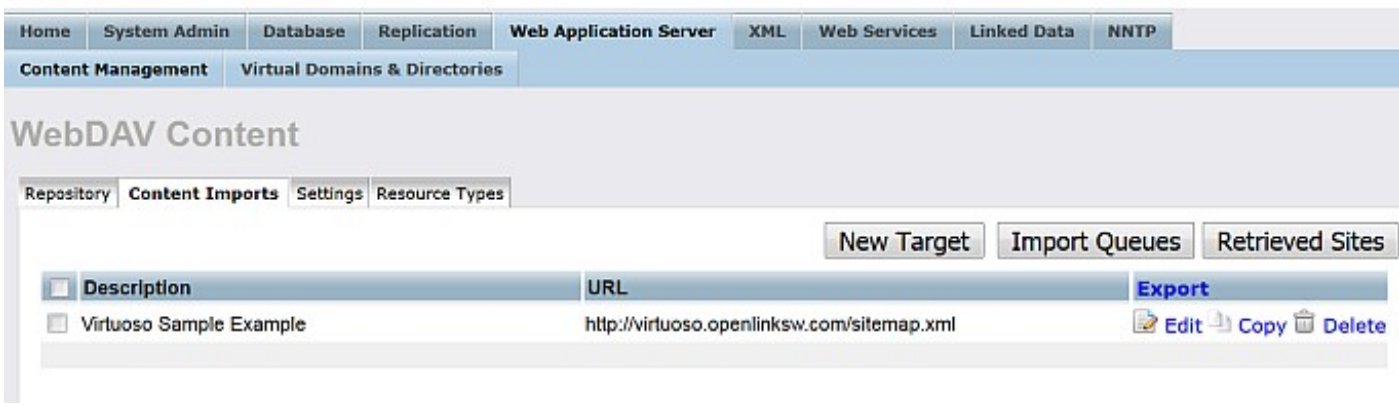


Figure 6.22. Setting up a Content Crawler Job to Retrieve Sitemaps



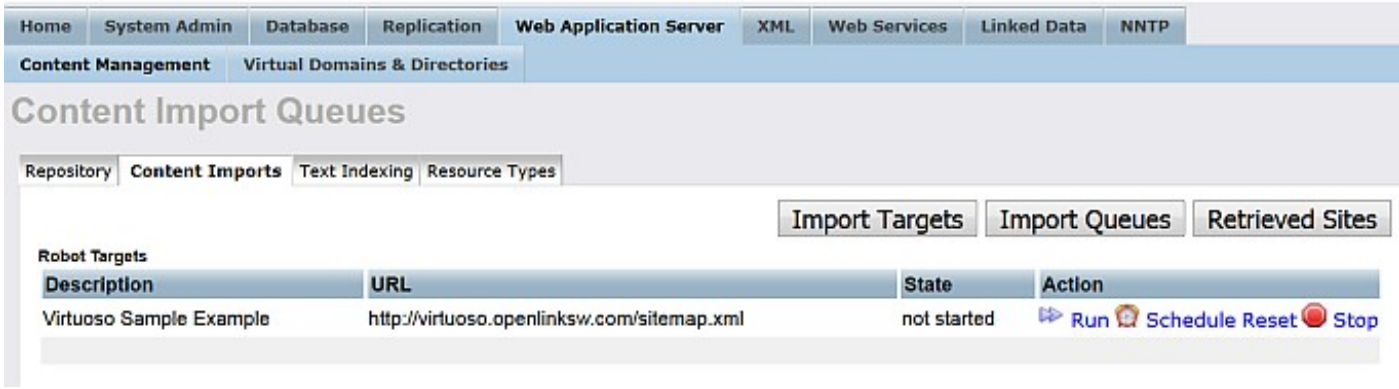
6. Click the "Create" button to create the import.

Figure 6.23. Setting up a Content Crawler Job to Retrieve Sitemaps



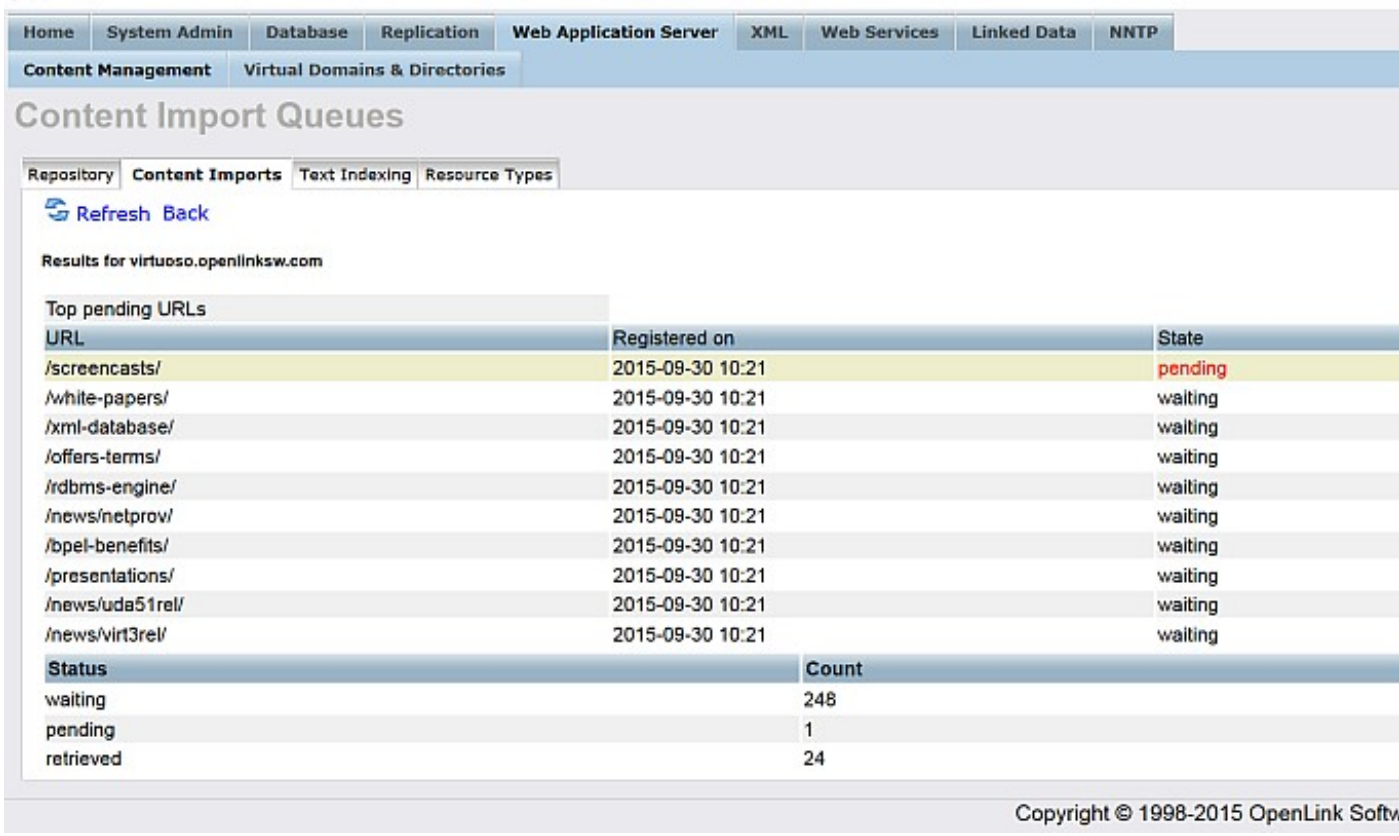
7. Click the "Import Queues" button.

Figure 6.24. Setting up a Content Crawler Job to Retrieve Sitemaps



8. For the "Robot targets" with label "Virtuoso Sample Example" click the "Run" button.
9. This will result in the Target site being crawled and the retrieved pages stored locally in DAV and any network resource triples being fetched in the RDF Quad store.

Figure 6.25. Setting up a Content Crawler Job to Retrieve Sitemaps

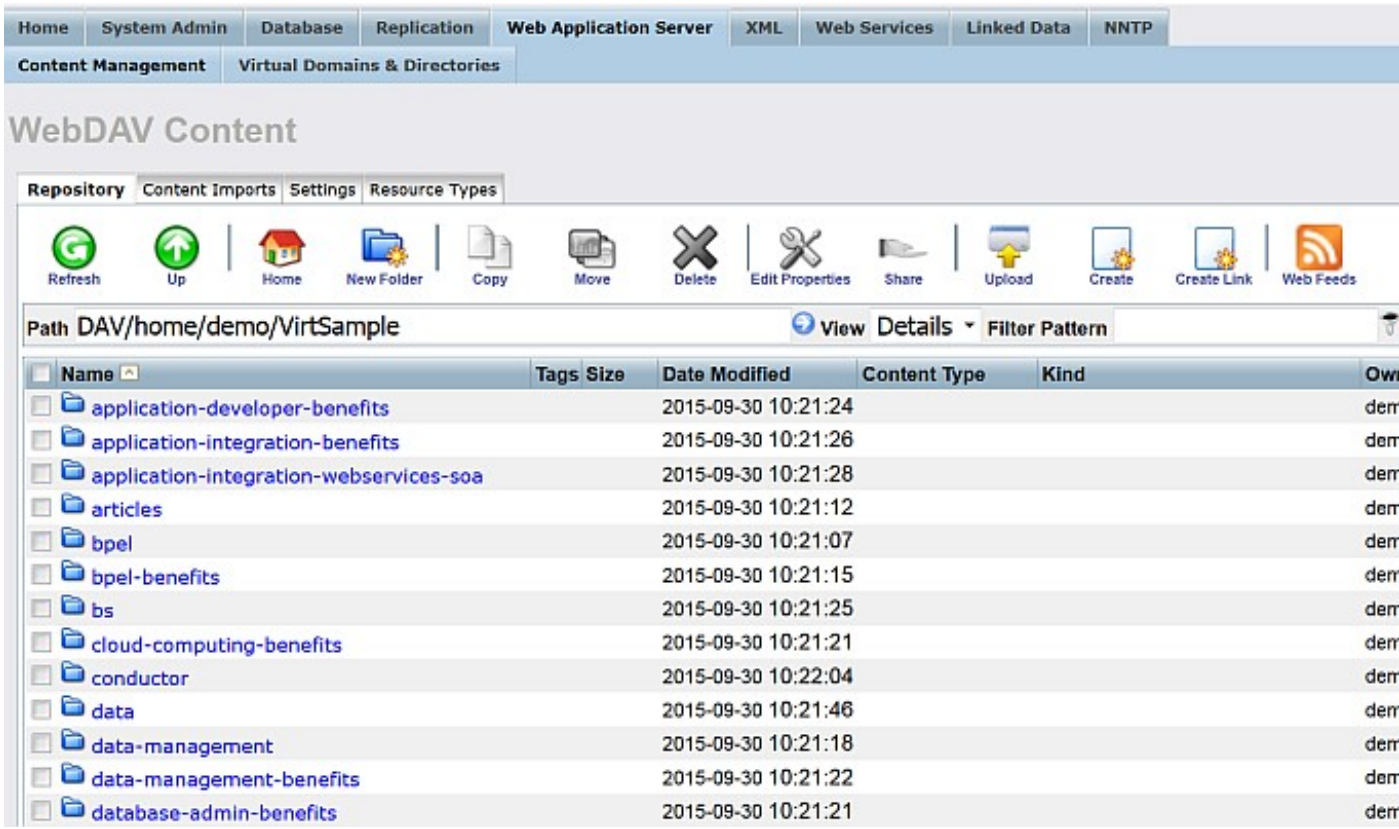


10. Go to the "Web Application Server" -> "Content Management" tab and navigate to the location of newly created DAV collection:

    ```
    /DAV/home/demo/VirtSample/
    ```

11. The retrieved content will be shown:

Figure 6.26. Setting up a Content Crawler Job to Retrieve Sitemaps

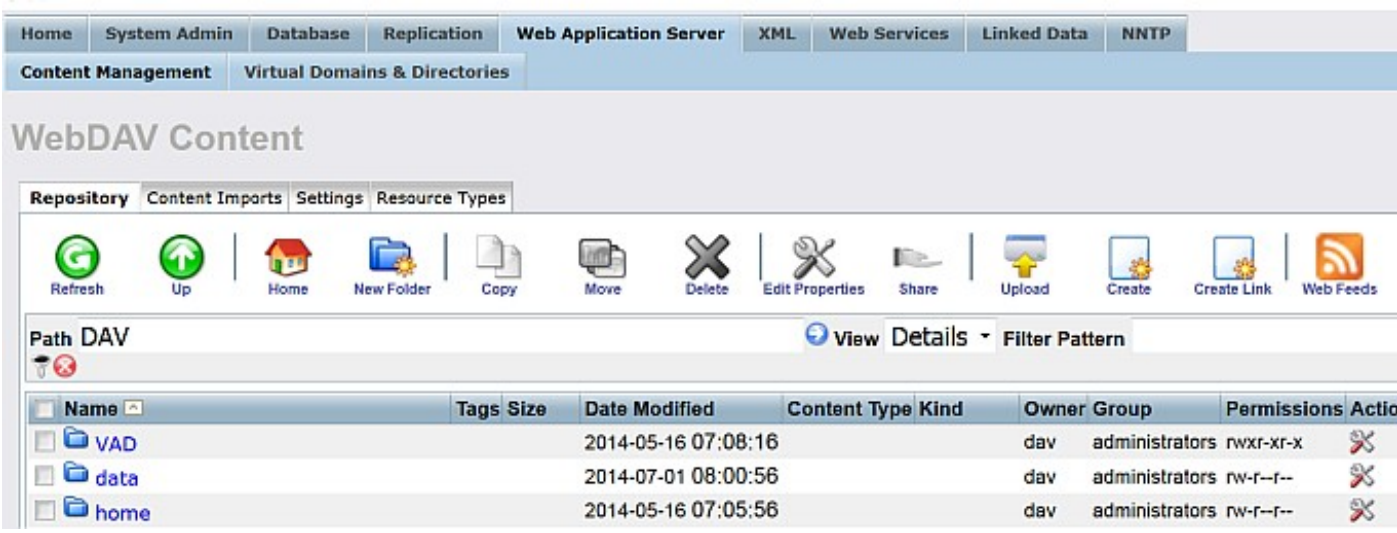


Setting up a Content Crawler Job to Retrieve Semantic Sitemaps (a variation of the standard sitemap)

The following section describes how to set up crawler job for getting Semantic Sitemap's content: a variation of standard sitemap:

1. From the Virtuoso Conductor User Interface i.e. http://cname:port/conductor, login as the "dba" user.
2. Go to "Web Application Server".

Figure 6.27. Setting up a Content Crawler Job to Retrieve Semantic Sitemap content

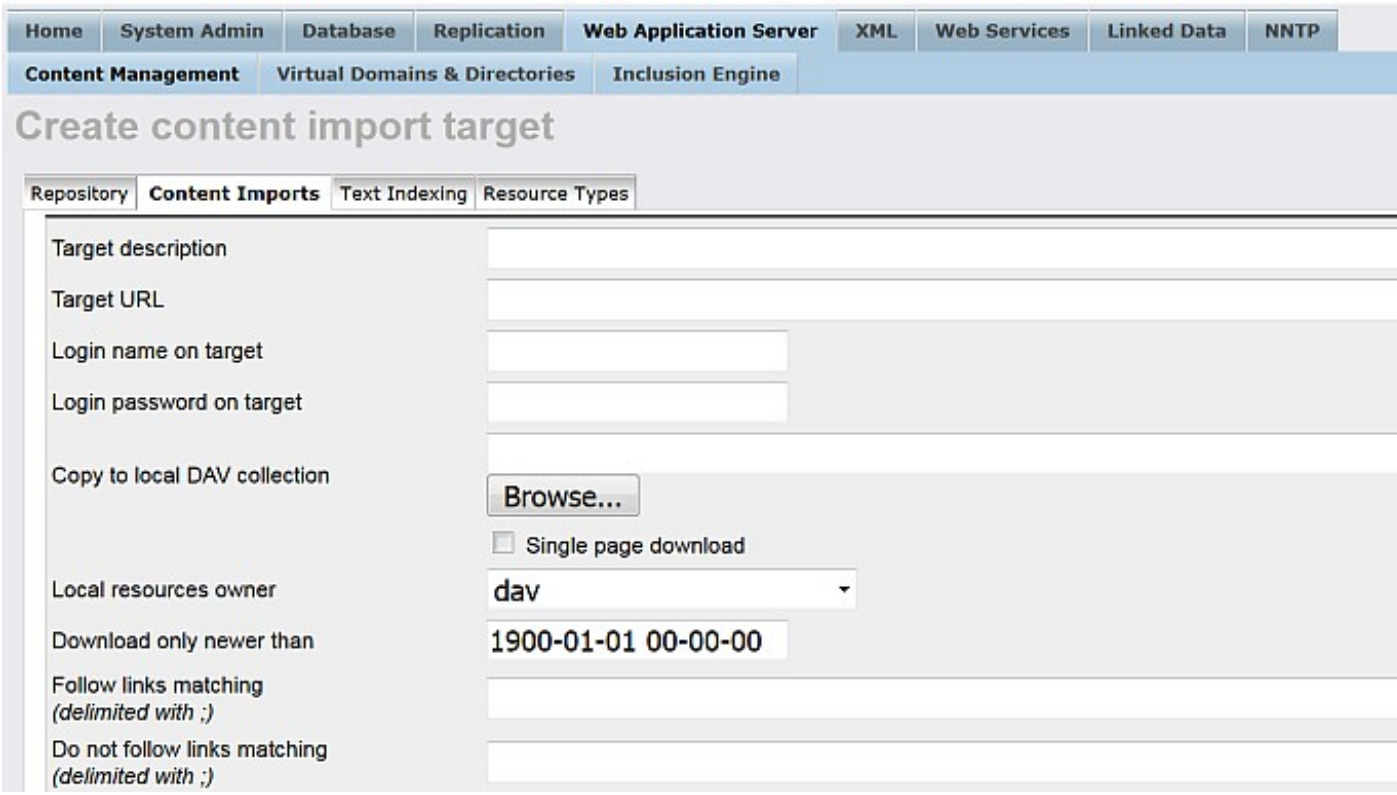


3. Go to "Content Imports".

Figure 6.28. Setting up a Content Crawler Job to Retrieve Semantic Sitemap content

4. Click "New Target".

Figure 6.29. Setting up a Content Crawler Job to Retrieve Semantic Sitemap content



5. In the shown form:

   - Enter for "Target description":

     ```
     Semantic Web Sitemap Example
     ```

   - Enter for "Target URL":

     ```
     http://virtuoso.openlinksw.com/sitemap-semantic.xml
     ```

   - Enter the location in the Virtuoso WebDAV repository the crawled should stored in the "Copy to local DAV collection" text box, for example, if user demo is available, then:

     ```
     /DAV/home/demo/semantic_sitemap/
     ```

   - Choose the "Local resources owner" for the collection from the list box available, for ex: user demo.
   - Hatch "Semantic Web Crawling"

     - Note: when you select this option, you can either:

       a. Leave the Store Function and Extract Function empty - in this case the system Store and Extract functions will be used for the Semantic Web Crawling Process, or:
       b. You can select your own Store and Extract Functions, for ex:

```
-- Example of Extract Function
use WS;

create procedure WS.WS.SITEMAP_BB_PARSE (
  in _host varchar,
  in _url varchar,
  in _root varchar,
  inout _content varchar,
  in _c_type varchar := null,
  in lev int := 0))
{
  --pl_debug+
  declare xt, xp, graph any;
  declare inx int;

--  dbg_obj_print ('WS.WS.GET_URLS_SITEMAP', _url);

  declare exit handler for sqlstate '*'
```

```
    {
--      dbg_obj_print (__SQL_MESSAGE);
      return;
    };

  if (_url like '%.xml.gz')
    {
      _content := gzip_uncompress (_content);
    }

  if (_url like '%.xml' or _url like '%.xml.gz' or _url like '%.rdf')
    {
      xt := xtree_doc (_content);
      if (xpath_eval ('/urlset/dataset', xt) is not null)
        {
          xp := xpath_eval ('/urlset/dataset/dataDumpLocation/text()', xt, 0
          graph := cast (xpath_eval ('/urlset/dataset/datasetURI/text()', xt
          if (length (graph))
            update VFS_SITE set VS_UDATA = serialize (vector ('graph', graph
          inx := 0;
          foreach (any u in xp) do
            {
              declare hf, host, url varchar;

              u := cast (u as varchar);
              hf := WS.WS.PARSE_URI (u);
              host := hf[1];
              --dbg_obj_print ('WS.WS.GET_URLS_SITEMAP PARSE', u);
              url := hf[2];
              insert soft VFS_QUEUE (VQ_HOST, VQ_TS, VQ_URL, VQ_STAT, VQ_ROO
                  values (host, now (), url, 'waiting', _root, NULL);
              if (row_count () = 0)
                update VFS_QUEUE set VQ_STAT = 'waiting', VQ_TS = now () whe
              inx := inx + 1;
            }
        }
      if (xpath_eval ('/sitemapindex/sitemap/loc', xt) is not null)
        {
          xp := xpath_eval ('/sitemapindex/sitemap/loc/text()', xt, 0);
          inx := 0;
          foreach (any u in xp) do
            {
              declare hf, host, url varchar;

              u := trim (cast (u as varchar));
              hf := WS.WS.PARSE_URI (u);
              host := hf[1];
--            dbg_obj_print ('WS.WS.GET_URLS_SITEMAP', host, _host);
              url := hf[2];
              if (url <> '')
                {
                  insert soft VFS_QUEUE (VQ_HOST, VQ_TS, VQ_URL, VQ_STAT, VQ
                      values (host, now (), url, 'waiting', _root, NULL);
                  if (row_count () = 0)
                    update VFS_QUEUE set VQ_STAT = 'waiting', VQ_TS = now ()
                  inx := inx + 1;
                }
            }
        }
    }
  commit work;
}
;

-- Example of Store Function
use WS;

create procedure WS.WS.SITEMAP_BB_STORE (
  in _host varchar,
  in _url varchar,
  in _root varchar,
  inout _content varchar,
  in _s_etag varchar,
```

```
    in _c_type varchar,
    in store_flag int := 1,
    in udata any := null,
    in lev int := 0)
{
  --pl_debug+
  declare graph varchar;

--  dbg_obj_print ('WS.WS.SITEMAP_BB_STORE', _url, udata);
  if (isarray (udata))
    graph := get_keyword ('graph', udata);
  else
    graph := null;

  if (graph is not null and _url like '%.rdf')
    {
      DB.DBA.RDF_LOAD_RDFXML (_content, graph, graph);
      DB.DBA.VT_INC_INDEX_DB_DBA_RDF_OBJ ();
    }
  insert soft VFS_URL (VU_HOST, VU_URL, VU_CHKSUM, VU_CPTIME, VU_ETAG, VU_RO
      values (_host, _url, md5 (_content), now (), _s_etag, _root);
  if (row_count () = 0)
    update VFS_URL set VU_CHKSUM = md5 (_content), VU_CPTIME = now (), VU_ET
        VU_HOST = _host and VU_URL = _url and VU_ROOT = _root;
  commit work;
}
;
```

6. Optionally you can select "Convert Link" to make all HREFs in the local stored content relative.
7. Hatch "Accept RDF"

Figure 6.30. Setting up a Content Crawler Job to Retrieve Semantic Sitemap content

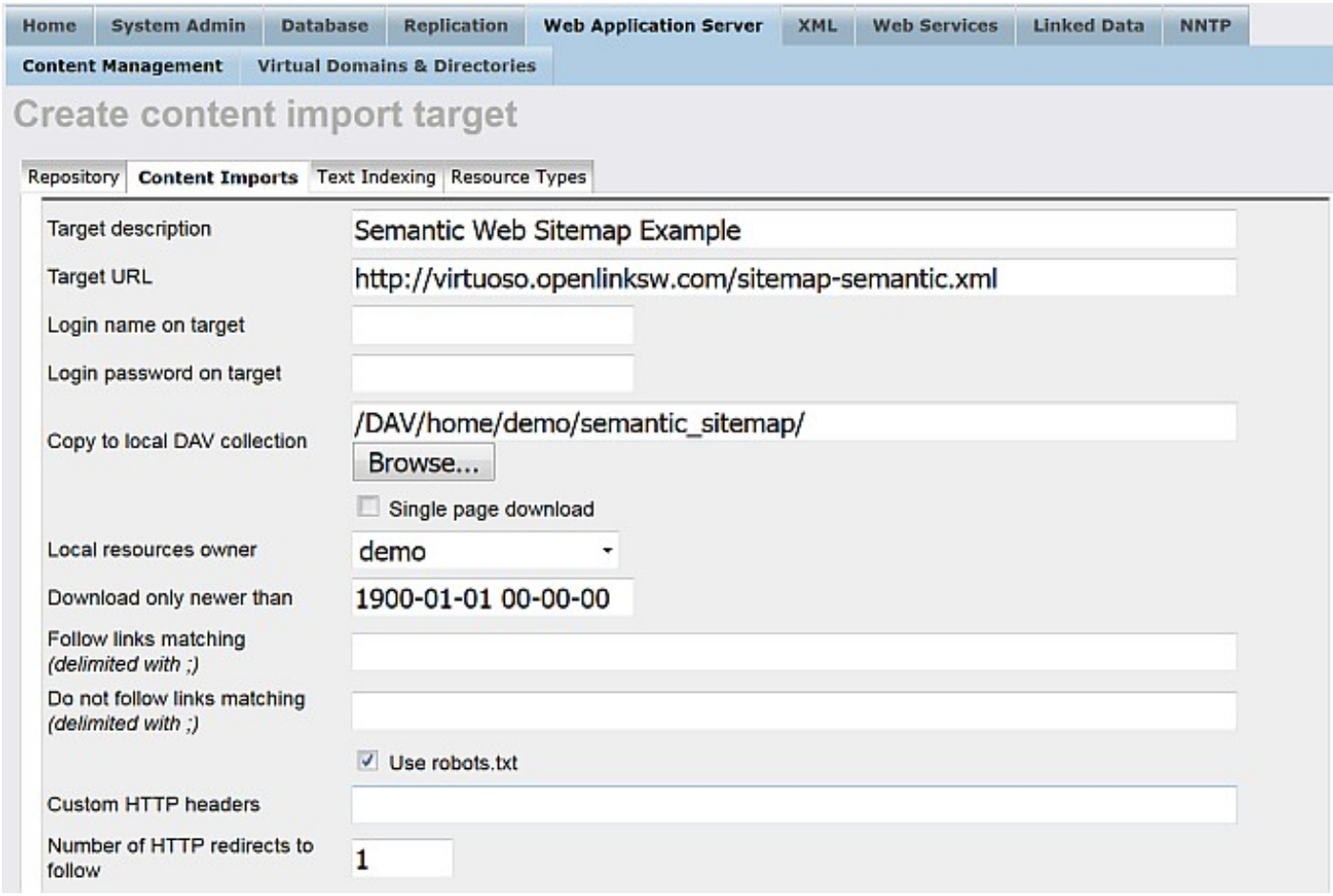


Figure 6.31. Setting up a Content Crawler Job to Retrieve Semantic Sitemap content

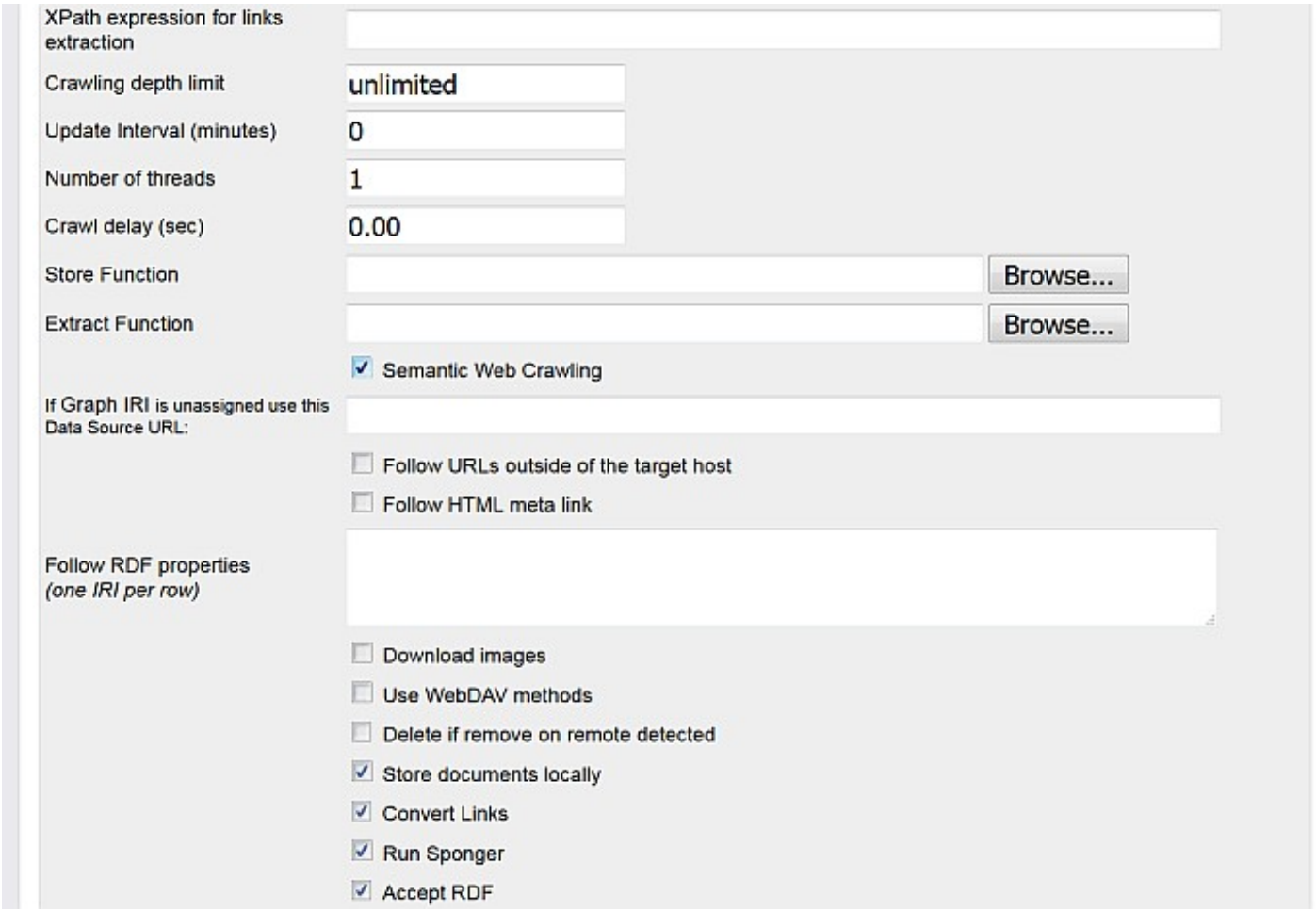
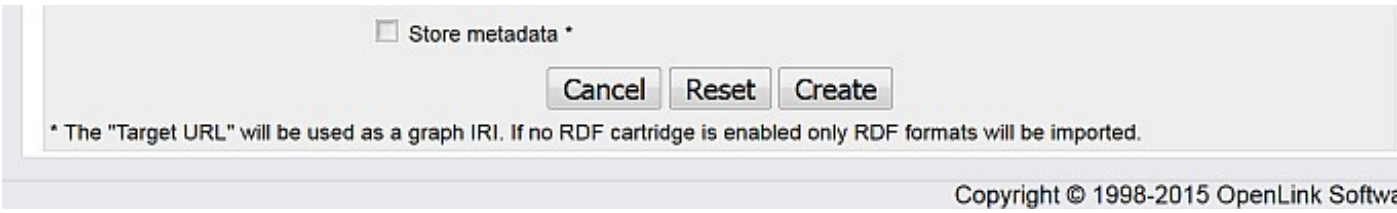
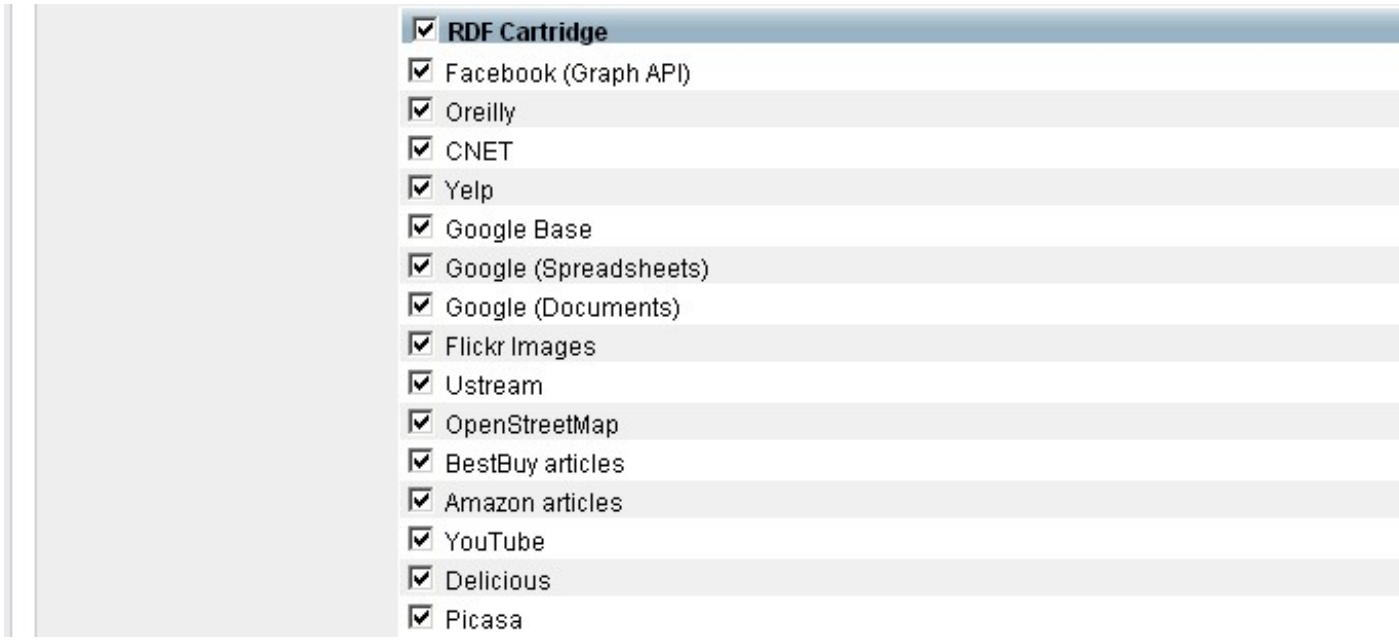


Figure 6.32. Setting up a Content Crawler Job to Retrieve Semantic Sitemap content



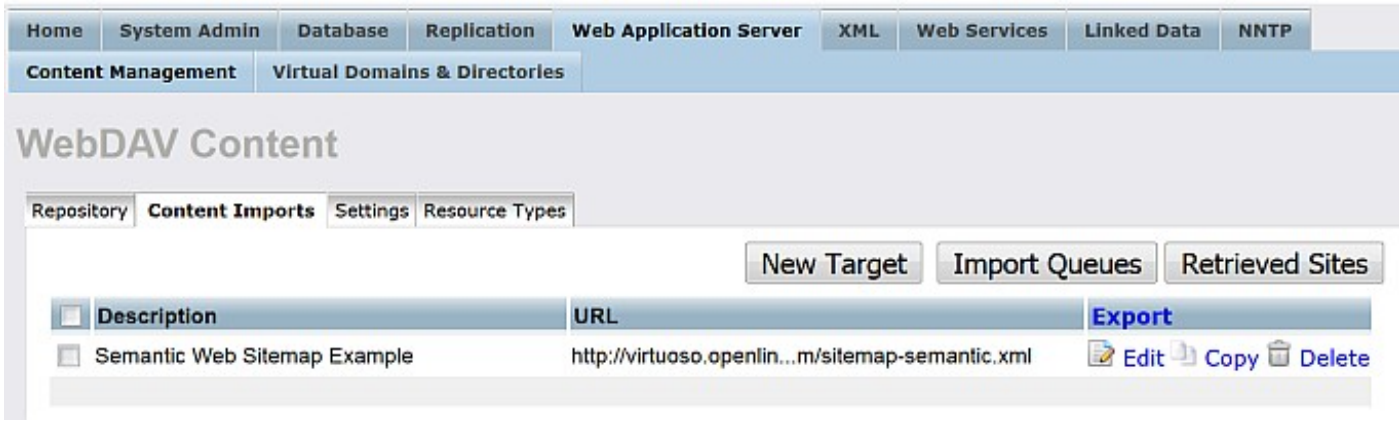
8. Optionally you can hatch "Store metadata *" and specify which RDF Cartridges to be included from the Sponger:

Figure 6.33. Setting up a Content Crawler Job to Retrieve Semantic Sitemap content



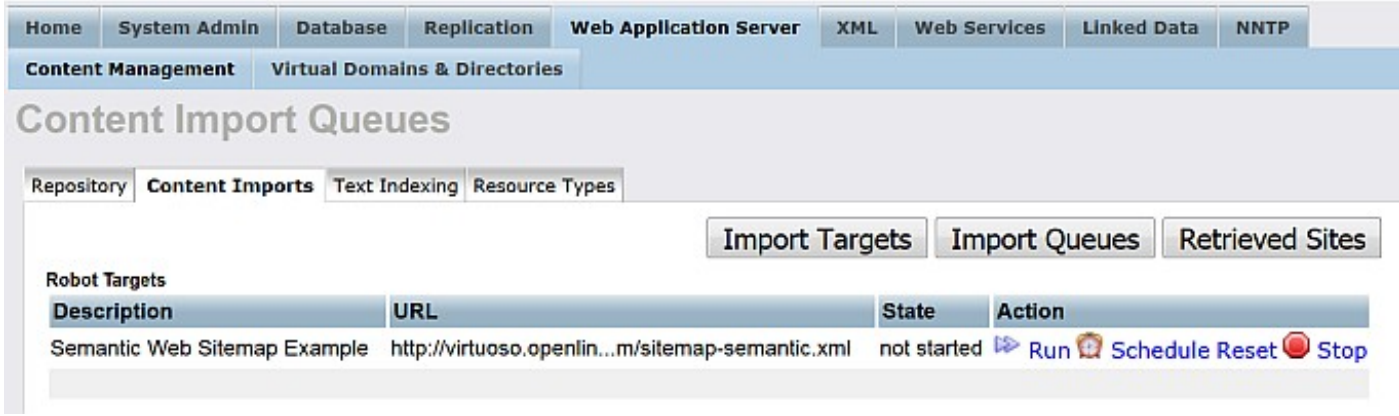
9. Click the button "Create".

Figure 6.34. Setting up a Content Crawler Job to Retrieve Semantic Sitemap content



10. Click "Import Queues".

Figure 6.35. Setting up a Content Crawler Job to Retrieve Semantic Sitemap content



11. For "Robot target" with label "Semantic Web Sitemap Example" click "Run".
12. As result should be shown the number of the pages retrieved.

Figure 6.36. Setting up a Content Crawler Job to Retrieve Semantic Sitemap content

13. Check the retrieved RDF data from your Virtuoso instance sparql endpoint http://cname:port/sparql with the following query selecting all the retrieved graphs for ex:

```
SELECT ?g
FROM <http://host:port/>
WHERE
  {
    graph ?g { ?s ?p ?o } .
    FILTER ( ?g LIKE <http://virtuoso.openlinksw.com/%> )
  }
```

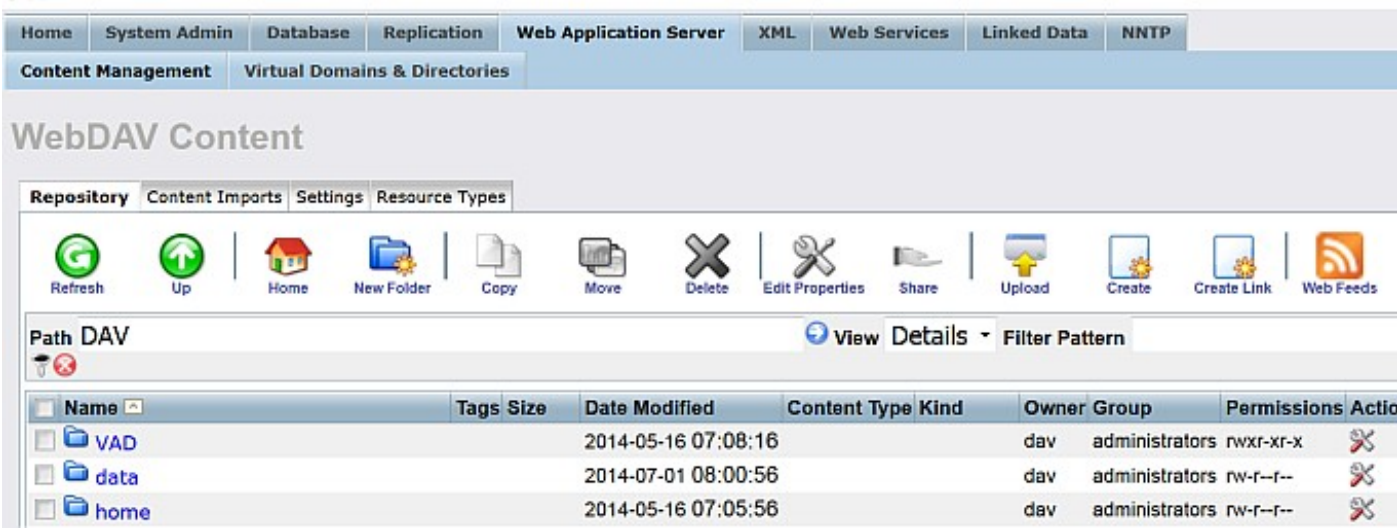
Figure 6.37. Setting up a Content Crawler Job to Retrieve Semantic Sitemap content

Figure 6.38. Setting up a Content Crawler Job to Retrieve Semantic Sitemap content

The following section describes how to set up crawler job for getting directories using Conductor.

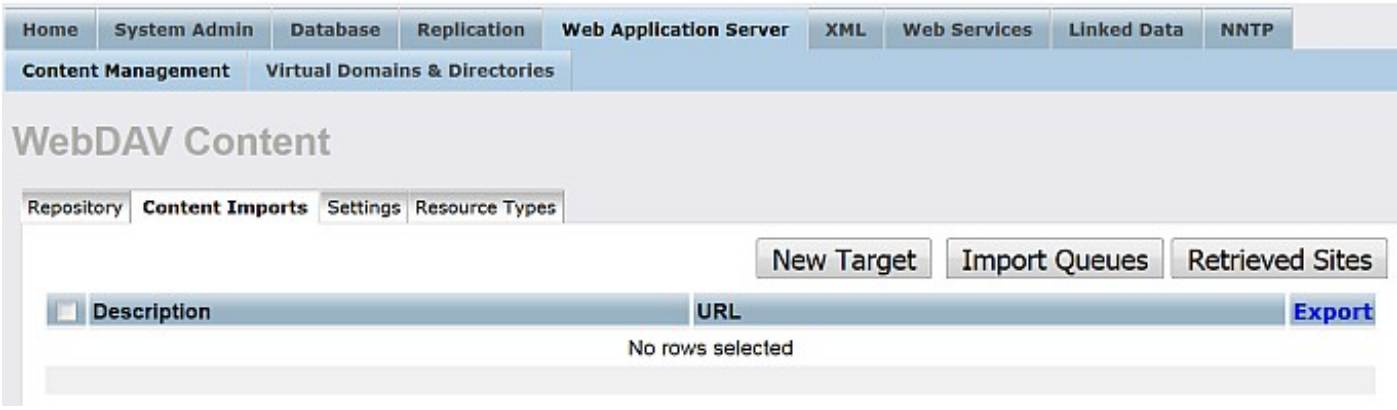
1. From the Virtuoso Conductor User Interface i.e. http://cname:port/conductor, login as the "dba" user.
2. Go to "Web Application Server".

Figure 6.39. Setting up a Content Crawler Job to Retrieve Content from Specific Directories



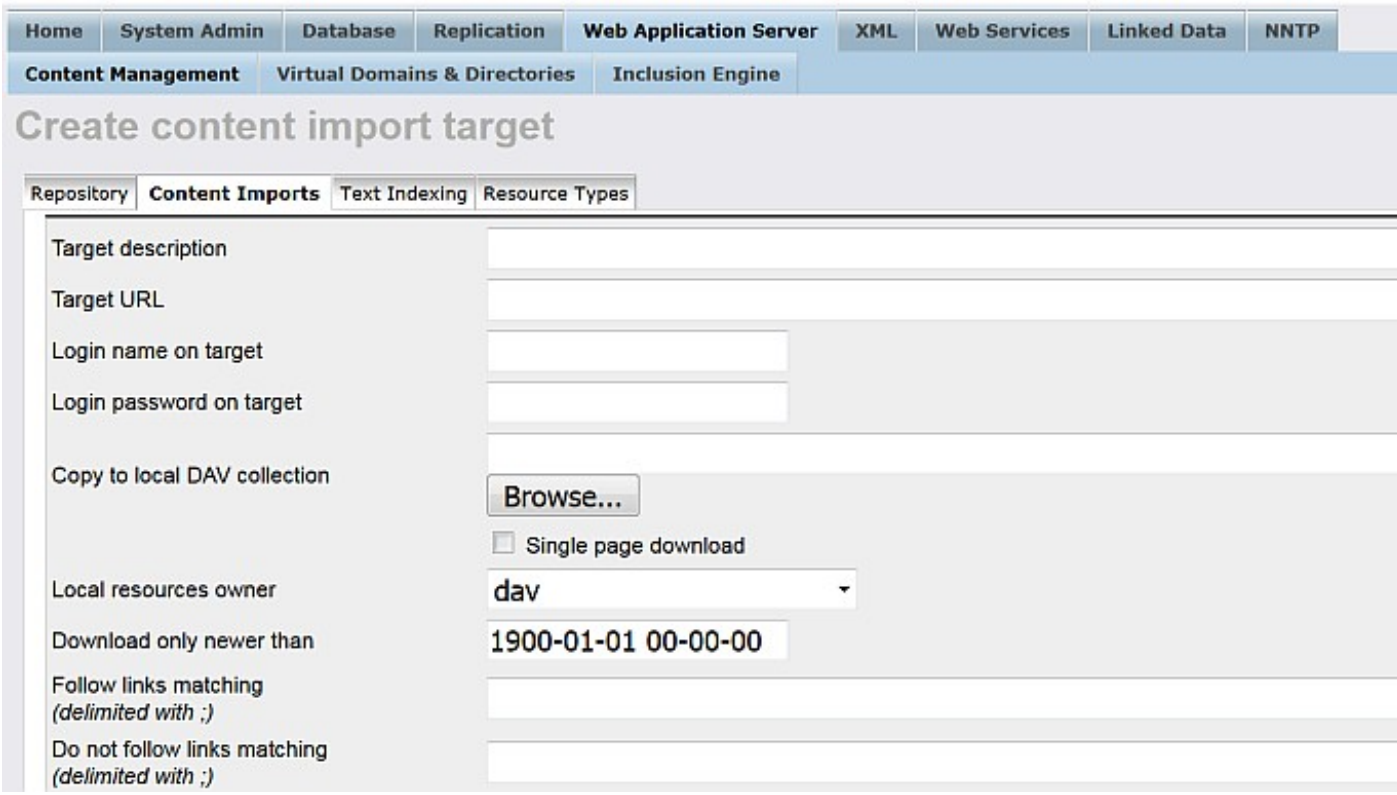
3. Go to "Content Imports".

Figure 6.40. Setting up a Content Crawler Job to Retrieve Content from Specific Directories



4. Click "New Target".

Figure 6.41. Setting up a Content Crawler Job to Retrieve Content from Specific Directories



5. In the shown form:

   - Enter for "Target description":

     ```
     UDA WebSite
     ```

   - Enter for "Target URL":

     ```
     http://uda.openlinksw.com/
     ```

   - Enter for "Copy to local DAV collection" for available user, for ex. demo:

     ```
     /DAV/home/demo/uda/
     ```

   - Choose from the available list "Local resources owner" an user, for ex. demo.

Figure 6.42. Setting up a Content Crawler Job to Retrieve Content from Specific Directories

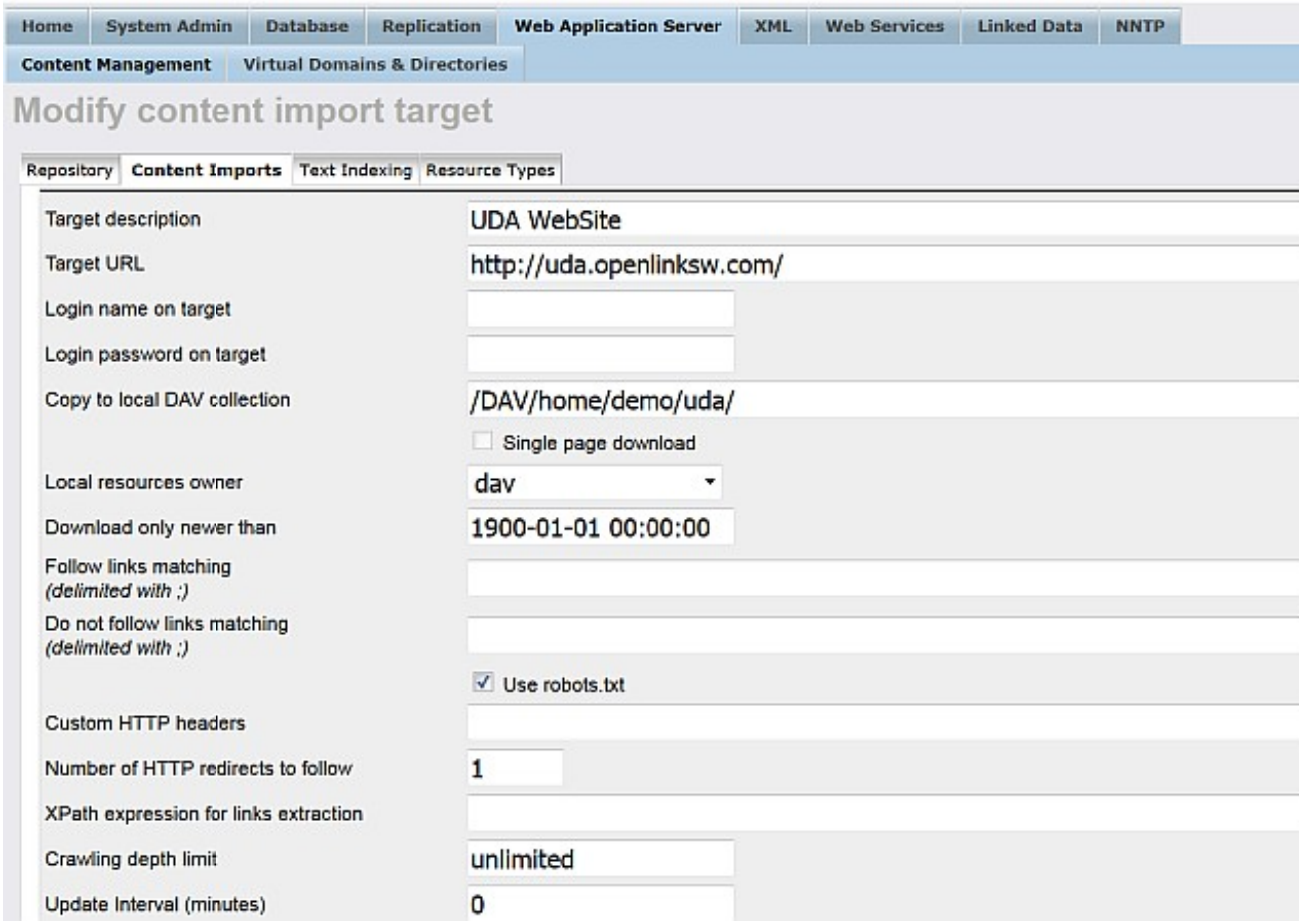
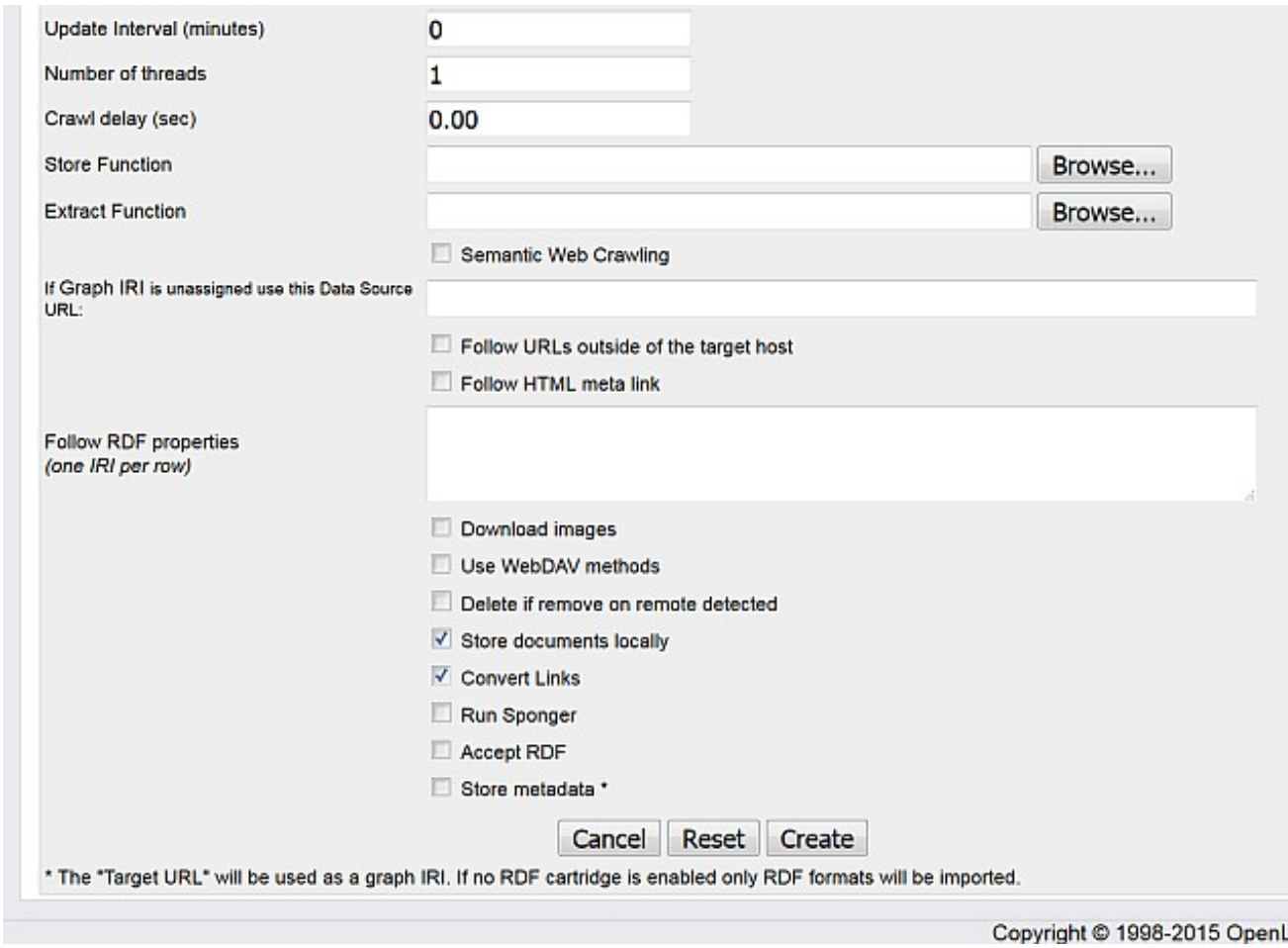


Figure 6.43. Setting up a Content Crawler Job to Retrieve Content from Specific Directories



- Optionally you can select "Convert Link" to make all HREFs in the local stored content relative.
- Click the button "Create".

6. As result the Robot target will be created:

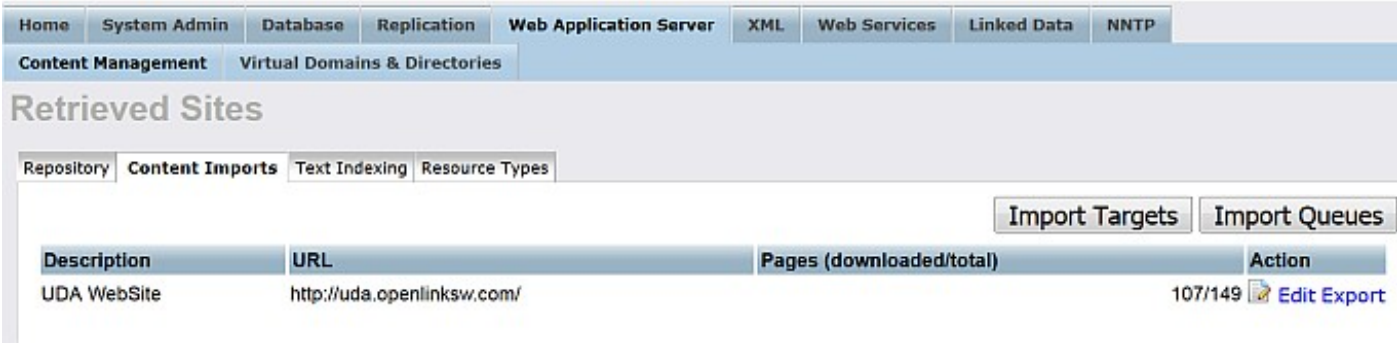
Figure 6.44. Setting up a Content Crawler Job to Retrieve Content from Specific Directories

7. Click "Import Queues".

Figure 6.45. Setting up a Content Crawler Job to Retrieve Content from Specific Directories

8. Click "Run" and go to "Retrieved Sites".
9. As result should be shown the number of the total pages retrieved.

Figure 6.46. Setting up a Content Crawler Job to Retrieve Content from Specific Directories

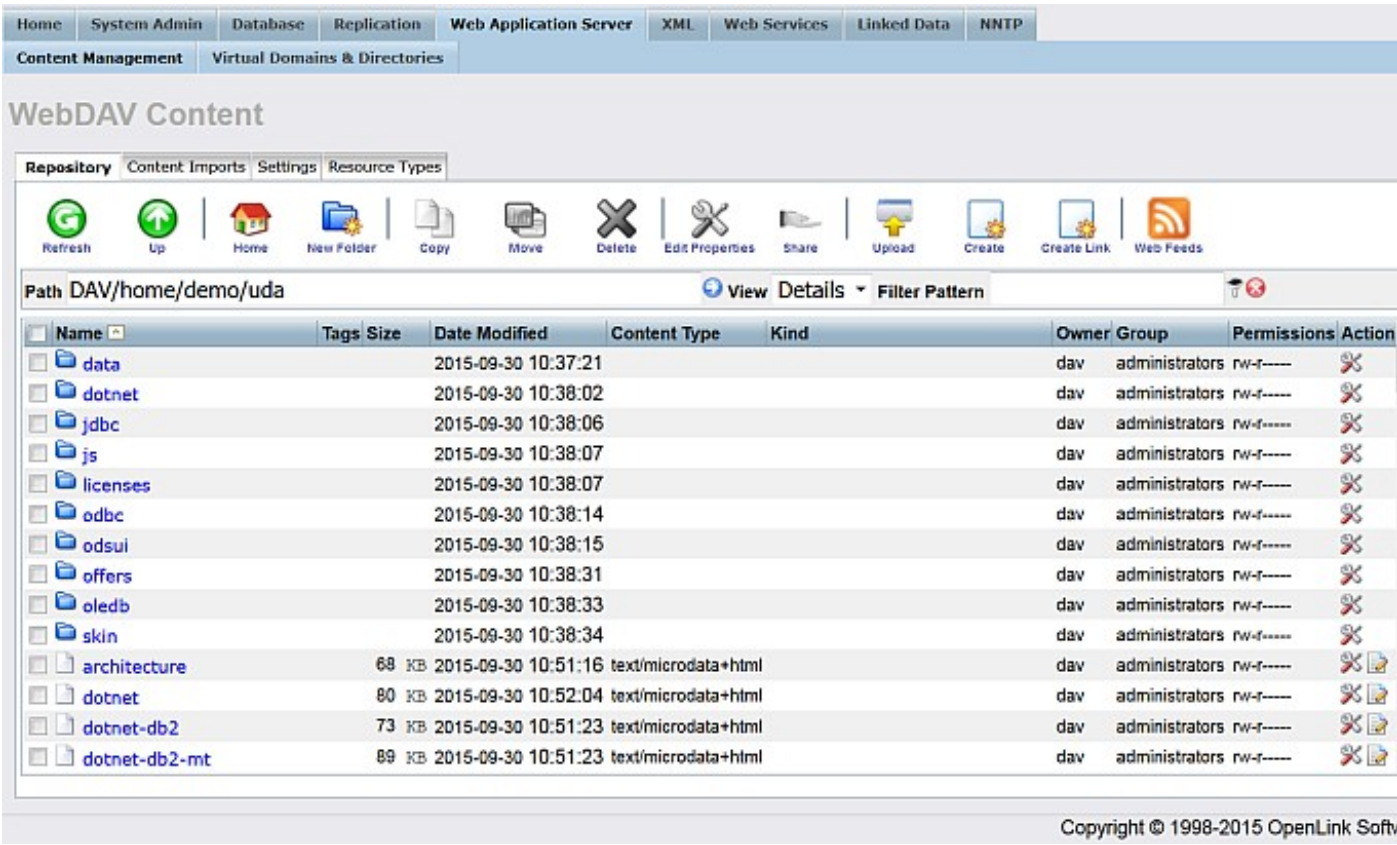


10. Go to "Web Application Server" -> "Content Management" and go to path:

```
DAV/home/demo/uda/
```

11. As result the retrieved content will be shown:

Figure 6.47. Setting up a Content Crawler Job to Retrieve Content from Specific Directories



Setting up a Content Crawler Job to Retrieve Content from ATOM feed

This section demonstrates populating the Virtuoso Quad Store using ATOM feed.

Populating the Virtuoso Quad Store can be done in different ways Virtuoso supports. The Conductor -> Content Import UI offers plenty of options, one of which is the XPath expression for crawling RDF resources URLs and this feature is a powerful and easy-to-use for managing the Quad Store.

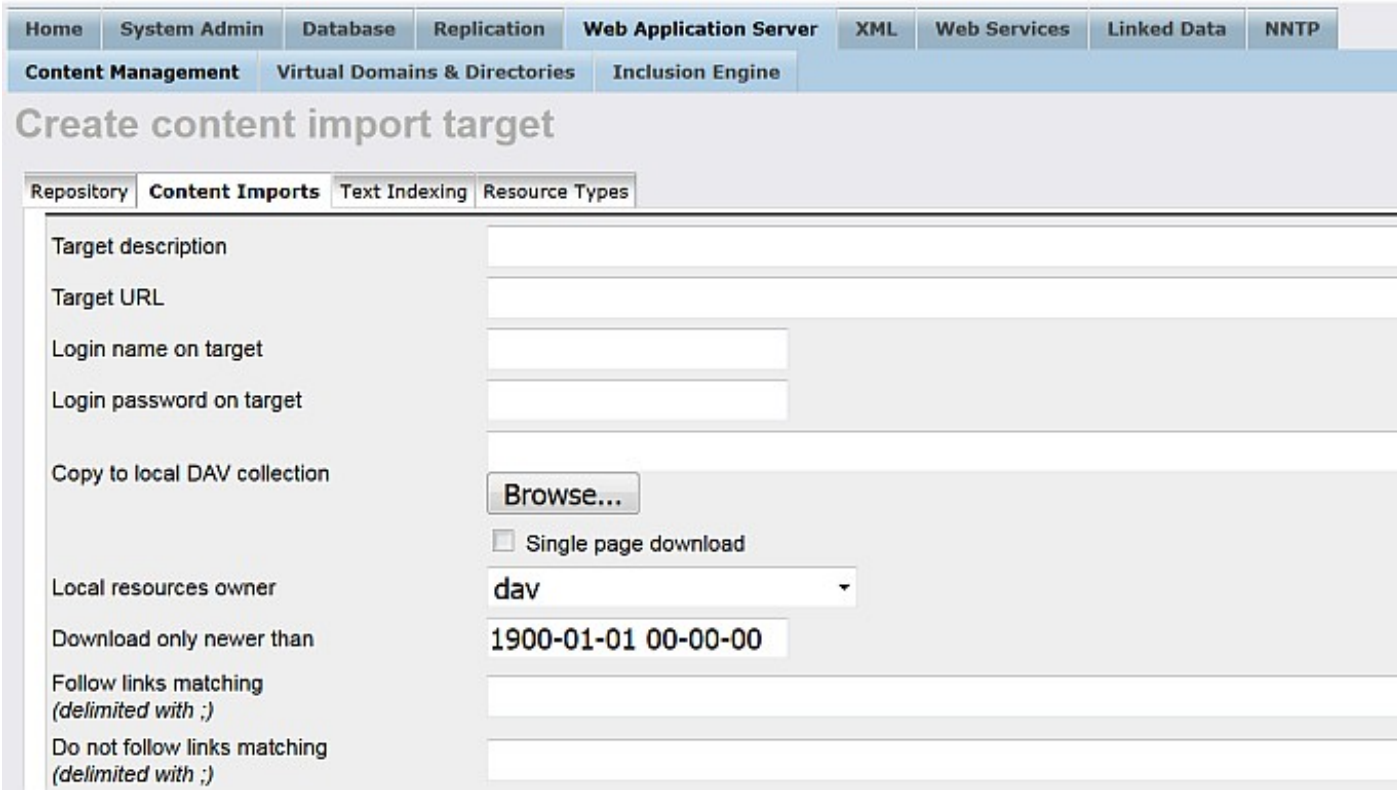
To populate the Virtuoso Quad Store, in this Guide we will use a XPAth expression for the URLs of the RDF resources references in a given ATOM feed. For ex. this one of the "National Bibliography" Store.

1. Go to http://cname/conductor
2. Enter dba credentials
3. Go to Web Application Server -> Content Management -> Content Imports:

Figure 6.48. Crawling ATOM feed

4. Click "New Target":

Figure 6.49. Crawling ATOM feed



5. In the presented form specify respectively:

   - "Target description": for ex. National Bibliography ;
   - "Target URL": for ex. http://data.libris.kb.se/nationalbibliography/feed/ ;

     Note: the entered URL will be the graph URI for storing the imported RDF data. You can also set it explicitly by entering another graph URI in the "If Graph IRI is unassigned use this Data Source URL:" shown as option in this form.
   - "Copy to local DAV collection": for ex.

     ```
     /DAV/temp/nbio/
     ```
   - "XPath expression for links extraction:":

     ```
     //entry/link/@href
     ```
   - "Update Interval (minutes)": for ex. 10 ;
   - "Run Sponger": hatch this check-box ;
   - "Accept RDF": hatch this check-box ;
   - Optionally you can hatch "Store metadata" ;
   - "RDF Cartridge": hatch this check-box and specify what cartridges will be used.

Figure 6.50. Crawling ATOM feed

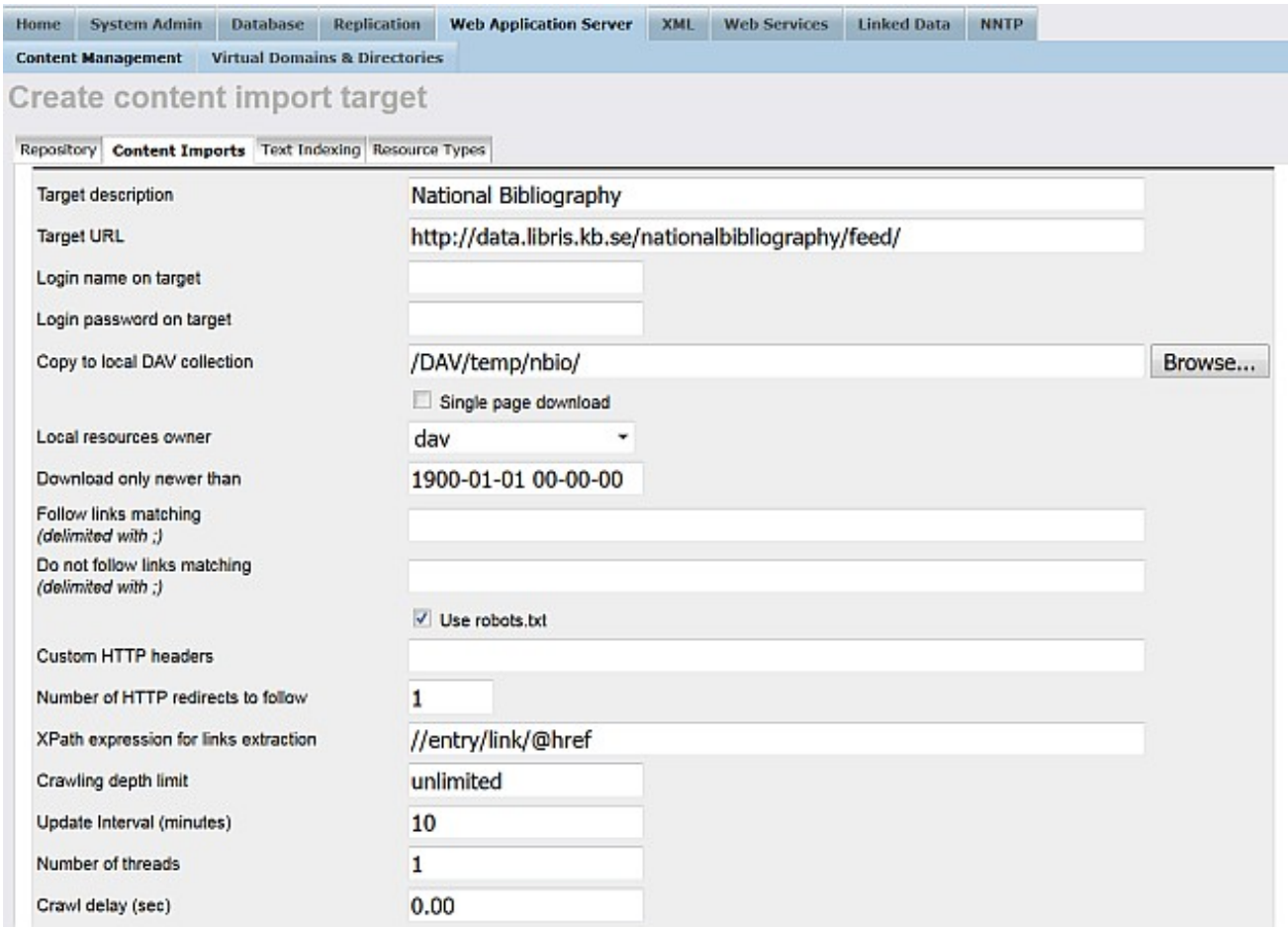
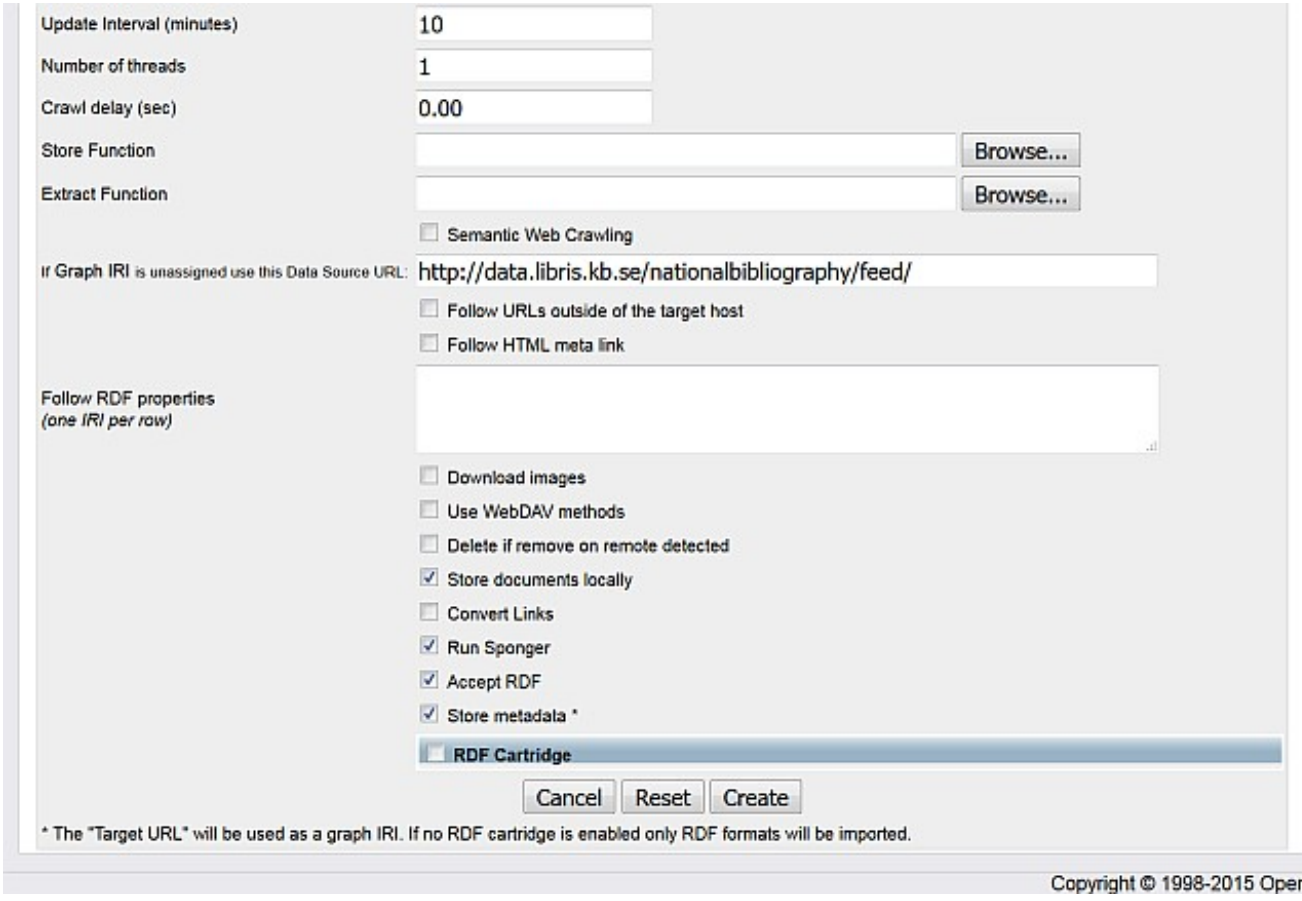
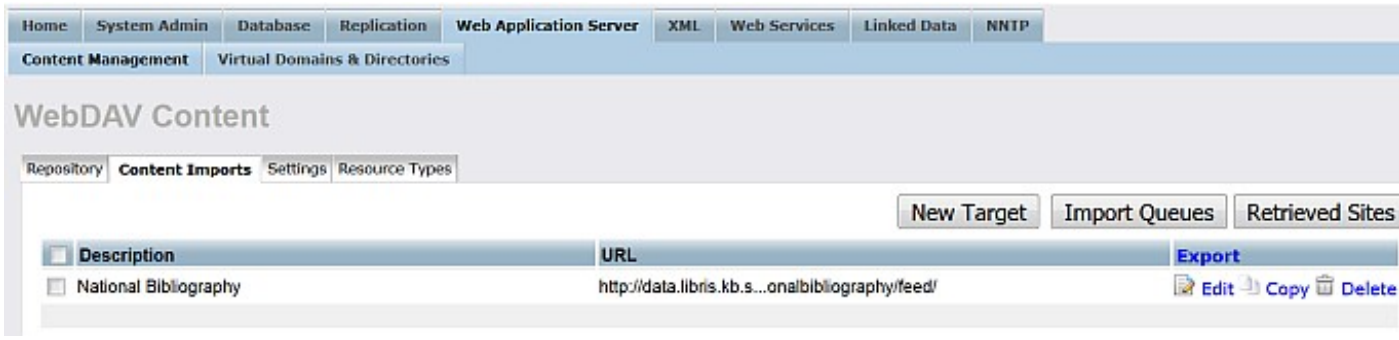


Figure 6.51. Crawling ATOM feed



6. Click "Create".
7. The new created target should be displayed in the list of available Targets:

Figure 6.52. Crawling ATOM feed

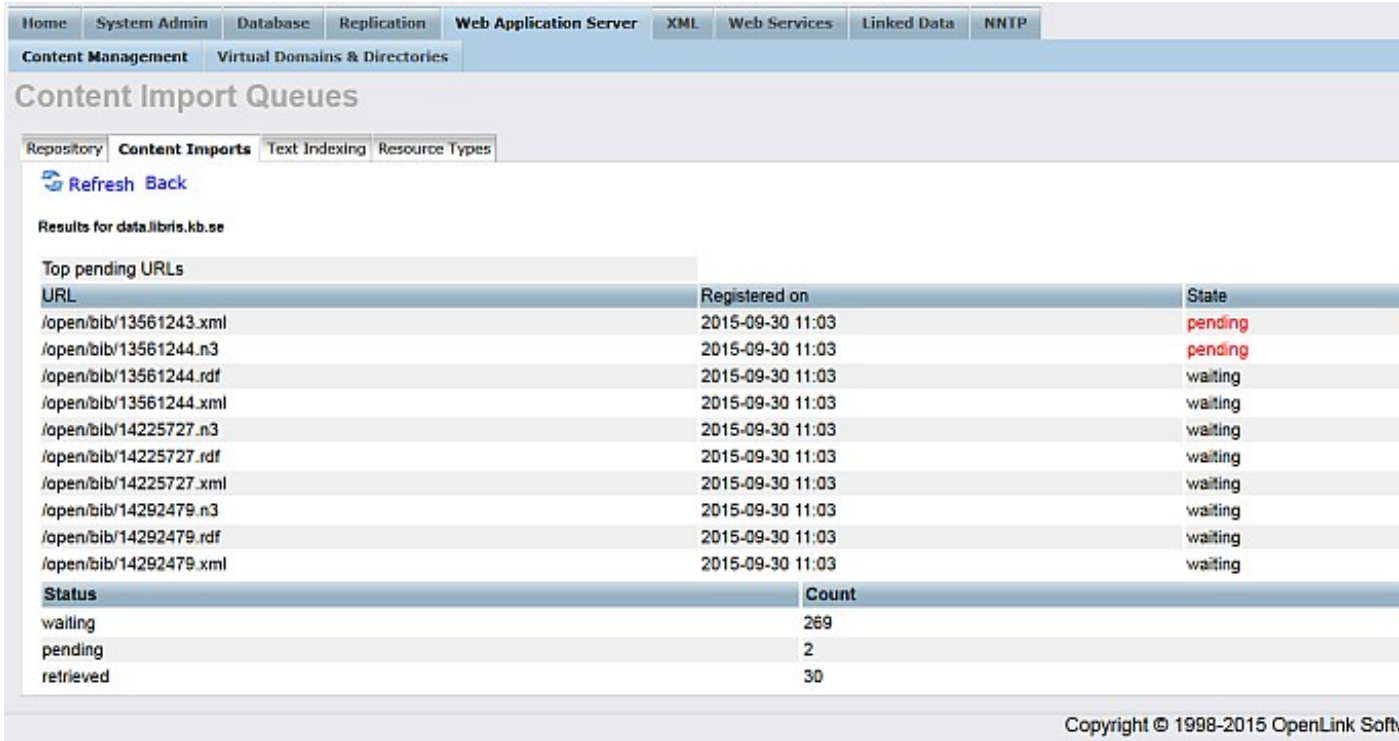


8. Click "Import Queues":

Figure 6.53. Crawling ATOM feed

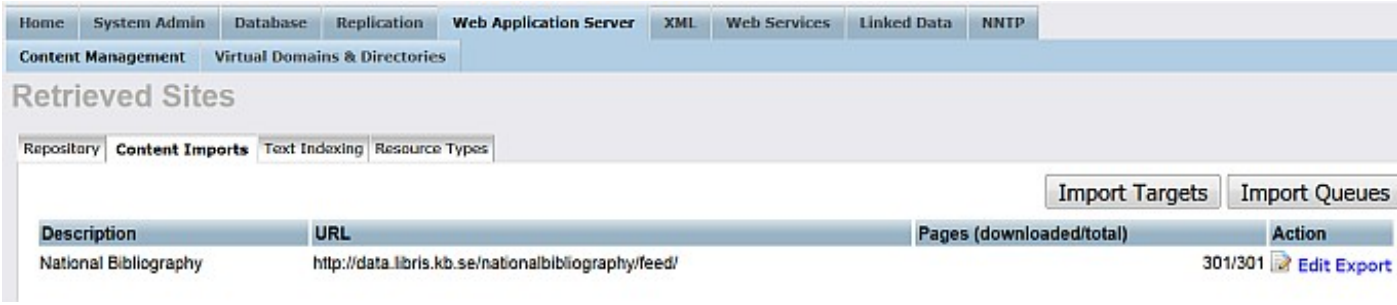
9. Click for "National Bibliography" target the "Run" link from the very-right "Action" column.
10. Should be presented list of Top pending URLs:

Figure 6.54. Crawling ATOM feed



11. Go to "Retrieved Sites" to check the total URLs that were processed:

Figure 6.55. Crawling ATOM feed



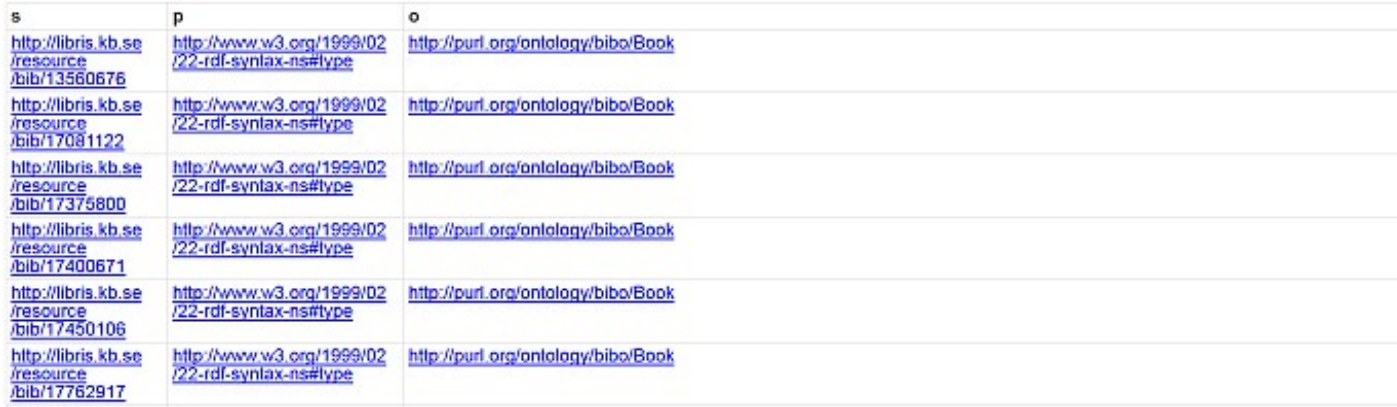
12. To view the imported RDF data, go to http://example.com/sparql and enter a simple query for ex.:

```
SELECT *
FROM <http://data.libris.kb.se/nationalbibliography/feed/>
WHERE
  {
    ?s ?p ?o
  }
```

Figure 6.56. Crawling ATOM feed

13. Click "Run Query".
14. The imported RDF data triples should be shown:

Figure 6.57. Crawling ATOM feed



Setting up a Content Crawler Job to Retrieve Content from SPARQL endpoint

The following step-by section walks you through the process of:

- Populating a Virtuoso Quad Store with data from a 3rd party SPARQL endpoint
- Generating RDF dumps that are accessible to basic HTTP or WebDAV user agents.

1. Sample SPARQL query producing a list SPARQL endpoints:

```
PREFIX rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:     <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:      <http://www.w3.org/2002/07/owl#>
PREFIX xsd:      <http://www.w3.org/2001/XMLSchema#>
PREFIX foaf:     <http://xmlns.com/foaf/0.1/>
PREFIX dcterms:  <http://purl.org/dc/terms/>
PREFIX scovo:    <http://purl.org/NET/scovo#>
PREFIX void:     <http://rdfs.org/ns/void#>
PREFIX akt:      <http://www.aktors.org/ontology/portal#>

SELECT DISTINCT ?endpoint
WHERE
  {
    ?ds a void:Dataset .
    ?ds void:sparqlEndpoint ?endpoint
  }
```

2. Here is a sample SPARQL protocol URL constructed from one of the sparql endpoints in the result from the query above:

```
http://void.rkbexplorer.com/sparql/?query=PREFIX+foaf%3A+%3Chttp%3A%2F%2Fxmlns.com%2Ffoaf%2F0.1%2F
```

3. Here is the cURL output showing a Virtuoso SPARQL URL that executes against a 3rd party SPARQL Endpoint URL:

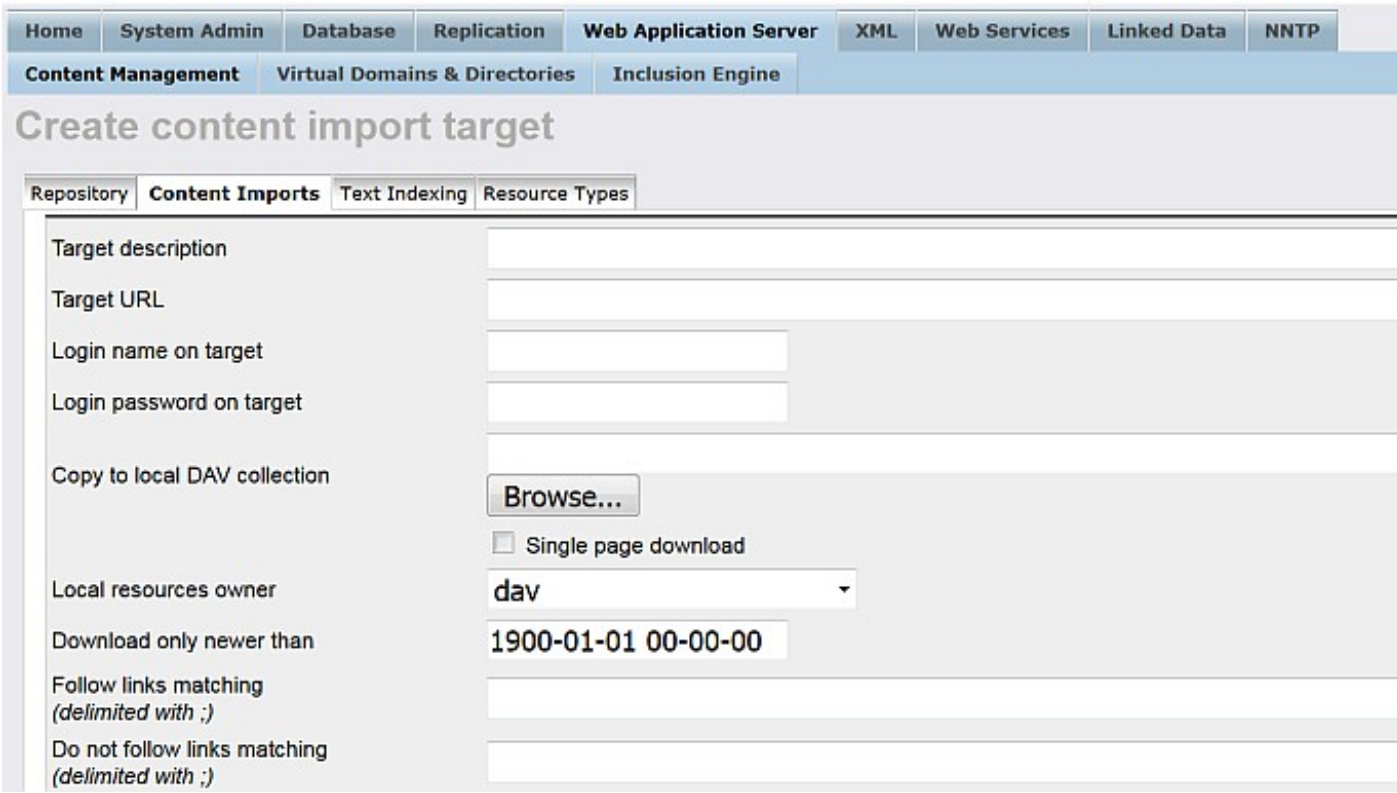
```
$ curl "http://void.rkbexplorer.com/sparql/?query=PREFIX+foaf%3A+%3Chttp%3A%2F%2Fxmlns.com%2Ffoaf%
%3A+++++%3Chttp%3A%2F%2Frdfs.org%2Fns%2Fvoid%23%3E++%0D%0ASELECT+distinct+%3Furl++WHERE+%7B+%3Fds+
omepage+%3Furl+%7D%0D%0A&format=sparql"
<?xml version="1.0" ?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head>
    <variable name="url"/>
  </head>
  <results ordered="false" distinct="true">
    <result>
      <binding name="url"><uri>http://kisti.rkbexplorer.com/</uri></binding>
    </result>
    <result>
      <binding name="url"><uri>http://epsrc.rkbexplorer.com/</uri></binding>
    </result>
    <result>
      <binding name="url"><uri>http://test2.rkbexplorer.com/</uri></binding>
    </result>
    <result>
      <binding name="url"><uri>http://test.rkbexplorer.com/</uri></binding>
    </result>
    ...
    ...
    ...
  </results>
</sparql>
```

4. Go to Conductor UI -- http://example.com/conductor and provide dba credentials;
5. Go to "Web Application Server"-> "Content Management" -> "Content Imports"

Figure 6.58. Crawling SPARQL Endpoints

6. Click "New Target":

Figure 6.59. Crawling SPARQL Endpoints



7. In the presented form enter for ex.:

   a. "Target description": voiD store
   b. "Target URL": the url from above i.e.:

      ```
      http://void.rkbexplorer.com/sparql/?query=PREFIX+foaf%3A+%3Chttp%3A%2F%2Fxmlns.com%2Ffoaf%2
      ```

   c. "Copy to local DAV collection":

      ```
      /DAV/void.rkbexplorer.com/content
      ```

   d. "Follow links matching (delimited with ;)":

      ```
      %
      ```

   e. Un-hatch "Use robots.txt" ;
   f. "XPath expression for links extraction":

      ```
      //binding[@name="url"]/uri/text()
      ```

   g. Hatch "Semantic Web Crawling";
   h. "If Graph IRI is unassigned use this Data Source URL:": enter for ex:

      ```
      http://void.collection
      ```

i. Hatch "Follow URLs outside of the target host";
j. Hatch "Run "Sponger" and "Accept RDF"

Figure 6.60. Crawling SPARQL Endpoints

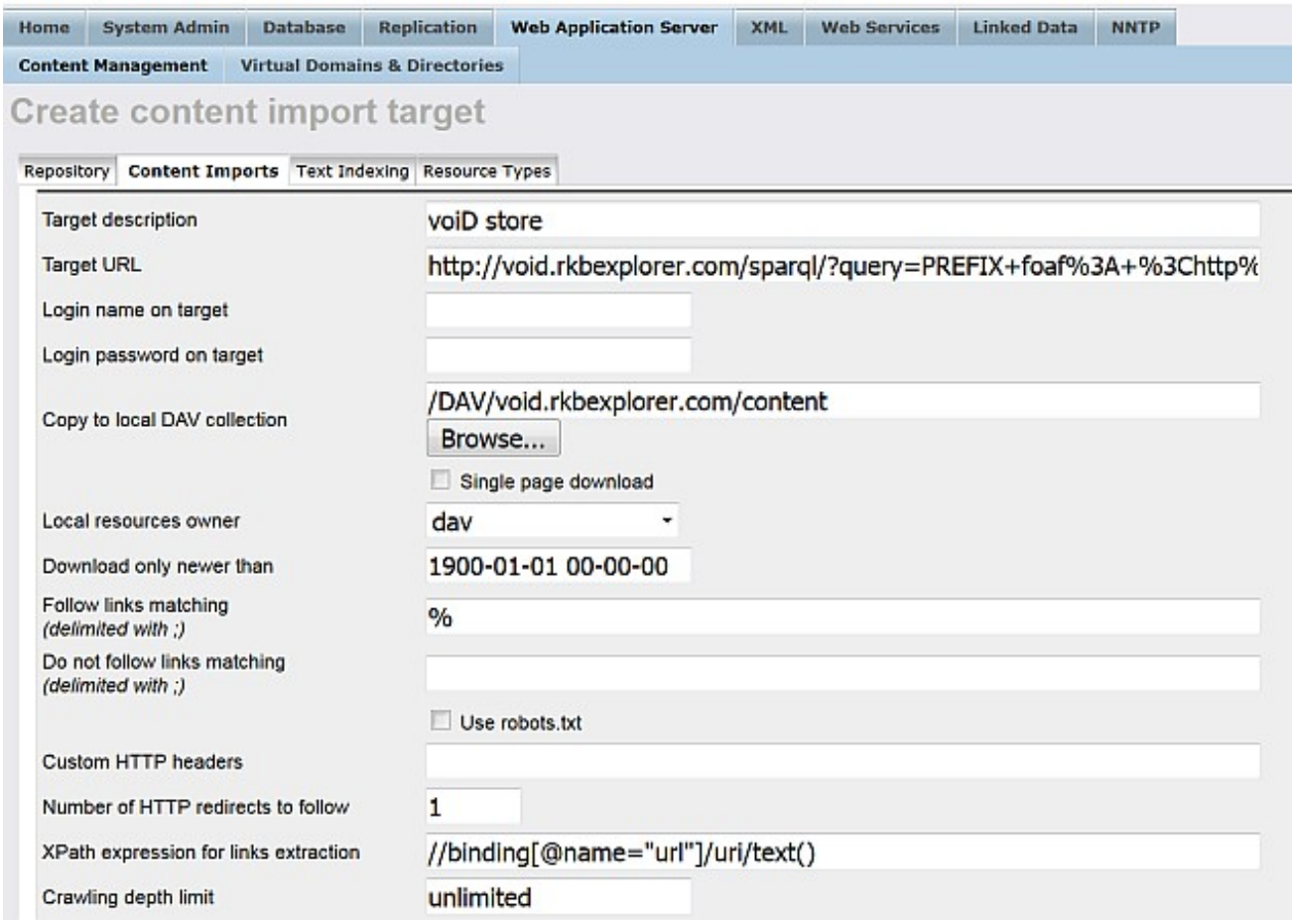
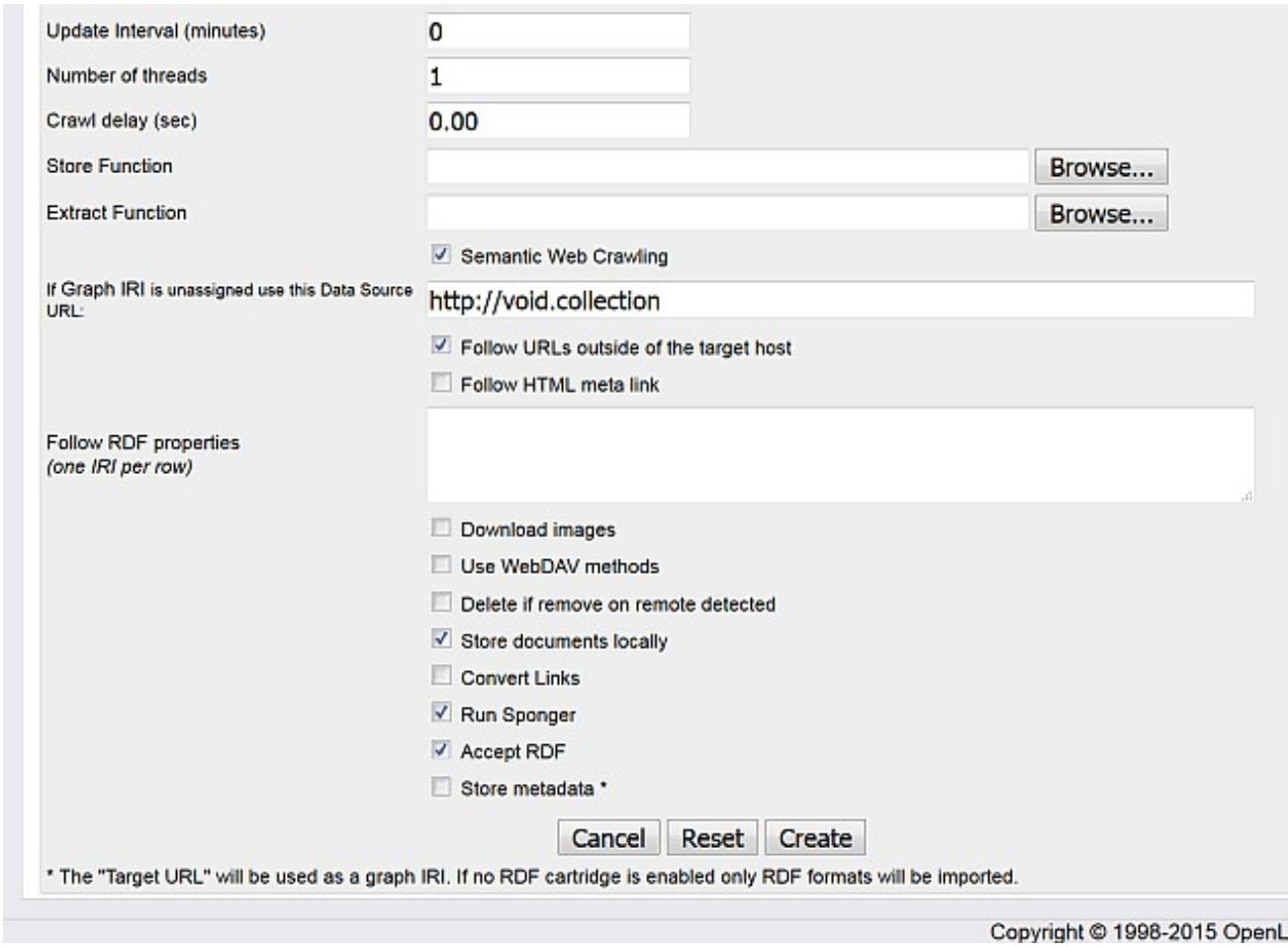
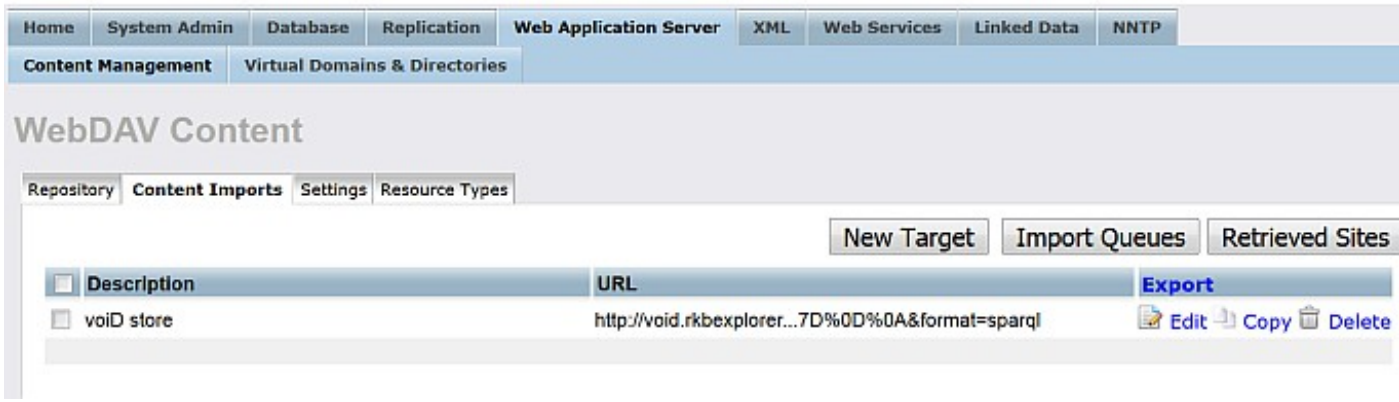


Figure 6.61. Crawling SPARQL Endpoints



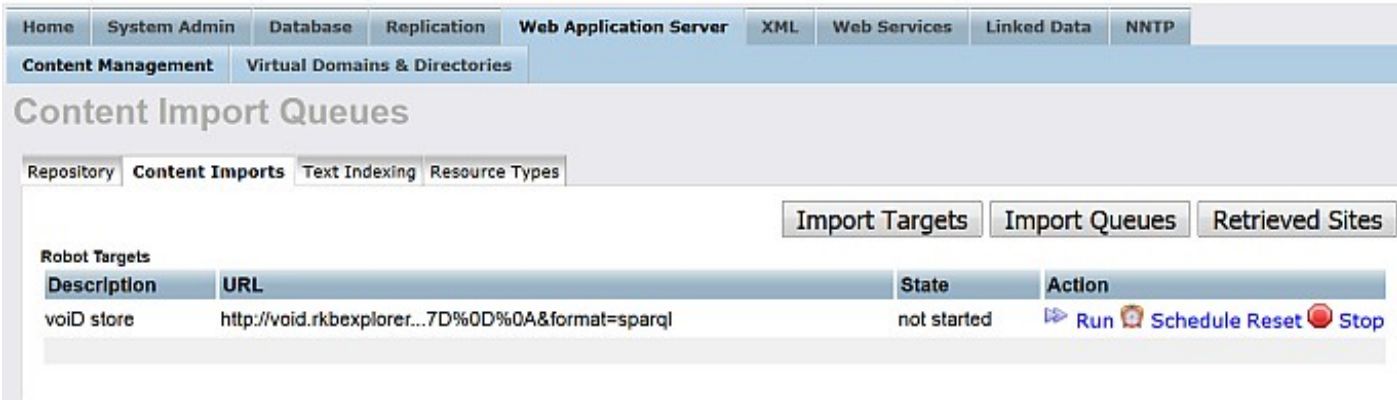
8. Click "Create";
9. The target should be created and presented in the list of available targets:

Figure 6.62. Crawling SPARQL Endpoints



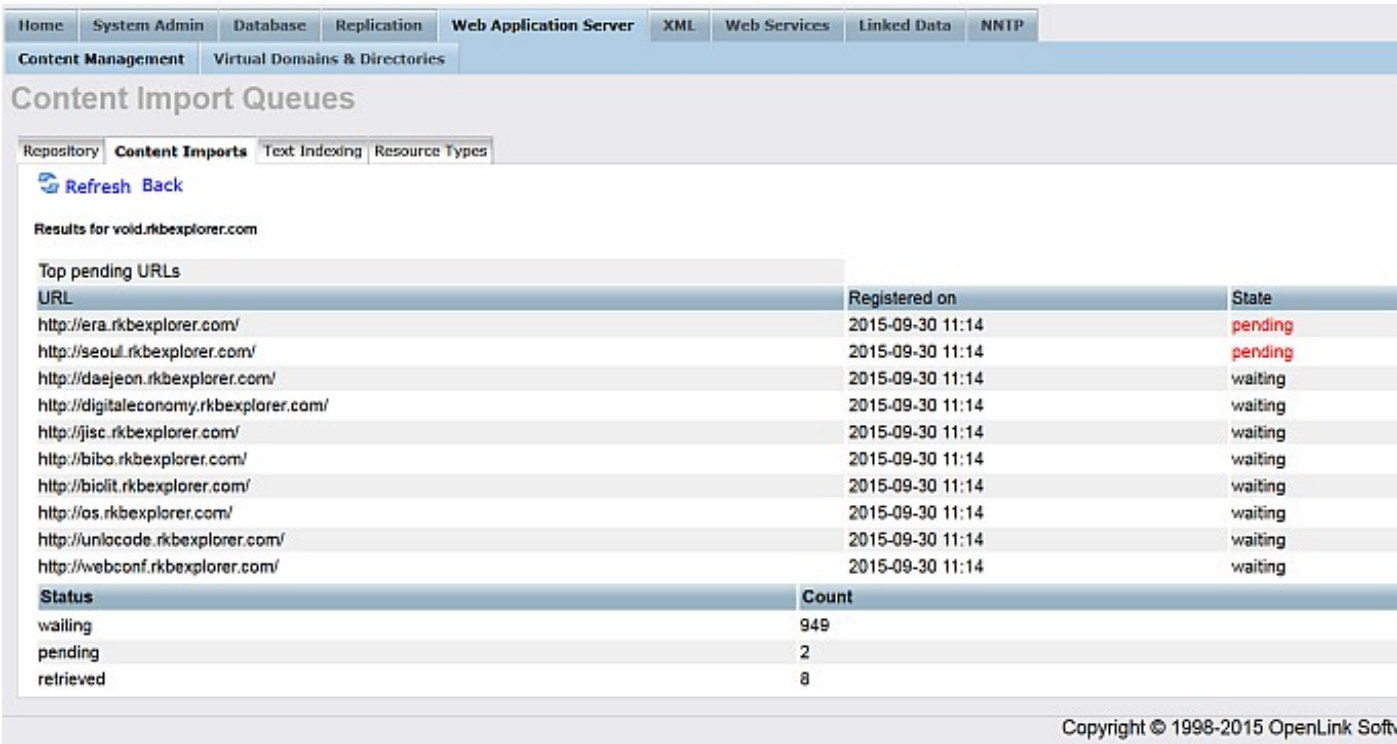
10. Click "Import Queues":

Figure 6.63. Crawling SPARQL Endpoints



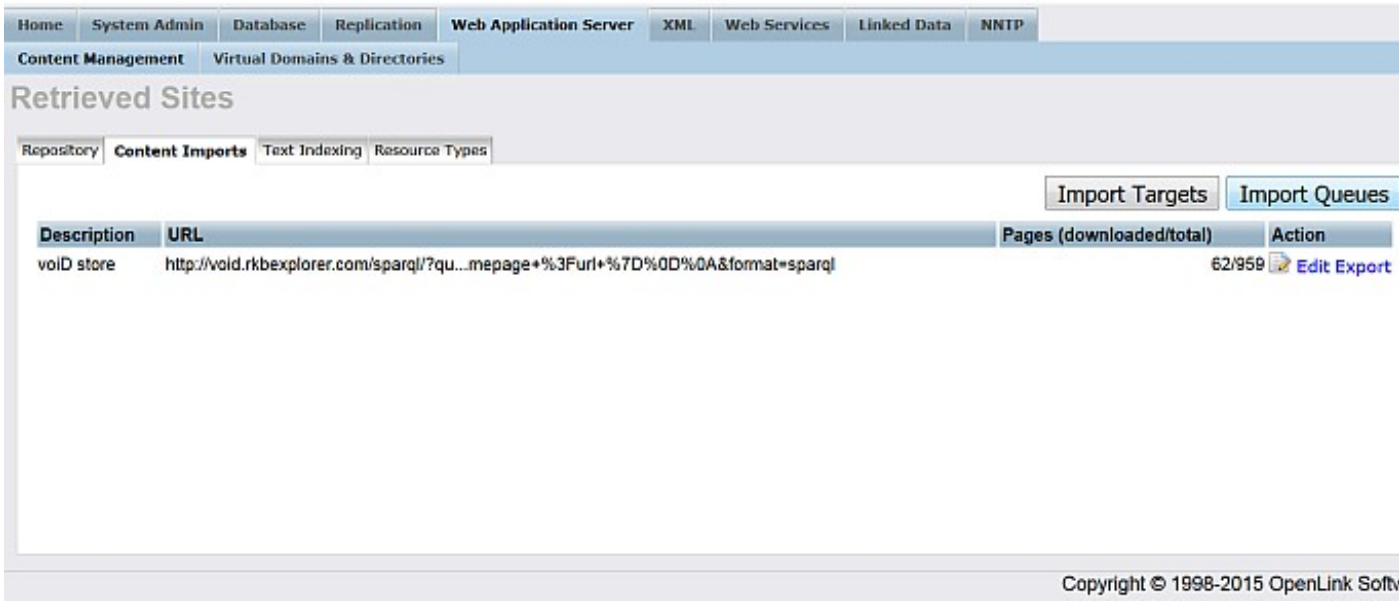
11. Click "Run" for the imported target:

Figure 6.64. Crawling SPARQL Endpoints



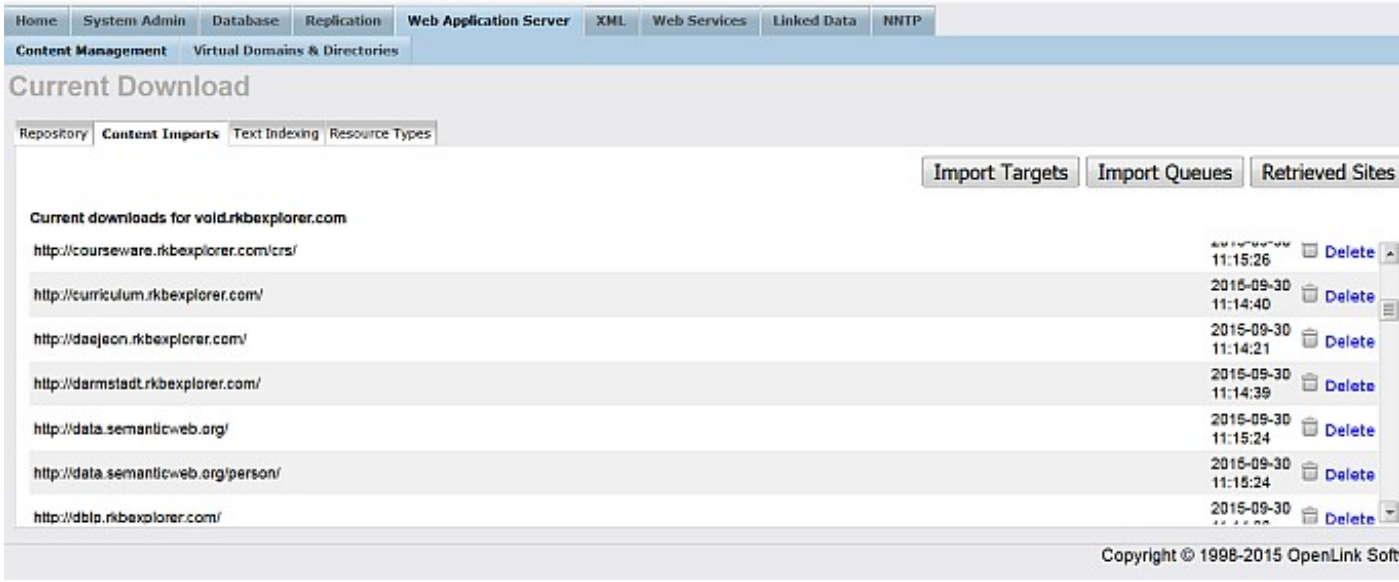
12. To check the retrieved content go to "Web Application Server"-> "Content Management" -> "Content Imports" -> "Retrieved Sites":

Figure 6.65. Crawling SPARQL Endpoints



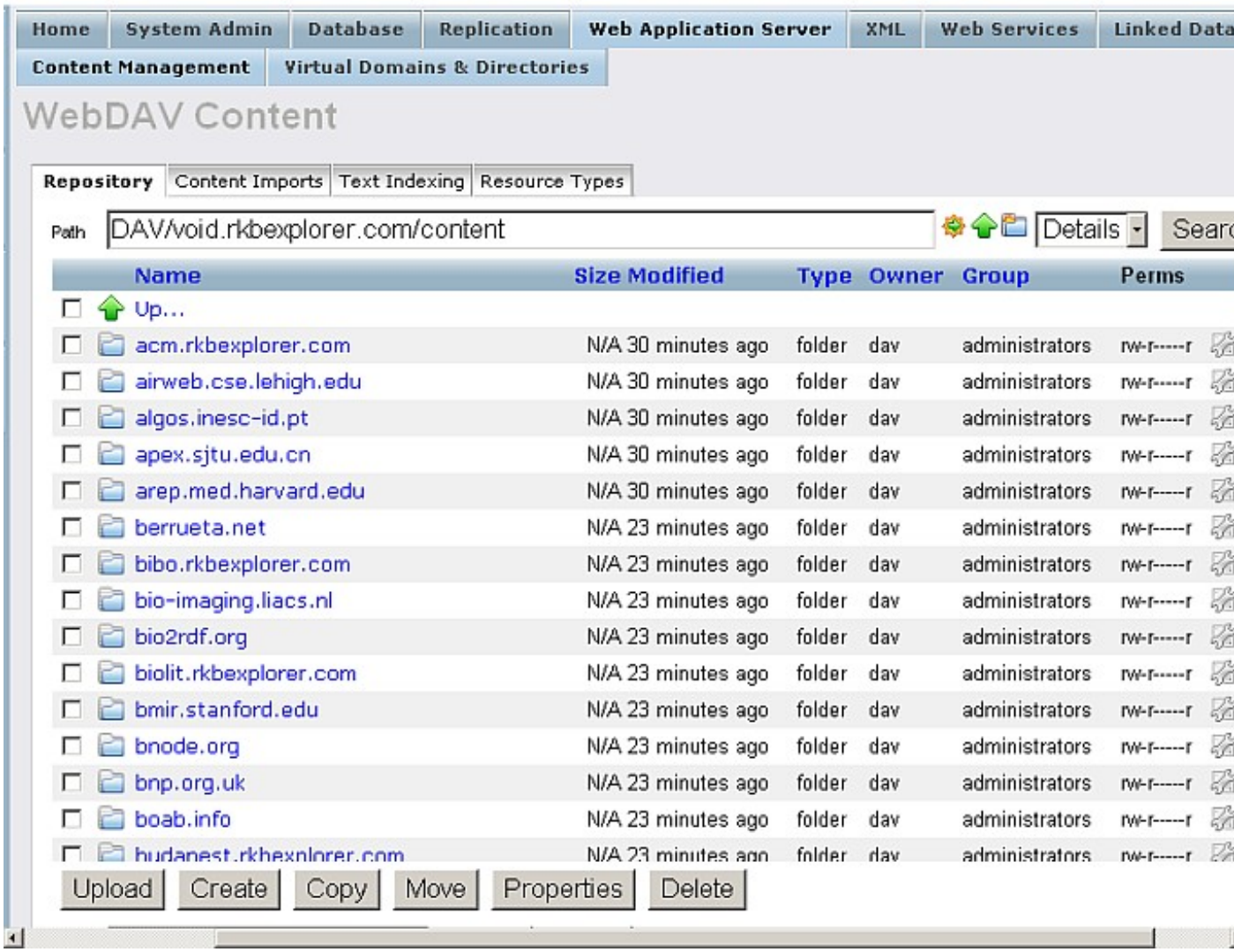
13. Click "voiD store" -> "Edit":

Figure 6.66. Crawling SPARQL Endpoints



14. To check the imported URLs go to "Web Application Server"-> "Content Management" -> "Repository" path DAV/void.rkbexplorer.com/content/:

Figure 6.67. Crawling SPARQL Endpoints



15. To check the inserted into the RDF QUAD data go to http://cname/sparql and execute the following query:

```
SELECT *
FROM <http://void.collection>
WHERE
  {
    ?s ?p ?o
  }
```

Figure 6.68. Crawling SPARQL Endpoints

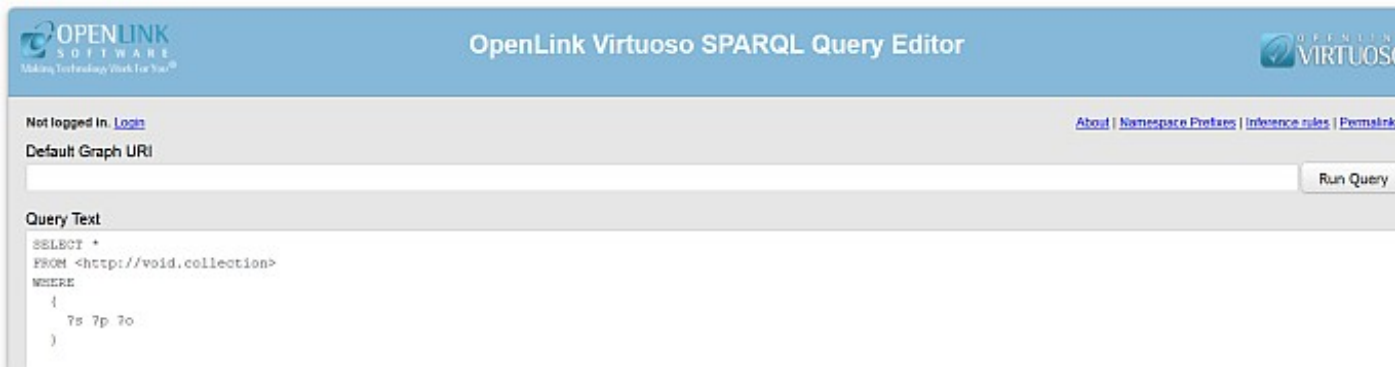


Figure 6.69. Crawling SPARQL Endpoints

s	p	o
nodeID://b1776905	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://rdfs.org/sioc/ns#UserAccount
http://wiki.dbpedia.org/sites/default/files/styles/flexslider/public/okfest-logo.png?itok=T-o7tzAw	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://xmlns.com/foaf/0.1/Image
http://wiki.dbpedia.org/misc/feed.png	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://xmlns.com/foaf/0.1/Image
http://revyu.com/tags/-	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.holygoat.co.uk/owl/redwood/0.1/tags/Tag
http://revyu.com/tags/0	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.holygoat.co.uk/owl/redwood/0.1/tags/Tag
http://revyu.com/tags/007	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.holygoat.co.uk/owl/redwood/0.1/tags/Tag
http://revyu.com/tags/11	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.holygoat.co.uk/owl/redwood/0.1/tags/Tag
http://revyu.com/tags/11e-arrondissement	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.holygoat.co.uk/owl/redwood/0.1/tags/Tag
http://revyu.com/tags/11th	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.holygoat.co.uk/owl/redwood/0.1/tags/Tag
http://revyu.com/tags/12	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.holygoat.co.uk/owl/redwood/0.1/tags/Tag
http://revyu.com/tags/12-arrondissement	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.holygoat.co.uk/owl/redwood/0.1/tags/Tag

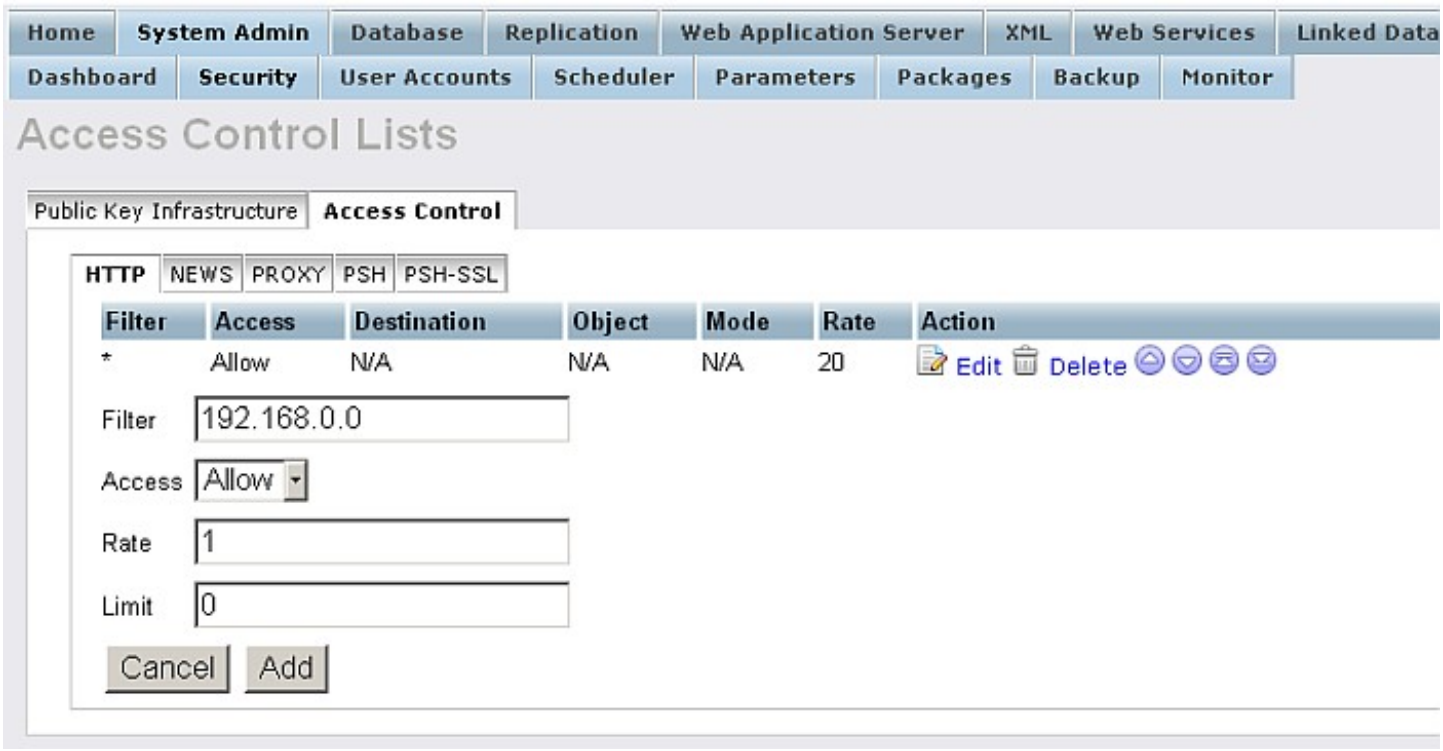
Access Control

From "System Admin" -> Security -> "Access Controls" you can manage Rules and ACL respectively for HTTP, News, Proxy. The tabs PSH and PSH-SSL are available only when the pubsubhub_dav.vad is installed.

Figure 6.70. Access Control Lists

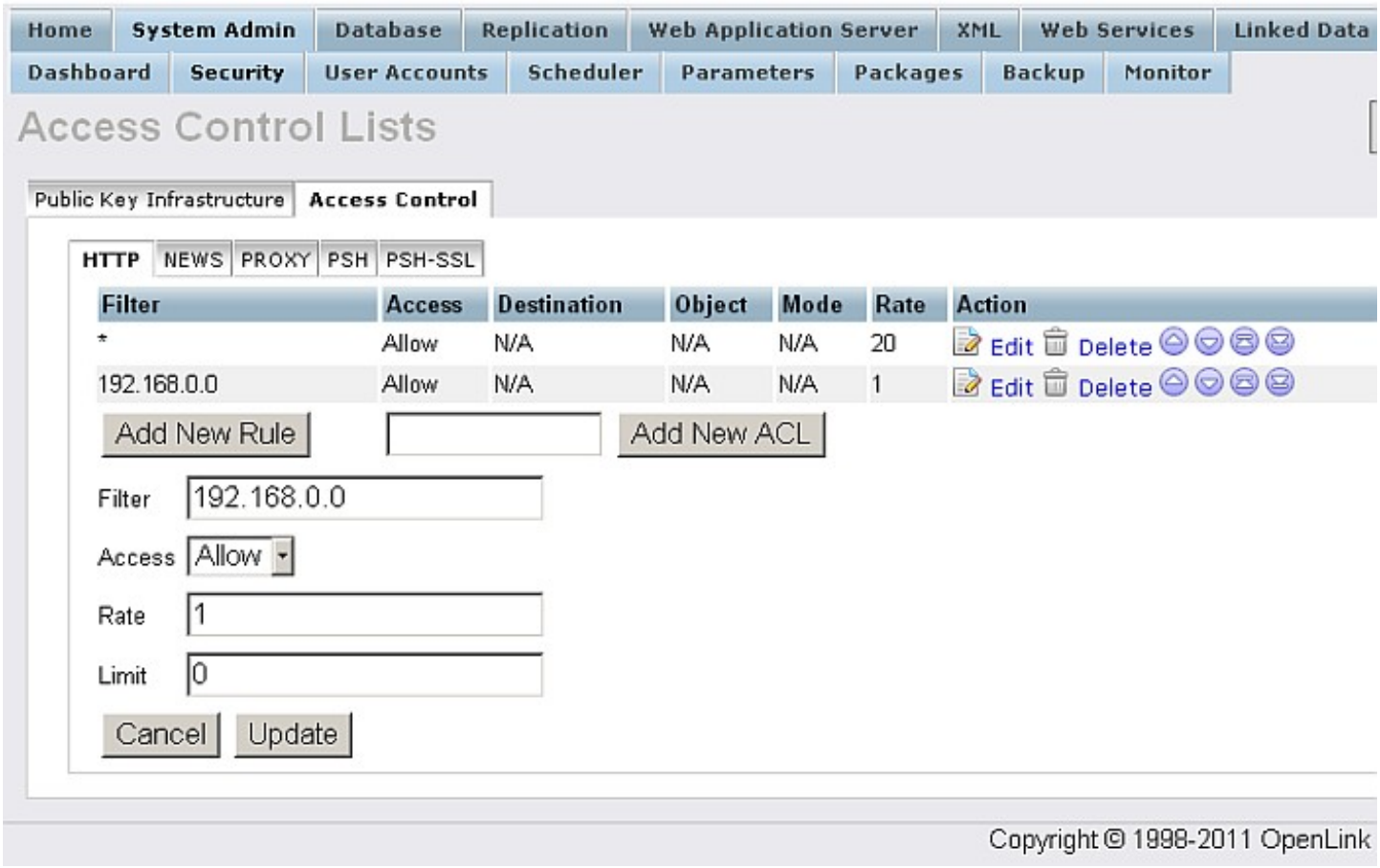
For each of the tabs "HTTP", "NEWS", "PROXY" the created rules will be shown in a list with Filter, Access, Destination, Object, Mode, Rate values. You can also add/delete rules, re-arrange rules order.

Figure 6.71. Access Control List for HTTP



Click the link "Edit" for a rule. Then specify the filter and access values.

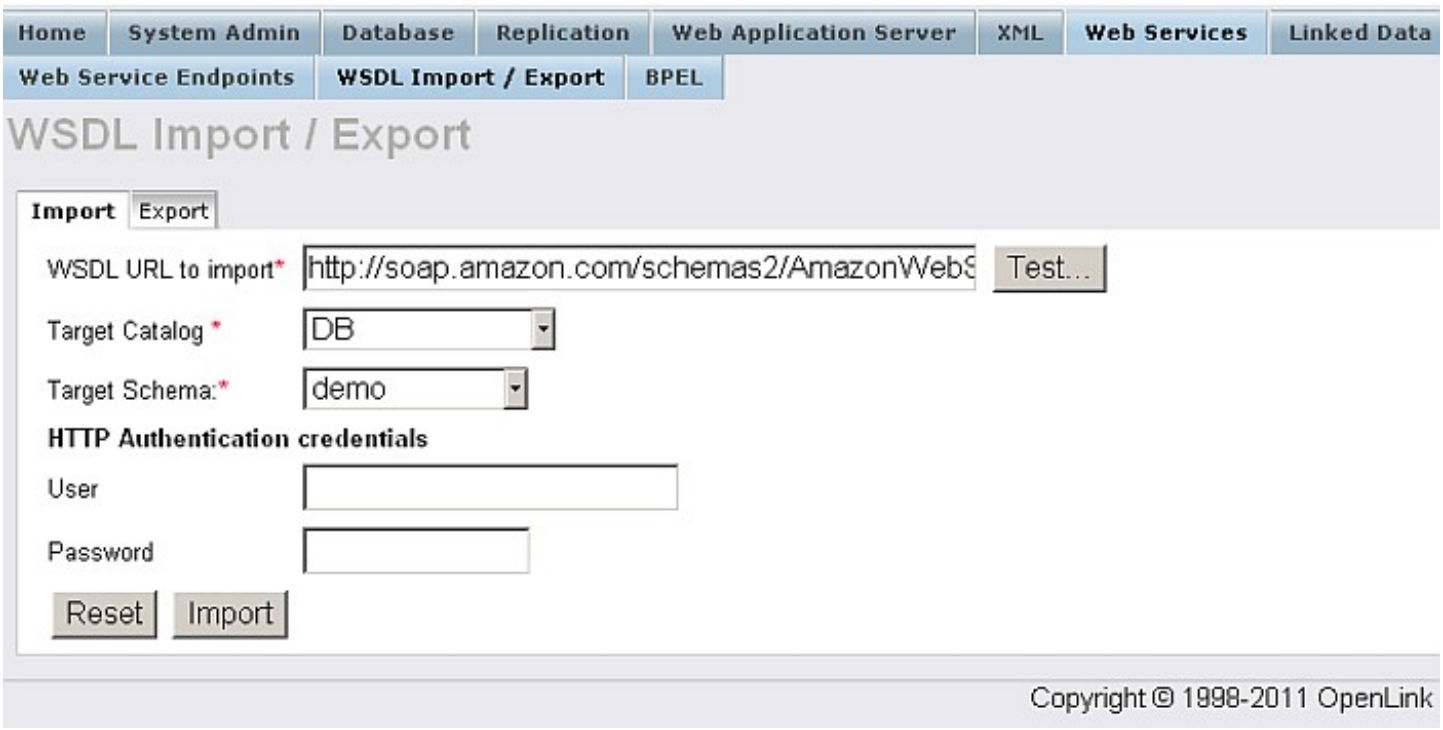
Figure 6.72. Access Control Lists



Import WSDL

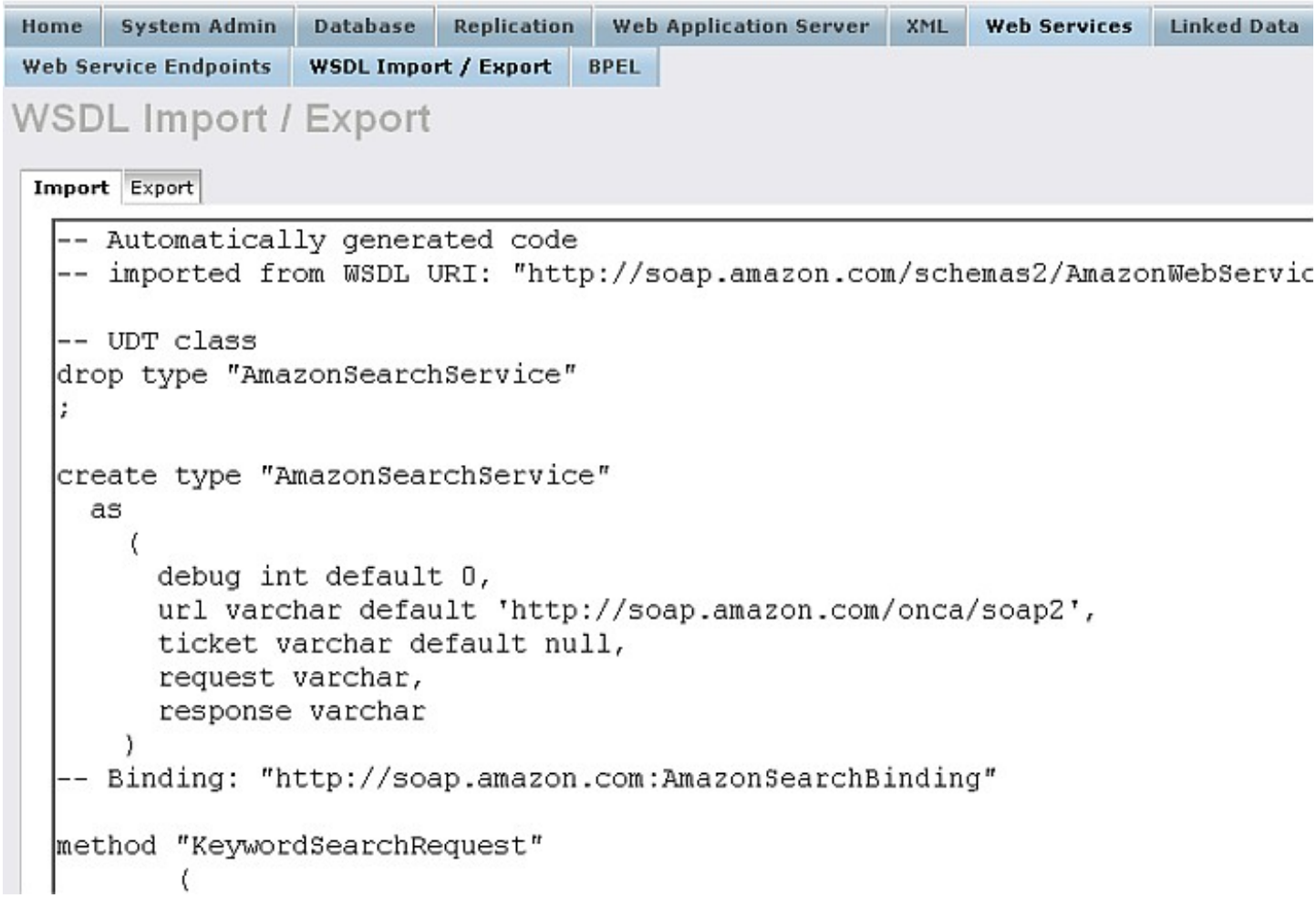
From "Web Services" / "WSDL Import/Export" you can provide a URL to a WSDL description. In return Virtuoso will automatically provide a wrapper for the services available, hence stored procedures and user-defined types that are callable within Virtuoso while the processing and mechanics of the services are actually handled at the source.

Figure 6.73. WSDL Import



After Virtuoso examines the supplied URL to a WSDL you are presented with the source code for the PL wrappers and Virtuoso user-defined types to be created. You have the chance to edit the code for more specific needs and then you can either save this to a file for later work, or execute it in Virtuoso to create the procedures and types.

Figure 6.74. WSDL Import



Any errors in the code will be highlighted if you try and execute it.

If you wish to save the file the appropriate file system ACLs must be in place for the destination.

6.2.4. WebDAV Administration

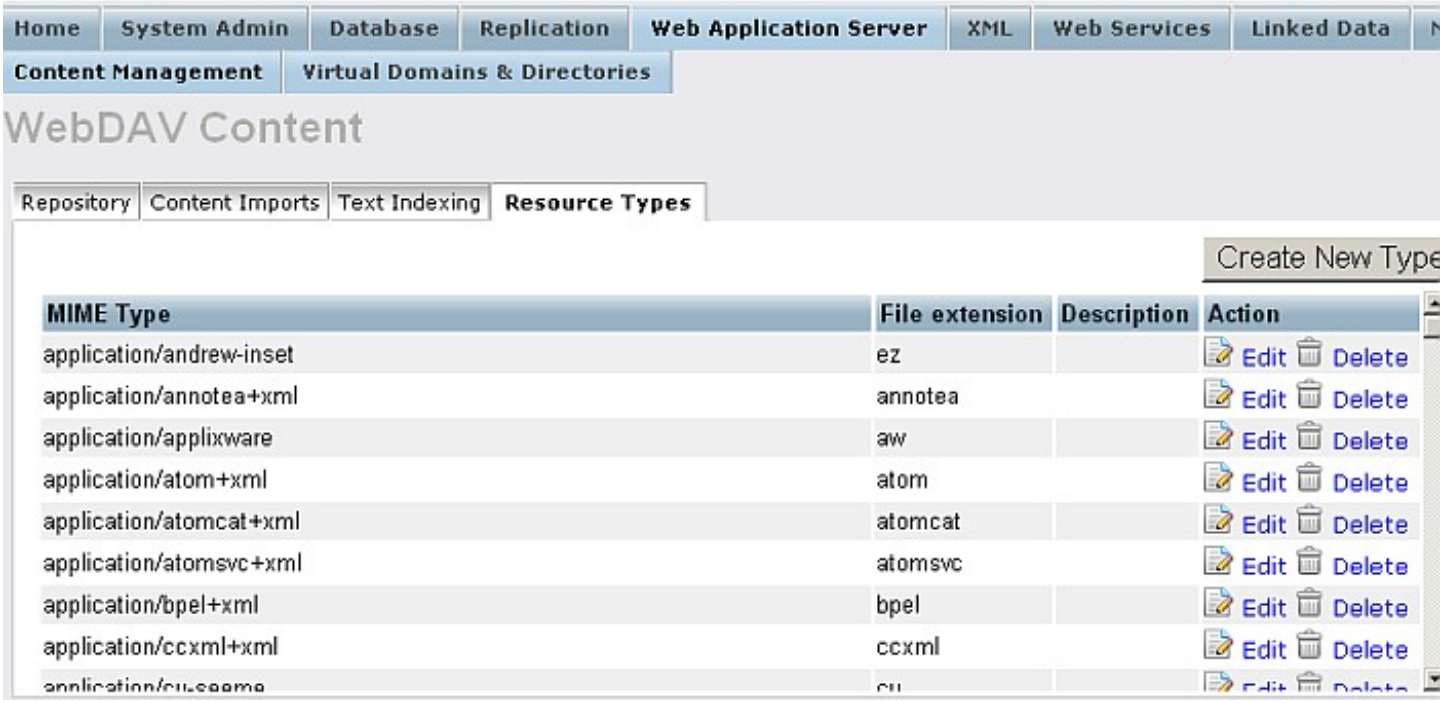
DAV, or WebDAV, is a protocol for Web-based Distributed Authoring and Versioning. The location where content items are placed is called the *repository* . Content elements are called *documents* , corresponding to files, and *folders/collections* , corresponding to directories. Collectively these documents and folders (collections) are known as *resources* .

Virtuoso implements the DAV protocol, allowing you to create and manage resources either directly through repository manipulations or indirectly, through a variety of WebDAV services.

DAV Resource Types

To make sure that when Virtuoso serves files to client user agents the type of file is conveyed properly so that the right application can be used with that file a list or file types are maintained in the server. This list is used when sending any content via the HTTP server which include content in DAV and under VSP.

Figure 6.75. DAV Resource Types



The Web Application Server/Content Management/Resource Types page shows the list of currently defined resource types in the Virtuoso server. You can edit or remove these types by using the action links on the right most column of the list next to the type applicable.

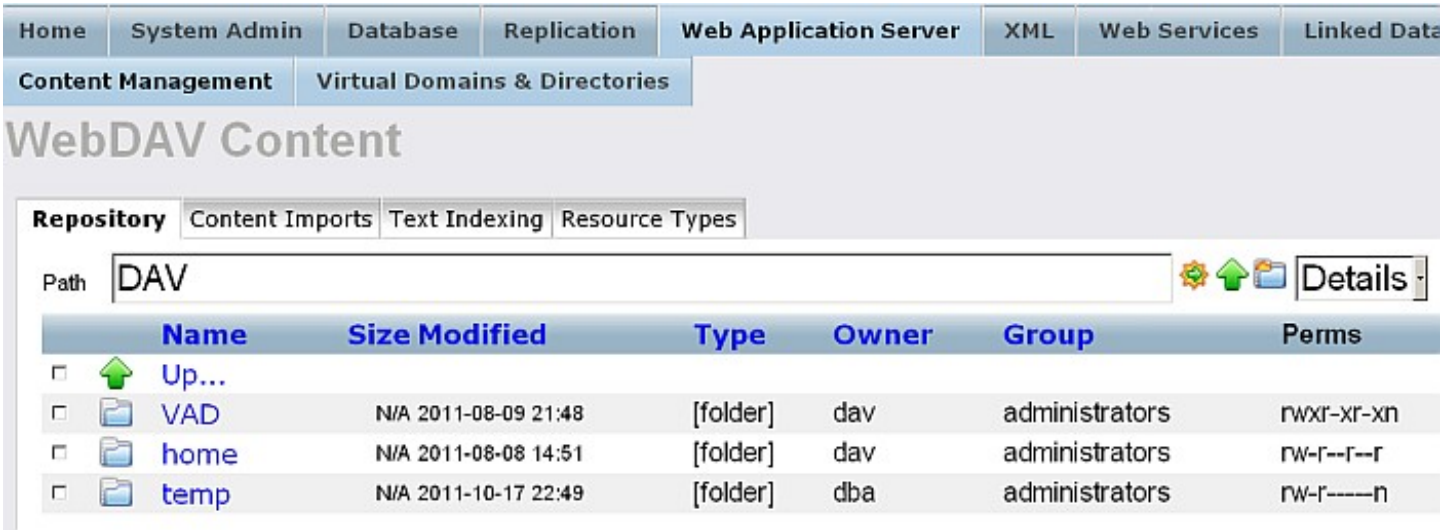
Add new types by typing the details into the fields provided and pressing the *Save* button.

Content Management

The content management page gives you an interface to the WebDAV repository resources. From here you can navigate or create directories, commonly referred to as collections or folders in DAV, alter properties, upload or remove files, or edit documents.

The Repository tab lists the current location within the tree and the current login name. The root of the repository is usually `/DAV/` .

Figure 6.76. Content Management



Resource Names

Resource names are given with collection (folder) names listed first, then individual documents. Permissions on resources are presented in a style similar to Unix, with (r)ead, (w)rite, and e(x)ecute permissions listed for the resource owner (the user), the resource's group, and for the general public. If a permission is present, the letter is shown; if not, it is replaced with a dash.

Resource Permissions

Unlike Unix, the Repository does not use the designation 'd' for directories, which in DAV are more commonly referred to as collections. Collections are distinguished by a different icon - a folder-like icon - and by having the type named "collection".

In addition, the permissions string has a trailing letter designating the indexing status of the resource. This letter is 'n' to designate that indexing is off, 'r' for recursive indexing, and 't' for direct indexing.

Resource User and Group

By default, the user and group of a DAV resource are those set by the service that created the resource, or they are the ID and primary group of the user who was logged in when the resource was created.

Resource Size

The size of a document resource is its size in disk bytes. Note that this does not necessarily correspond to characters displayed on the screen because of encodings. For example, the HTML token &-a-m-p-semicolon is five bytes on disk but displays as a single screen character.

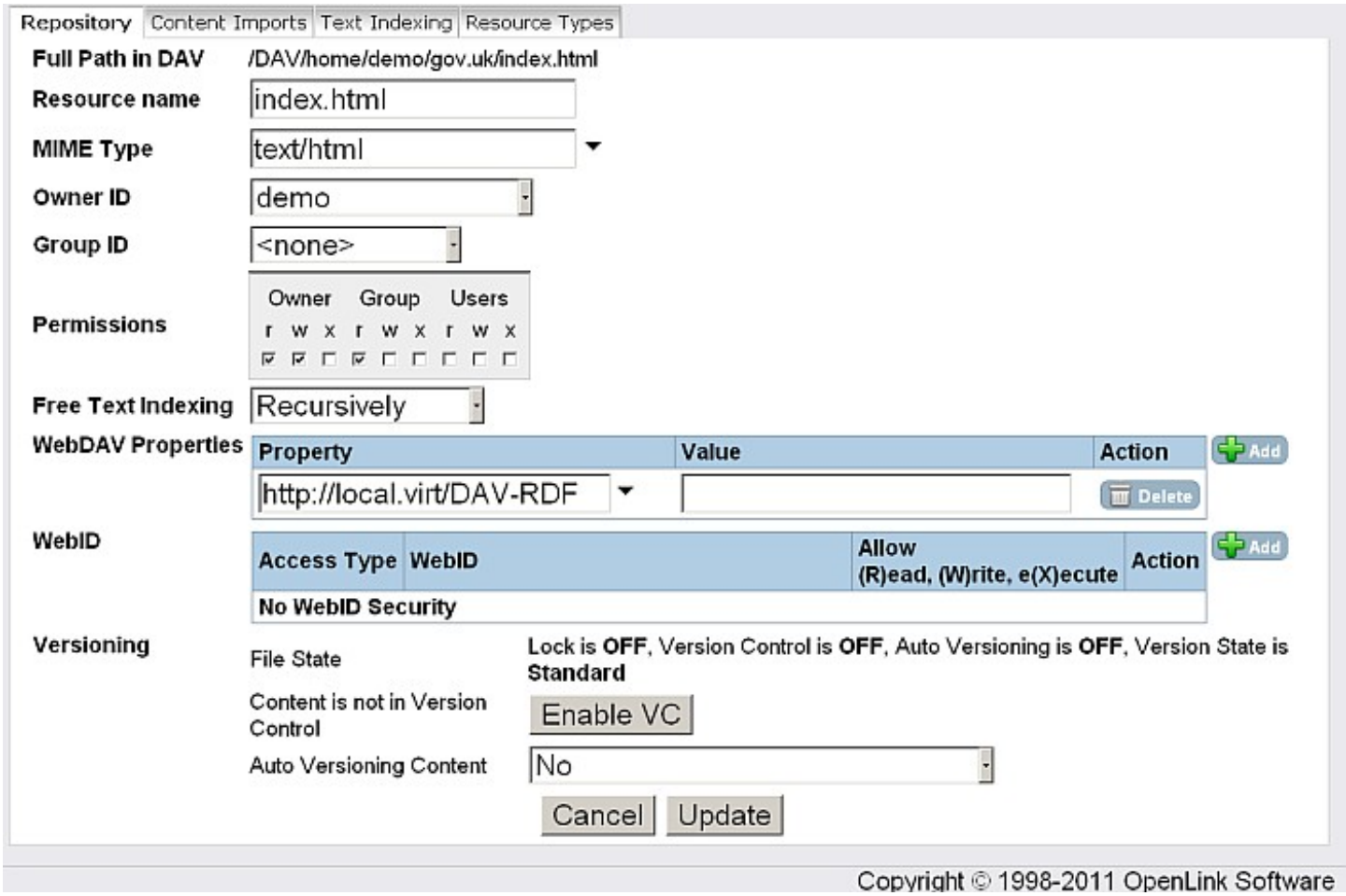
Resource Type

The type of a resource is always "collection" for collections (folder). For documents it can be any of the known resource types (see Resource Types below). The resource type is usually based on the resource's extension; for example, .xml files are usually assumed to be `text/xml` . If Virtuoso does not recognize the extension of a resource, it assigns the default type of `application/octet-stream` .

Editing Properties

Within the Content Management screen, you can change any of the properties of a resource, other than its name, by selecting the resource via the checkbox to the left of the icon, and pressing the "Properties" button.

Figure 6.77. Edit Resource Properties



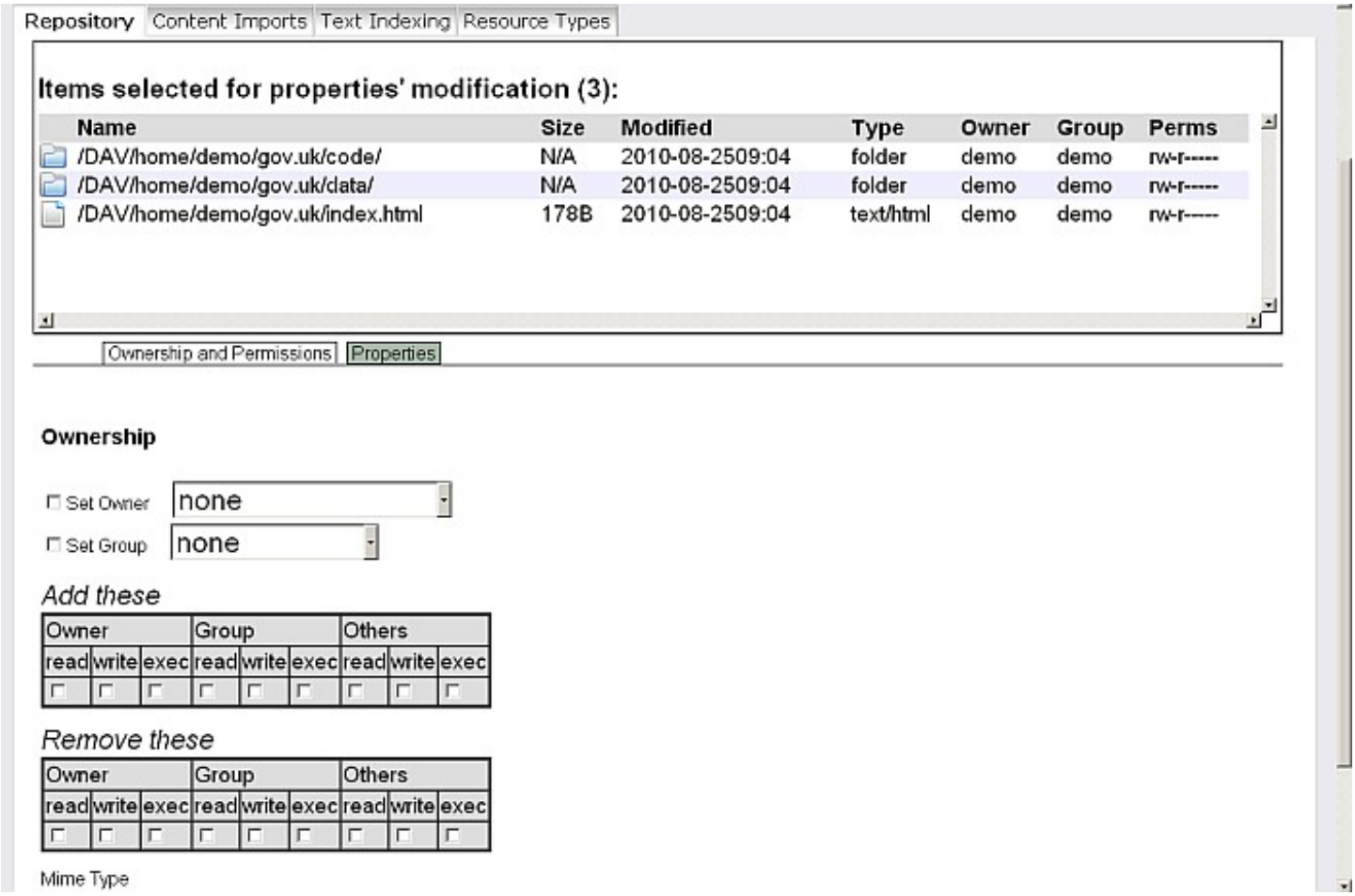
XML documents also permit you to edit their XML properties specifically. This can be done on any document of type text/xml by clicking on the icon for the document.

Properties

When clicked this button from the bottom of the page, you can edit the properties for one or a group of resources which should be selected. The name of a resource cannot be changed.

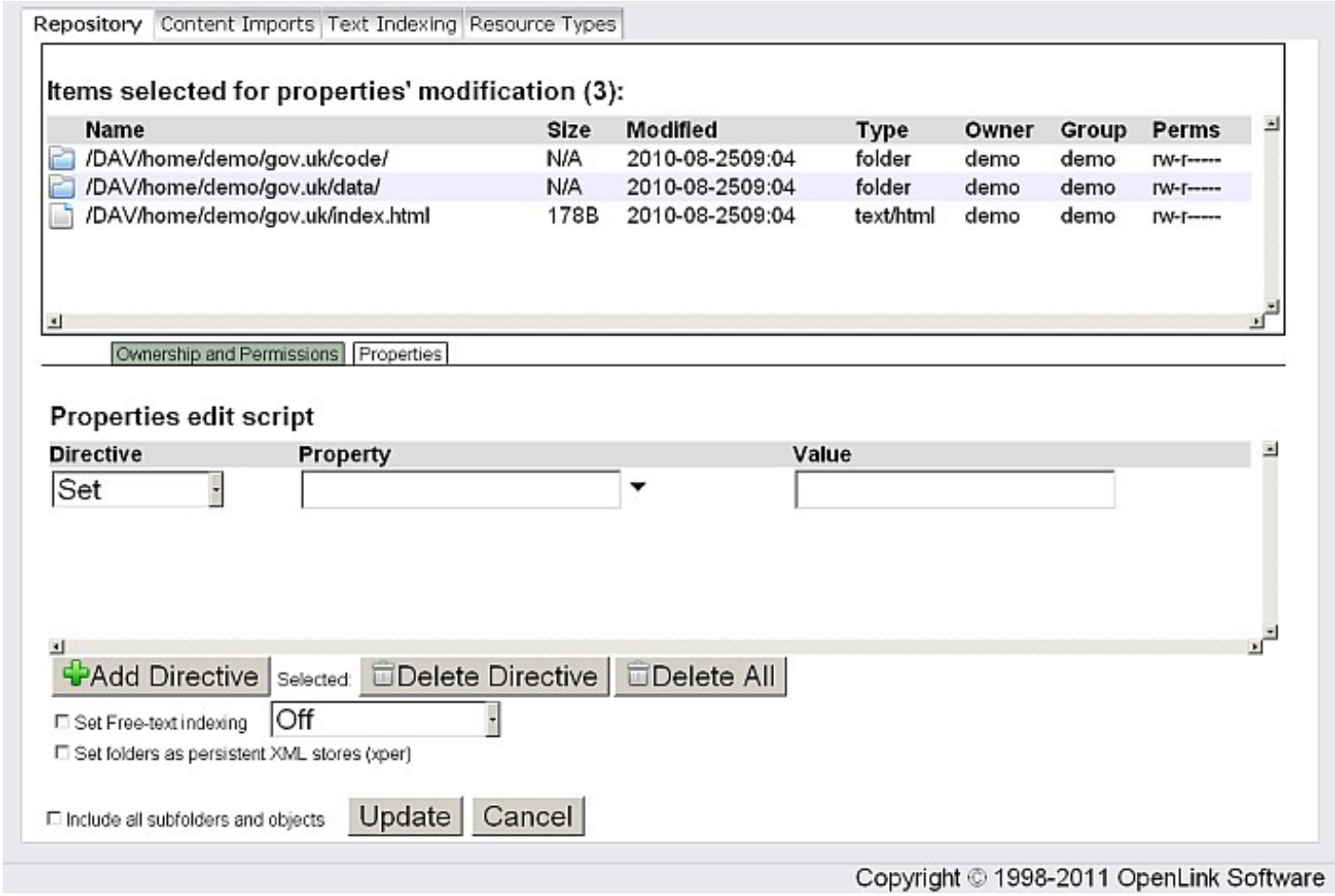
The owner, group, permissions, and indexing controls on this form apply to the appropriate properties of resources, as documented above. Changes to the type of a collection (folder) are ignored.

Figure 6.78. Multiple Resources Edit Properties



The "Property" control permits you to change or add additional properties. There is a pulldown of predefined XML-related properties, or you may create your own property.

Figure 6.79. Multiple Resources Edit Properties



WebDAV Properties

This control permits you to manipulate the specific properties of XML documents. Existing properties are shown with their values, and can be removed. New properties can be added.

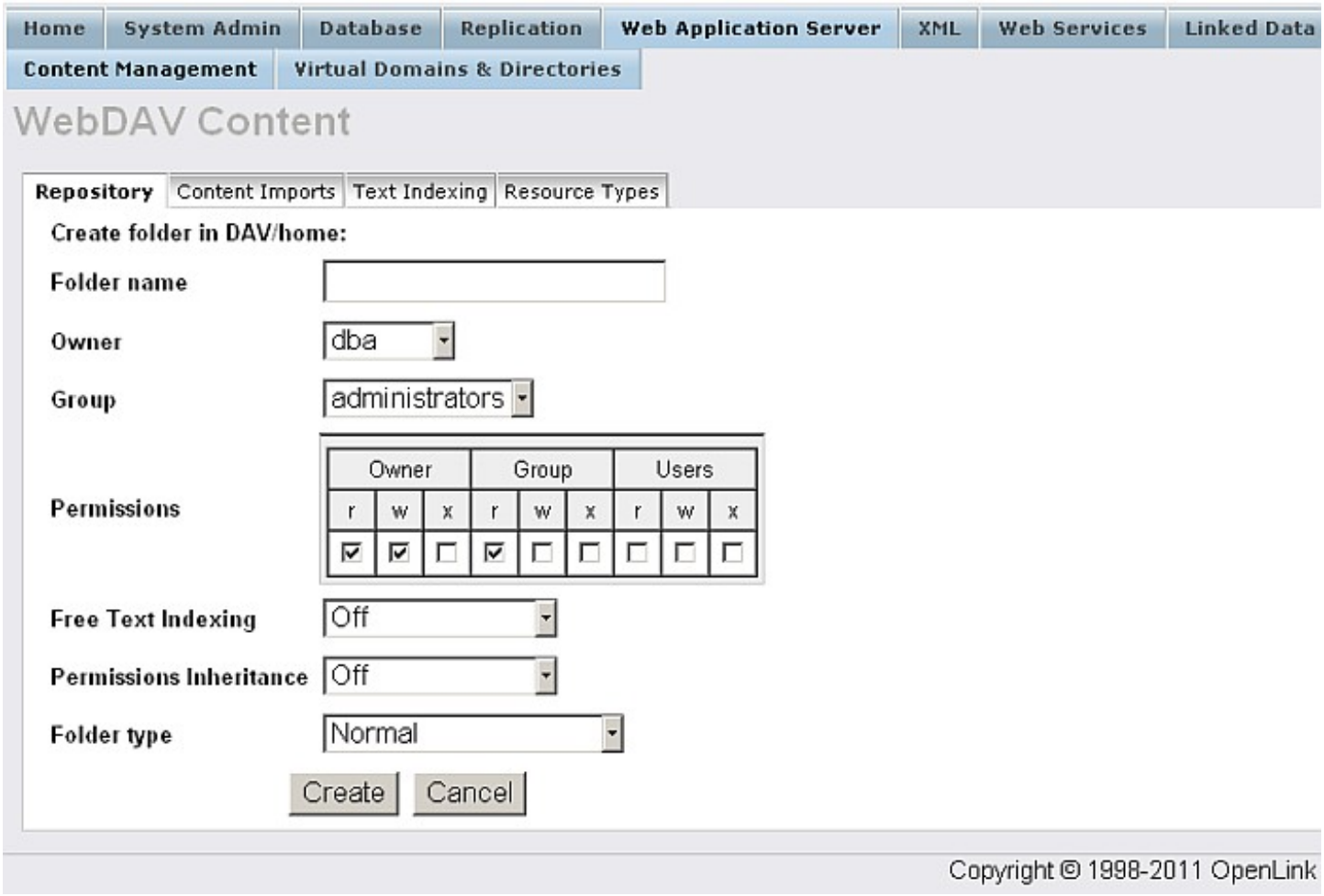
XML-related properties are generally set by specific XML-related services and do not need to be edited directly by users; however, this interface provides a quick means to correct a minor typo or other change without re-running the entire service. For example, you can change the `xml-sql-root` property, which controls the name of the root XML element for the document.

Adding New Resources

Resources may be added to the repository in two ways: by uploading files or by creating new collections (folders). Buttons for both these methods are on the page.

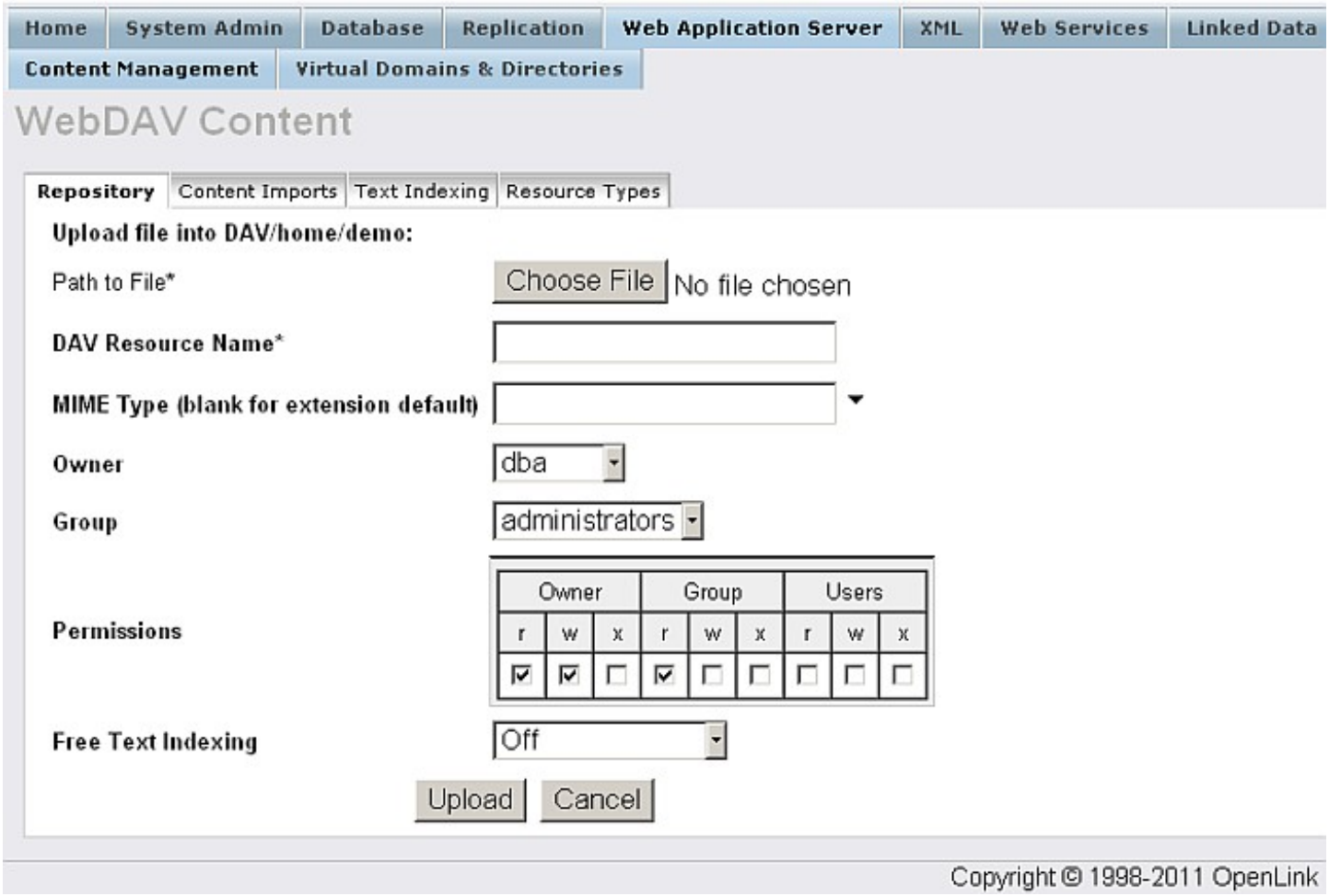
The "Create Folder" button brings up a form in which you can specify the name of the collection (folder), its owner and group, and the initial permissions. You may also turn on indexing for the folder's contents at this point.

Figure 6.80. Create Folder



The "Upload File" button brings up a form in which you can specify the name and location of a file on your local computer that is to be copied into the DAV repository. You need to specify the name of the resource in the repository and give it a type. You can also set the basic repository properties here.

Figure 6.81. Upload File



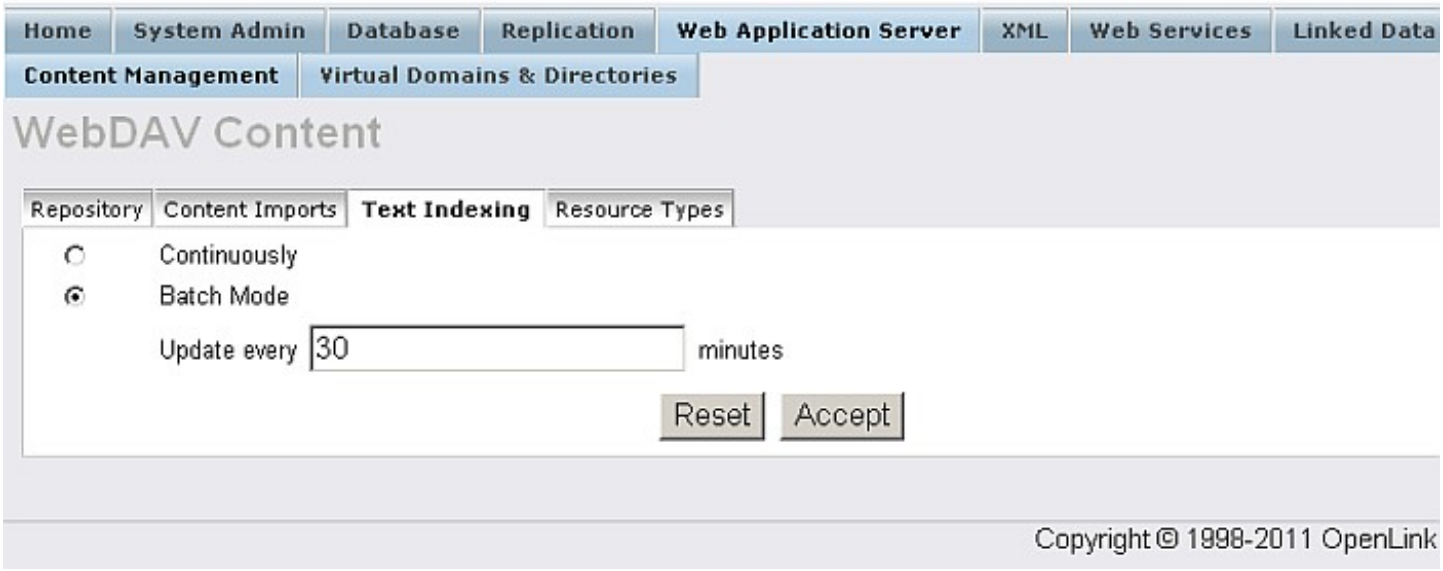
Free Text

Indexing Mode

When files are inserted into Virtuoso's WebDAV repository, if their type is a type of text file such as plain text TXT, XML, or HTML, then they may be automatically free text indexed.

By default files are automatically free text indexed as they are inserted into Virtuoso. This is very convenient but can be time consuming if you frequently insert or update text files. For this reason Virtuoso can be set to index in batch mode at a particular interval in minutes.

Figure 6.82. DAV Free Text Index Configuration



To change the free-text index mode to batch mode check the check-box and provide a non-zero time interval (in minutes). Press the *Accept* button to save the changes into the server.

6.2.5. Internet Domains

HTTP Virtual Directories

From Virtual Domains & Directories you can define Virtual HTTP directories. Virtual Directories let you define multiple HTTP server listeners in Virtuoso for either the same network interface or another one. Virtual directories can respond logically to a name or directly by IP address. Both types as well as default responses can be defined here.

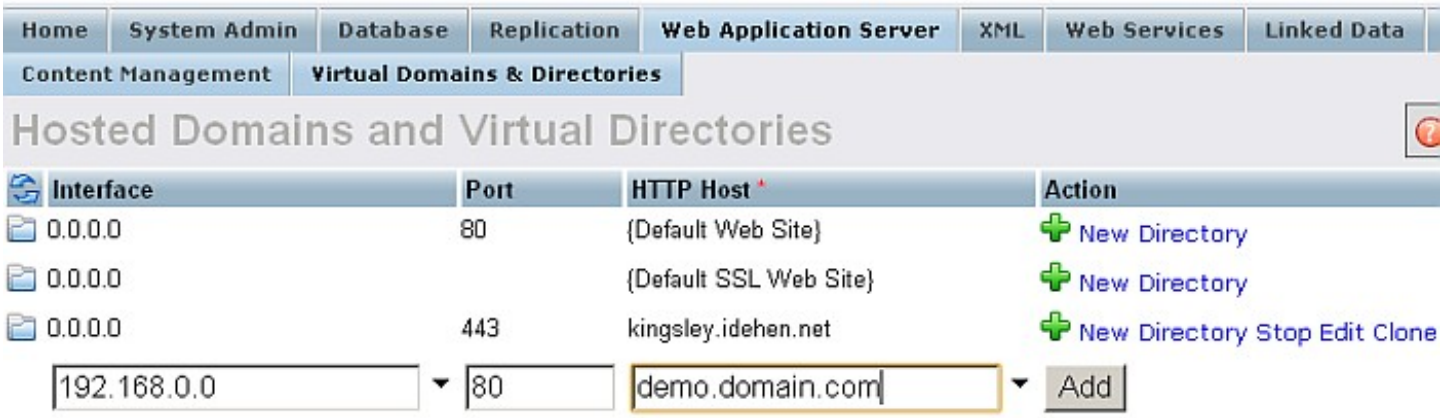
Each virtual directory can also have HTTP maps defined for it. This allow you to set logical paths on an HTTP directory to point to specific directories available to Virtuoso through the file system or DAV.

You can also publish stored procedures to a SOAP defined directory during the virtual directory definition.

Figure 6.83. Virtual Directories

Click on the *Add New Site* button to start adding a new Virtual Web Site and its directories.

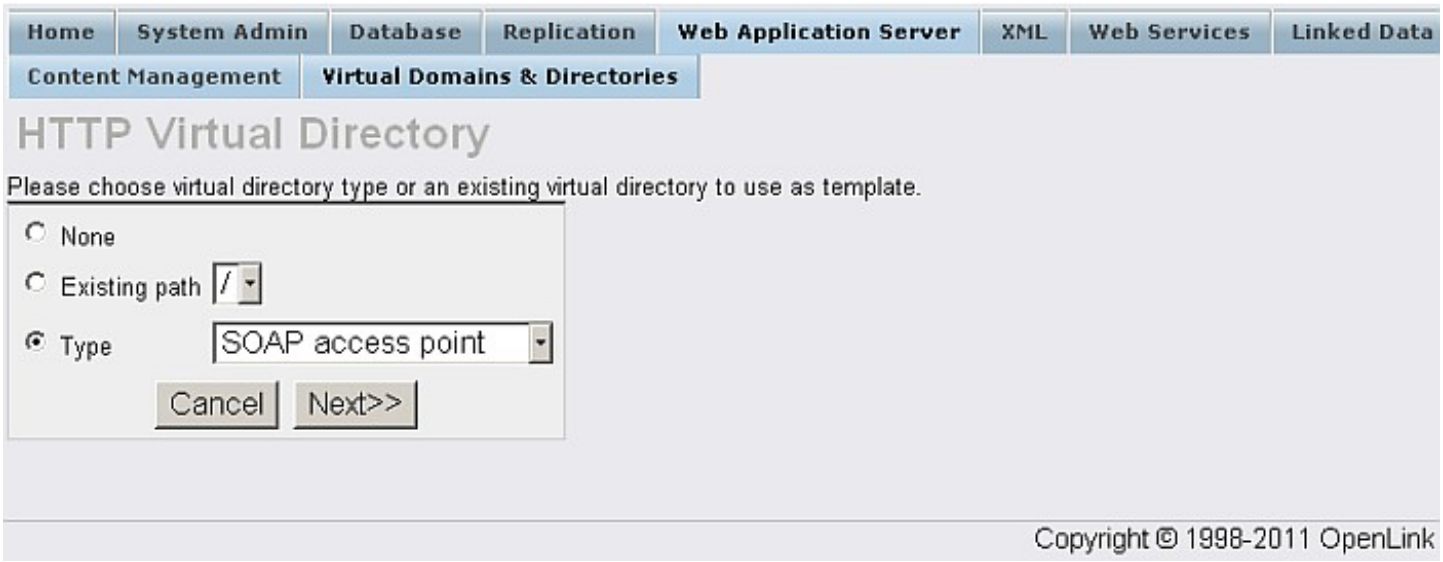
Figure 6.84. Virtual Directories: Site Details



When adding or editing a web site you must supply a host.domain name, which will be used to match again incoming requests to produce the correct response, the IP address of the network interface, to set-up the listener on, and the TCP port number that will be used to listen for incoming requests. Although Virtuoso will be listening on the specified interface you can set up multiple sites on this interface. The site required by client web browsers will be determined by the host name specified in the request. This provides the virtual site.

Click on "New Directory" to continue.

Figure 6.85. Virtual Directories Mappings



Before the directory settings are configured you can select from a few types to help configure the details to follow quicker and easier. For SOAP virtual directories this step is particular useful.

Select "Type" and then click "Next" to continue.

Figure 6.86. Virtual Directories

The "Virtual Directory Information" tab lets you configure most aspects of the virtual directory.

The default directory checkbox can be checked if you want the site being defined to act as the default site for the interface. This means that if a request is made to the interface that does not match a hostname defined for the interface, the default will be returned.

Logical Path will be the path that Virtuoso will respond to for this virtual directory mapping. This is what will be placed on the URL. Physical Path or URL is what Virtuoso will actually supply the content from. In either case you can use the Browse buttons to traverse the file systems graphically. Use the WebDAV Source Checkbox to instruct Virtuoso to use the WebDAV store for the physical location. "Default Page" will be returned if no page is specified in the incoming URL.

Note:
Virtual directories for SOAP must always use a physical path of /SOAP/.

The physical path of /SOAP/ does not need to exist in the filesystem under the VSP-root directory for normal SOAP operation. If it does existing it can be used to answer non-SOAP requests. Thus, configuring the virtual directory for SOAP with a "Default Page" can be used to avoid SOAP clients receiving HTTP 404 errors when testing the SOAP endpoint using standard HTTP only. Some SOAP applications assume the SOAP endpoint is down if they received HTTP 404 without checking the SOAP endpoint itself.

The permissions panel lets you choose whether to enable various abilities in the directory.

In the SOAP Options section you can publish or unpublish procedures and/or templates, both native and remote to the virtual directory using the Publish/Unpublish buttons respectively. The SOAP Options text-area allows you to specify other SOAP options such as DIME encapsulations and WS-security settings. These options are supplied as name=value pairs terminated with a semi-colon and a carriage-return. Here is an example of the options used for the default Interop test based demo virtual directory:

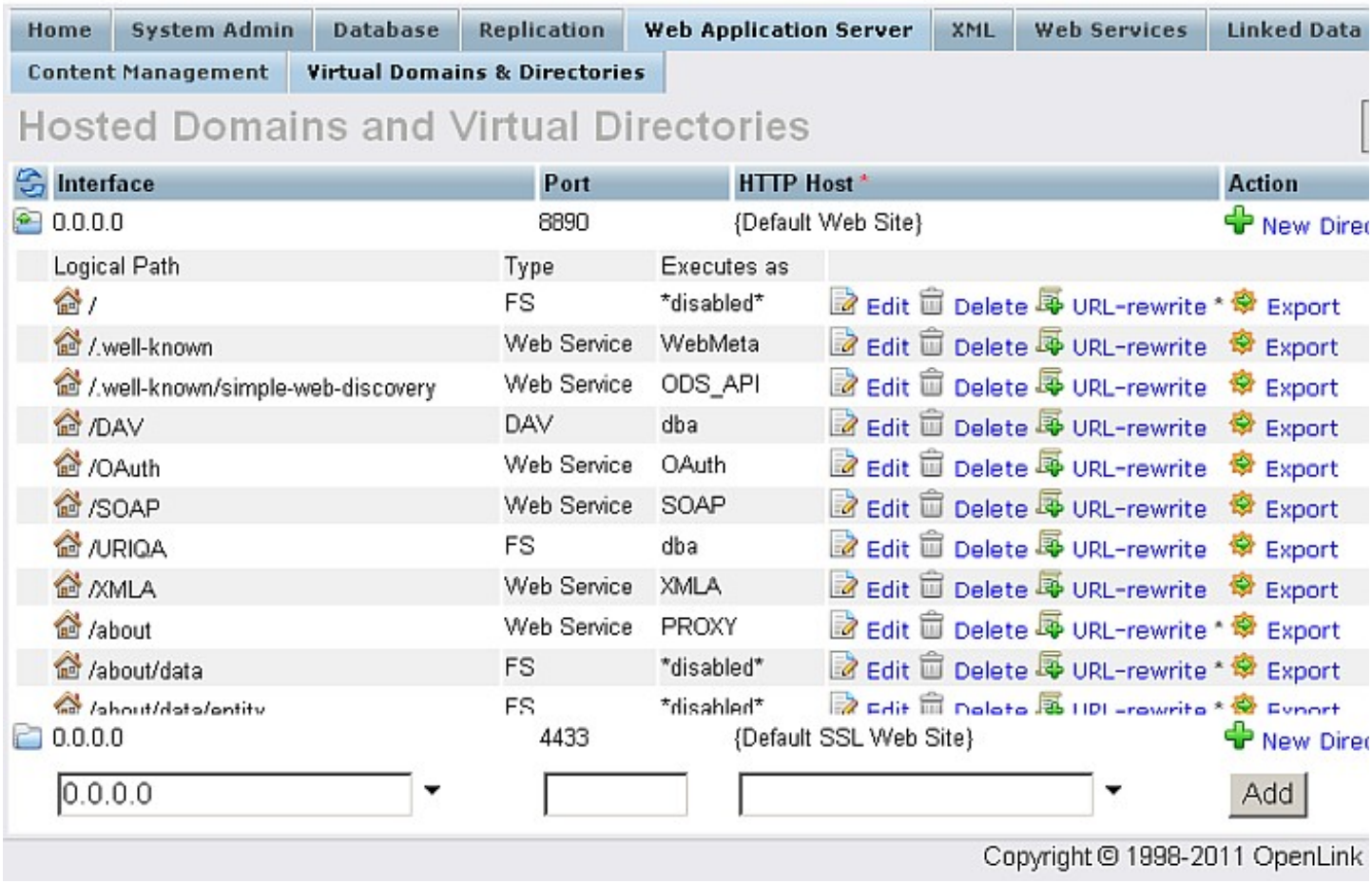
```
Namespace=http://soapinterop.org/;
MethodInSoapAction=no;
ServiceName=InteropTests;
HeaderNS=http://soapinterop.org/echoheader/;
CR-escape=yes;
```

See Also:
For a list of available SOAP Options review the end section of the SOAP chapter: Optional Parameters to the SOAP Endpoint .

The Authentication Options panel lets you define the authentication rules for the Virtual Directory.

Once the form details have been completed press on the Add button to save the them and proceed to configure mappings for the directory.

Figure 6.87. Virtual Directories Mappings



This screen lists mappings that have been defined for the virtual site. If you have just created a fresh site then only one line will be displayed. The "Add Virtual Directory" button will let you define more. Back returns you to the start page, Edit and Delete allow you to edit or remove existing mappings as their link suggests.

See Also:
Virtual Directories

For example, here are the basic steps to be performed, in order to mount FS folder to DAV:

1. Suppose there is a folder with name "test" in your FS and it is under the root of the ServerRoot defined in your virtuoso ini file.
2. Also suppose in the folder "test" there is a file index.html with simple content:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
  <head>
    <title>OpenLink Virtuoso Conductor Simple Test</title>
  </head>
  <body>
    <table cellspacing="0" cellpadding="0" border="1" width="50%">
      <tr>
       <td><b>Name</b></td><td></b>State</b></td>
      </tr>
      <tr>
       <td>Simple test for mounting FS to DAV</td><td>Successful.</td>
      </tr>
    </table>
  </body>
</html>
```

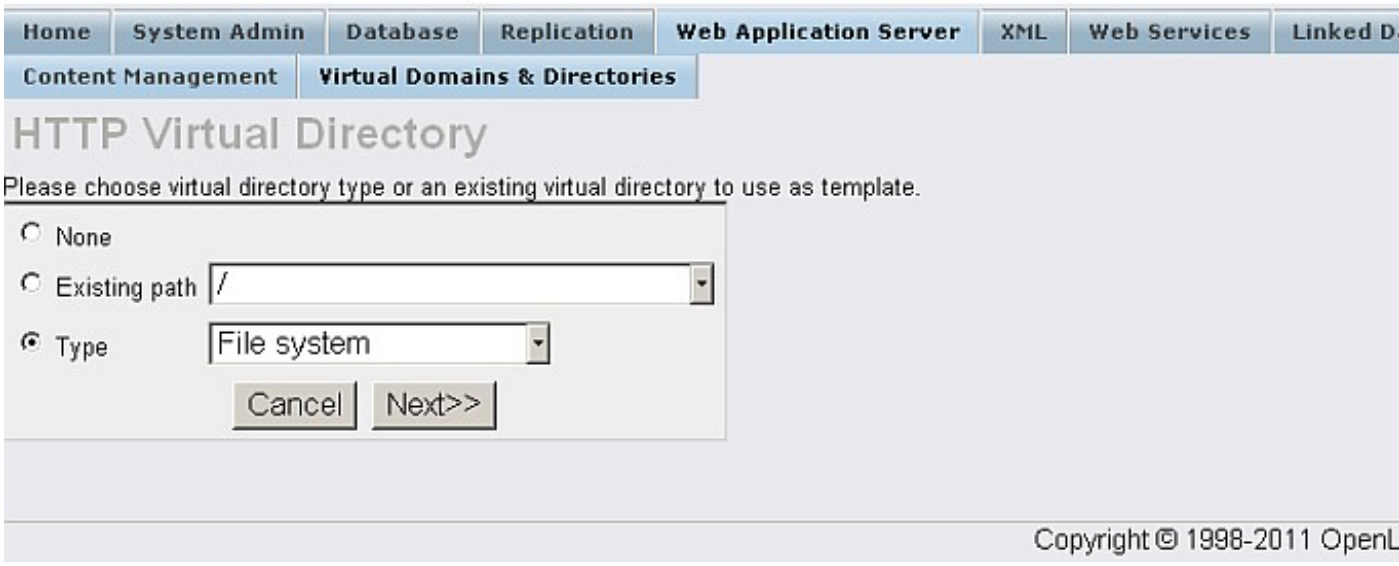
3. Install the Conductor package
4. Go to http://host:port/conductor and login as dba user.

5. Go to Web Application Server -> Virtual Domains & Directories.

Figure 6.88. Mount FS to DAV

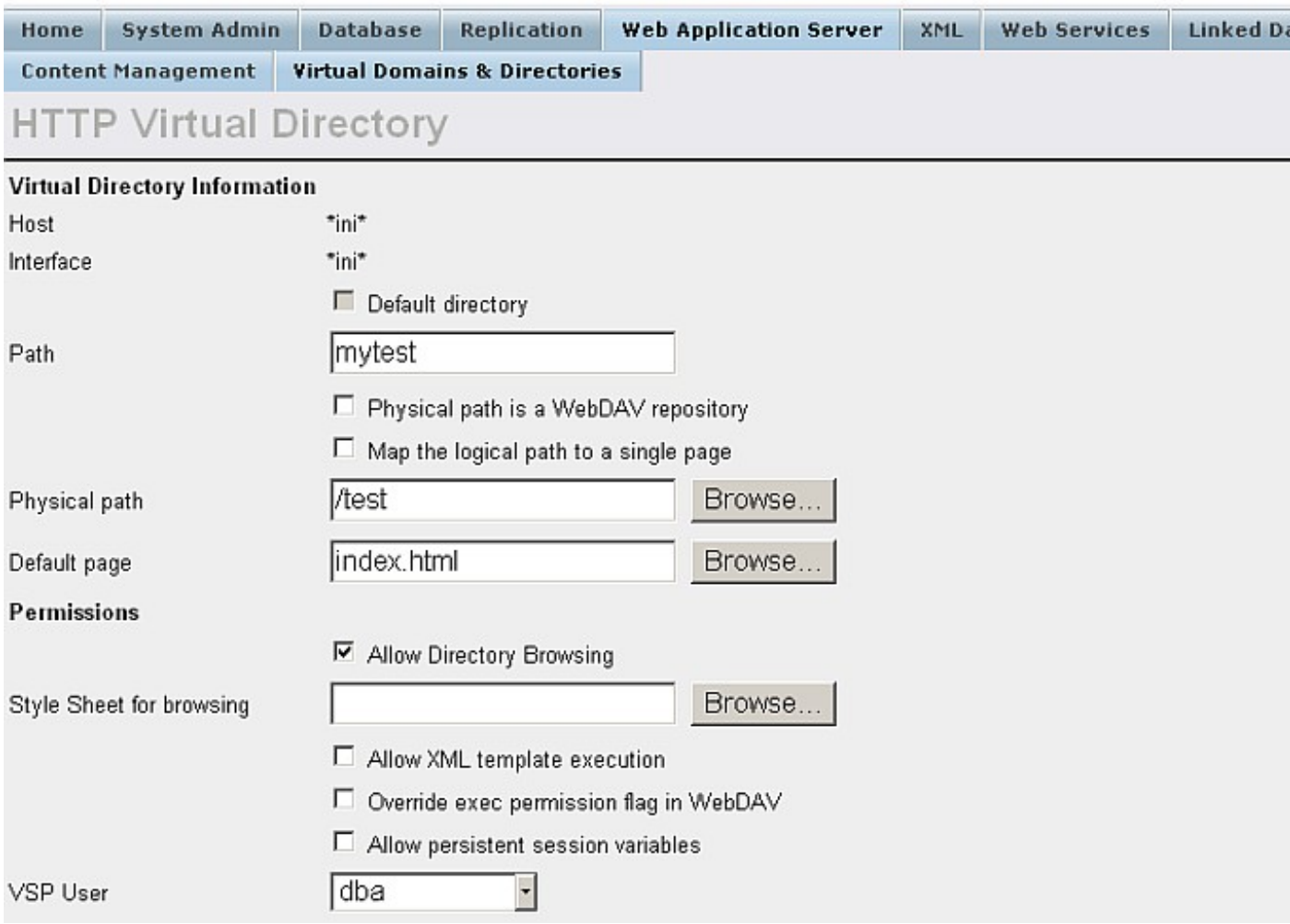
6. For your {Default Web Site} click the link "New Directory".
7. Check the check-box "Type" and select from the drop-down list "Filesystem".

Figure 6.89. Mount FS to DAV



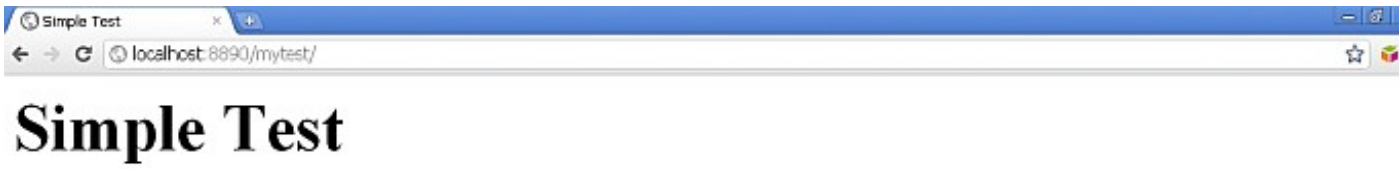
8. Click "Next"
9. In the shown form: Enter for the field "Path": mytest; Enter for the field "Physical path": /test; Enter for the field "Default page": index.html; Check the check-box "Allow Directory Browsing"; Leave the rest of the fields with their default values.
10. Click "Save Changes"

Figure 6.90. Mount FS to DAV



11. From your browser access the url: http://host:port/mytest/
12. As result the content of the index.html file will be shown:

Figure 6.91. Mount FS to DAV



6.2.6. XML Services

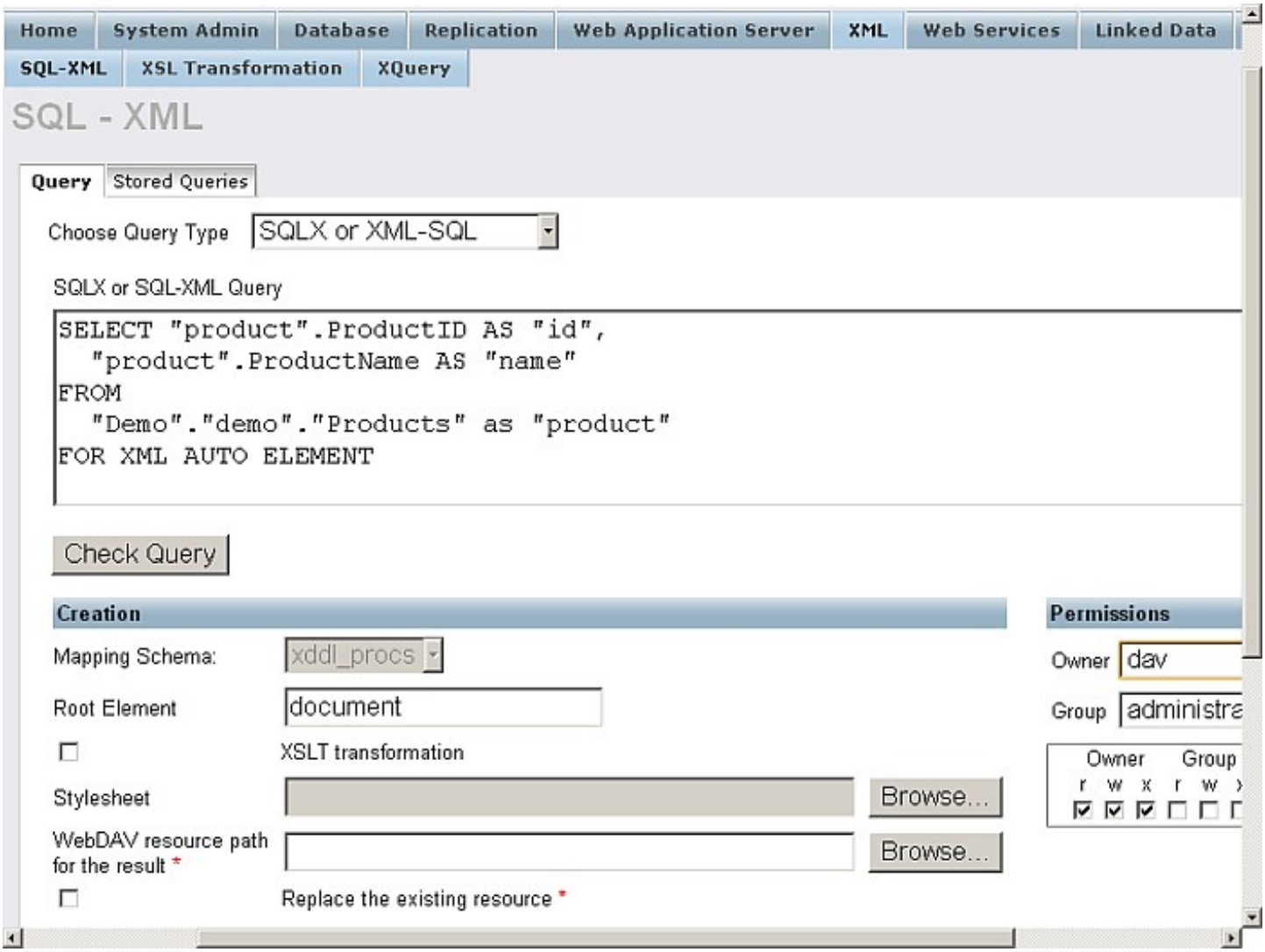
SQL-XML Statements

Go to XML/SQL-XML and enter a SQL to XML statement in the *SQLX or SQL-XML Query* text-area:

Example 6.15. For example:

```
SELECT "product".ProductID AS "id",
  "product".ProductName AS "name"
FROM
  "Demo"."demo"."Products" as "product"
FOR XML AUTO ELEMENT
```

Figure 6.92. SQL-XML Statements



Type the root XML element name into the *Root Element* field. The results of the query will be contained within this root element.

Type the full path and resource name where you want your XML resource to reside under WebDAV in the *WebDAV Resource path for the result* field. Collection(s) described in the full path must already exist.

You may use the *Browse* button next to the *WebDAV Resource path for the result* field to navigate existing WebDAV collections (folders) or resources to store the result of query.

From the "Stored Queries" tab if you choose a resource that already exists as a query in WebDAV resource, after click the "Edit" link, the form will automatically acquire the details from the query.

If an XSLT transformation should be performed on retrieval time then you may either type the location of the stylesheet in the *Stylesheet* field or press the *Browse* button next to it to search for a valid XSLT stylesheet from the WebDAV repository.

Provide the appropriate *WebDAV owner* and *group* of stored result using the drop downs provided.

Select appropriate *permissions* for the stored result.

The query can be set to update itself at specific intervals of time or execute every time the resource is retrieved. If *Persist XML* is checked the query will be executed on *Update every* 10 minutes by default.

Press the *Execute* button to execute the statement and store as a WebDAV resource. Use *Reset* to clear the form

If you want to build a schema definition for the result then press the *Create XML Schema* radio-button.

Virtuoso can provide the generated XML output with a full DTD for the data. Press *Create External DTD* to enable this option for

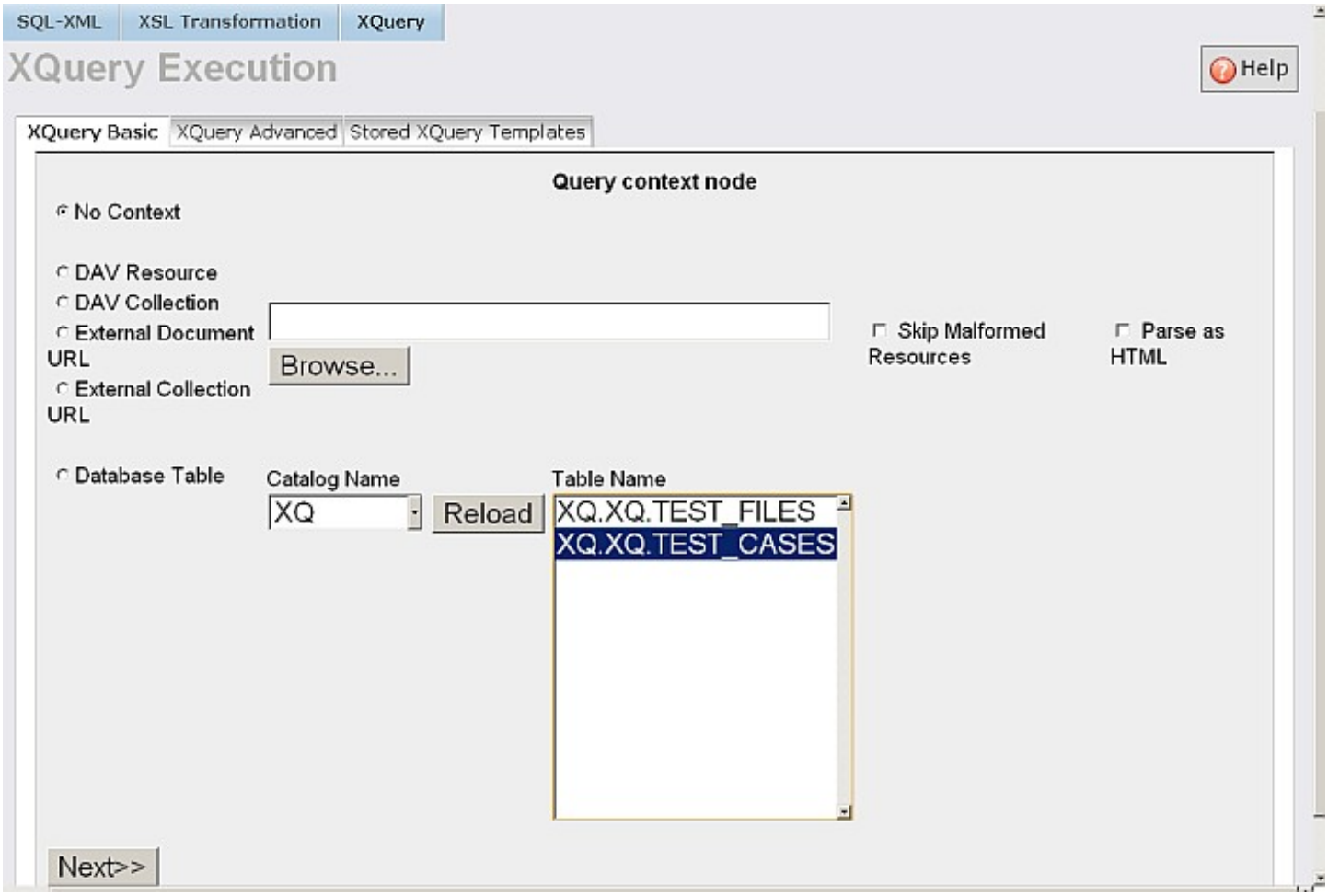
the query.

XQuery Search

Go to XML / XQuery /Xquery Basic.

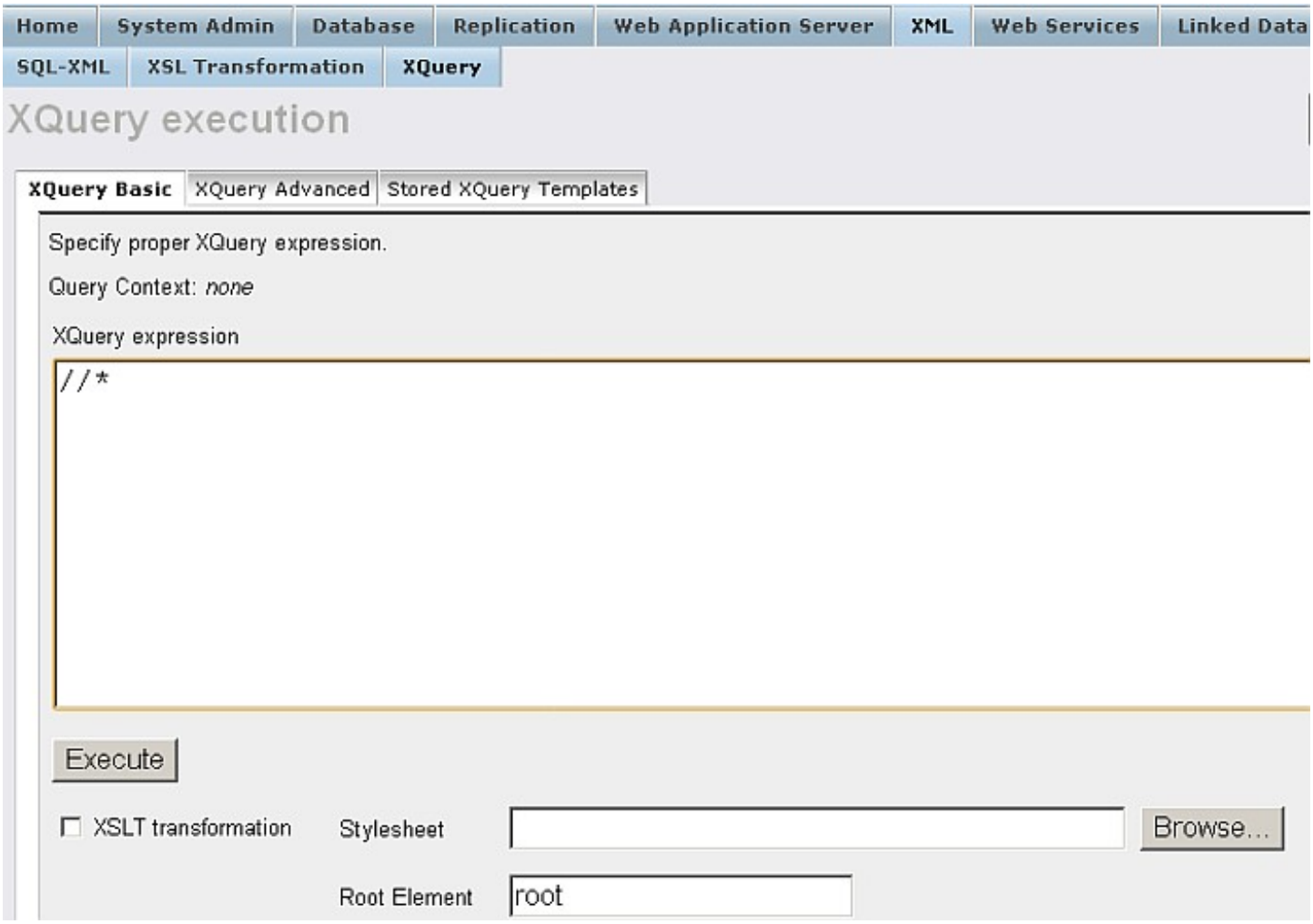
XQuery Document Search applies the XPATH expression to every realized XML contained within the Query Scope to qualify search hit results. Note that this does not include XML Views unless persistent.

Figure 6.93. XPATH Query of DAV



Choose the *Scope of Query* from within the WebDAV repository. You may define the scope as either No Context, DAV Resource (file), DAV Collection (folder), External Document URL, External Collection URL or Database Table. Use the *Browse* button to browse the contents of WebDAV to make a selection. The query will then be confined to the selected resource or collection and its children. Then Click the button "Next" in order to continue.

Figure 6.94. XPATH Query of DAV



Enter that XPATH query expression into the text area that you wish to interrogate your WebDAV XML documents with. e.g. *//** or *//title*

The *Root Element* field allows you specify the name of the root element to contain document element matches for each document found. This aids stylesheet design.

Choose an *Output Style Sheet* by either typing its URL or using *Browse* to select an XSL resource from WebDAV. You only need to specify a style sheet if you want to transform your XML results using XSLT.

Click the button "Next" in order to continue.

Choose an *Store into* location by either typing its URL or using *Browse* to select a DAV resource from WebDAV

Set *Permissions* for the output result. Check "r" for read rights for Group and Users in order to view later the saved xml file.

You may want to *Replace the existing resource* by checking the shown check-box with this label.

Choose *Output Type* that you wish to obtain. *Persist XML* should be selected with Update interval im minutes or if you want to be created as XML Template, fill in the "Create as XML Template Description" field.

Click the button "Save".

Figure 6.95. XPATH Query of DAV

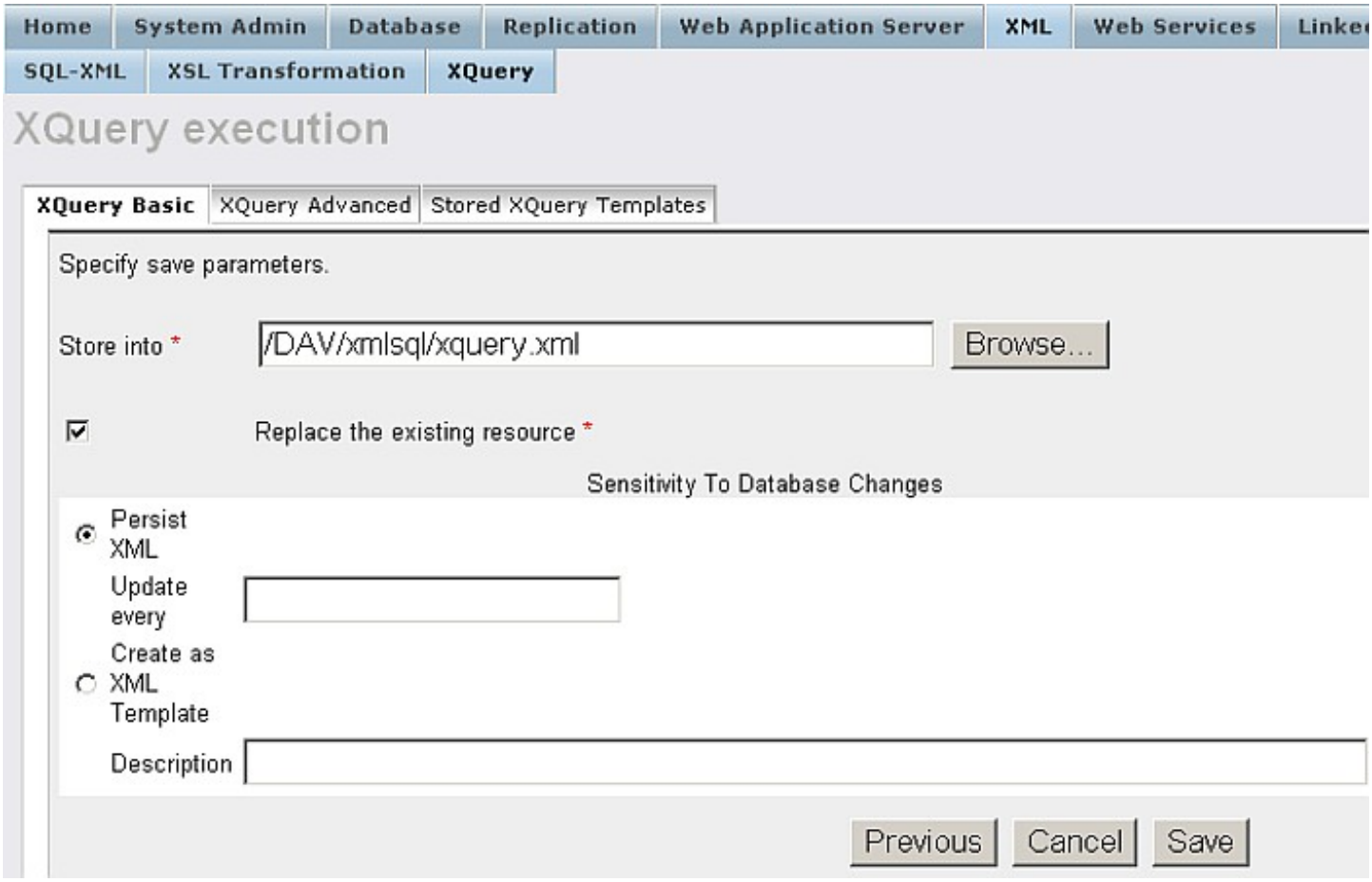
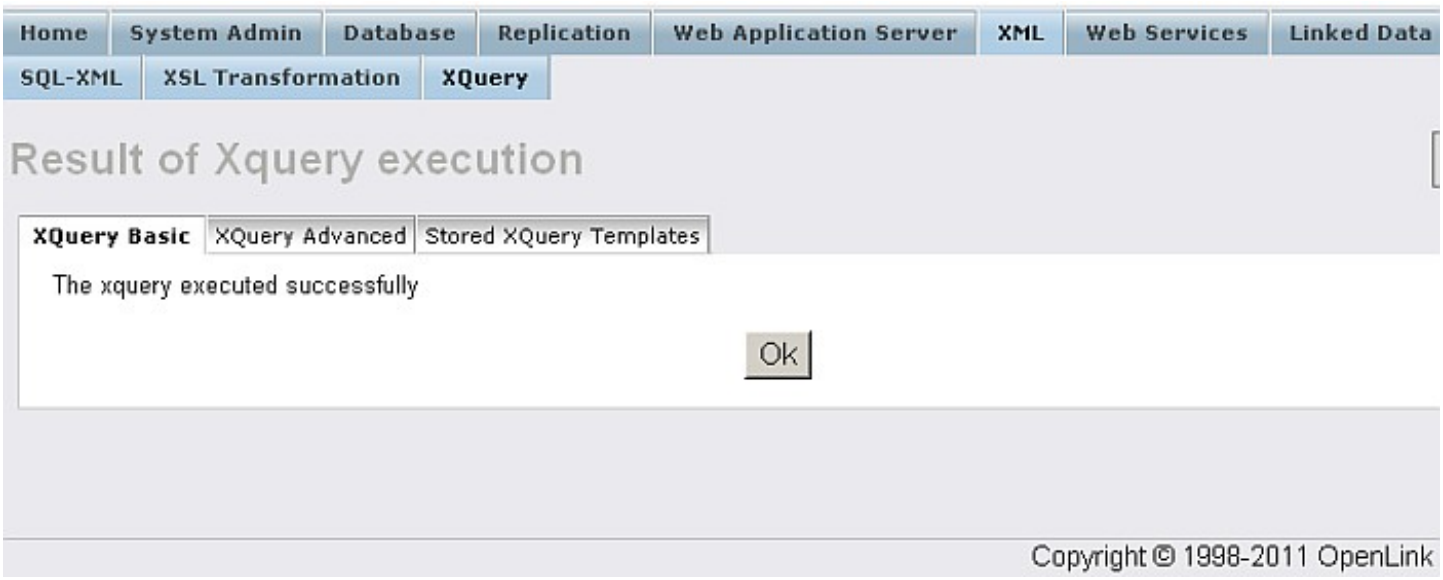


Figure 6.96. XPATH Query of DAV



If for location you have chosen /DAV/xmlsql/xquery.xml, you can view the saved file accessing the url: http://host:port/DAV/xmlsql/xquery.xml

Figure 6.97. XPATH Query of DAV

```
<?xml version="1.0" encoding="UTF-8" ?>
- <root>
  - <bib>
    - <book year="1994">
        <title>TCP/IP Illustrated</title>
      - <author>
          <last>Stevens</last>
          <first>W.</first>
        </author>
        <publisher>Addison-Wesley</publisher>
        <price>65.95</price>
      </book>
    - <book year="1992">
        <title>Advanced Programming in the Unix environment</title>
      - <author>
          <last>Stevens</last>
          <first>W.</first>
        </author>
        <publisher>Addison-Wesley</publisher>
        <price>65.95</price>
      </book>
    - <book year="2000">
        <title>Data on the Web</title>
      - <author>
          <last>Abiteboul</last>
          <first>Serge</first>
        </author>
      - <author>
          <last>Buneman</last>
          <first>Peter</first>
        </author>
      - <author>
          <last>Suciu</last>
          <first>Dan</first>
        </author>
        <publisher>Morgan Kaufmann Publishers</publisher>
        <price>39.95</price>
      </book>
```

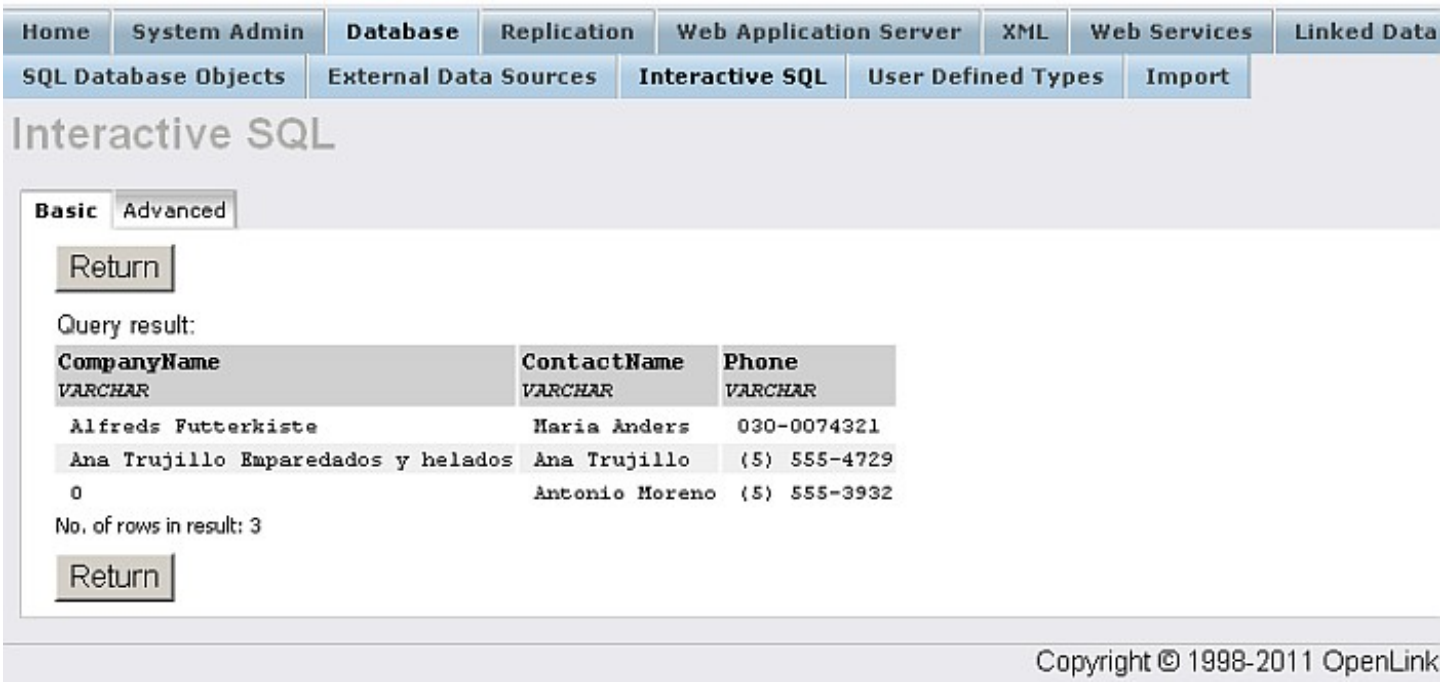
6.2.7. Query Tools

Relational Data using SQL

Conductor Interactive SQL allows you to quickly and directly query Virtuoso using SQL. It offers *Save* and *Load* facilities which allow SQL queries to be saved as an XML template, and read back later. With a query in the SQL Statement box click on the *Execute* for the results which will in the "Base" tab with option to return back to the query area. The *Clear* clears the SQL Statement text box.

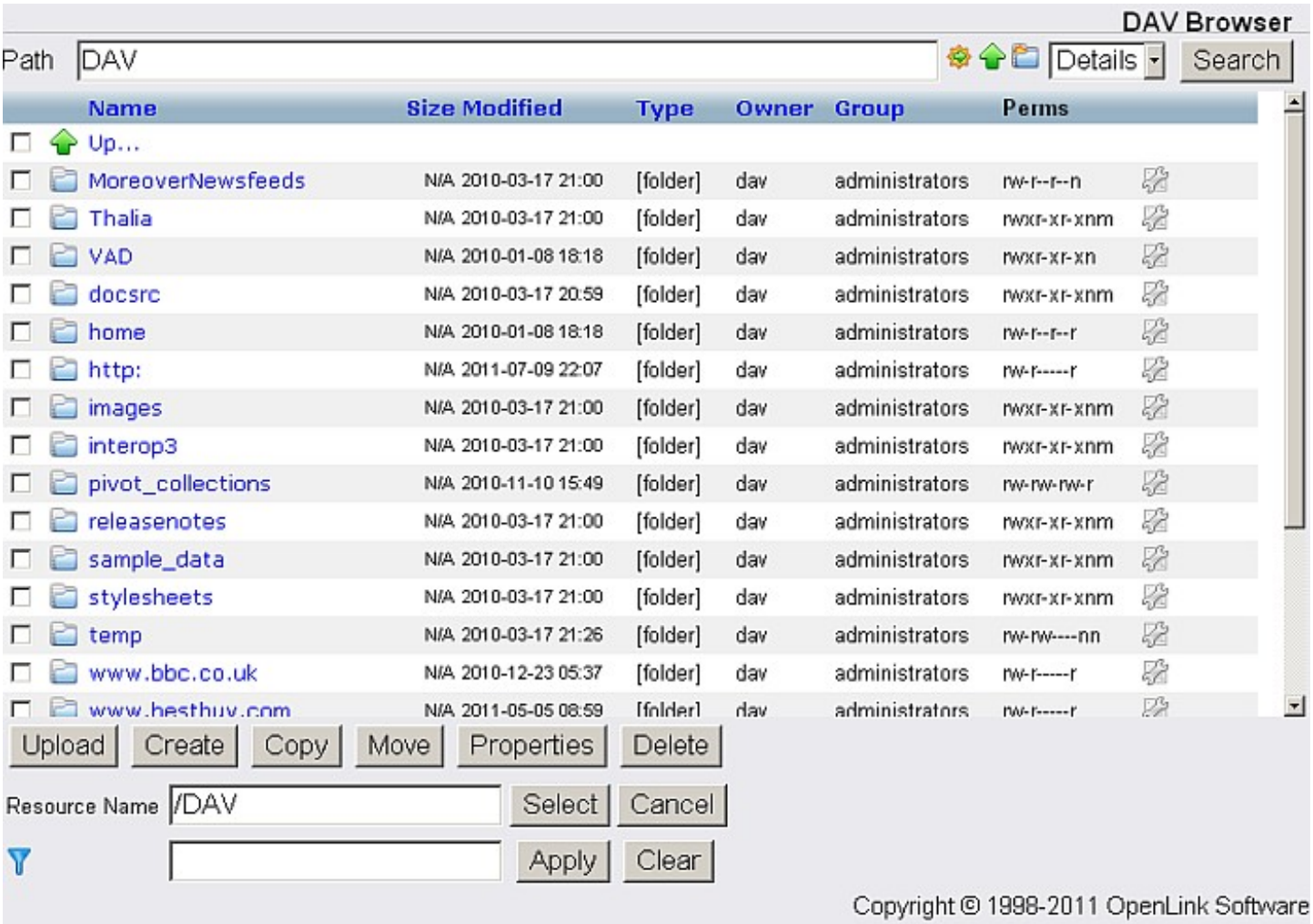
Figure 6.98. Querying Relational Database Using SQL

Figure 6.99. Results



Specify the location for the file to be saved to by selecting the "WebDAV source" or "Local file" check-box.

Figure 6.100. Saving SQL in an XML Template



Click the "Browse" button. As result will be opened the Virtuoso WebDAV/File Browser where you should define the XML template based on the SQL Query. Specify a *Root Element* that will contain the resulting XML tree. Specify the file name and location of the XML Template.

XML Data Using XQuery

The Conductor Interactive XQuery facility allows you to create, execute, save and reload queries using the evolving W3C XML Query (XQuery) Language . Virtuoso currently supports the 1.0 version of this language.

This language uses XPath-like expressions, as well as a set of functions and operators, to permit effective parallel searching of a set of XML documents. Where XPath works with one XML "tree", XQuery searches a "forest". The result is an XML document.

In order to create an XQuery you must both create the query statement - by typing or pasting it into the text box - and specify the document context. Since Virtuoso's XQuery implementation operates over XML data in relational tables, this means the tables and columns that are to be searched.

Note that the XQuery language also allows a query to specify all or part of the document context for the query. In the example below we will see how these can interact. The user interface form permits you to select a table - either one of the XQuery test data tables that come with Virtuoso, or the WS.WS.SYS_DAV_RES table, which stores Virtuoso's WebDAV Repository content.

The form specifies a *`Key Column`* and a *`Data Column`* . For the sample tables, the values for these are filled in for you. The *`Path`* is prepended to any `document()` function specified in the query text to find Key column values of XML trees against which the query is to be run.

Once a query has been written and debugged, it can be saved by pressing the Save button. This brings you to the form for saving a query as an XML Template in the DAV repository.

Pressing the Execute button causes the query result (an XML tree) to be shown on the page below the Statement type-in box.

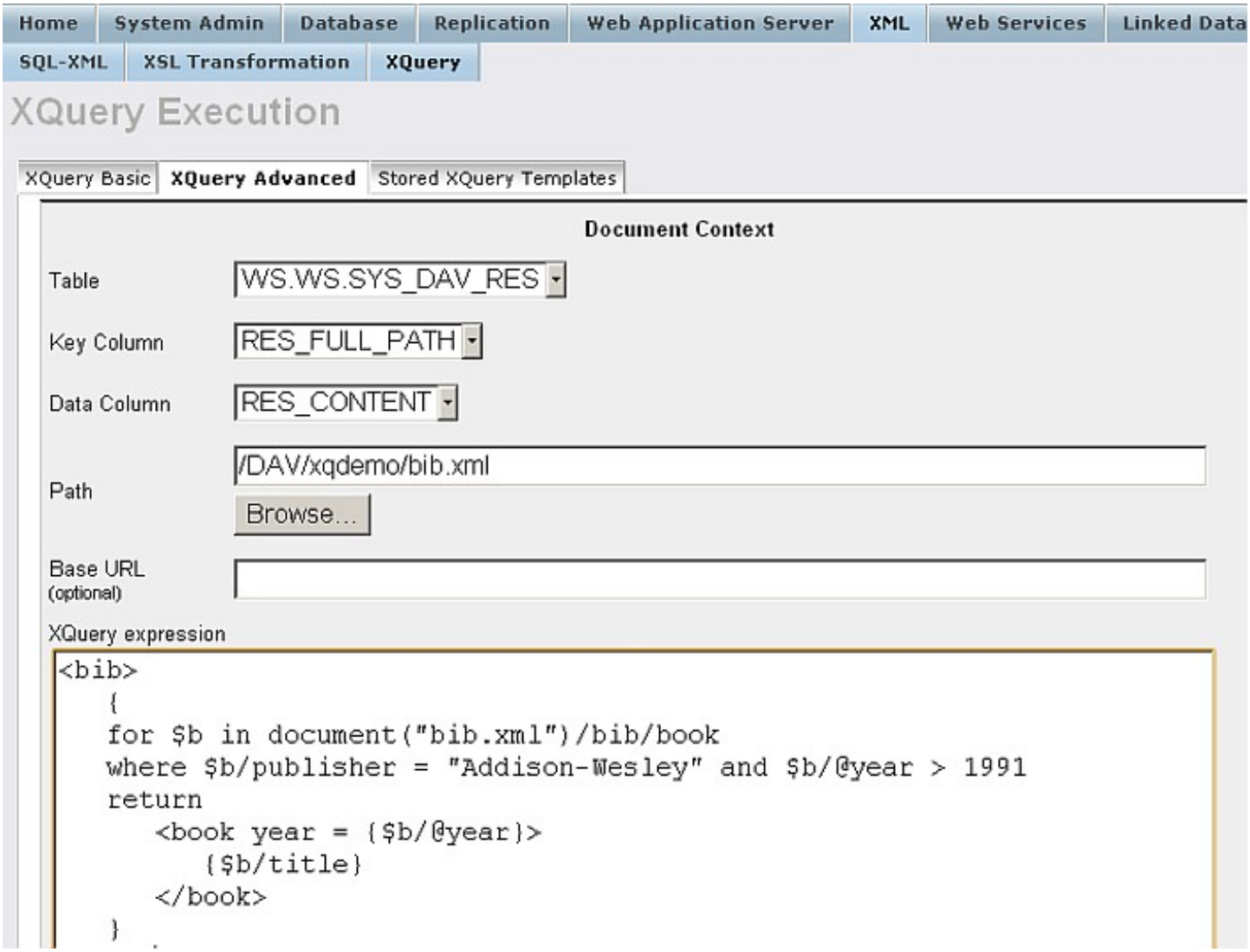
Example 6.16. XQuery Test File Example

In this example, we will query the table XQuery test files table, with "name" as the key column and "text" as the data column.

The query text, shown below, is a sample query from the W3C's XML Query Use Cases document (http://www.w3.org/TR/xmlquery-use-cases). This query contains a document() call specifying a document named "bib.xml". In order to have the query run properly, we first set the Path form value to "xqdemo/". This causes the query to find all rows in the table XQ.XQ.TEST_FILES that have the value "xqdemo/bib.xml" in their Name column.

```
<bib>
   {
   for $b in document("bib.xml")/bib/book
   where $b/publisher = "Addison-Wesley" and $b/@year > 1991
   return
      <book year = {$b/@year}>
         {$b/title}
      </book>
   }
</bib>
```

Figure 6.101. XQuery Test File Results



6.2.8. Replication & Synchronization

Snapshot Replication

Conductor "Replication" offers manage Virtuoso Replication. You are offered with "Basic", "Incremental", "Bidirectional Snapshot" and "Transactional" sub-tabs.

- From

  Replication/Basic/Query(Table select)to local

  to replicate local table(s) to another data source click

  Create New Snapshot

  and follow the wizard.
- To copy the changes from the local table to the remote select some replications and press

  Synchronize
- To drop the replication definition without dropping the destination table select some replications and press

  Remove
- To drop the replication definition and the destination table select some replications and press

  Remove & Drop Remote
- To copy the changes from the local table to the remote automatically select some replications, enter minutes in

Scheduled Interval

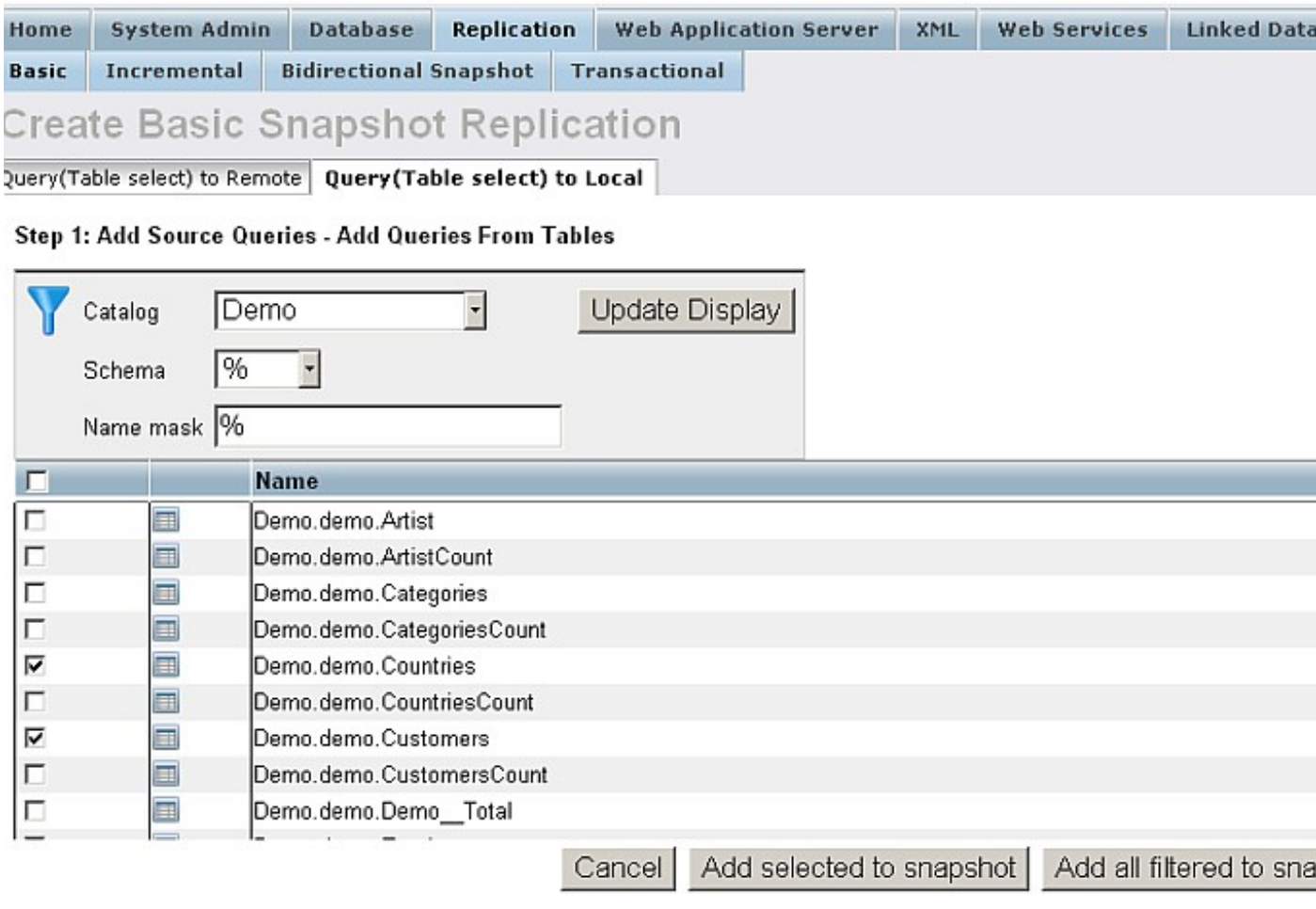
and press

Schedule

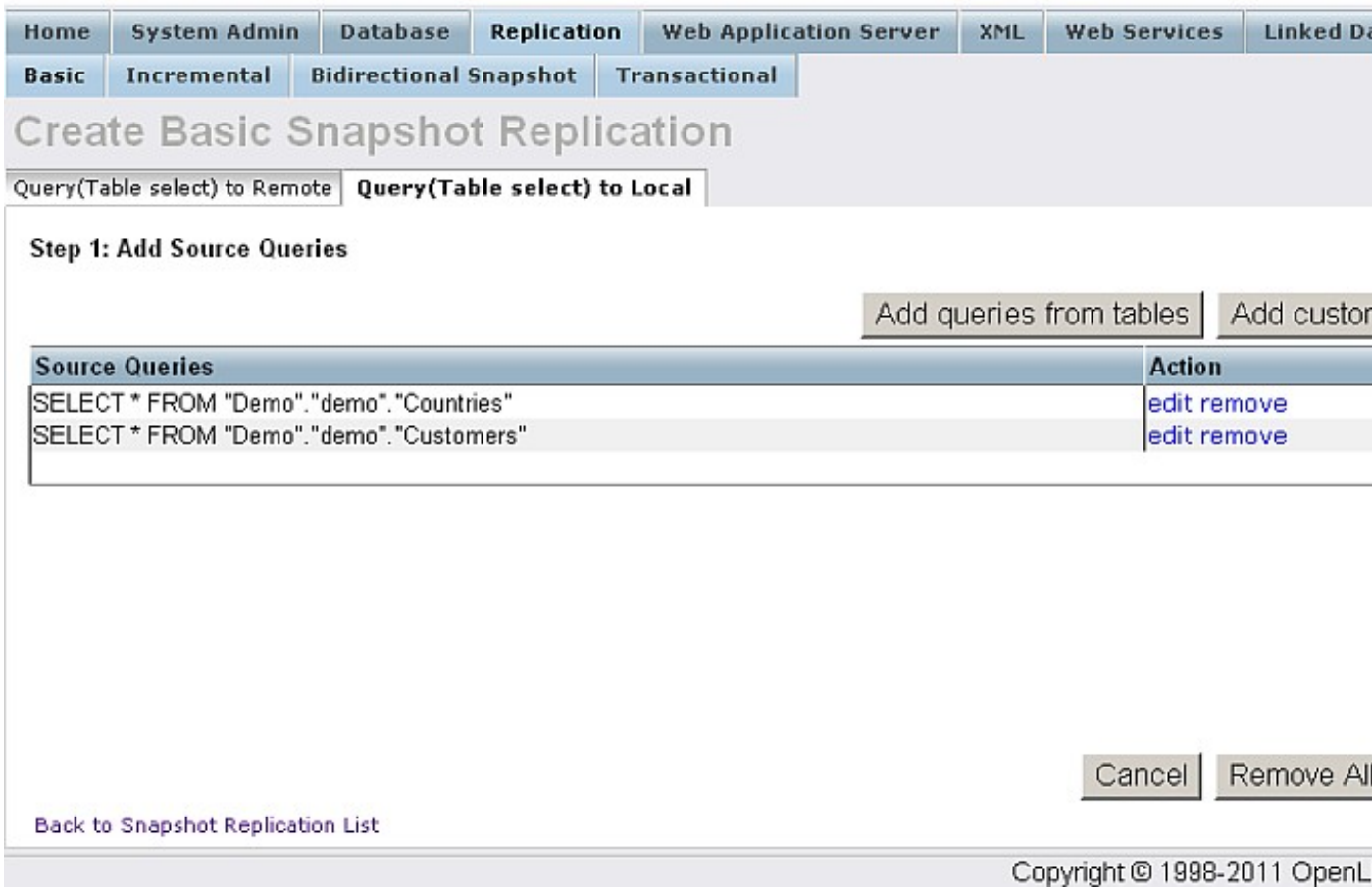
- To remove scheduled update select the replications enter 0 in "Scheduled Interval" and press Schedule

- **Figure 6.102. Snapshot Replication**

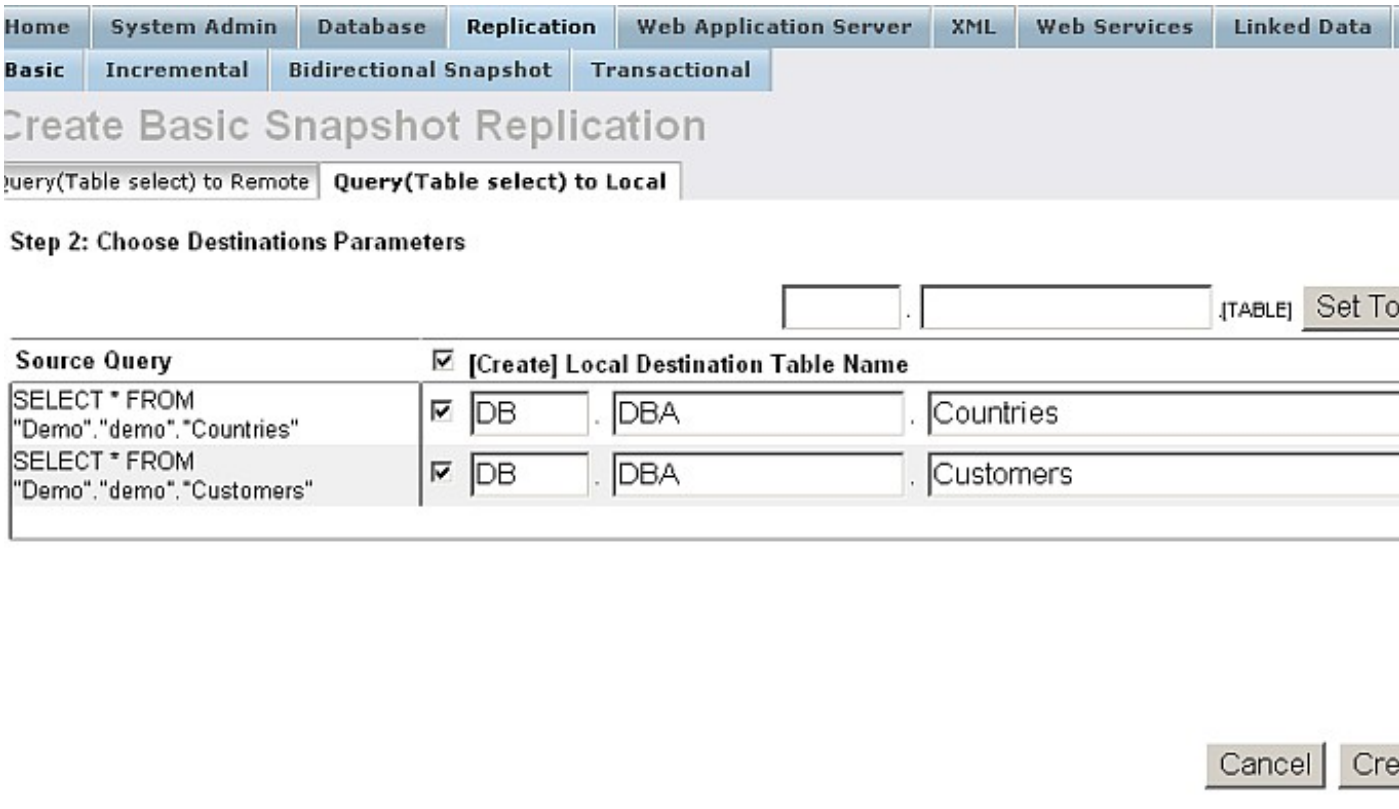
- **Figure 6.103. Snapshot Replication**



- **Figure 6.104. Snapshot Replication**

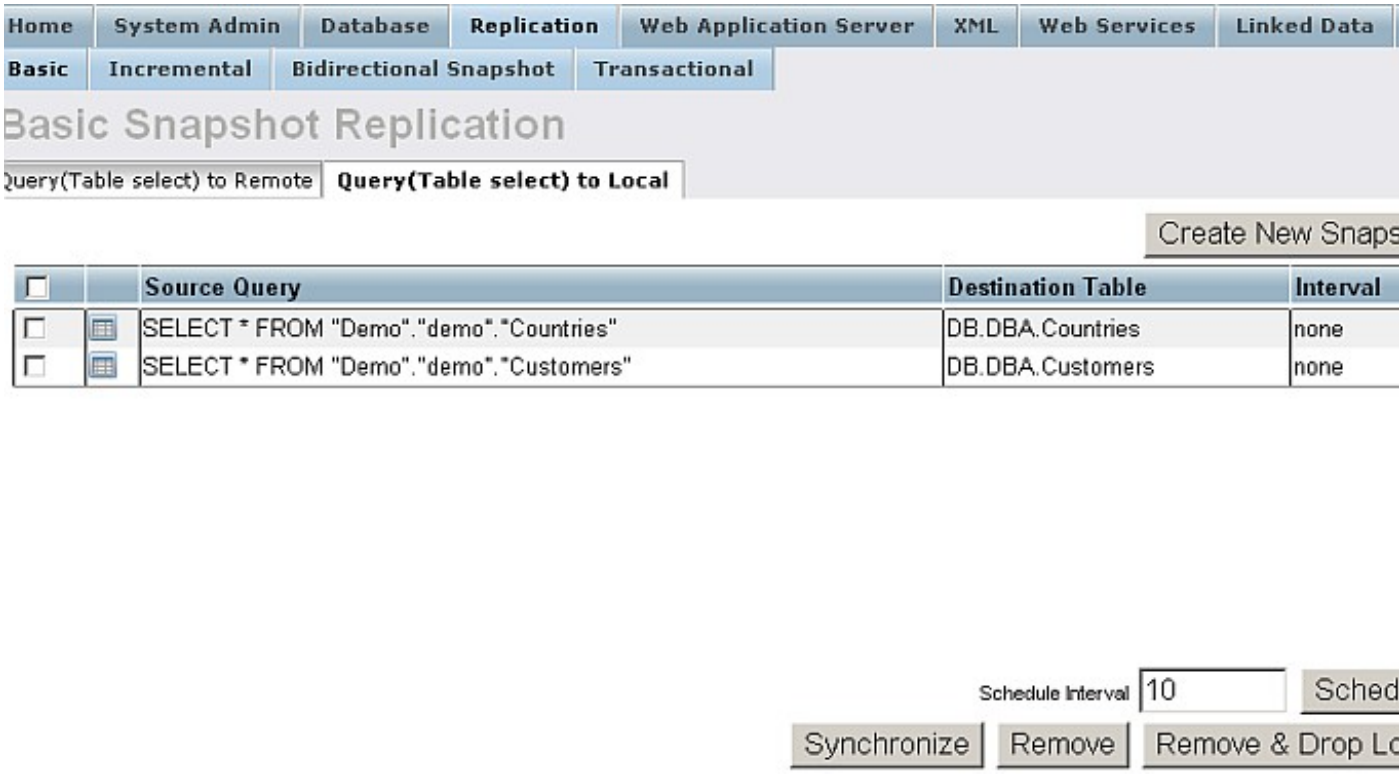


♦ **Figure 6.105. Snapshot Replication**



♦ **Figure 6.106. Snapshot Replication**

♦ **Figure 6.107. Snapshot Replication**



Transactional Replication

Publications

The Transaction Publication screen lists publications. You can add, edit or remove publications as necessary, each time using the guide-lines of the wizard.

Figure 6.108. Transaction Replication - Publications

Add a new publication by clicking on the *Create* button.

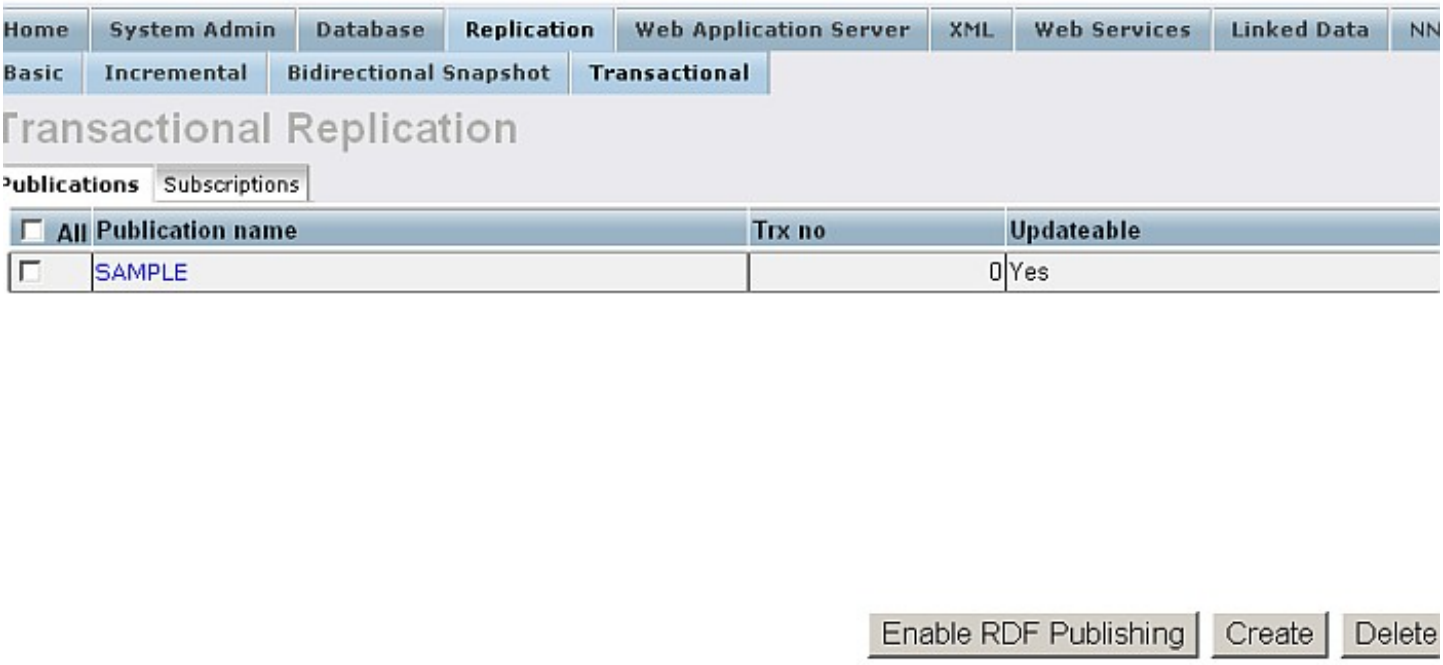
Edit an existing publication by clicking on publication name.

Remove publications by selecting their checkbox and clicking on the *Delete* button. Use caution, as this action cannot be reverted.

When creating a new publication you must supply a name. At this point you have the option to create an updateable publication - see the Bi-Directional Transactional Replication section for more details - which allows updates to come from subscribers rather than being limited to originating from the publisher only.

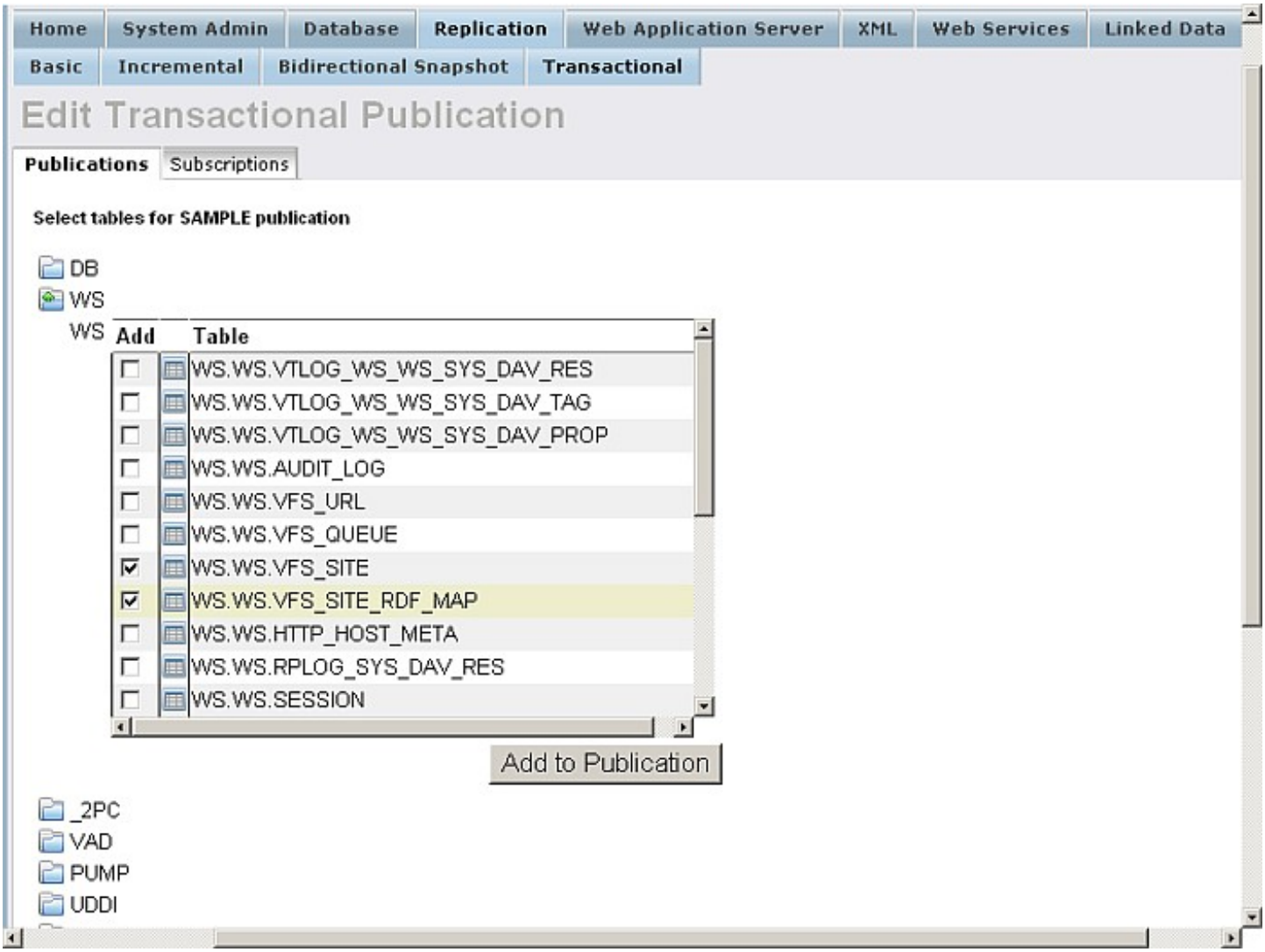
Figure 6.109. Transaction Replication - New Publication

Figure 6.110. Transaction Replication - List of Publications



Once a new publication has been made you can add database objects to it using the *Add ...* buttons, or remove them by selecting their checkbox and using the *Remove* button. To add a new table to the publication click on *Add Table/Procedure* . Follow the wizard by choosing the appropriate database/catalog by clicking on its name, you will then be presented with the tables within it.

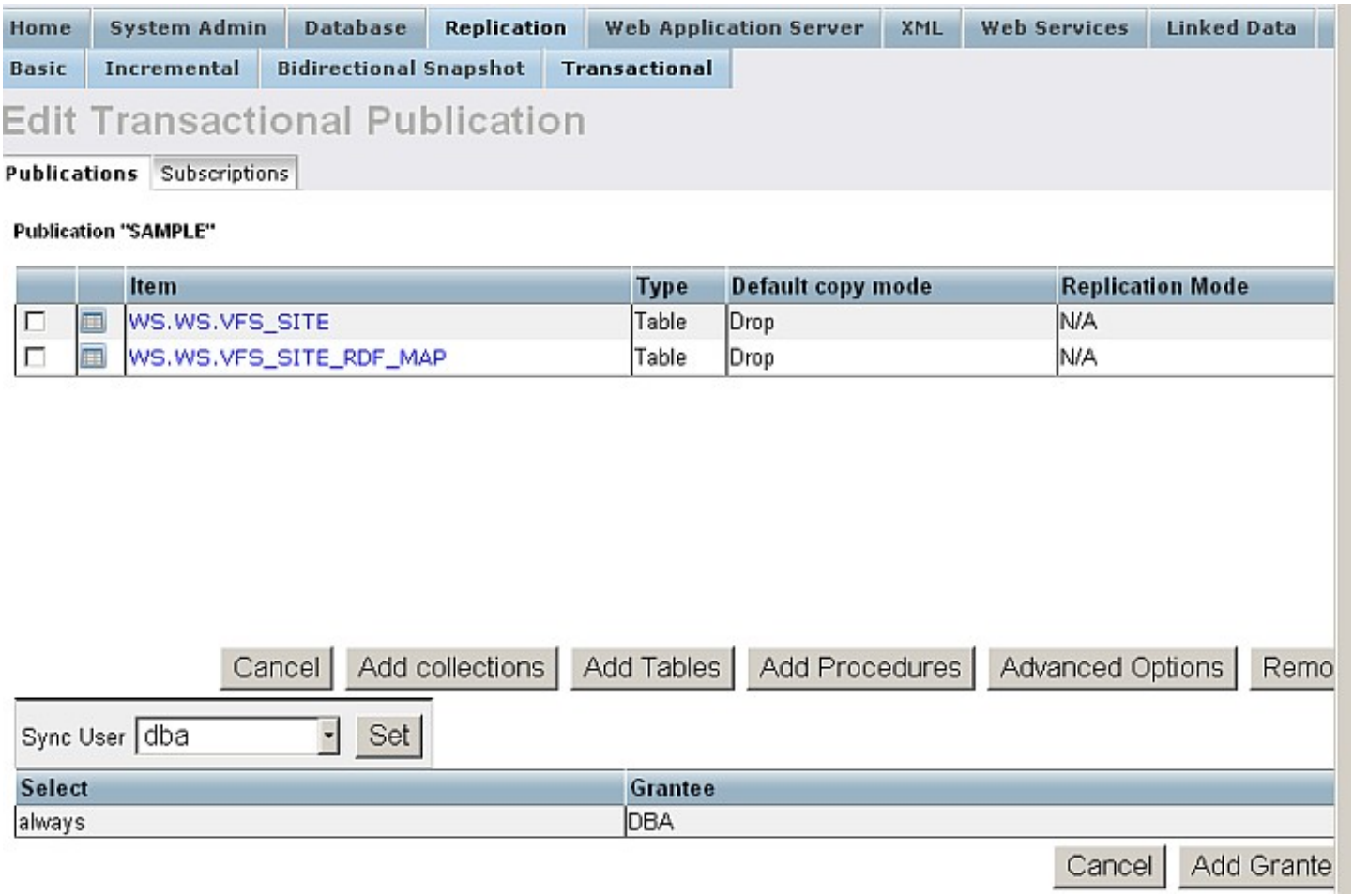
Figure 6.111. Transaction Replication - Publish Tables and Procedures



Select the tables to publish using the checkboxes and press *Add to Publication* to add the tables and continue.

Once returned the publication screen, the published tables will be listed. Updatable publication will need conflict resolvers in case of conflicting data arriving from a subscriber. Click on the table name to manage the resolvers list.

Figure 6.112. Transaction Replication - Published Items



Use the *Add /Remove* to add or remove selected resolvers.

Click on Publication Item -> *New Resolver* to add a new resolver. You have the following details to contend with:

- *Name Suffix*

  Conflict resolver name suffix.
- *Order*

  Resolver order number.
- *Type*

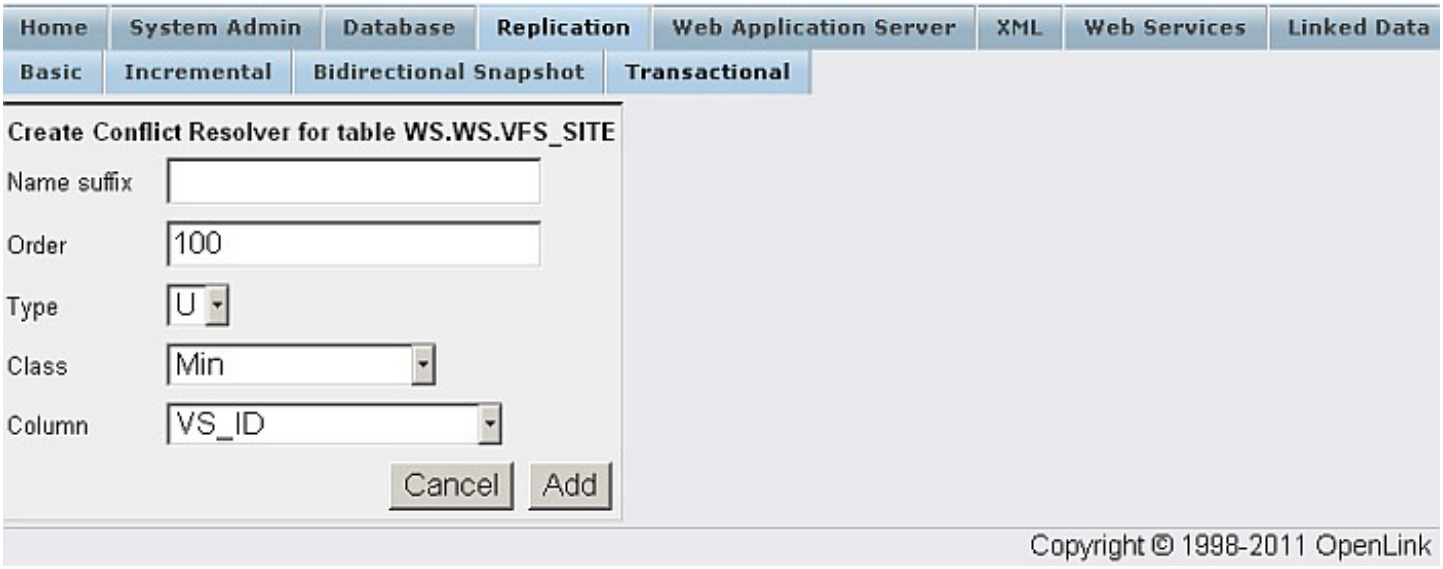
  The resolver type, one of: Insert, Update, or Delete for resolving such SQL queries.
- *Class*

  The kind of resolver. Class can be one of the following:

  'max' row with maximum value of specified column wins
  'min' row with minimum value of specified column wins
  'ave' new value of specified column is calculated as: `current_val = (current_val + new_val) / 2`
  'add' new value of specified column is calculated as: `current_val = current_val + (new_val - old_val)`
  'pub_wins'
  'custom' publisher always wins
  'sub_wins' subscriber always wins
- *Column*

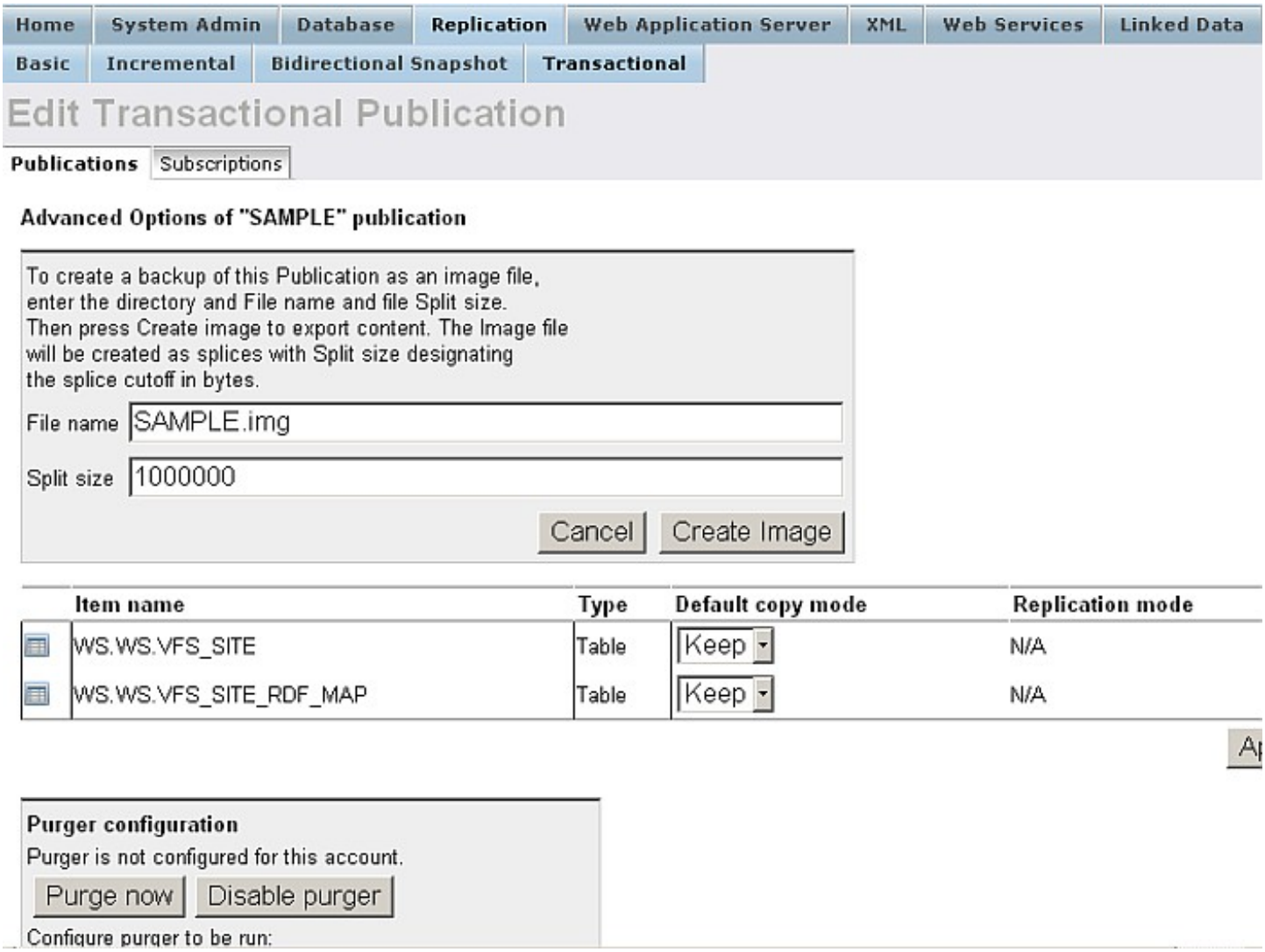
  This should specify the column name if `class` is not one of 'pub_wins', 'sub_wins' or 'custom'.

Figure 6.113. Transaction Replication - Resolvers



Click on Publication -> *Advanced Options* to set additional settings such as split size, default copy mode of publication item, etc.

Figure 6.114. Transaction Replication - Advanced

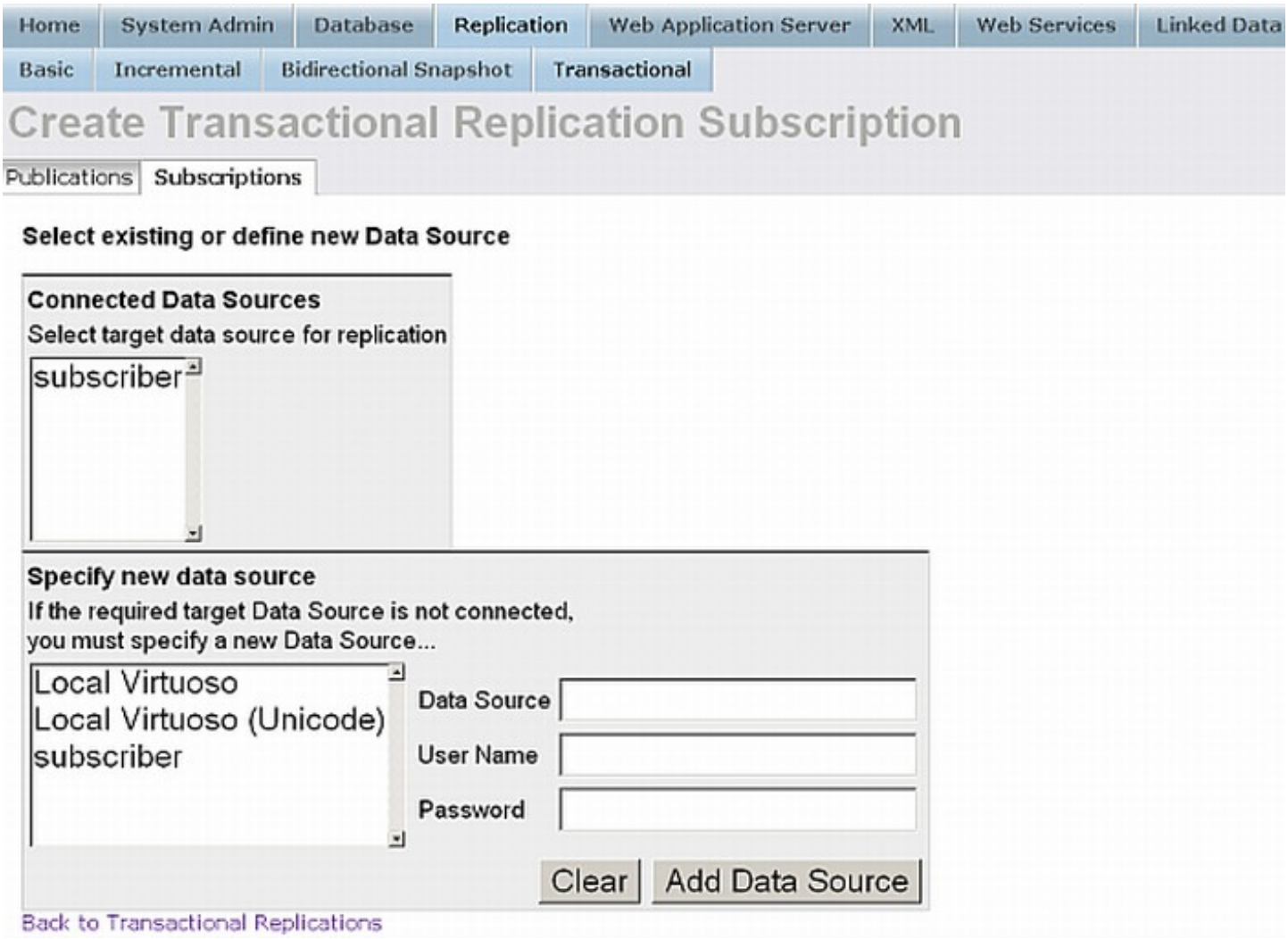


Subscriptions

- To add new subscription click at New Subscription and follow the wizard.
- Click Edit to change properties.
- To drop subscription click at link Drop.
- To synchronize subscription click the Sync button.
- To disconnect all subscriptions click at Disconnect all button.
- To load image file click at Load image button and follow the wizard.

Figure 6.115. Transaction Replication - Subscription

Figure 6.116. Transaction Replication - Subscription



6.2.9. Database Administration

Users & Group Accounts

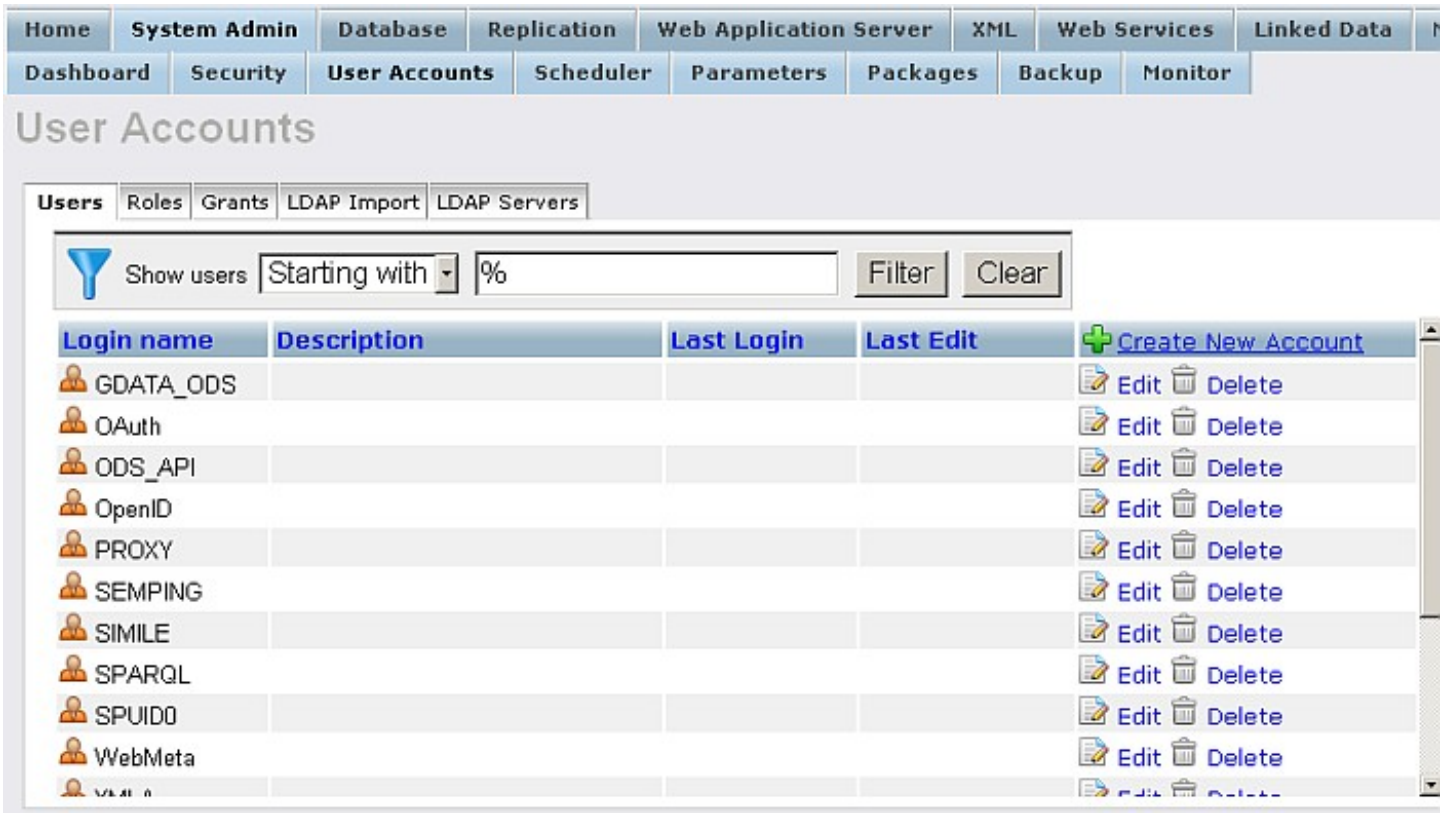
From System Admin / user Accounts you can alter the users that can access the Virtuoso database. You may add a new user by clicking the "Create New Account" link, edit an existing user, or delete users. You can associate users with groups or make new groups.

Groups are created and function like normal user accounts. To define a group you simply create a new user, and then to make use of the group you assign users to that group from the drop down which will contain available groups.

To create a new user or group you must enter a username, password and confirm the password by retyping it in the fields provided. You may optionally specify a group and default qualifier. You can use groups to control access to a set of users by maintaining the permissions granted to the group for which they are members. Specifying a default qualifier allows you to specify the catalog or database for which queries will be run against that do not explicitly refer to a particular catalog or database.

You can import users by adding LDAP Servers(s) from the "LDAP Servers" tab, and then from the "LDAP Import" to specify from which LDAP server the import should be done:

Figure 6.117. Users Accounts

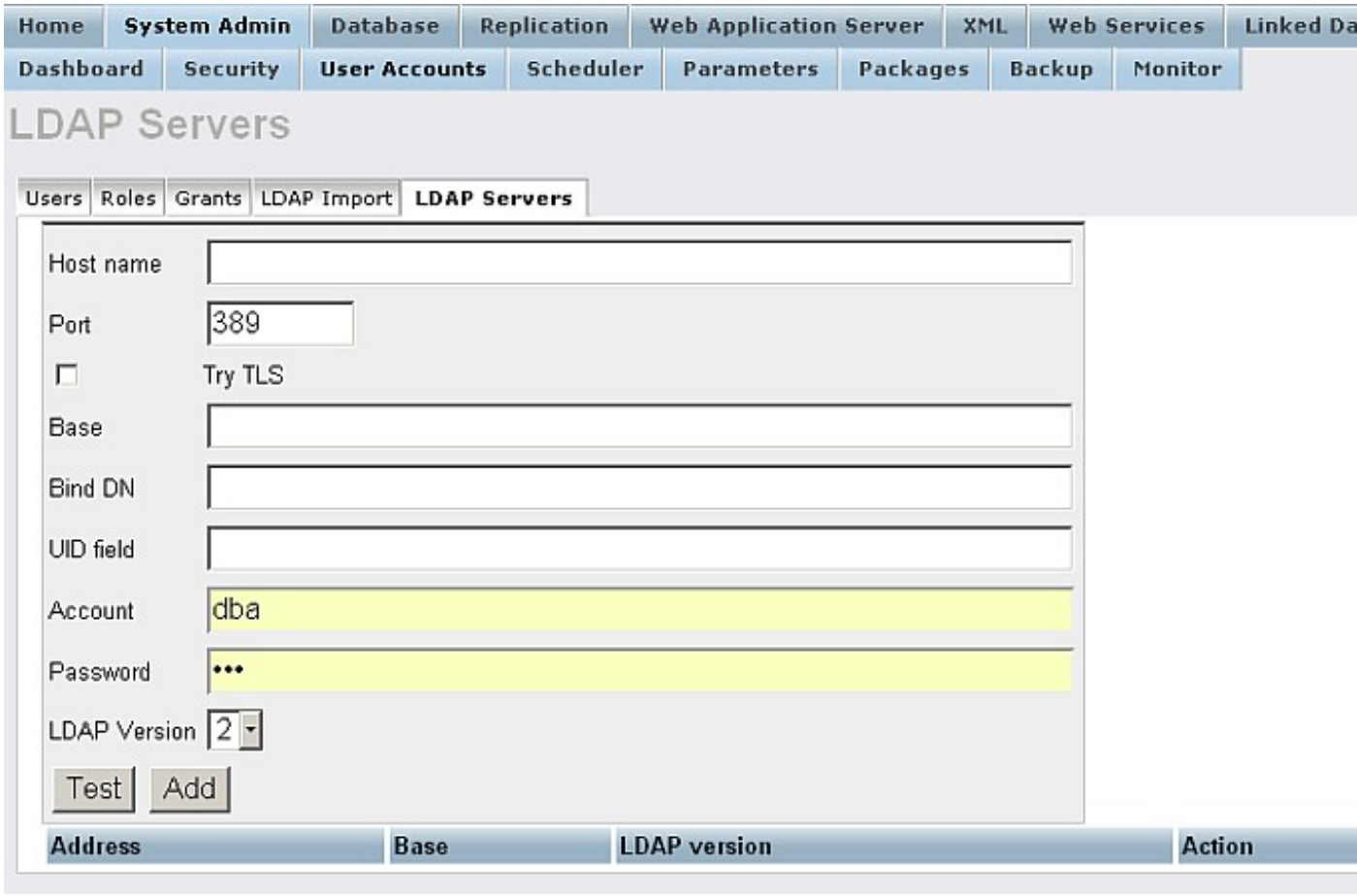


Configuring and import from LDAP server

The following steps describe how to configure LDAP server and then import user(s) using the Conducor UI:

1. Go to http://cname:port/conductor
2. Enter user dba credentials
3. Go to System Admin -> User Accounts -> LDAP Servers

Figure 6.118. LDAP Servers Configure and Import

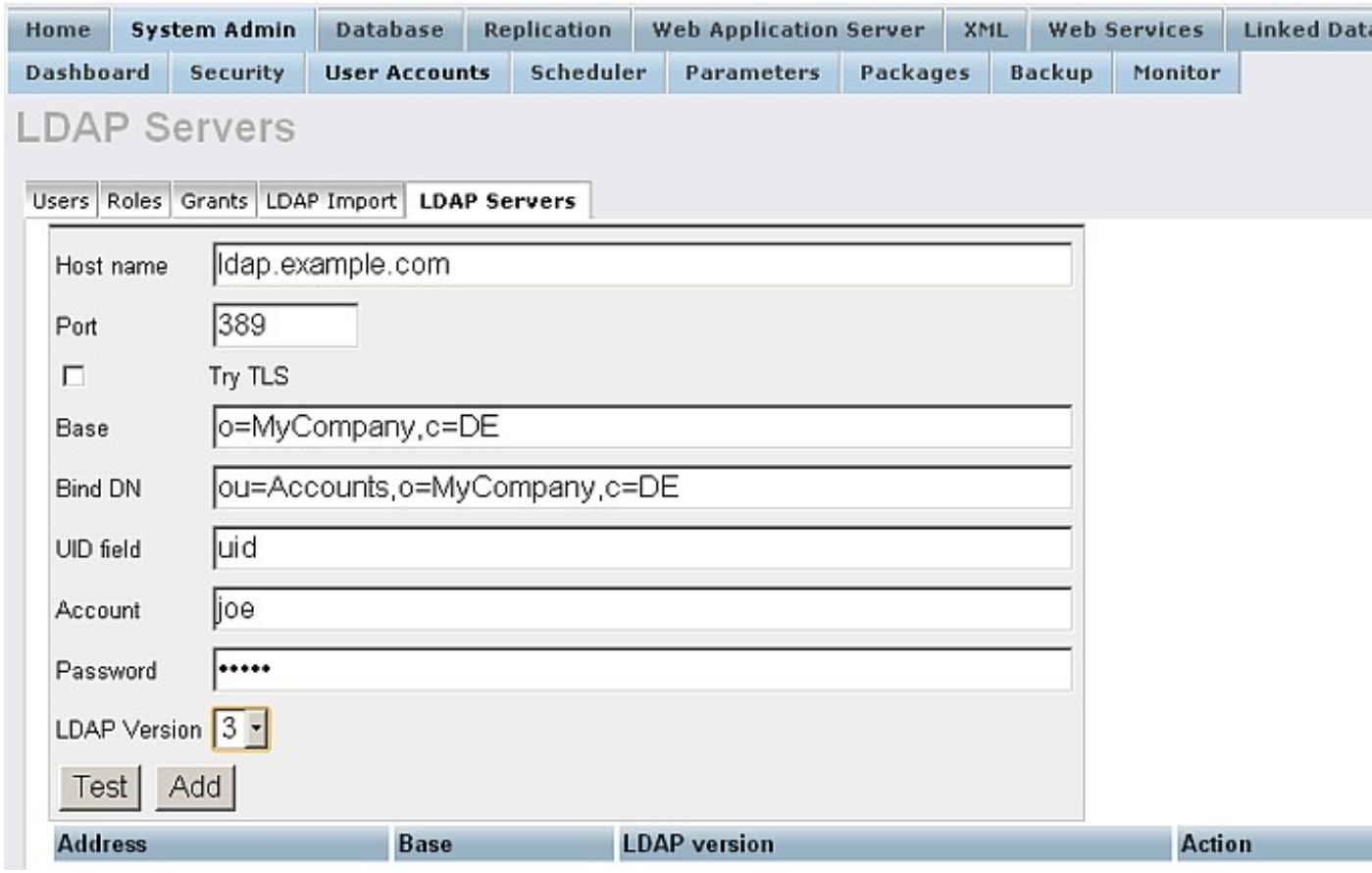


4. Enter in the LDAP servers form respectively:

    a. Host Name: for ex.: ldap.example.com

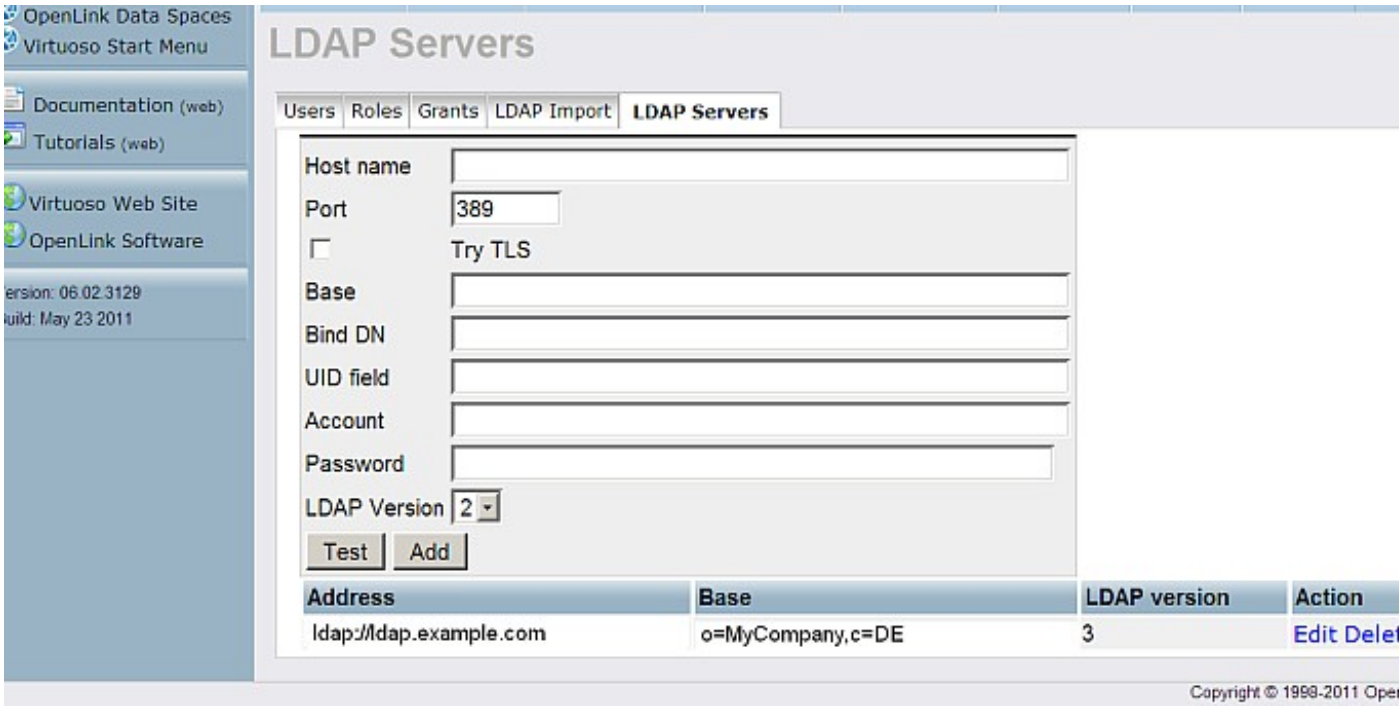
    Note: you can also use ldap:// or ldaps:// in front of Host name
    b. Base: for ex.: o=MyCompany,c=DE
    c. Base DN: for ex: ou=Accounts,o=MyCompany,c=DE
    d. UID field: uid
    e. Account: for ex: joe
    f. Password: *******
    g. LDAP version: choose for ex. 3
5. Check the connection by pressing "Test":

Figure 6.119. LDAP Servers Configure and Import



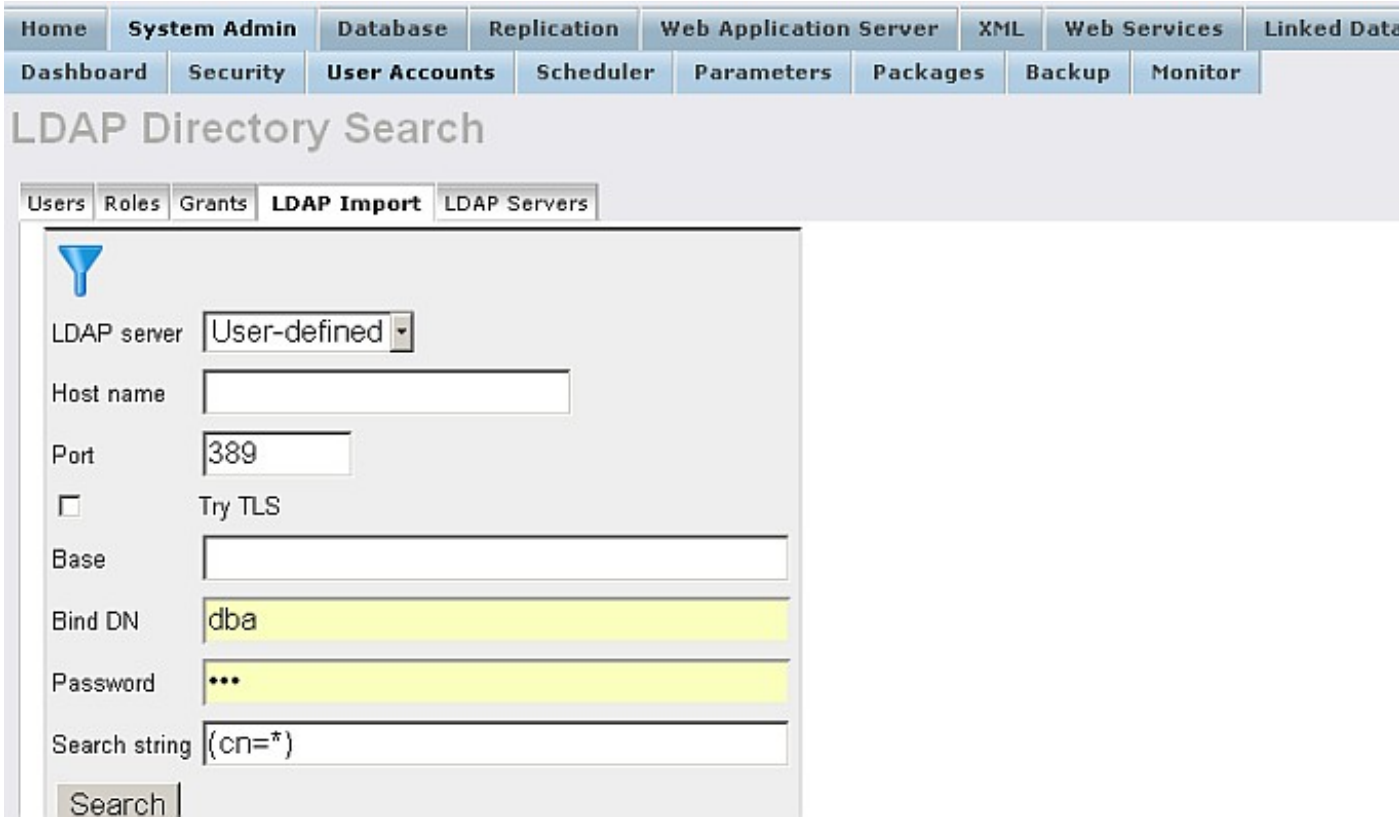
6. If the test connection is successful, click "Add".

Figure 6.120. LDAP Servers Configure and Import



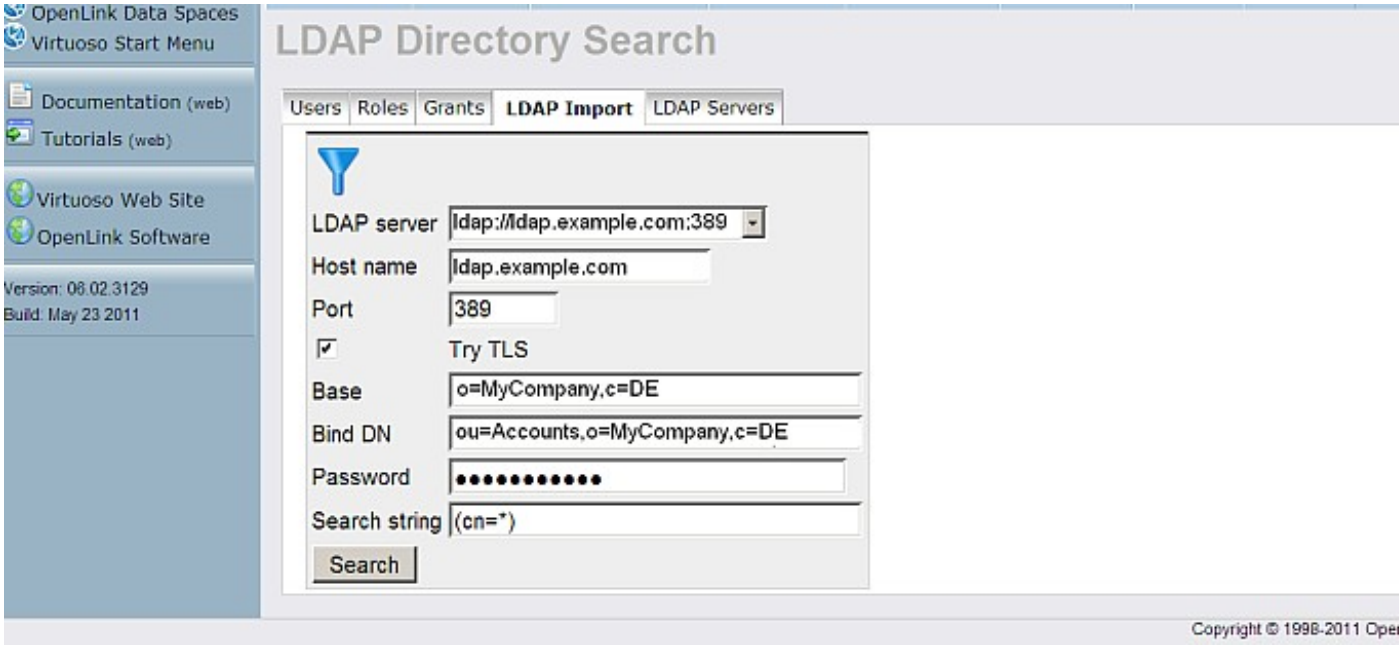
7. Next you can import various data from your LDAP server.
8. Go to System Admin -> User Accounts -> LDAP Import

Figure 6.121. LDAP Servers Configure and Import



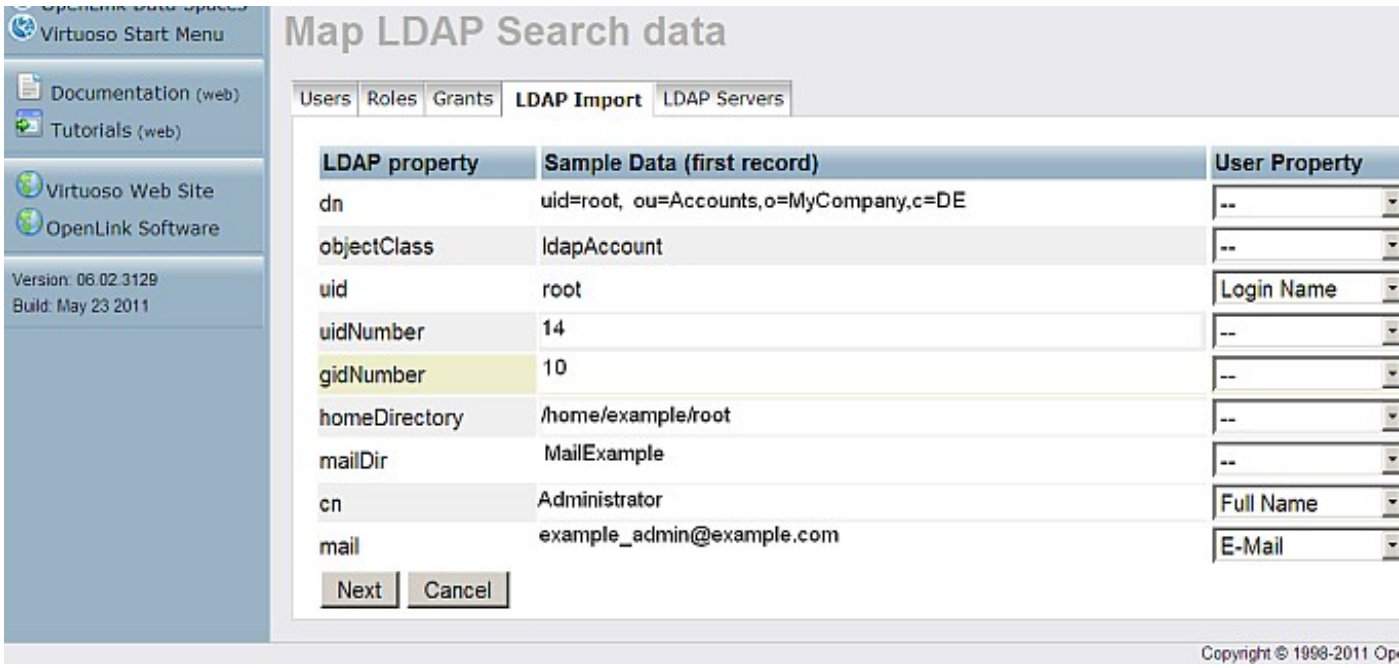
9. Select the LDAP server added from above:

Figure 6.122. LDAP Servers Configure and Import



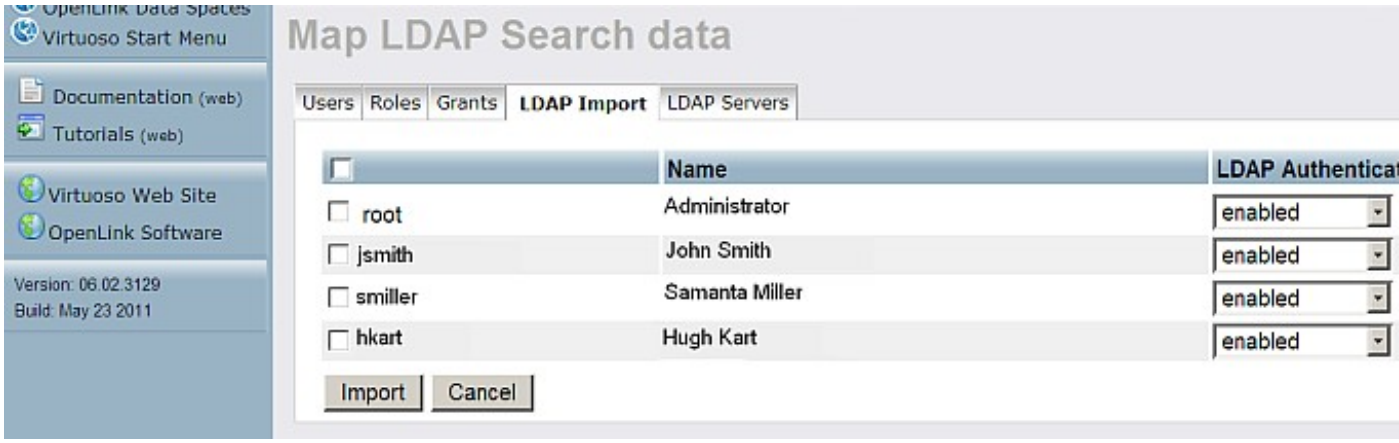
10. Click "Search".

Figure 6.123. LDAP Servers Configure and Import



11. Click "Next".

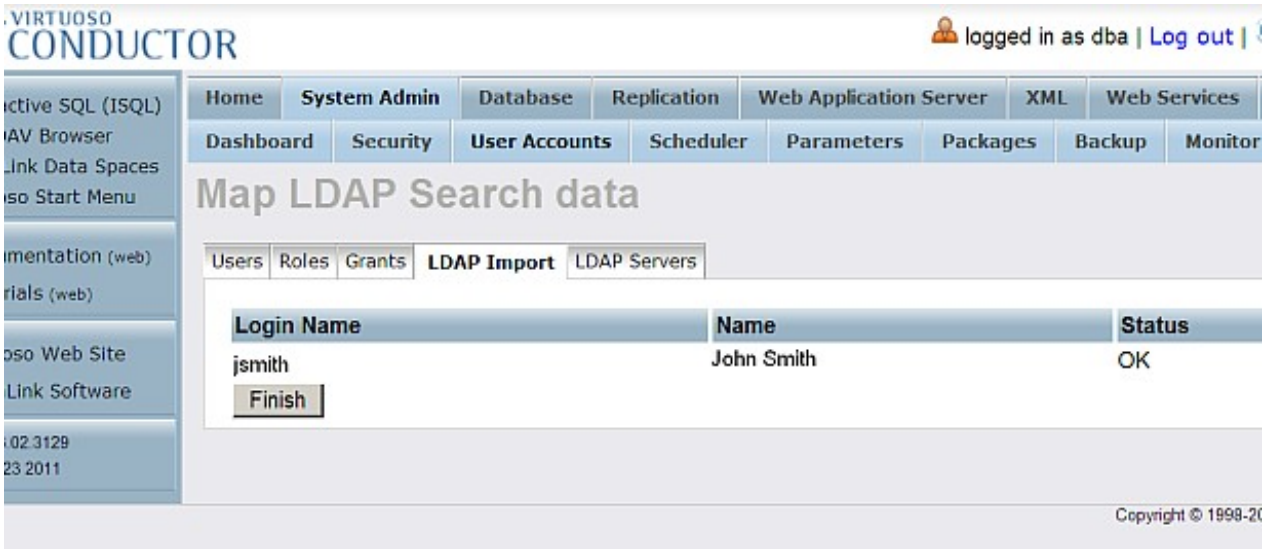
Figure 6.124. LDAP Servers Configure and Import



12. Select the desired profiles to be imported and set prefered "LDAP Authentication" for selected user:

    - If set to "enabled", then on an attempt to log in into Virtuoso, the system tries to log in into the LDAP server with the given username and password. If this succeeds, then the user logs in successfully in Virtuoso.
    - If set to "account check", then on an attempt to log in into Virtuoso, the system performs a check if there is such an user on the LDAP server. If yes, then the user logs in successfully in Virtuoso.

Figure 6.125. LDAP Servers Configure and Import



13. Click "Import".

    Note: By default an importing LDAP user has "User type" set to "WebDAV". To log in into iSQL with an imported LDAP user credentials you should:

    a. Make sure you have set in Virtuoso ini the SQL_ENCRYPTION_ON_PASSWORD parameter to 1 in the [Client] section, i.e.:

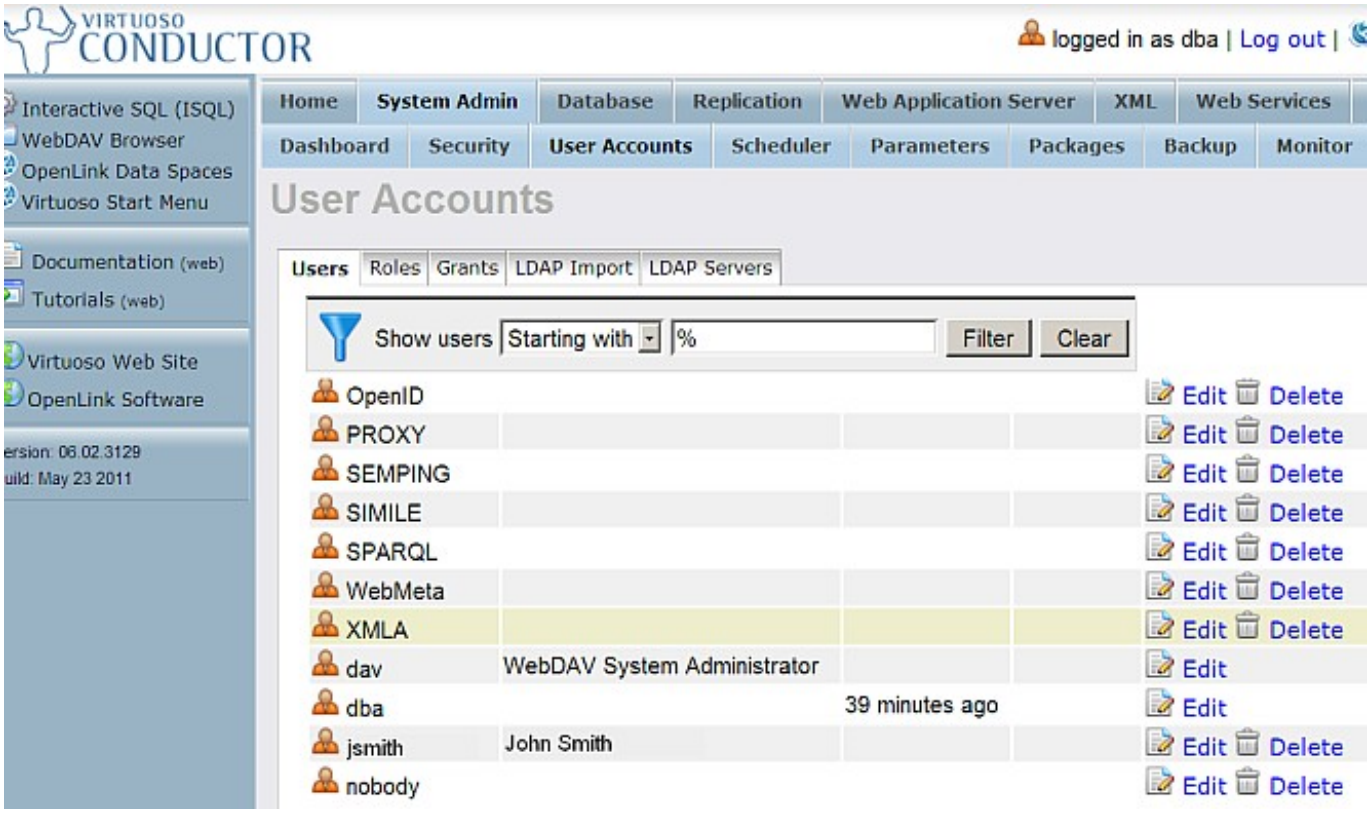
    ```
    ;virtuoso.ini
    ....
    [Client]
    SQL_ENCRYPTION_ON_PASSWORD  = 1
    ...
    ```

    b. Set the "User type" of the imported LDAP user to be "SQL/ODBC" by going to Virtuoso Conductor -> System Admin -> User Accounts -> user -> Edit -> field "User type".
    c. Log in into iSQL:

    ```
    isql host:port ldap-user-name ldap-user-password
    ```

14. Click "Finish".
15. The imported profile(s) should be displayed in the Users tab:

Figure 6.126. LDAP Servers Configure and Import



Databases

Each catalog (database) within Virtuoso will be listed under the Databases /Schema Objects. For each catalog you will be able to view and in some cases edit details about the tables, views, triggers, and store procedures stored within that catalog.

Figure 6.127. Viewing Tables details for the Demo catalogue in the Demo Database

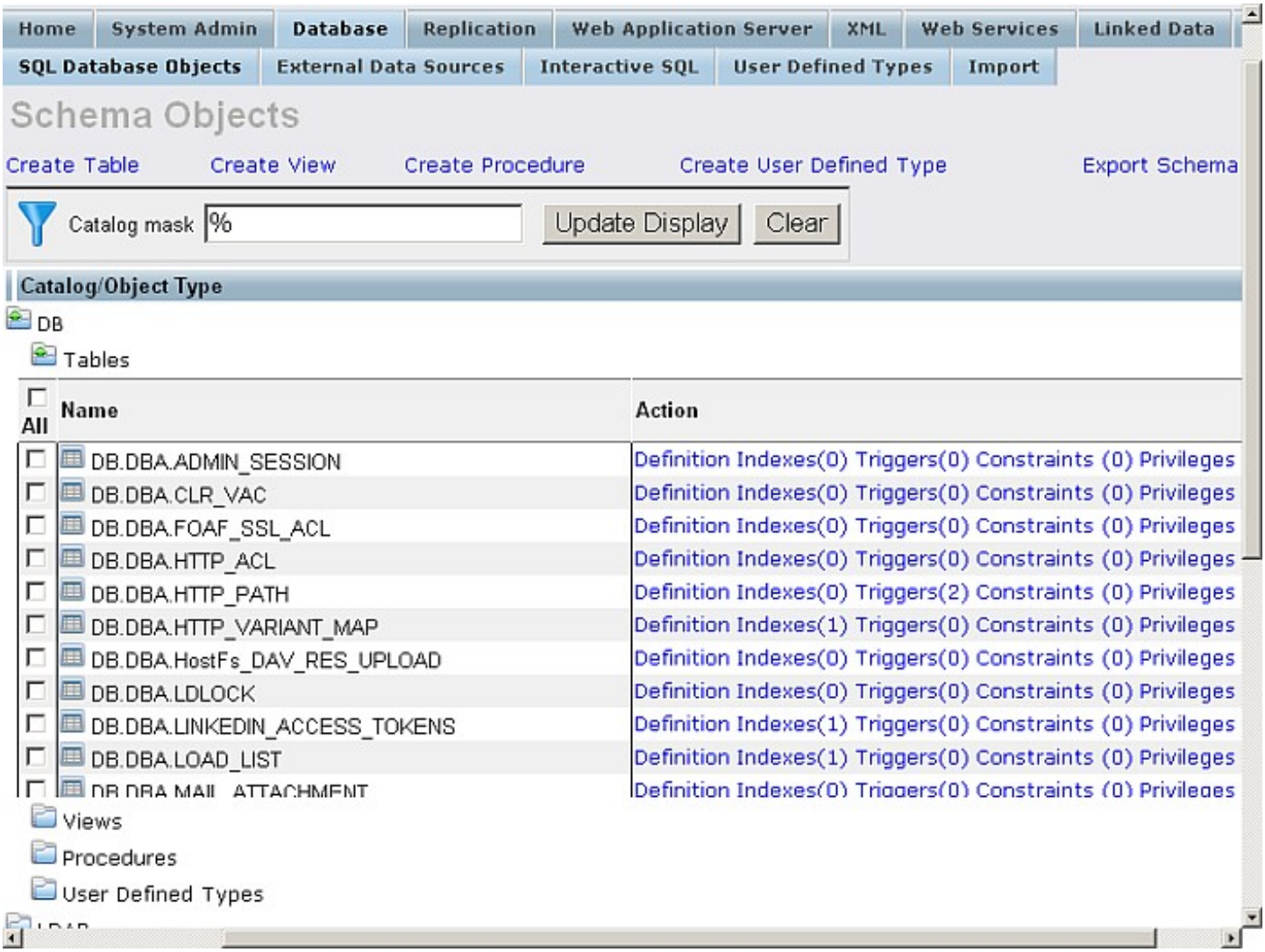
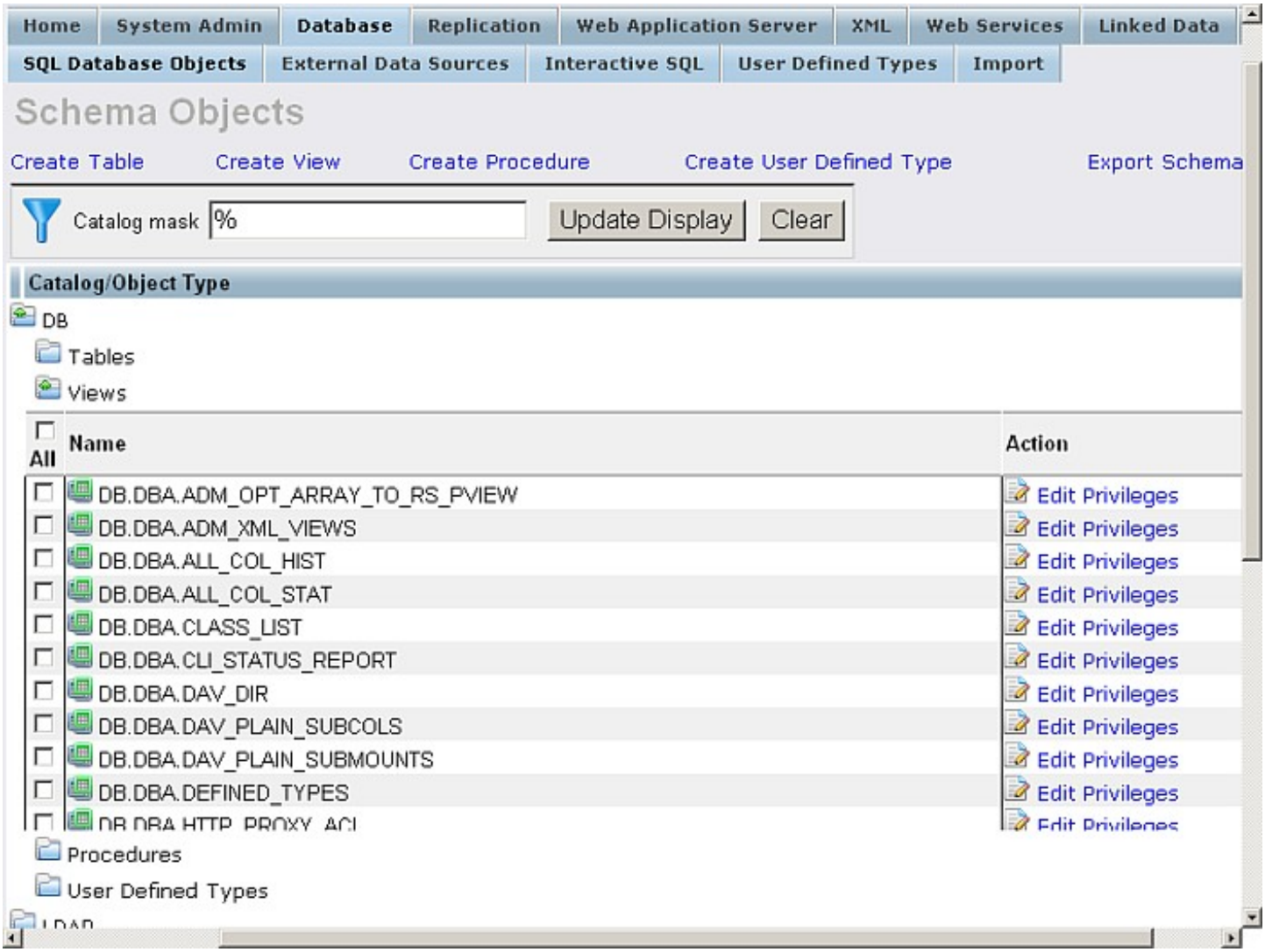


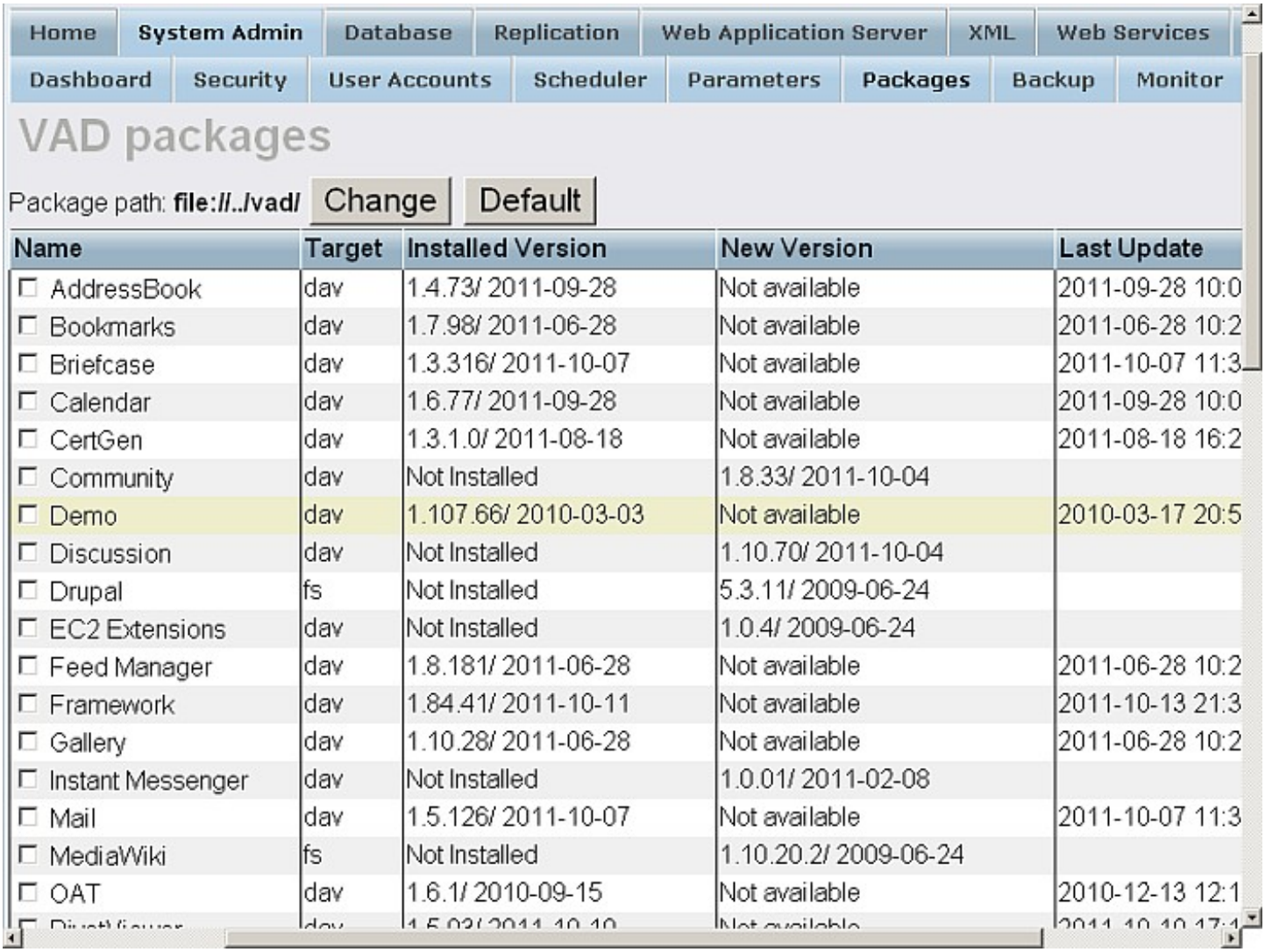
Figure 6.128. Viewing Views details for the Demo catalogue in the Demo Database



Demo Database Installation

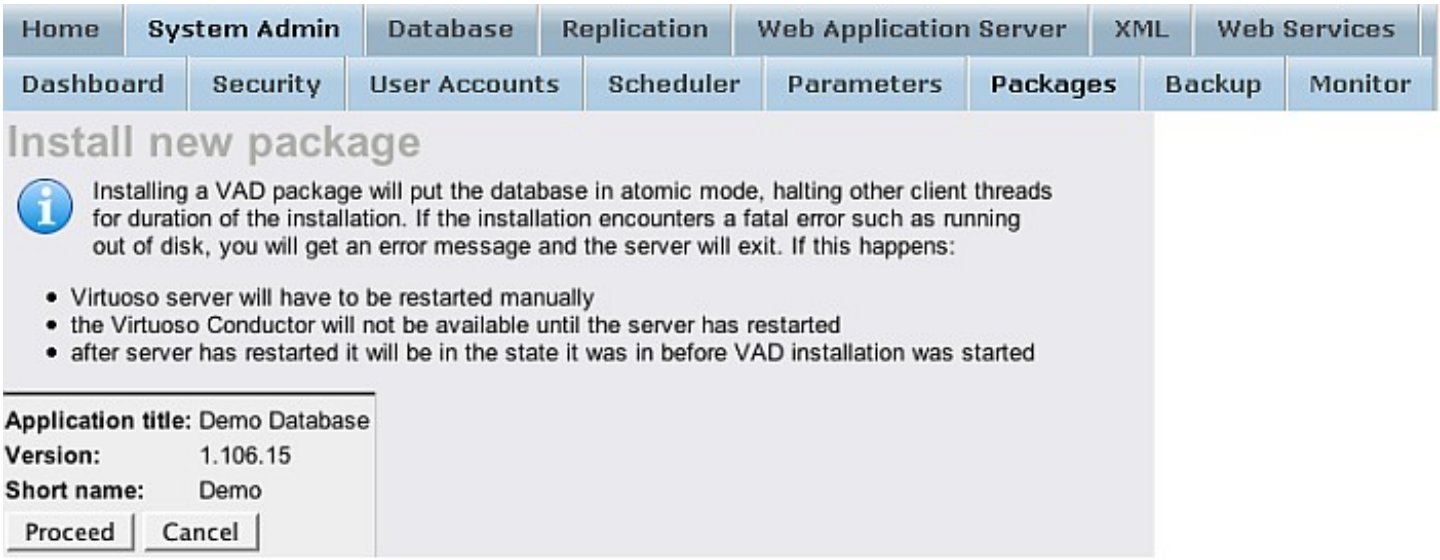
The Virtuoso Demonstration database can be installed with the Conductor by navigating to the "System Admin" -> "Packages" tab where a list of available Virtuoso applications are displayed, one of which is for the "Demo" database package as shown below:

Figure 6.129. Install Demo db using Conductor UI: "System Admin" -> "Packages"



Select the "Demo" package and click on the "Install" button to commence the installation process:

Figure 6.130. Install Demo db: confirmation

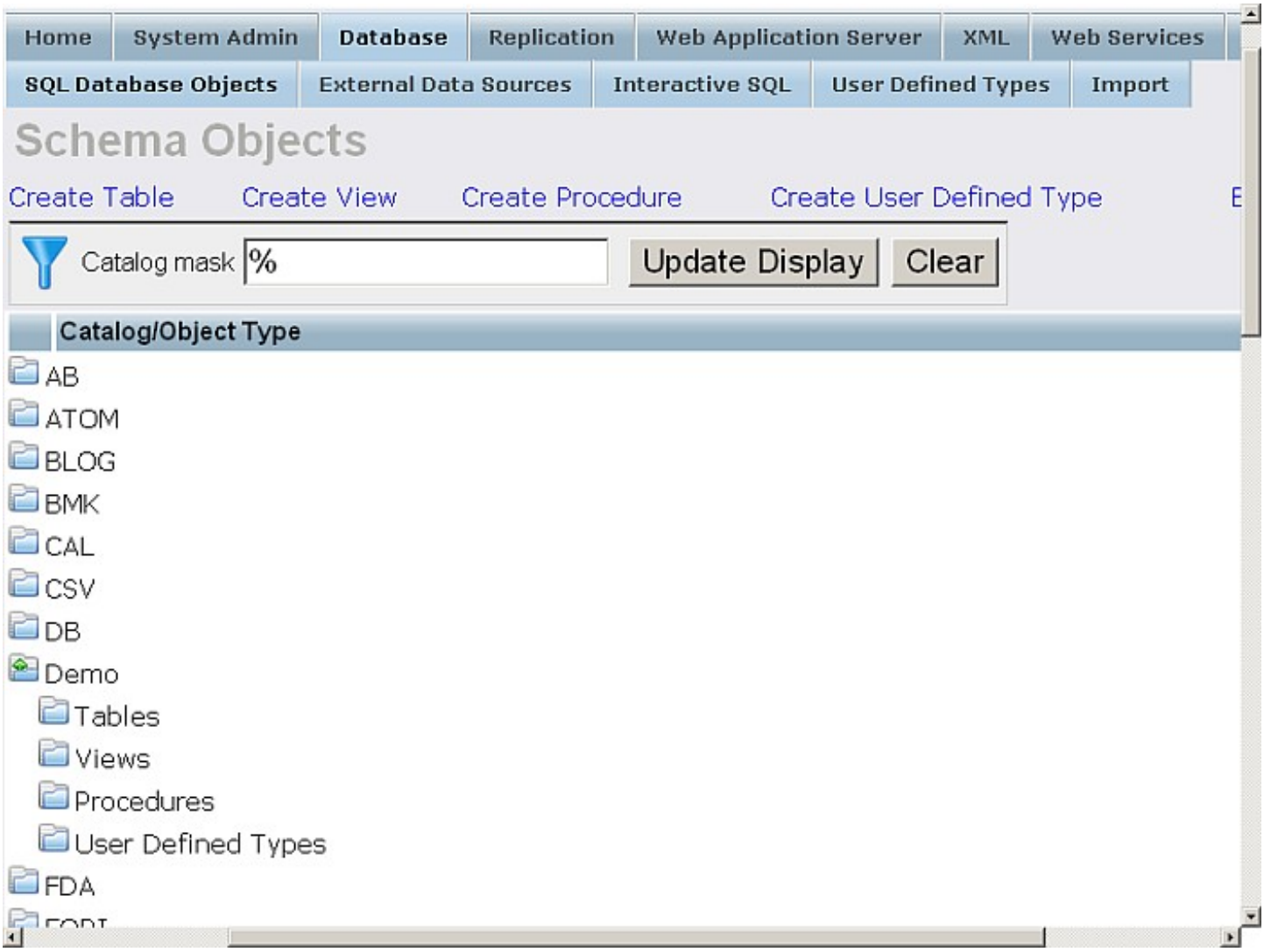


Click on the "proceed" button to install the indicated "Demo Database" application package:

Figure 6.131. Install Demo db: proceed

Once installed the demo database schema can be viewed from the "Database" -> "Schema Objects" tab of the Conductor:

Figure 6.132. Install Demo db: view schema objects



Editing Triggers

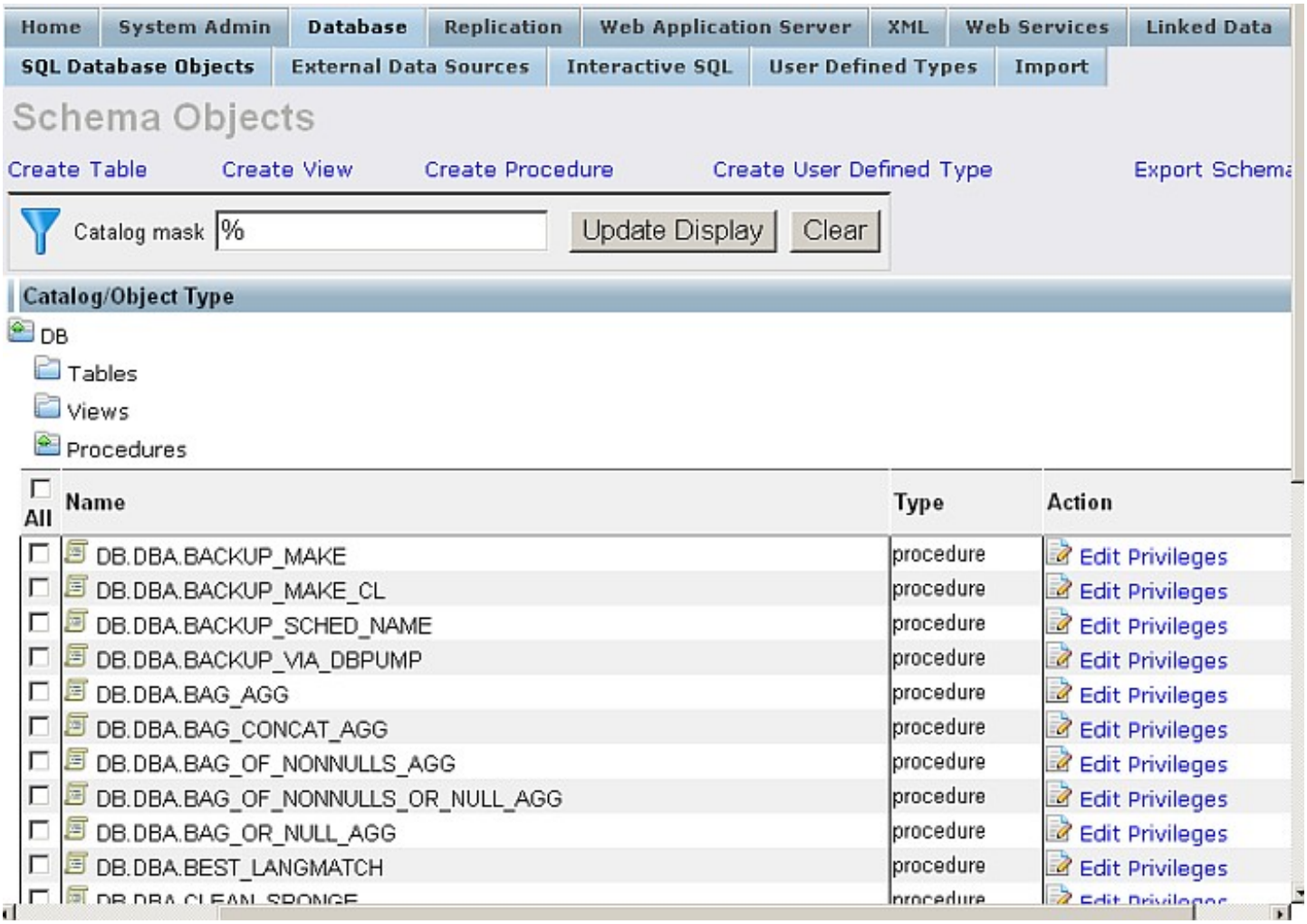
For each table, Virtuoso gives you the ability to edit the triggers linked to that table, as well as adding new triggers. The link text "Triggers" is followed by a number in parentheses that tells you how many triggers exist associated with that table. If no number is shown, there are no triggers. Clicking on the text takes you to the trigger edit page.

The triggers page shows you the name of each trigger, an excerpt from the text of the trigger, and permits you to edit or drop the triggers. In the last column of the shown form there is a "New Trigger" link for creation of a new trigger.

Editing Stored Procedures

The stored procedures link shows a page of information about existing stored procedures for the database you are viewing. If there are stored procedures, you can see their names and text excerpts, and you may edit or drop them. Adding new stored procedures can be done via the ISQL command-line interface or from the "Create Procedure" link shown above the list of objects for the relevant database.

Figure 6.133. Viewing Stored Procedures details for the DB catalogue



Stored procedures may also refer to modules that contain a group of related stored procedures. These are created and edited as a group, rather than individually. An example of this is the module DB.DBA.AmazonSearchService, which contains procedures such as KeywordSearchRequest() and BrowseNodeSearchRequest().

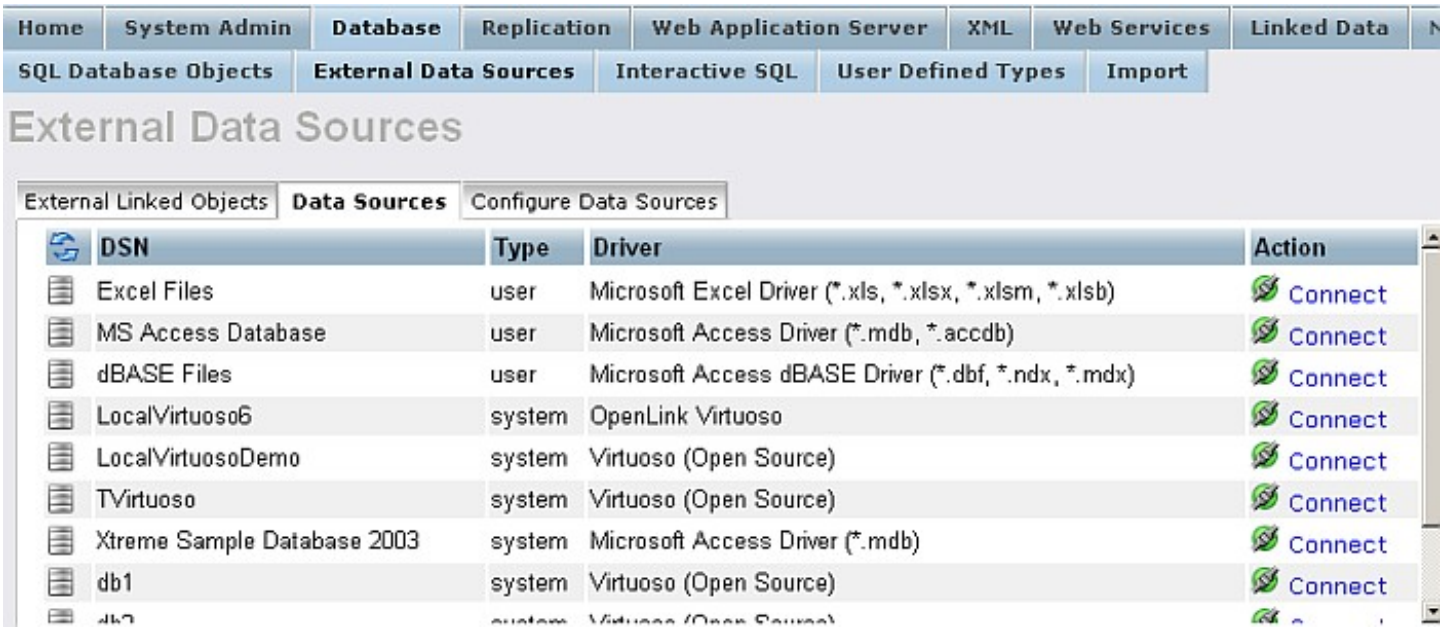
External Data Sources

From Database / External Data Sources you will be able to manage the Virtual Database feature of Virtuoso. You are able to administer ODBC Data Sources, how to link remote tables and view, and already linked remote connections.

Data Sources

This tab shows you the available Data Sources. For these one, to which there is no connection, will be shown the link"Connect". For the datasource for which there is already established connection, will be shown the links "Link objects", "Change Credentials" and "Disconnect".

Figure 6.134. Remote Datasources connected to Virtuoso

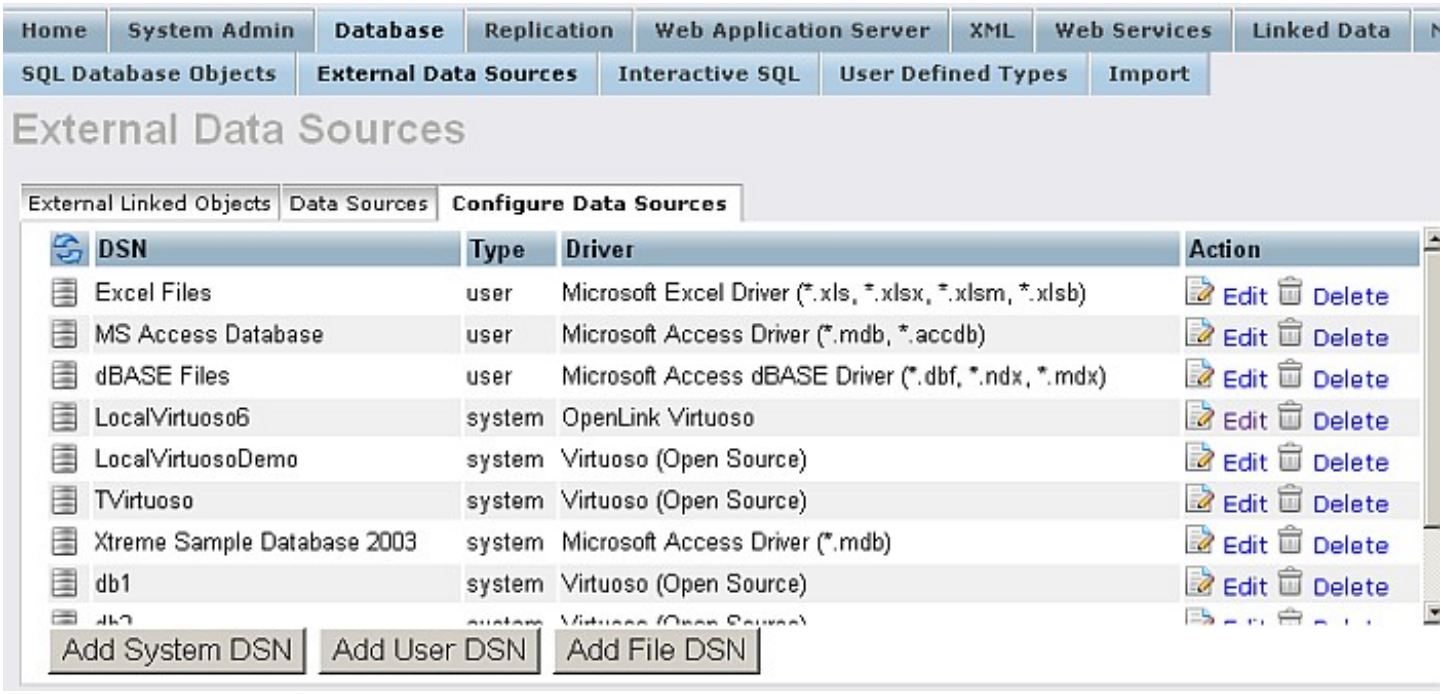


If you need to alter the authentication details of a particular Data Source then select the "Change Credentials" link. If you want to link new objects, select the "Link objects" link.

Configure Data Sources

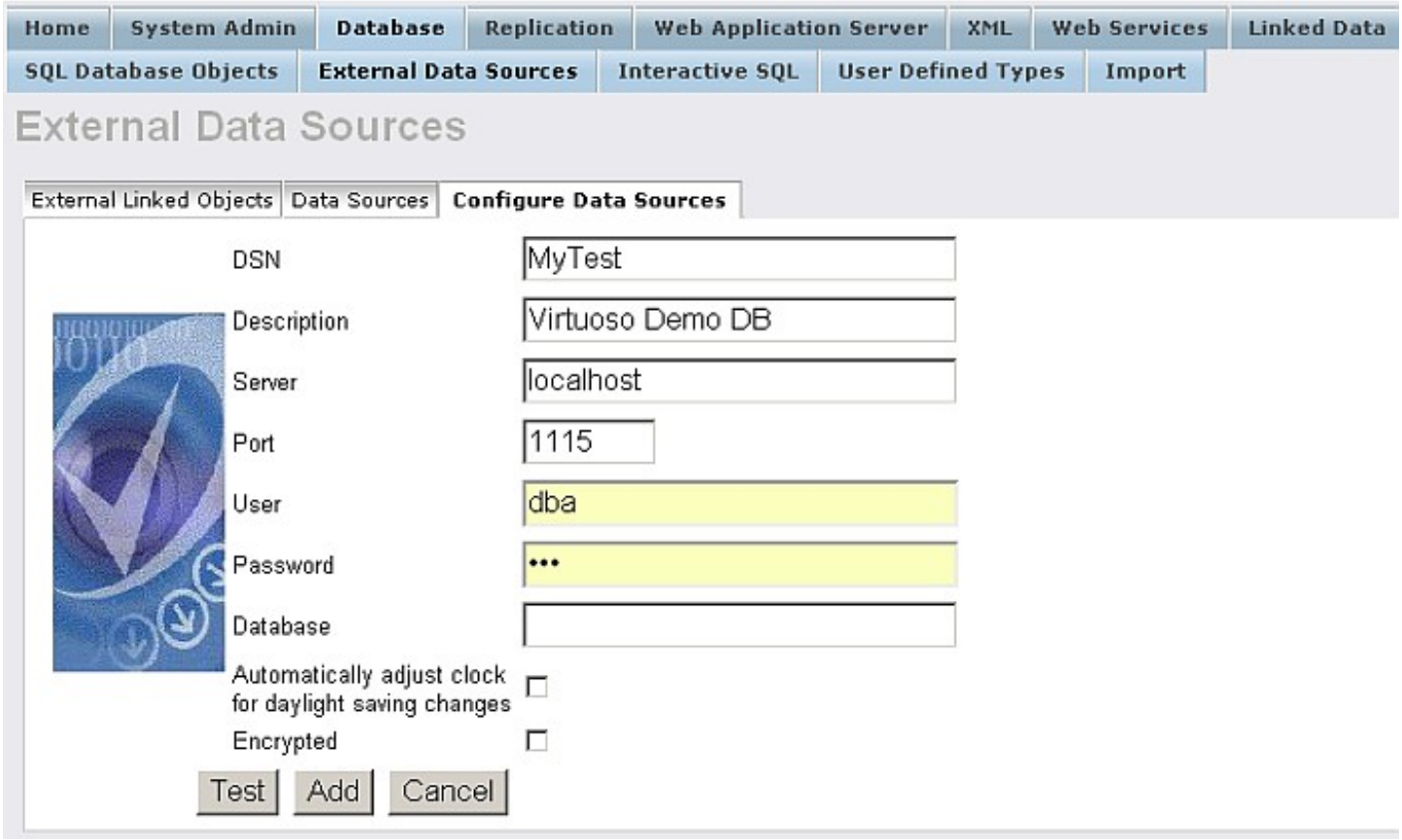
This section will allow you to configure the data sources themselves on the remote machine. The table that will be presented to you will contain both User and System data sources. You will be able to edit and create either User or System data sources, however, Virtuoso will only be able to use the User data sources that belong to the user that started the Virtuoso Server.

Figure 6.135. Configuring ODBC Datasources



Follow the buttons on the screen as to how you want to manage a data source. You can create new, edit or remove existing data sources.

Figure 6.136. Configuring A Virtuoso ODBC Datasources



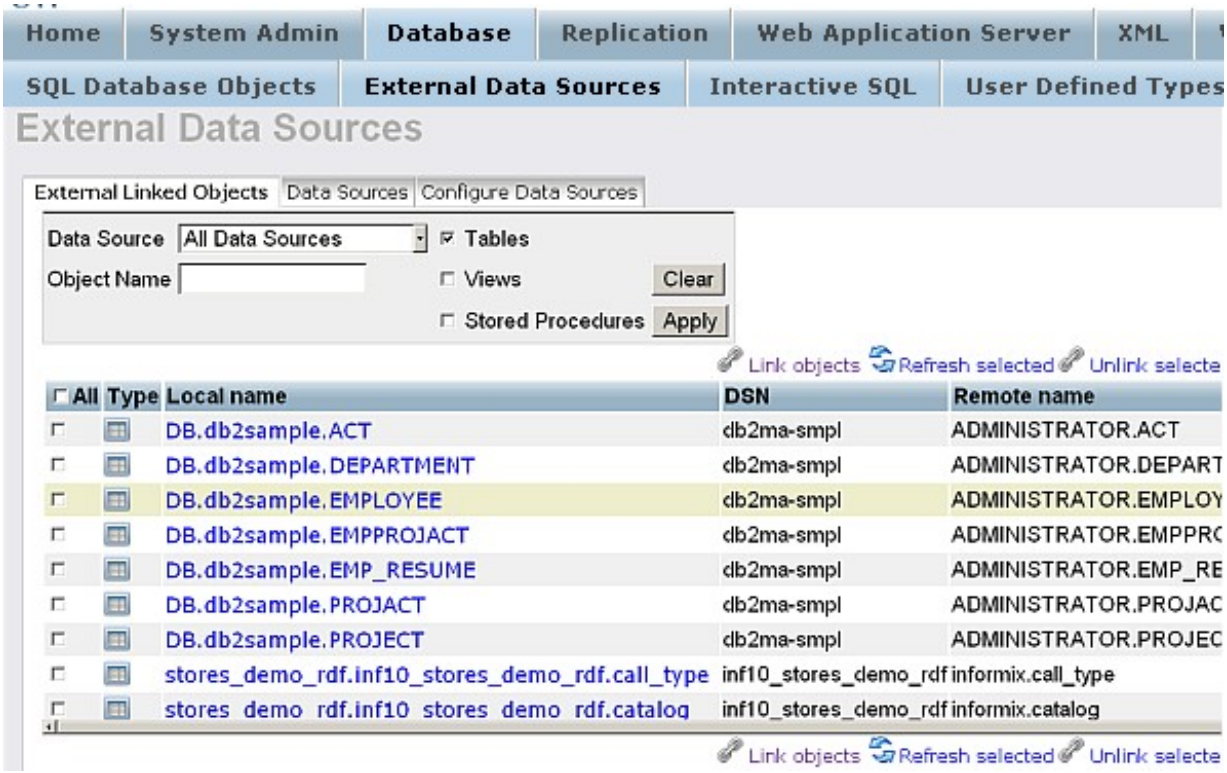
The Virtuoso Server can make use of File Data Sources too. This provides the usual associated conveniences. These enable you to migrate a Virtuoso database to another machine hosting the same ODBC Driver, Virtuoso then has enough information to connect using the installed driver to a remote data source, the tables would not need to be relinked. File DSN's are read from server's root directory. File DSN's can only be read if this directory is contained in the Virtuoso INI file parameter DirsAllows.

Note:
Windows NT or 2000 services by default start as the LocalSystem special user account. This account will not contain the same User DSNs as your own user. The Virtuoso service can be started with other users but you must be aware of any possible system permission problems. It is recommended that Virtuoso make use of System or File DSNs wherever possible.

External Linked Objects

This page will allow you to manage the remote objects that are linked into Virtuoso. You can unlink objects by selecting them and pressing the "Unlink selected" button.

Figure 6.137. Linking Tables from Remote Datasources



To link new tables into Virtuoso select "Link objects" link or go to the "Data Sources" tab. Click the "Connect" link for a data source. If this data source has been used before then the existing authentication details will automatically be provided. Otherwise you will have to provide these details in the username and password fields provided. When you are ready press the "Link objects" link.

As results you should be presented with list of available remote tables and optionally views and procedures. If the connection fails then an error will be returned to the top of the page instead. You should be able to link any of the listed tables or views into Virtuoso. In order for Virtuoso to be able to correctly query the underlying table it needs to be able to uniquely identify each row (record) within the table or view. Tables should always be created with a primary key, and details of which should be available from the data source. With a primary key available Virtuoso will be able to link the table in to the server easily. In some cases and certainly in the case of views Virtuoso may not be able to determine a primary key or what constitutes uniqueness in the table. On the right most column will be an action link to a page that allows you to alter the primary key information about the table or view that is to be linked. Some primary key information is italized to indicate the explicit definition of a primary key.

You will be required to enter primary key information for views before they can be linked.

Select the tables that you want to link into the Virtuoso Server by checking the check box on the left most column of the list for table or view in question. When you are ready press the *Link* ... button at the bottom of the page.

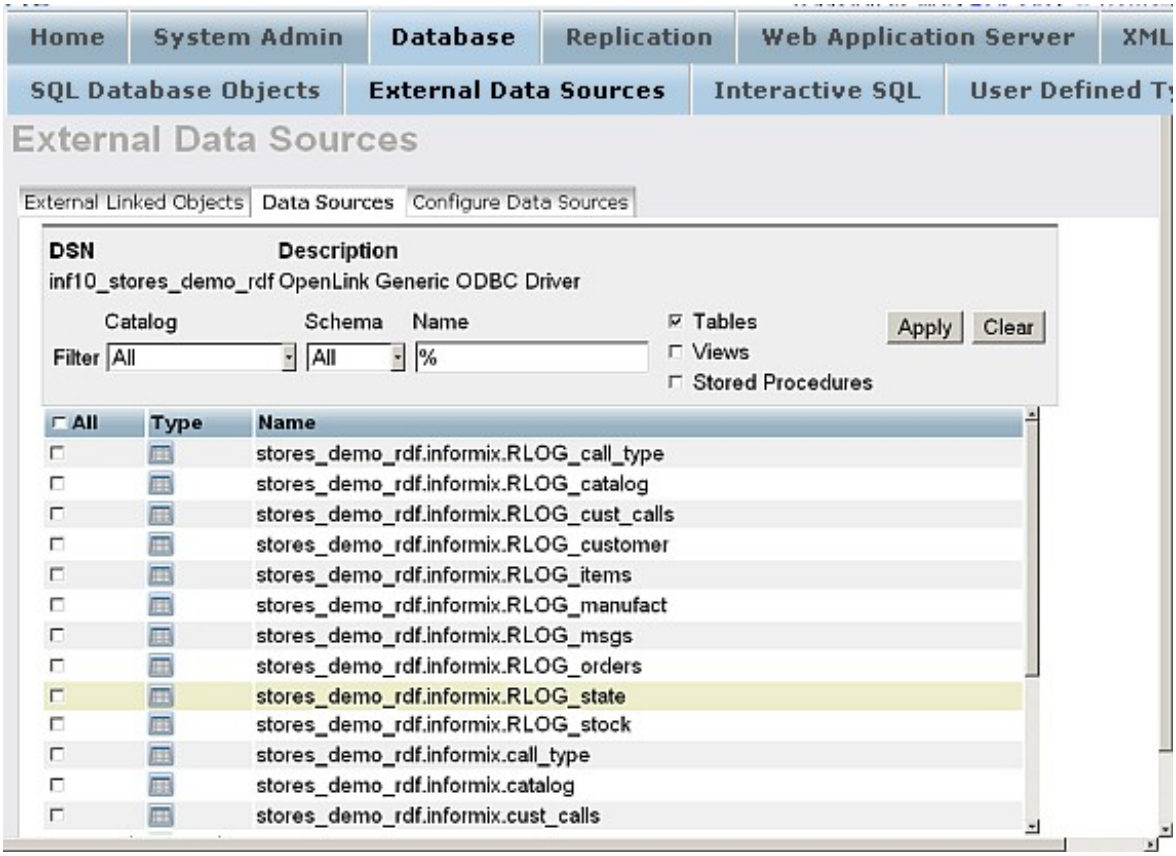
As result the selected tables will be linked and listed in the first list as Currently Linked.

Remote Procedures

In order to view the list of remote procedures, you should check the "Stored Procedures" check-box and click the button "Apply". As result the procedures that are already connected to Virtuoso will be shown with checked check-box. You can unlink these by selecting them and pressing the *Unlink* button.

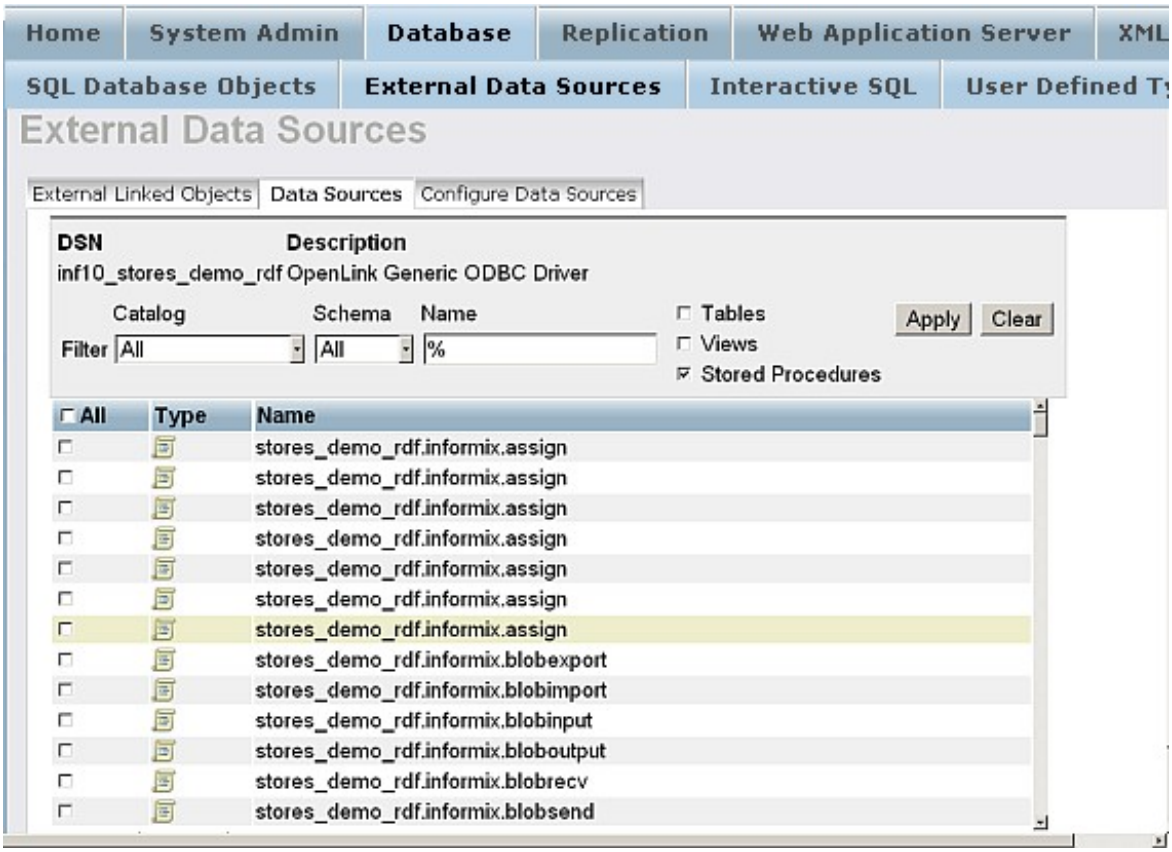
To link new procedures into Virtuoso select the Data Source from the list of available n t ab "Data Sources". If this data source has been used before then the existing authentication details will automatically be provided. Otherwise you will have to provide these details in the username and password fields provided. When you are ready press the *Link Objects* link. The page will next require you to select the catalog/owner combination to refine the search list of procedures on the remote datasource. Check the check-box "Stored procedures" and click the "Apply" button to list the available stored procedures.

Figure 6.138. Linking Procedures from Remote Datasources



When the page returns you will be presented with a list of available remote procedures. You should be able to link any of the listed procedures into Virtuoso.

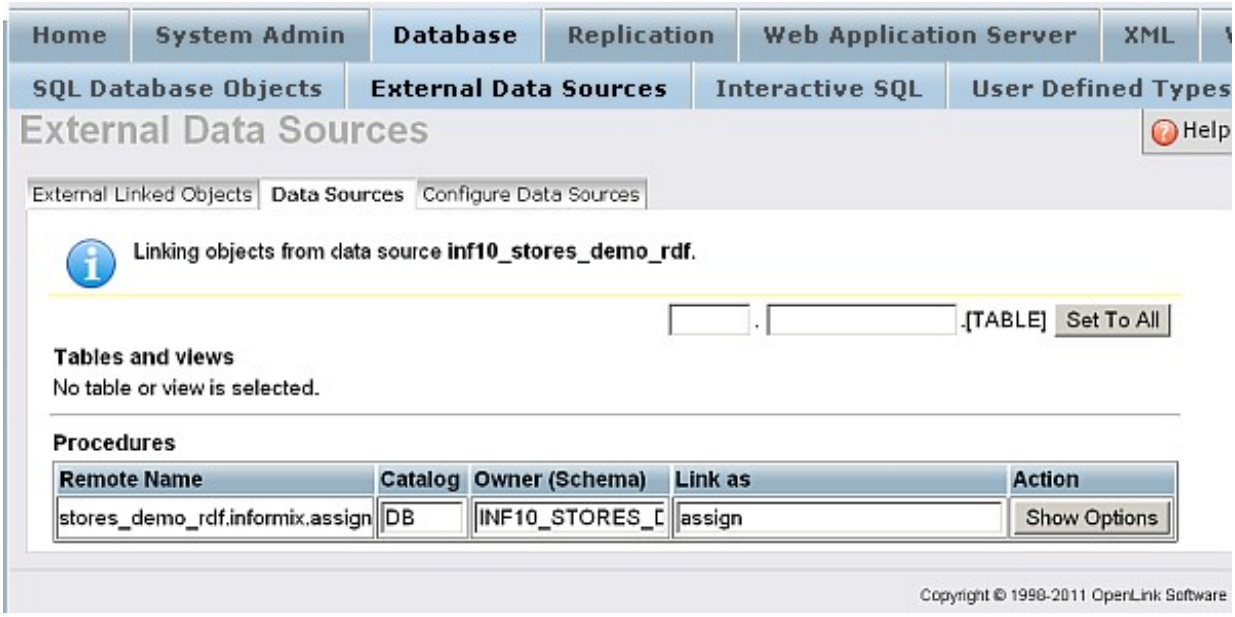
Figure 6.139. Linking Procedures from Remote Datasources



Select the procedures that you want to link into the Virtuoso Server by checking the check box on the left most column of the list for the procedure in question. When you are ready press the *Link ...* button at the bottom of the page.

You will be presented with a new page that lists the procedures chosen and the data type information regarding them. This gives you an opportunity to alter the data type mappings that Virtuoso will use both internally and for any future interactions with the SOAP server. If you do not want to specify any special type information the details can be left as default.

Figure 6.140. Linking Procedures from Remote Datasources



For each remote procedure you have the option to change how they will be referenced within Virtuoso by making changes to the fields for *Local Name* , *Database* , and *Schema* . These fields define how you will find the linked procedure locally to Virtuoso only and do not affect the remote data source.

For each procedure there are radio buttons for selecting the *PL Wrapper Requirements* . This option is of particular importance for remote procedures capable of returning a resultset. Remote procedures can be linked using a Virtuoso PL wrapper meaning that Virtuoso procedure language code provides a mechanism for negotiating the result set. The available options are:

Soap Execution - Generates a PL wrapper that can be published to Web Services.

SQL Execution - Generates a PL Wrapper that is more suitable for general SQL use.

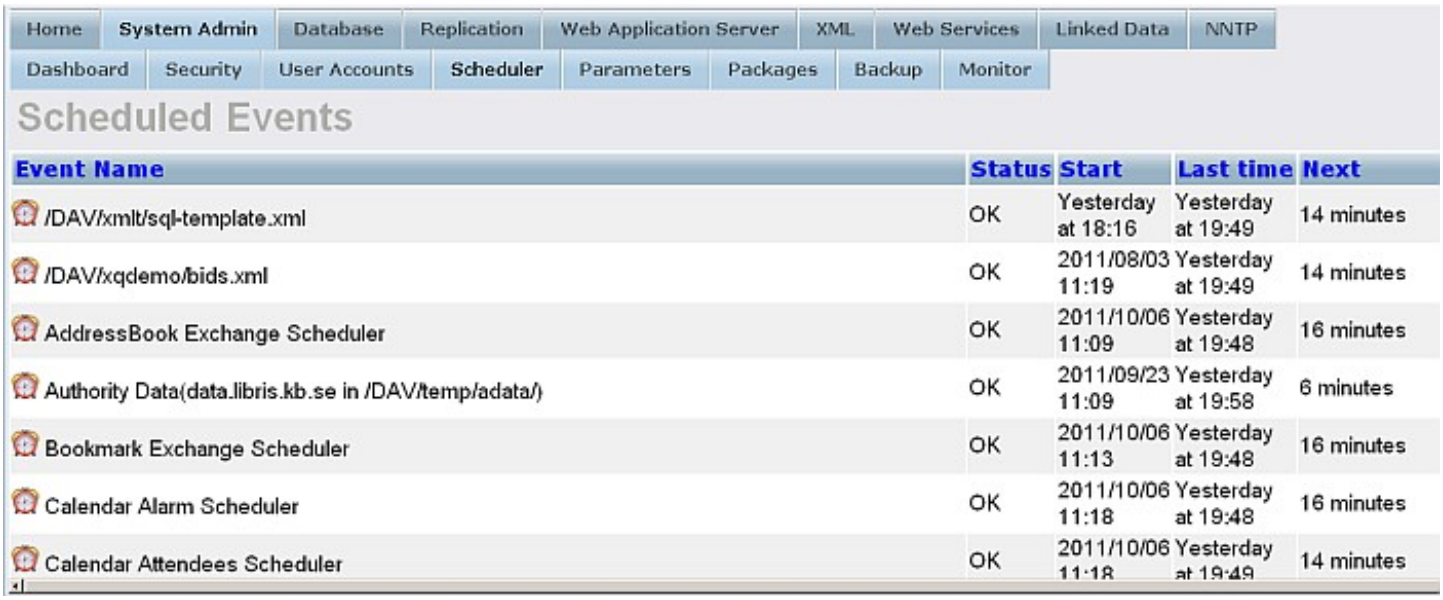
None - Does not generate and PL Wrapper code and simply links the procedure by reference.

There is a description input box for you to write a description or comment for the linked procedure. This comment will only be applicable to procedures that are linked using a PL Wrapper, and is only directly applicable for PL Wrappers for SOAP execution.

Event Scheduler

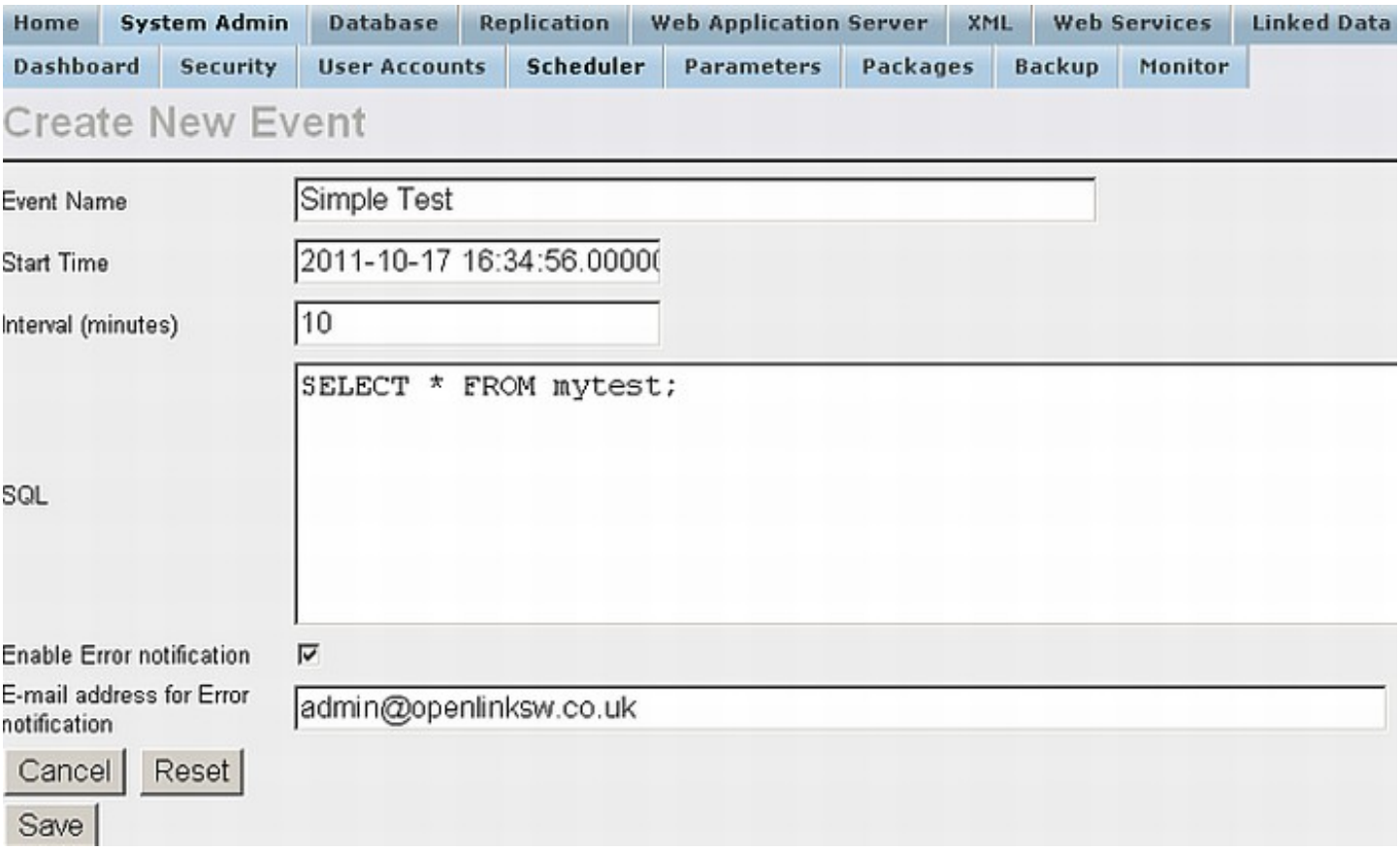
The System Admin / Scheduler tab allows you to view, edit and remove events that can be scheduled to run from a particular time at a defined interval.

Figure 6.141. Event Scheduler



To add a new scheduled event click the "Create New Event" link in tha last column of the shown form. The Event Name is a unique name required to uniquely identify each event. Start Time is when the event will be run for the first time and takes the form of a normal SQL timestamp value. Interval(minutes) is the length of time in minutes to wait to run the event again. SQL is a valid piece of SQL that you want to schedule to run. You can also check for "Enable Error Notification" and enter E-Mail for error notification.

Figure 6.142. Event Scheduler



Events write possible error messages into the Virtuoso log file.

Virtuoso Configuration Editor

From System Admin/Parameters you can change the Virtuoso Configuration settings:

Figure 6.143. Virtuoso (virtuoso.ini) Configuration File Editor

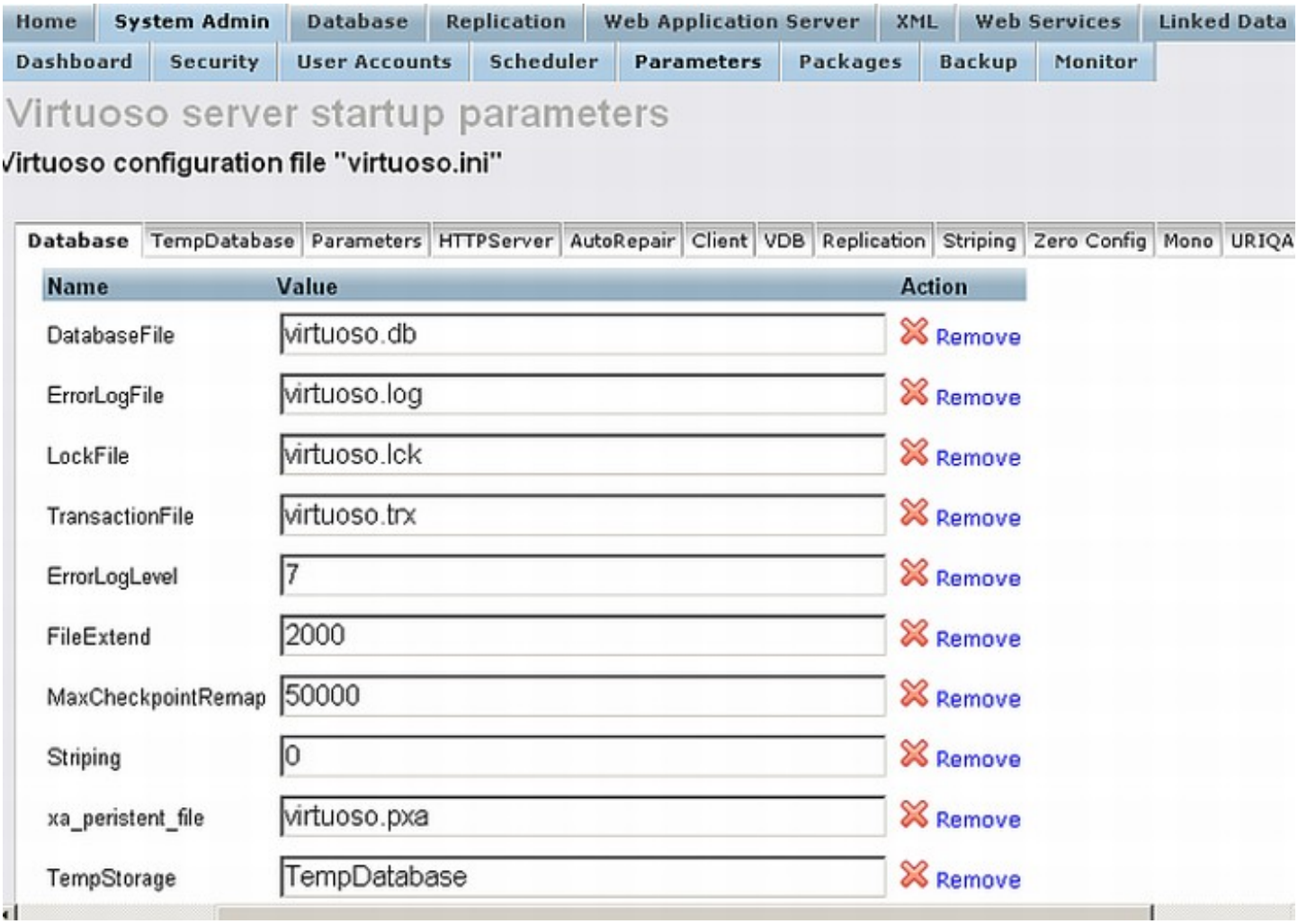


Figure 6.144. Virtuoso (virtuoso.ini) Configuration File Editor

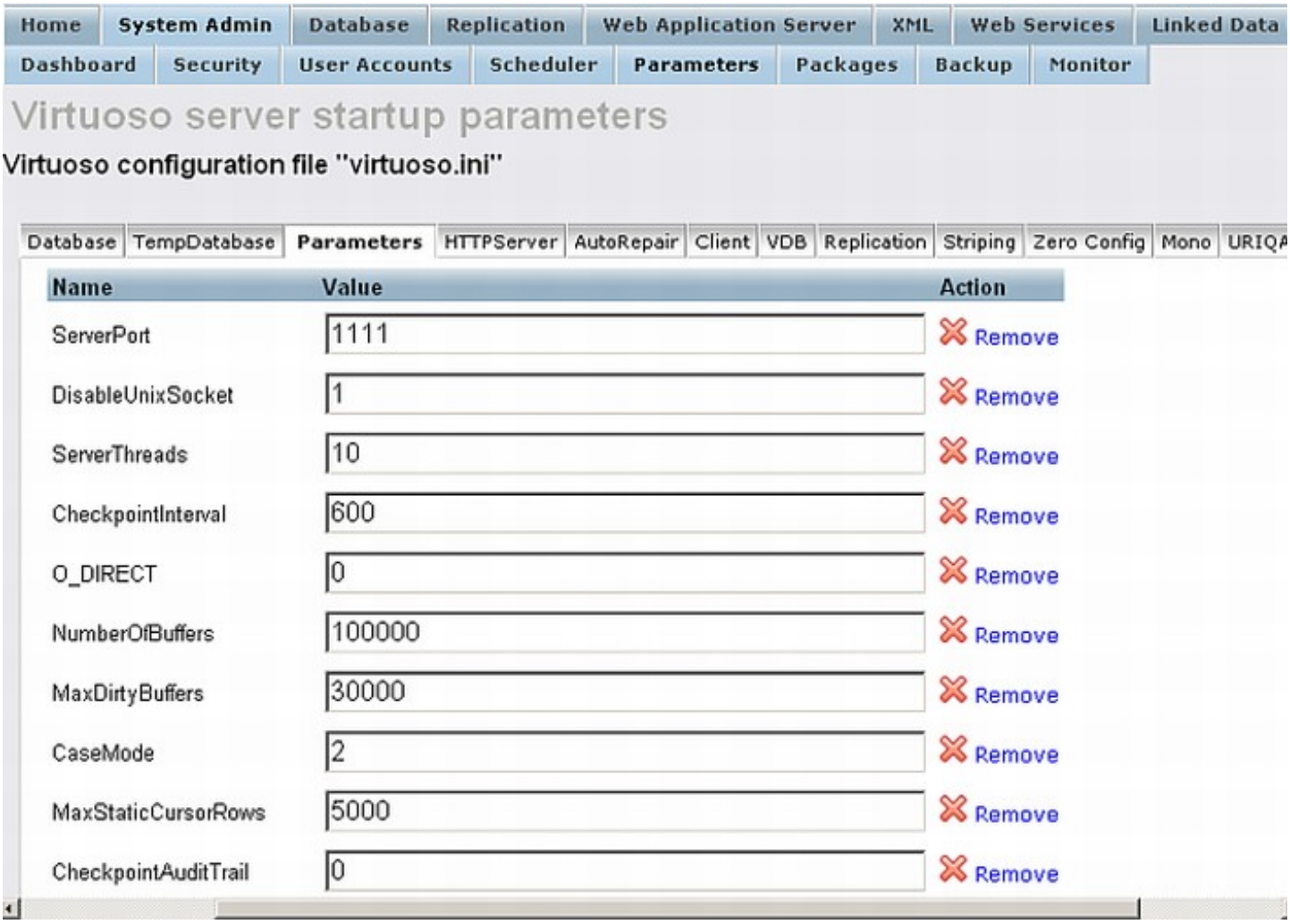


Figure 6.145. Virtuoso (virtuoso.ini) Configuration File Editor

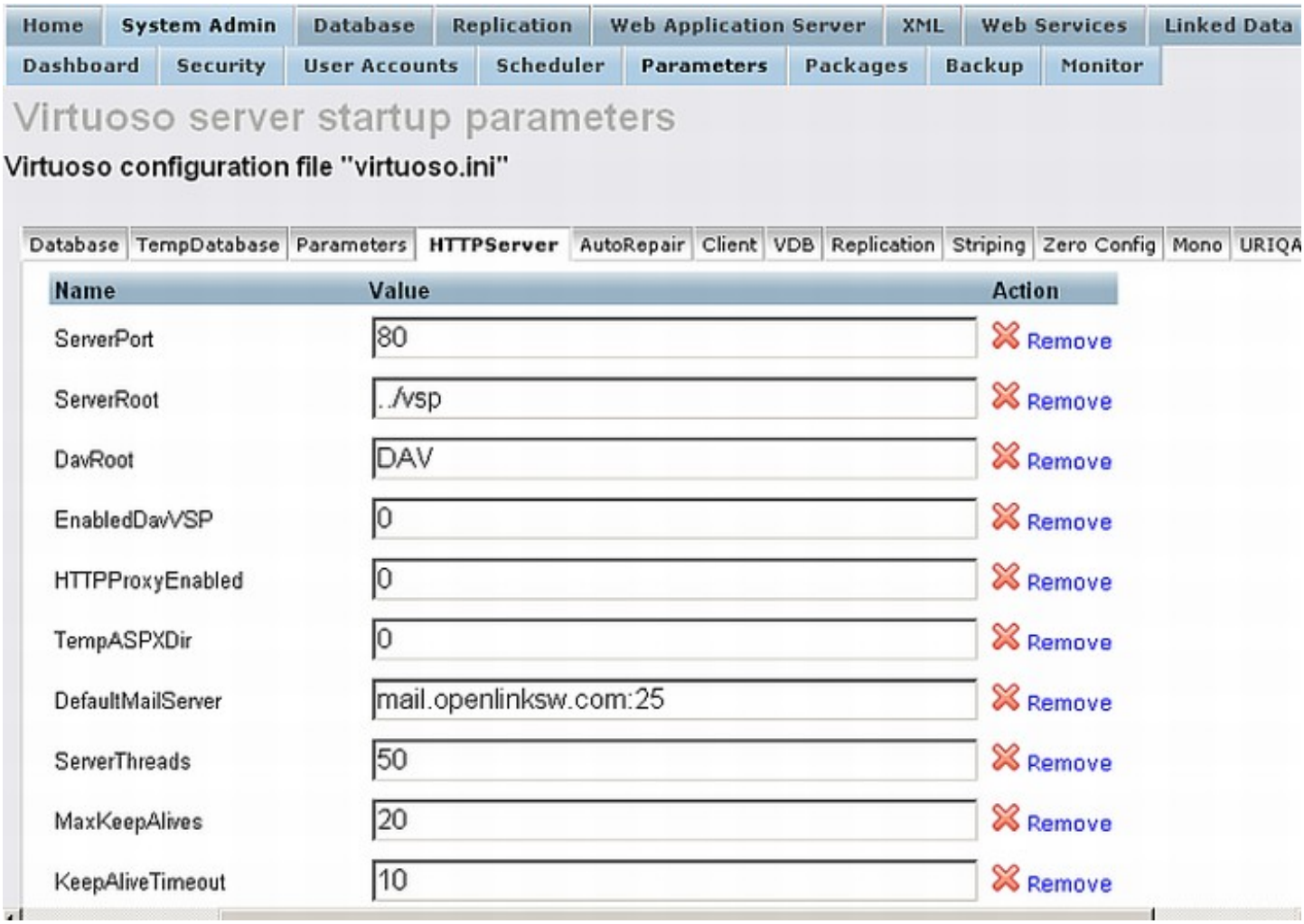


Figure 6.146. Virtuoso (virtuoso.ini) Configuration File Editor

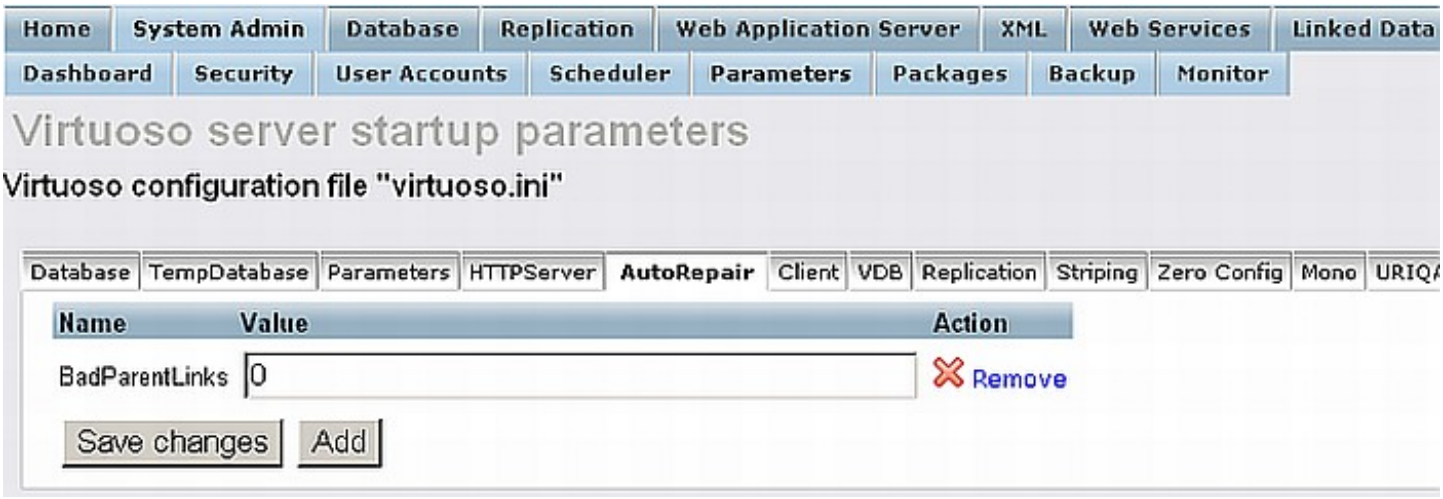


Figure 6.147. Virtuoso (virtuoso.ini) Configuration File Editor

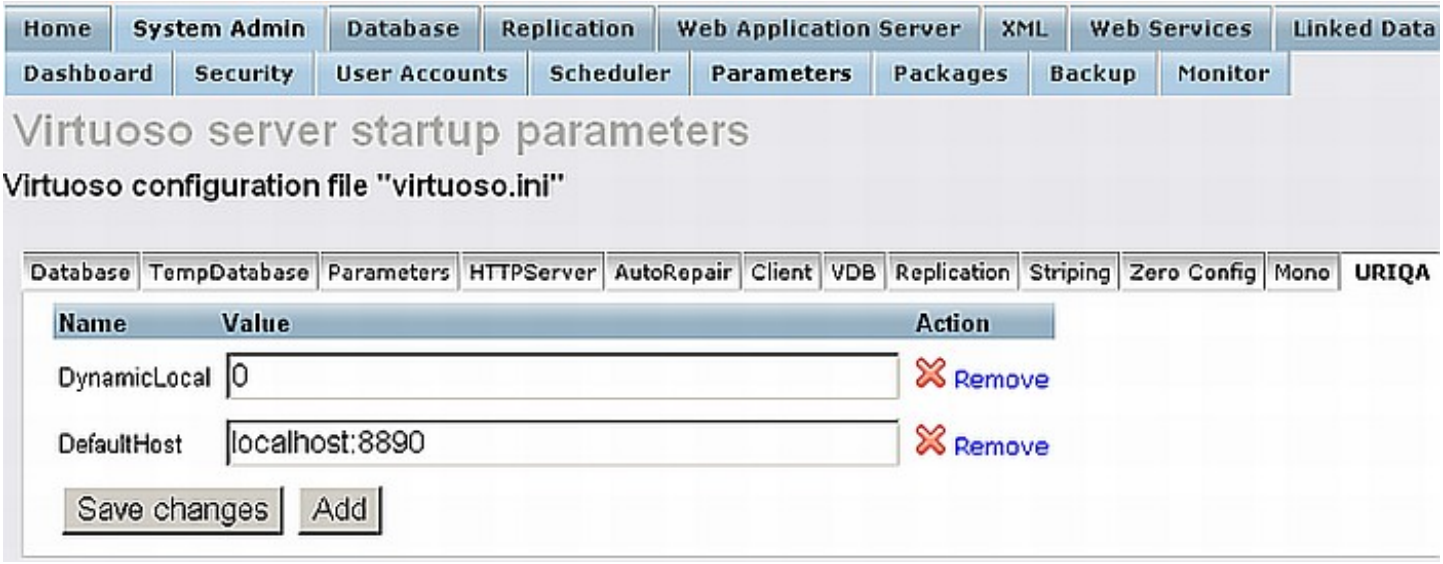


Figure 6.148. Virtuoso (virtuoso.ini) Configuration File Editor

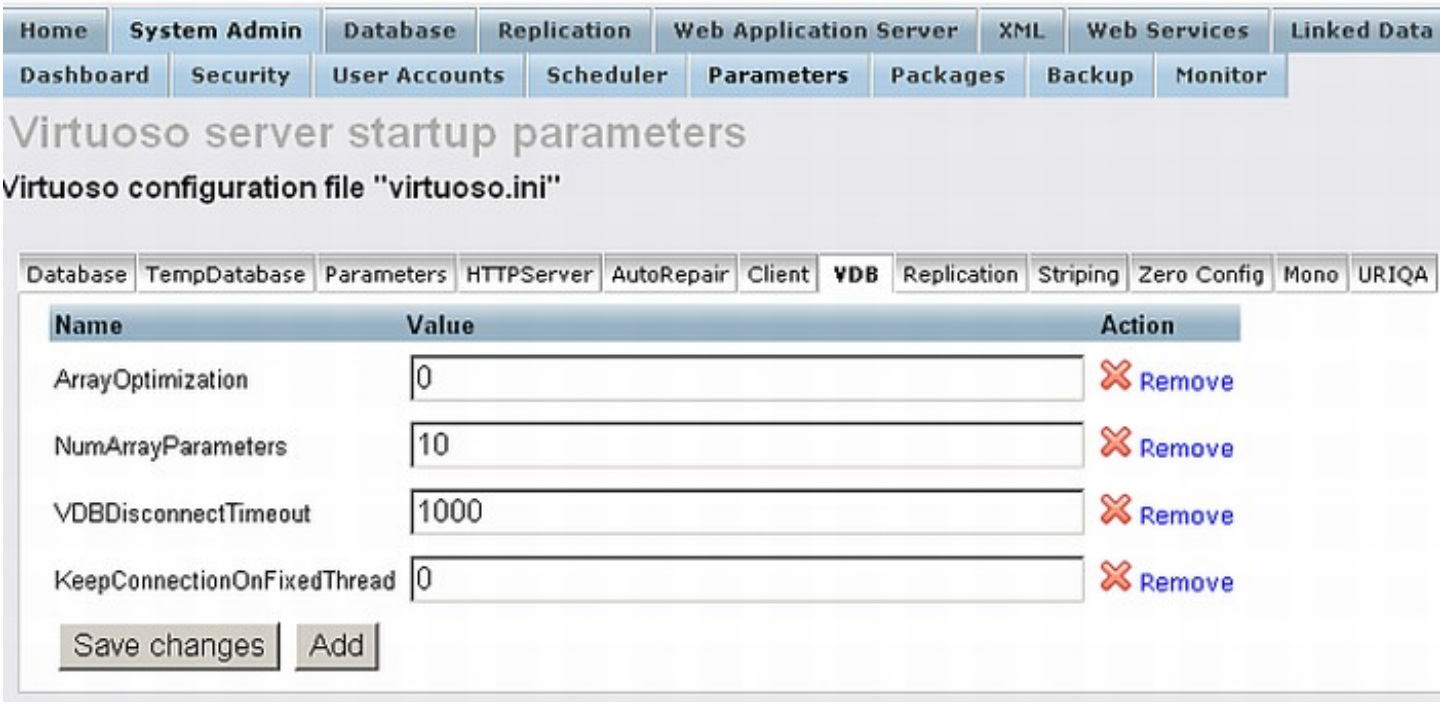
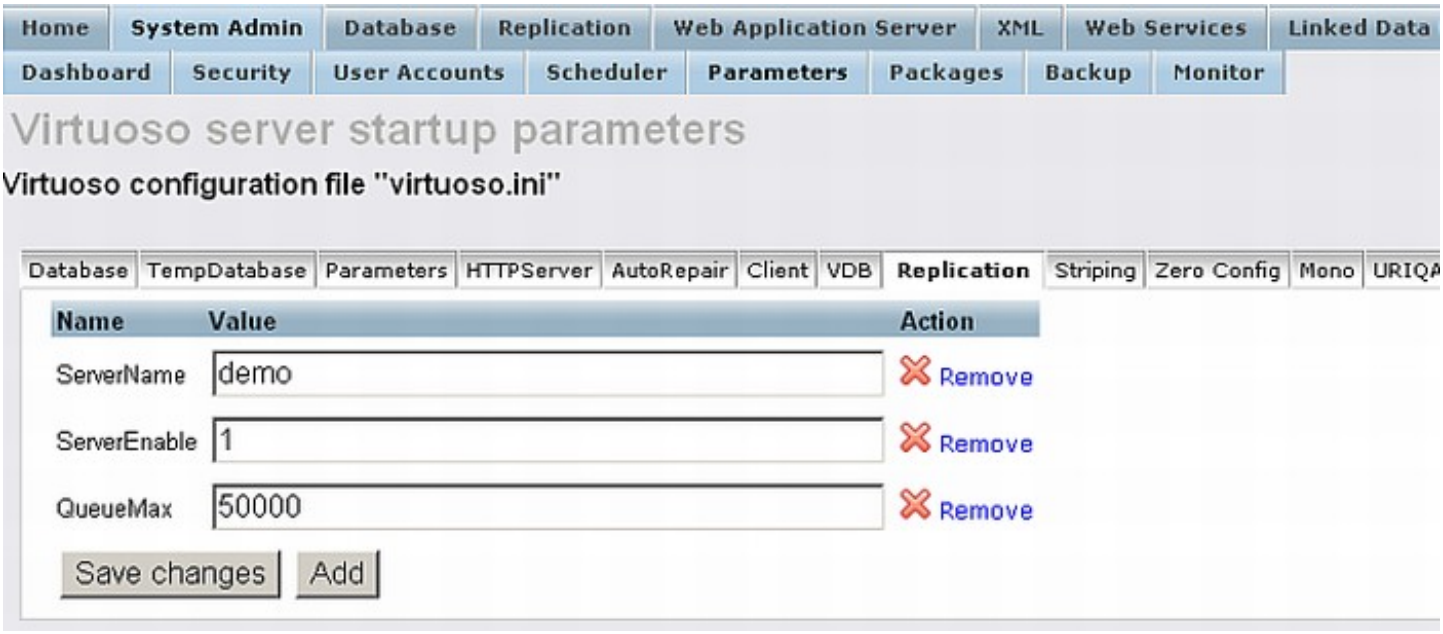


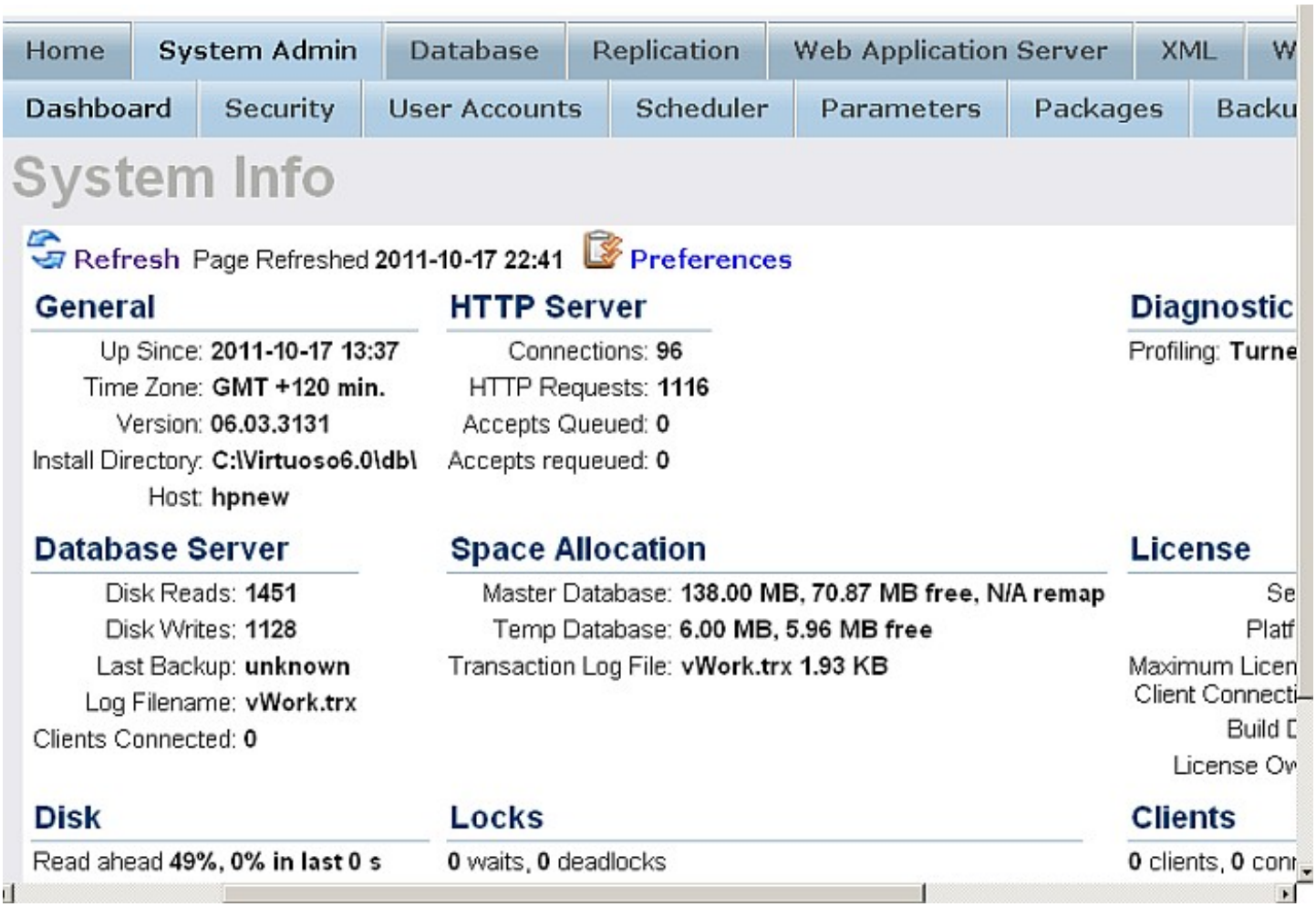
Figure 6.149. Virtuoso (virtuoso.ini) Configuration File Editor



Dashboard and Monitor

The System Admin/Dashboard consists of a collection of statistical information about your Virtuoso server. This includes general licensing information, locking, webserver hits statistics and more.

Figure 6.150. Dashboard



The System Admin/Monitor shows in details statistical information about your server.

Figure 6.151. Monitor - General Information

Figure 6.152. Monitor - HTTP Usage

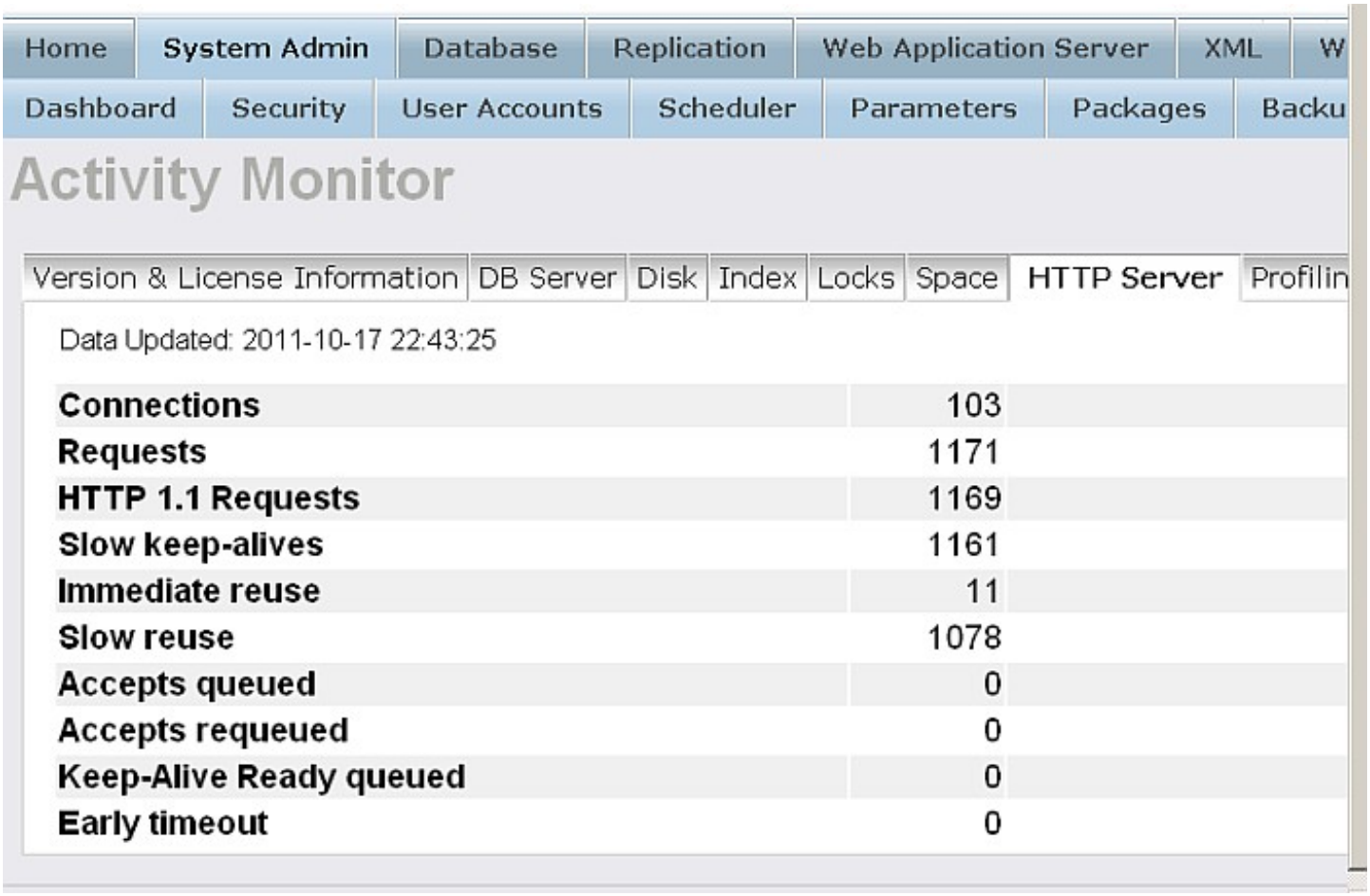


Figure 6.153. Monitor - Index Usage

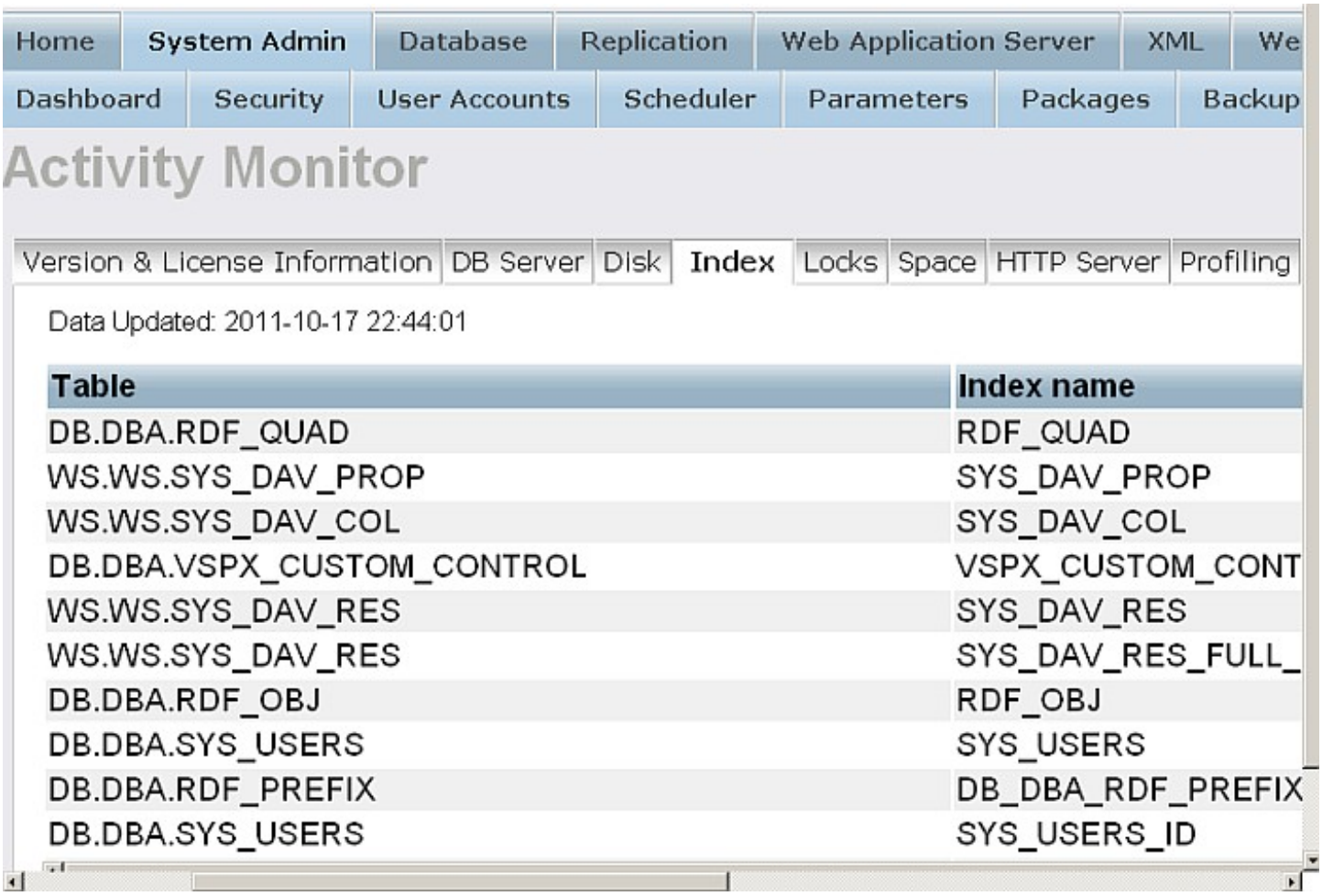
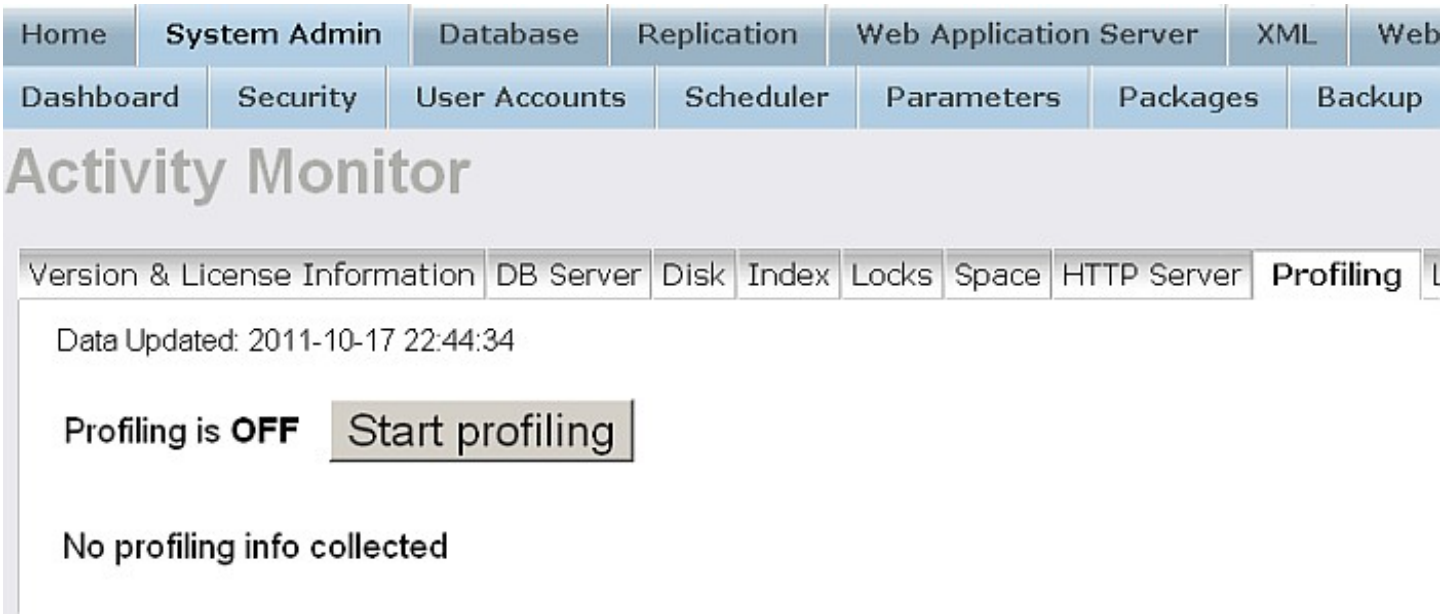


Figure 6.154. Monitor - Profiling



6.2.10. Conductor Linked Data Administration

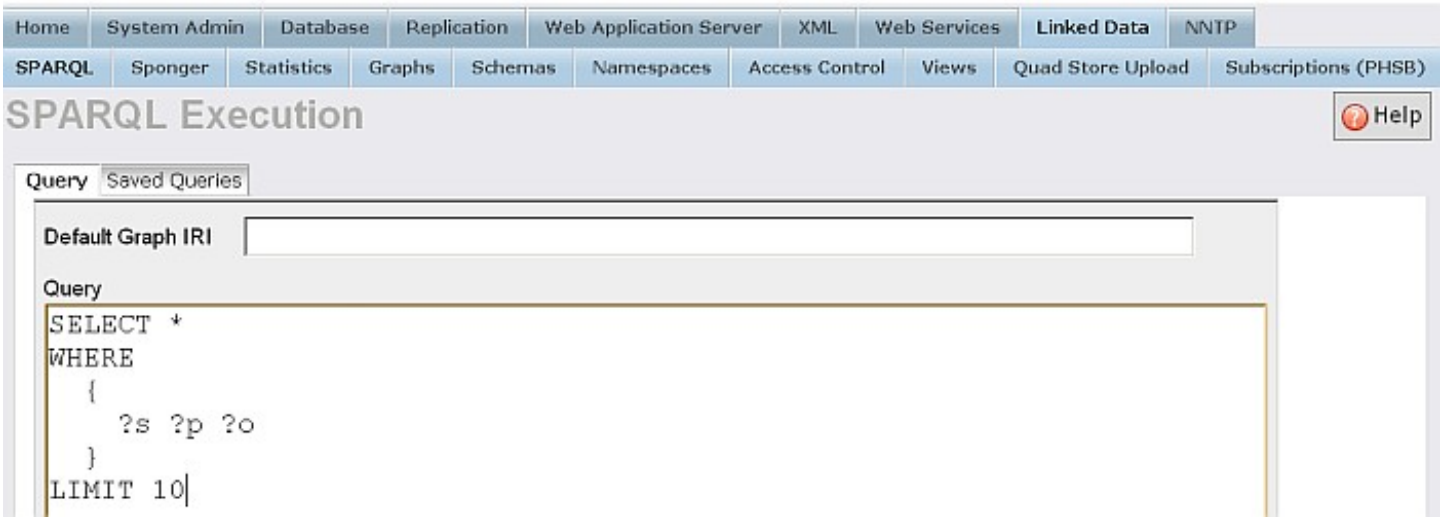
From Conductor the "Linked Data" tab allows you to execute/save/load SPARQL queries, to add/edit RDF Mapping, to perform statistics and manage graphs, to import schemas and define namespaces, to generated Linked Data Views and to upload to the Quad Store.

SPARQL

Query

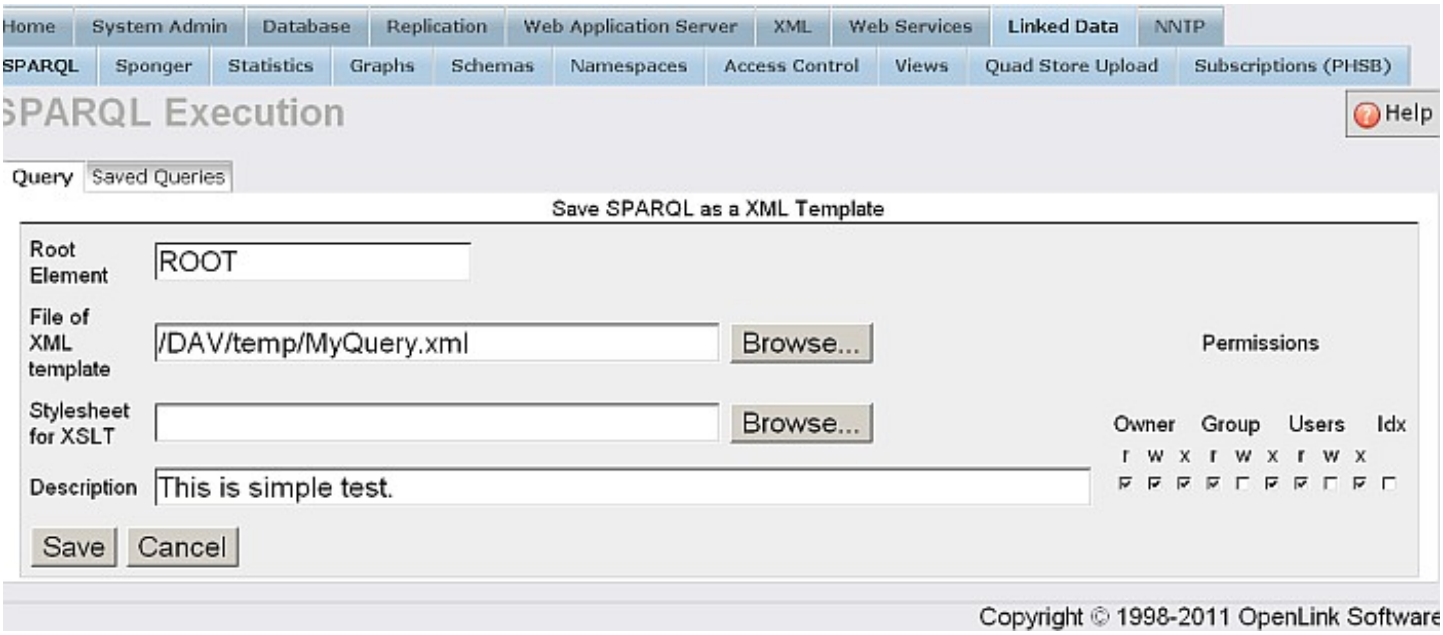
"SPARQL -> Query" offers form form SPARQL query execution:

Figure 6.155. RDF



Also you can save the results in a query or load such:

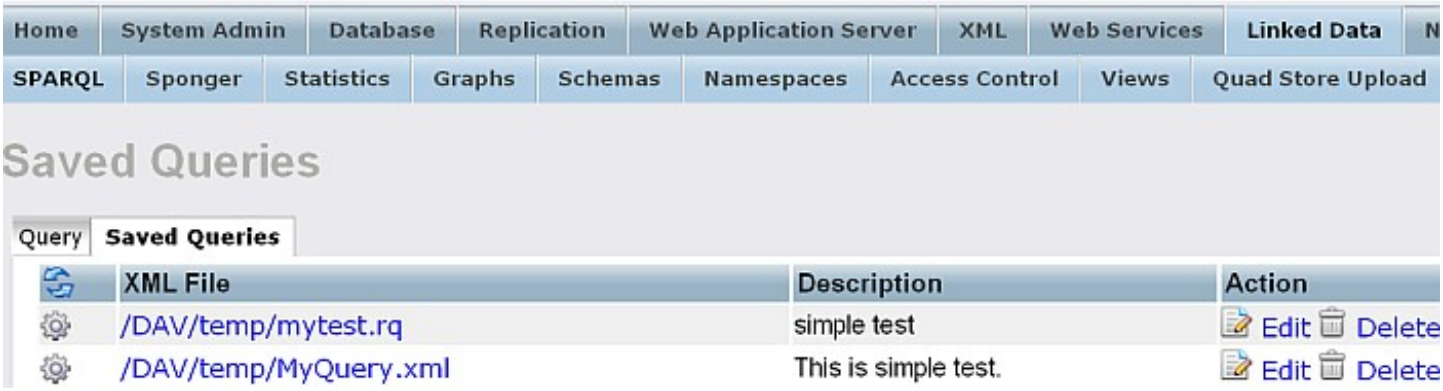
Figure 6.156. RDF



Saved Queries

"SPARQL -> Saved Queries" offers list of saved queries, which can be opened in the "Query" tab in order to be executed:

Figure 6.157. RDF

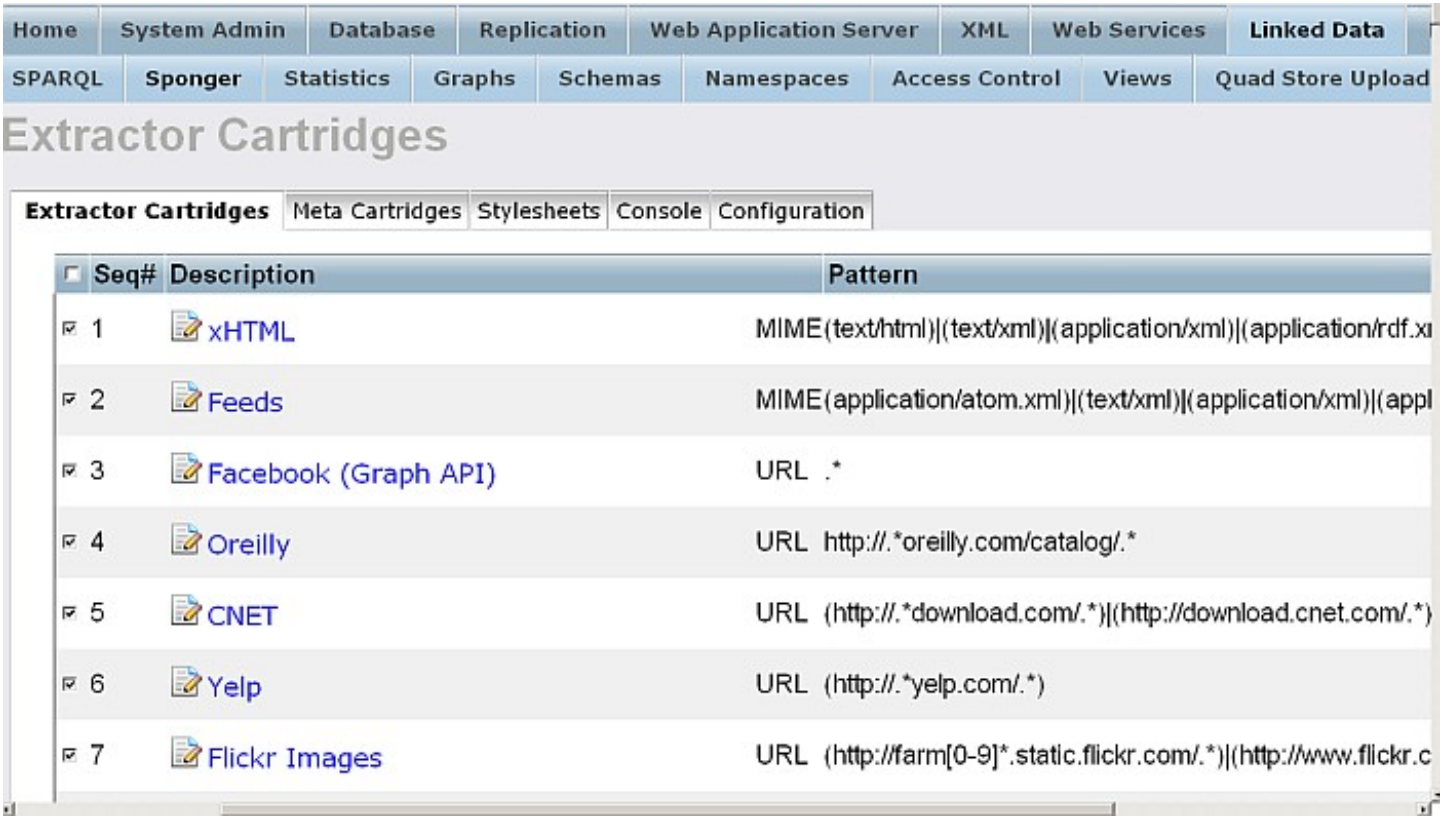


Sponger

Extractor Cartridges

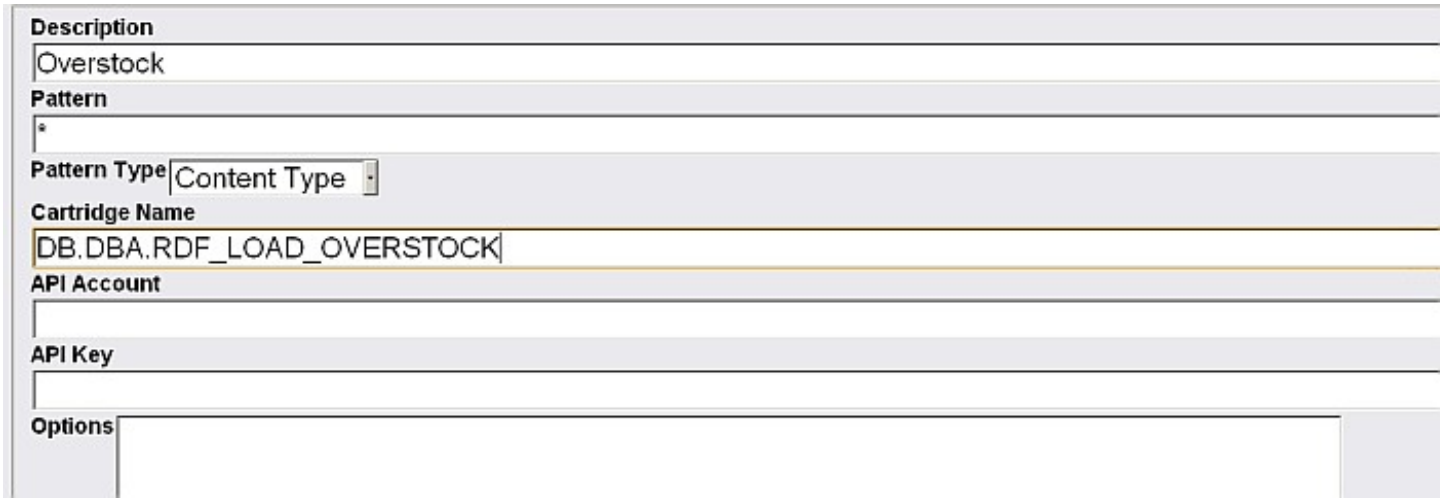
Shows list of available extractor cartridges. They can be re-ordered by place of appearance, edit or deleted:

Figure 6.158. RDF



You can also add a new extractor cartridge:

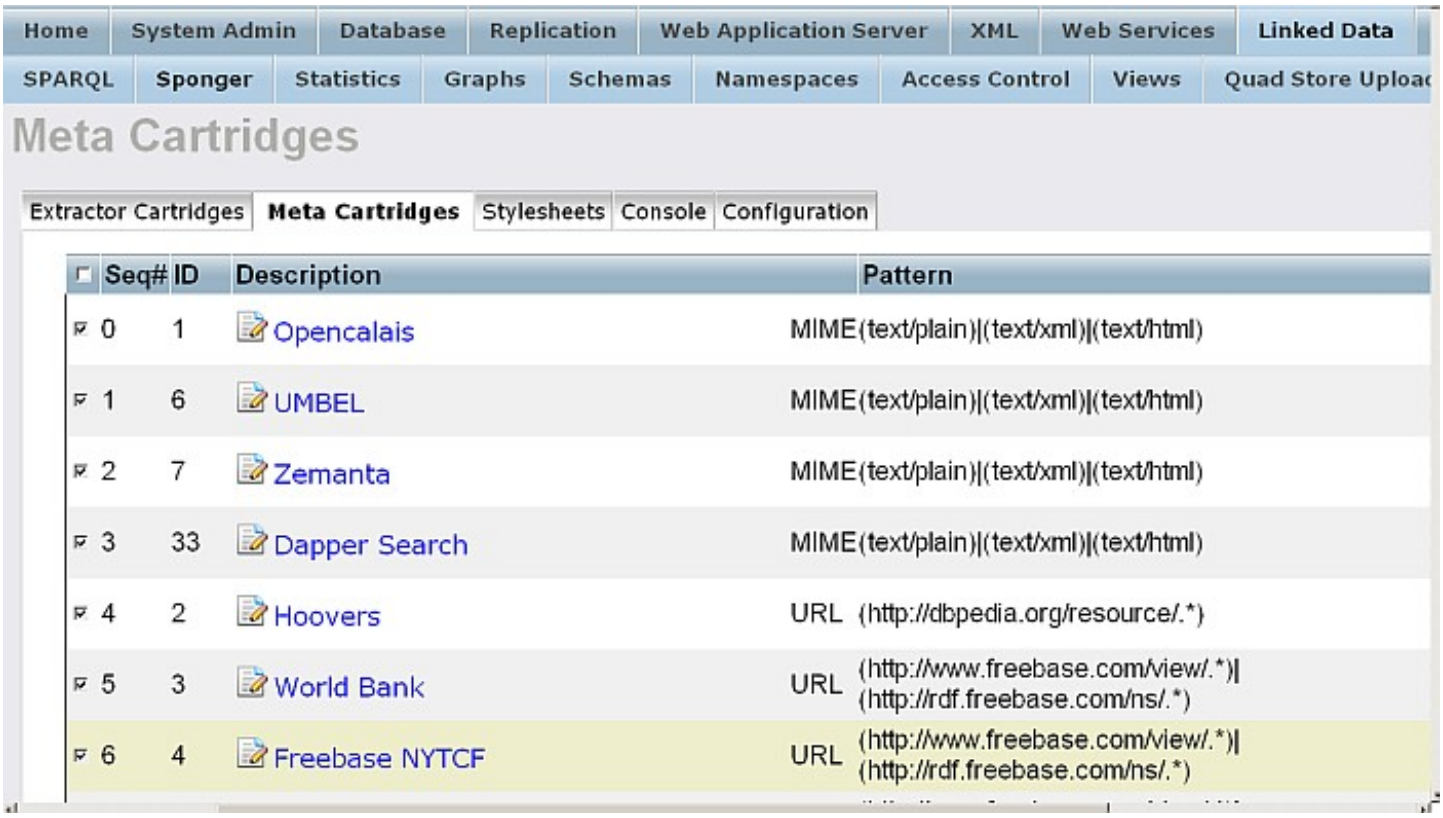
Figure 6.159. RDF



Meta Cartridges

Shows list of available meta cartridges. They can be re-ordered by place of appearance, edit or deleted:

Figure 6.160. RDF



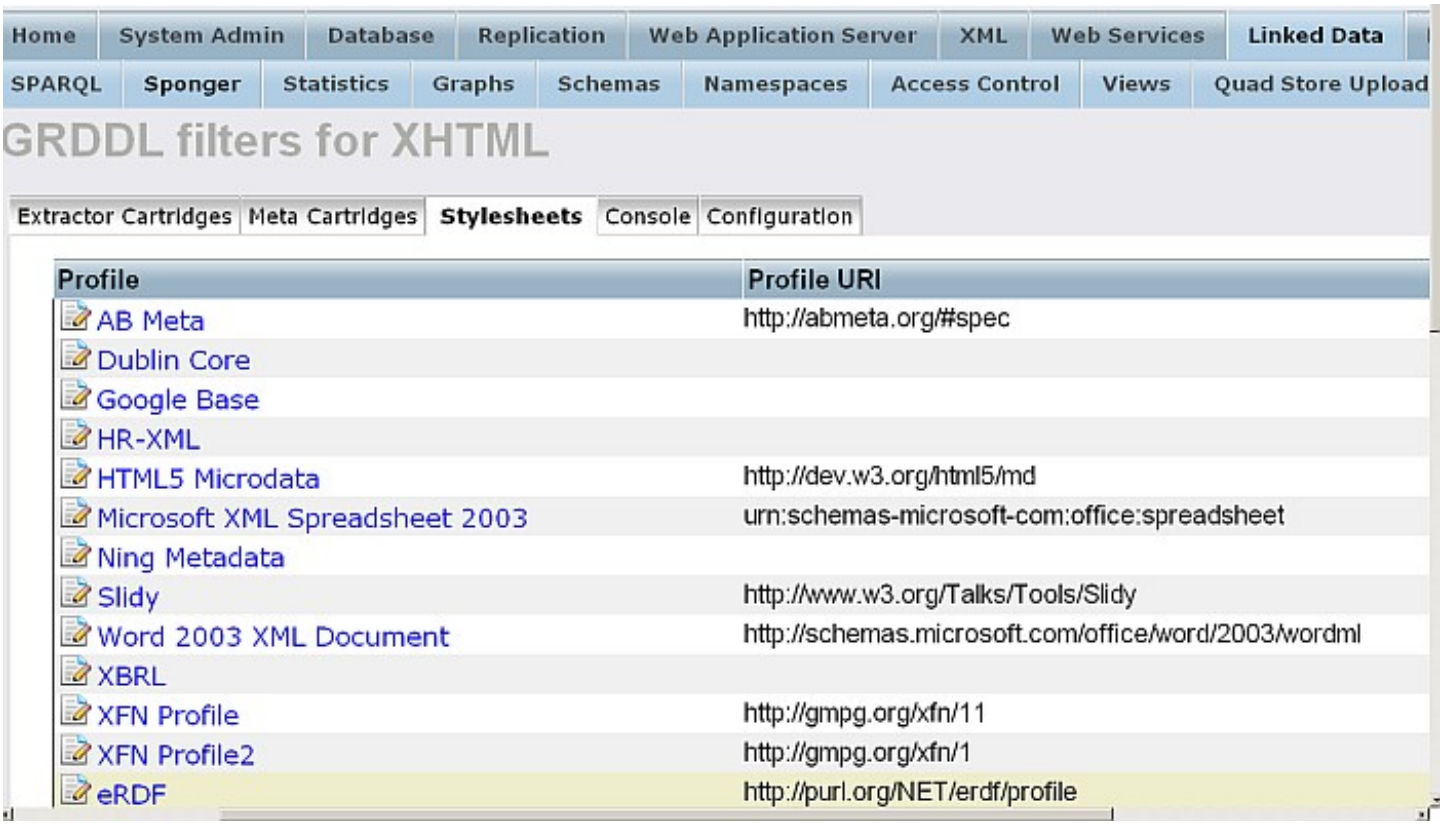
You can also add a new meta cartridge:

Figure 6.161. RDF

Stylesheets

Shows list of available stylesheets for processing cartridges. From here you can manage these stylesheets by adding new, editing existing ones or deleting them:

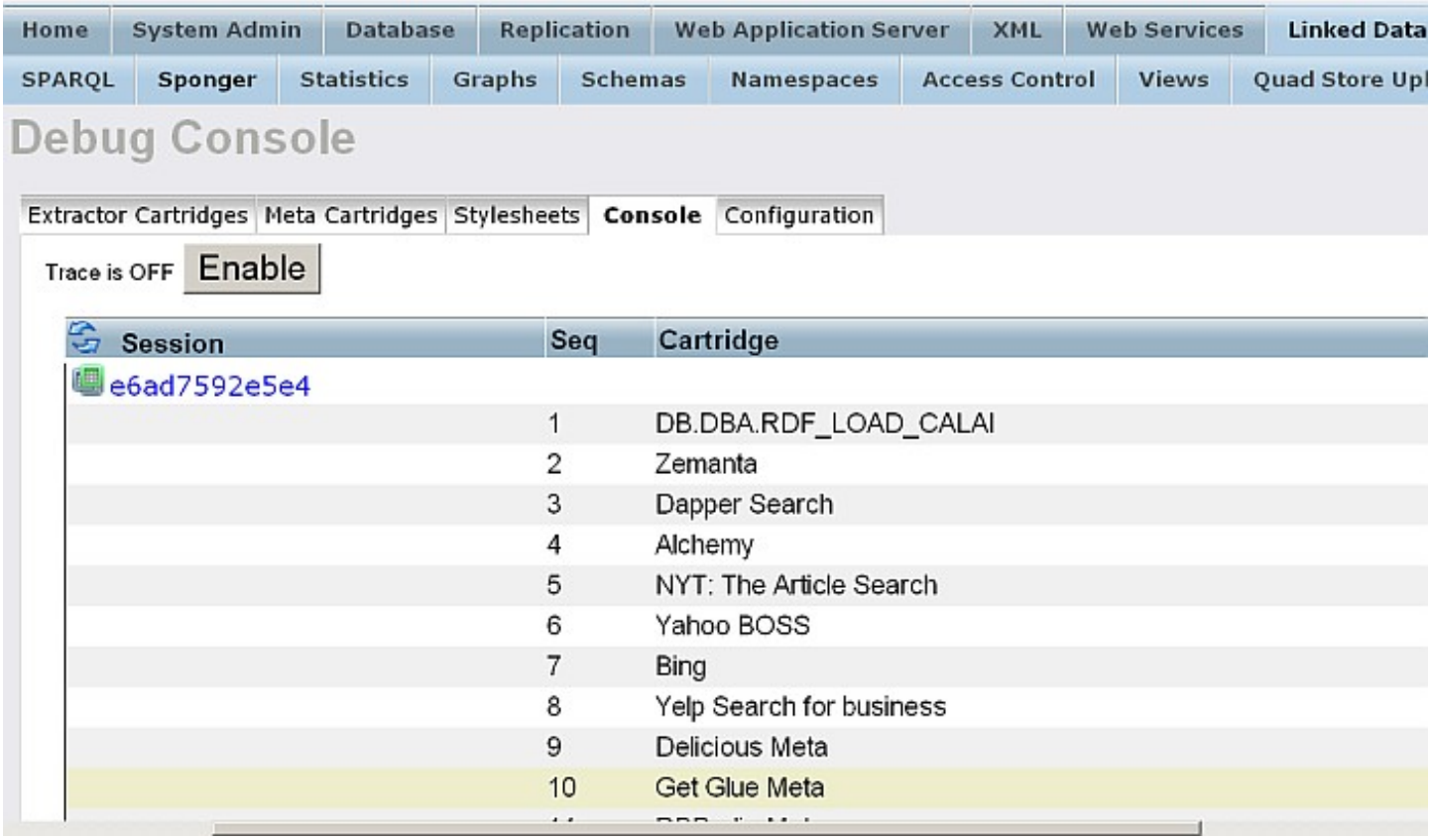
Figure 6.162. RDF



Console

Offers tracking cartridges exection. By default is off. When enabled, shows list of sessions per cartridge:

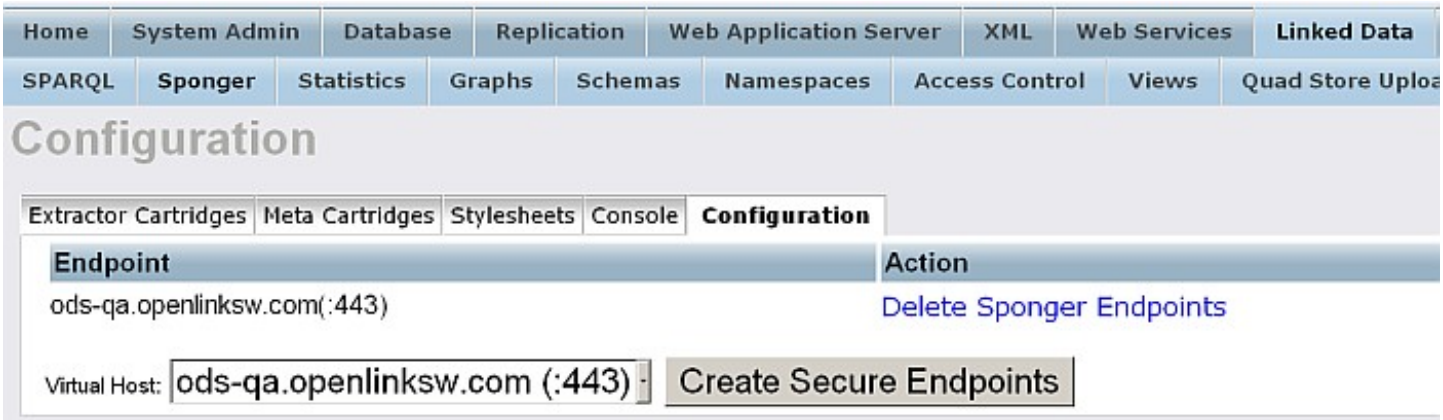
Figure 6.163. RDF



Configuration

Offers creation of secure endpoints for pre-defiend vitual host:

Figure 6.164. RDF



Statistics

Offers interface for `DB.DBA.RDF_VOID_STORE(graph, to_graph)` . In the new graph (field "Publish to") will be saved the VoID statistics data when generated:

Figure 6.165. RDF

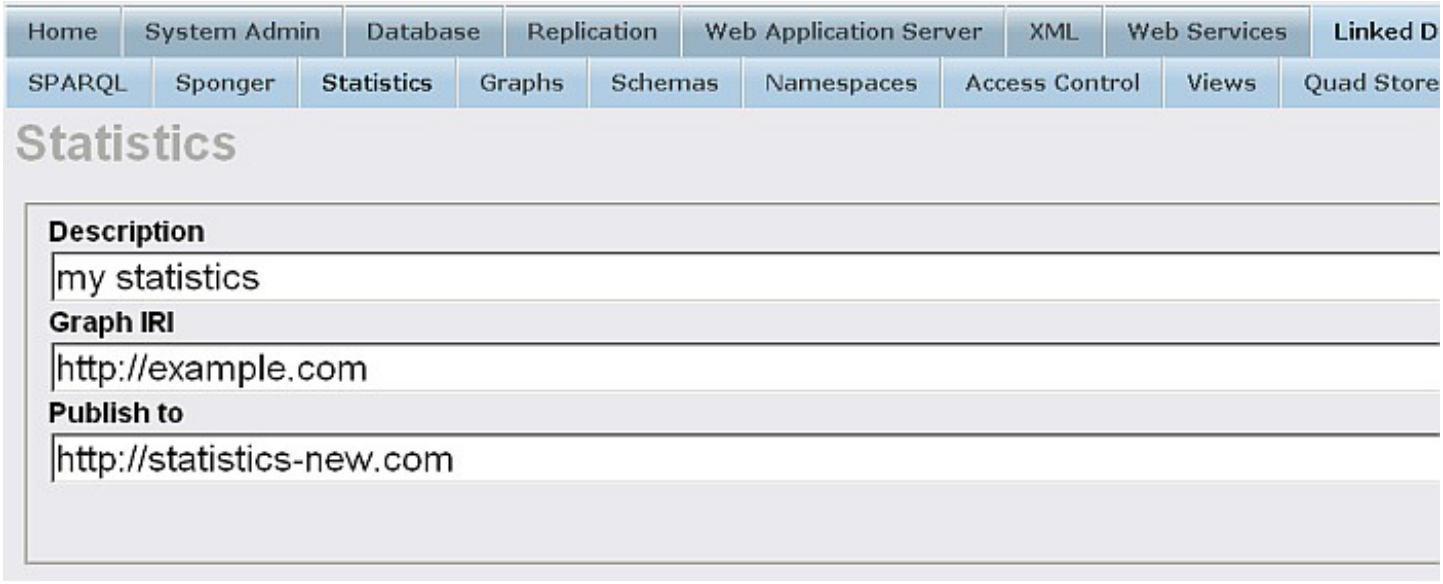
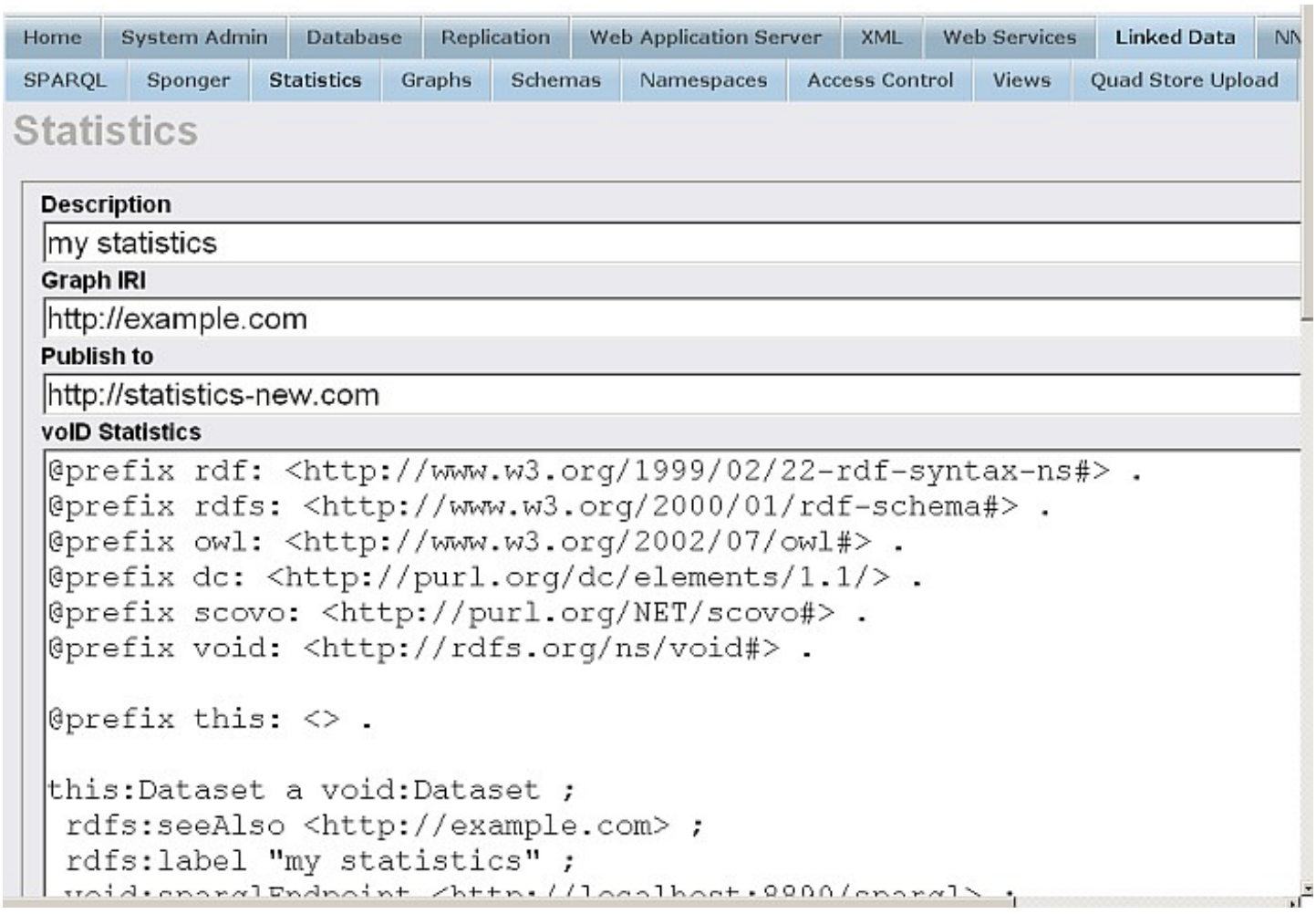


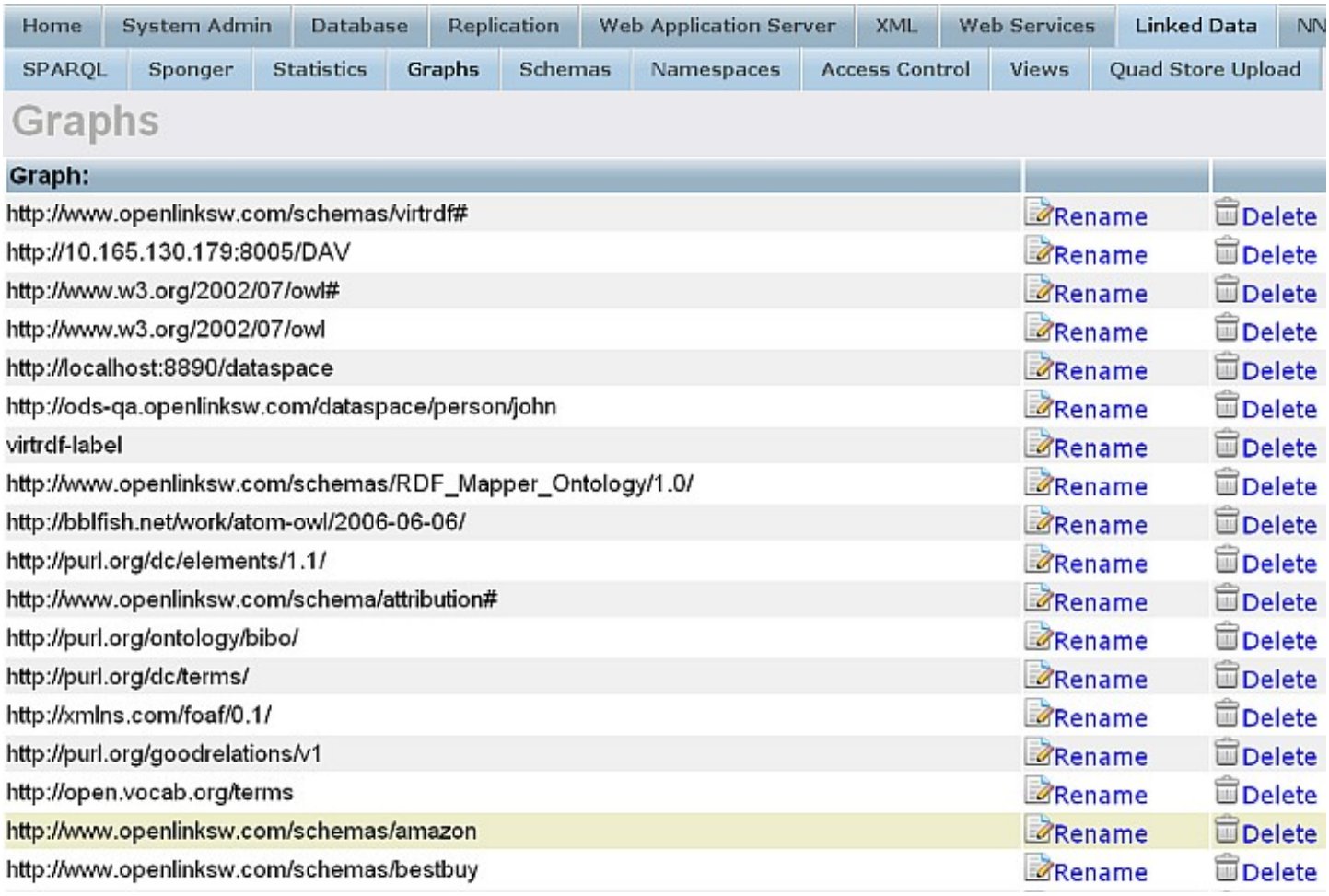
Figure 6.166. RDF



Graphs

"Graphs" tab shows list of current graphs in the RDF Quad Store. From here day can be renamed or deleted:

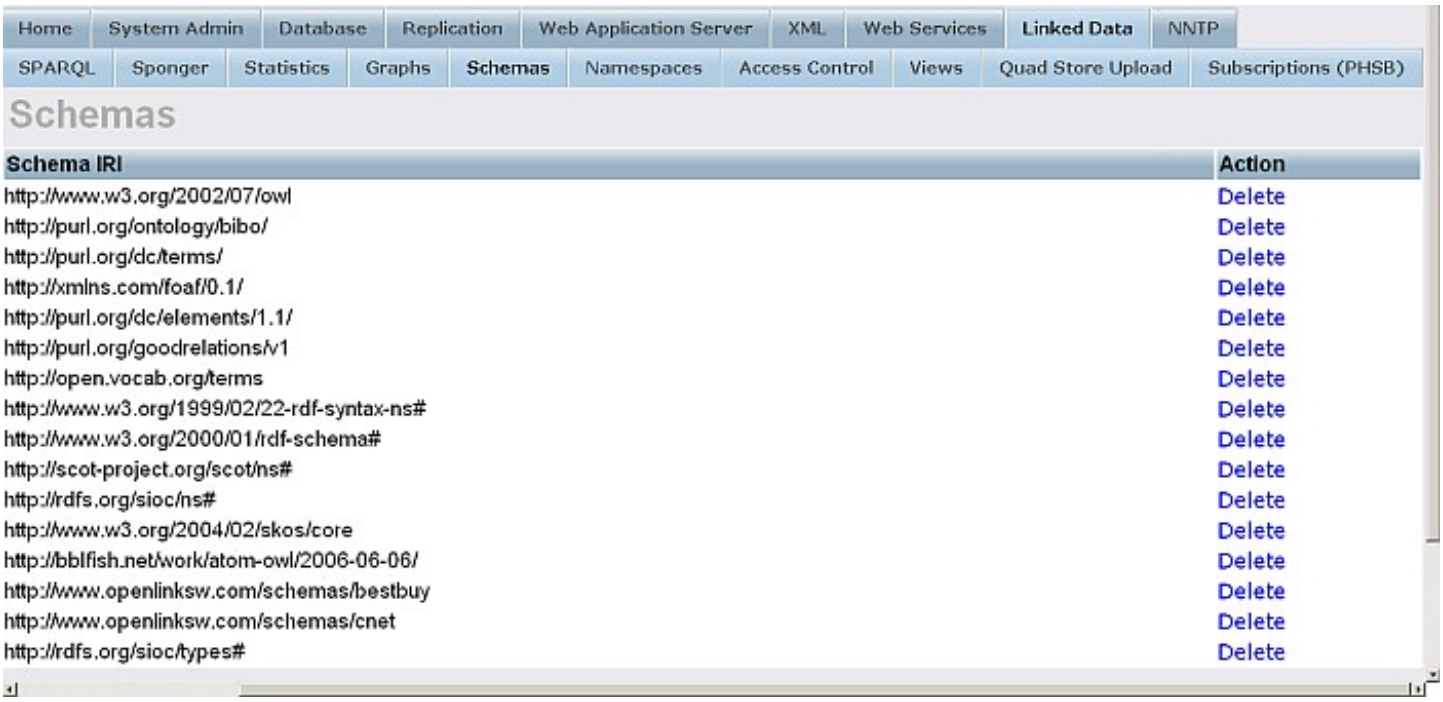
Figure 6.167. RDF



Schemas

Shows list of available schemas IRI-s and offers a new schema IRI to be imported:

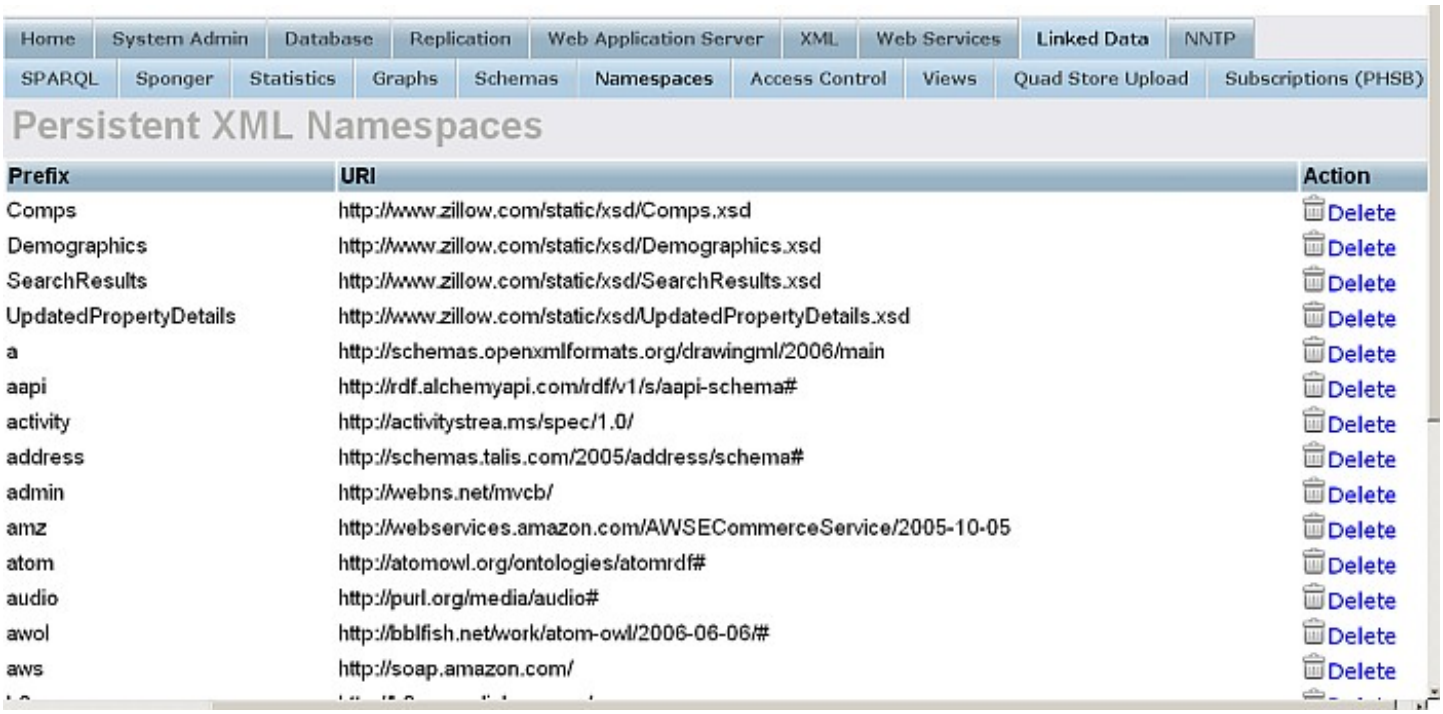
Figure 6.168. RDF



Namespaces

Shows list of namespace prefixes and offers a new namespace prefix to be imported:

Figure 6.169. RDF



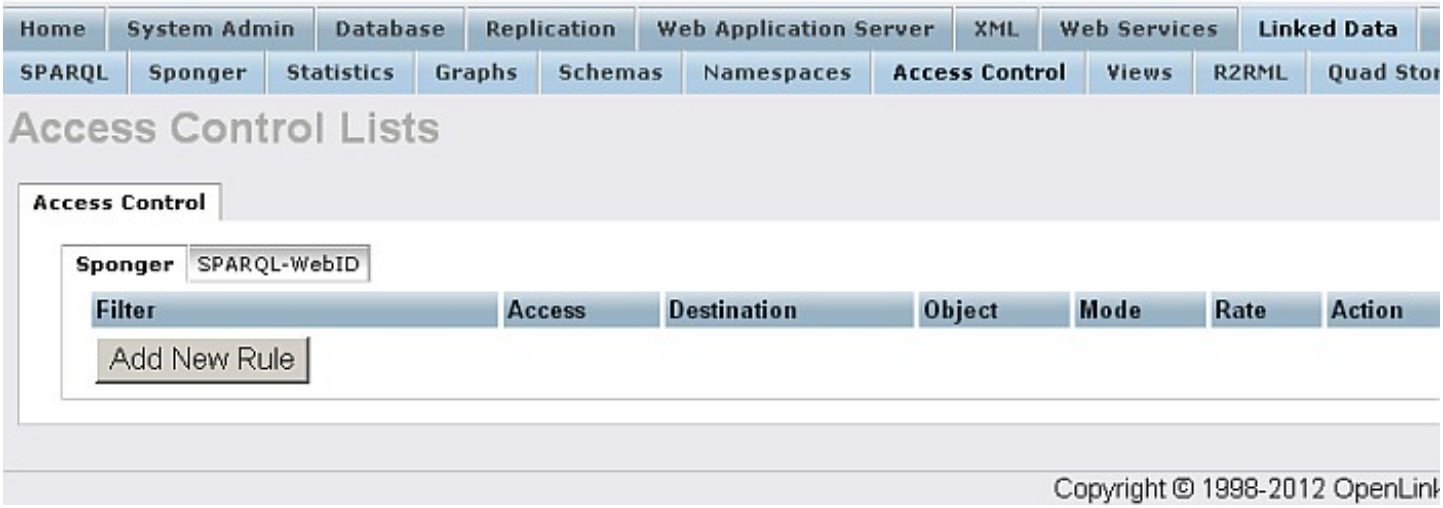
Access Control

Sponger

Manage Sponger rules by adding new ones, deleting, re-ordering, etc.

This tab is available only when the cartridges_dav.vad is installed.

Figure 6.170. RDF

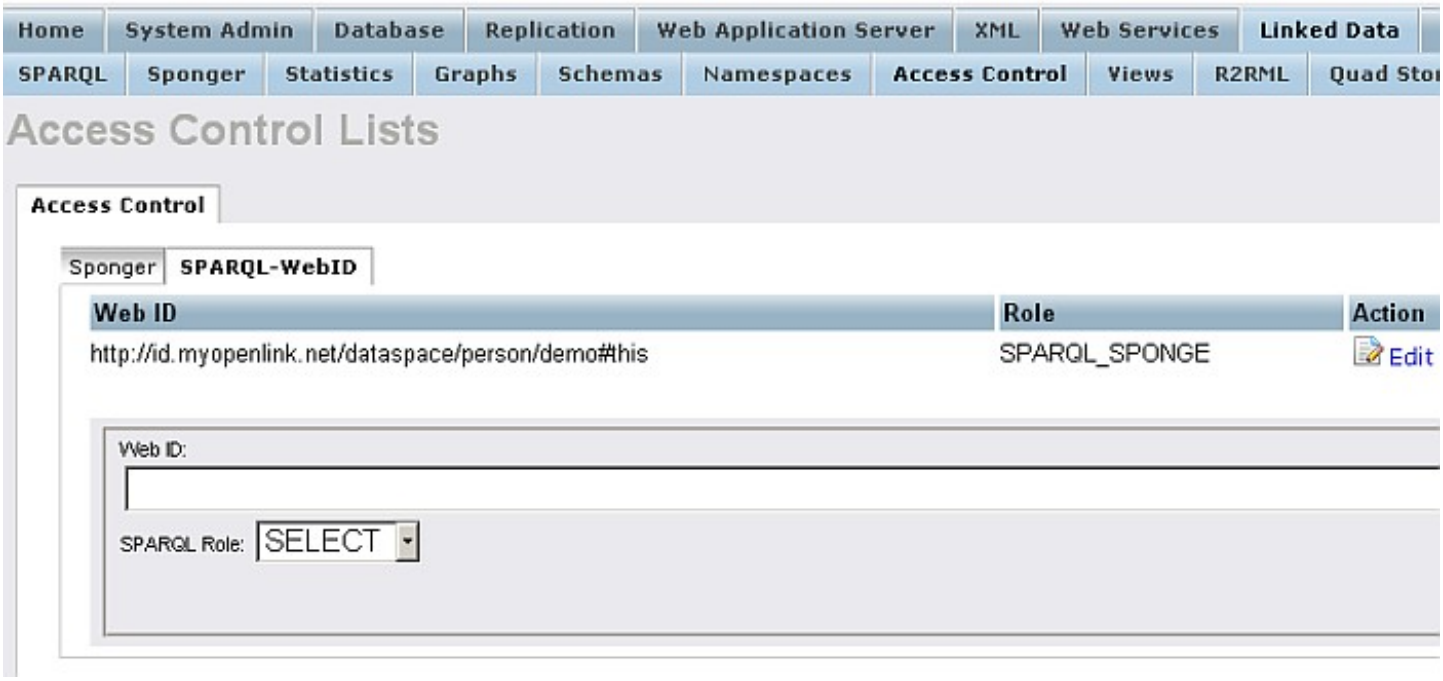


SPARQL-WebID

Here you can manage different WebID Protocol ACLs WebID ACL by granting SPARQL Roles such as "SELECT", "UPDATE", "SPONGE":

This tab is available only when the policy__manager_dav.vad is installed.

Figure 6.171. RDF



See Also:

Managing a SPARQL Web Service Endpoint

WebID Protocol ACLs

Creating and Using a SPARQL-WebID based Endpoint

Example with curl and SPARQL-WebID endpoint

Views

Virtuoso uses a SPARQL-based Meta Schema Language to provide RDBMS-to-RDF mapping functionality (also described as, Linked Data Views of SQL data). The language is an extension of the SPARQL query language meshed with Virtuoso's SPASQL (SPARQL-inside-SQL) functionality. The language enables you to declaratively map relational database schema tables, columns, rows, and foreign key relationships to Classes, Attributes, Relationships, and Instances (Objects/Entities/Individuals) defined by RDF Schemas or OWL Ontologies. The mapping process uses a special built-in Virtuoso function to generate IRIs / URIs in "Subject" or "Object" position of a SPARQL graph pattern, en route to building a set of SQL compiler rules and optimizations for translating SPARQL queries into SQL on the fly.

Figure 6.172. RDF

See Also:

Mapping Process steps.

Mapping Example.

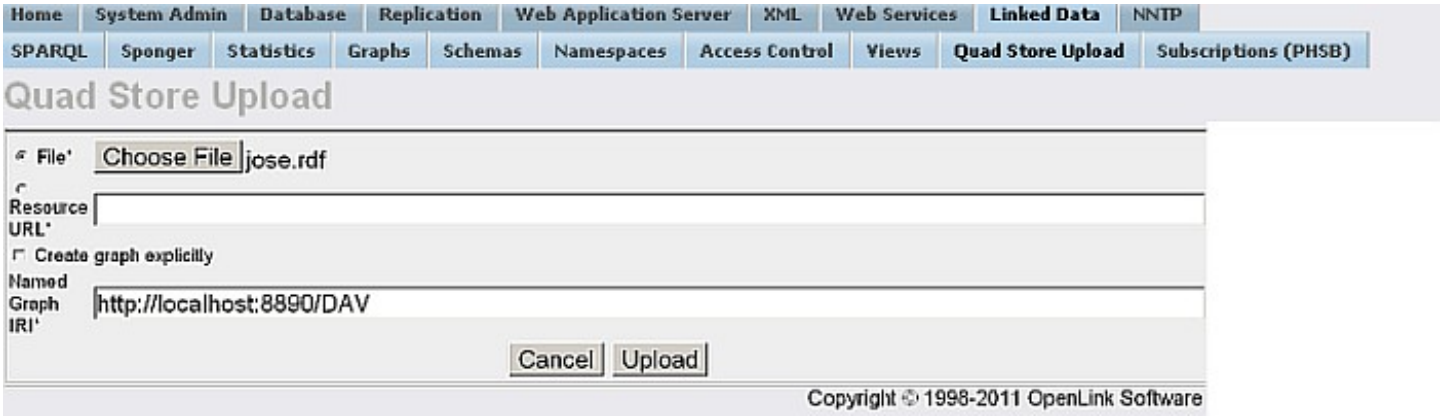
Mapping SQL Data to RDF.

Automated Generation of Linked Data Views over Relational Data Sources with Virtuoso.

Quad Store Upoload

Offers upload to Quad Store from file or Resource URL:

Figure 6.173. RDF

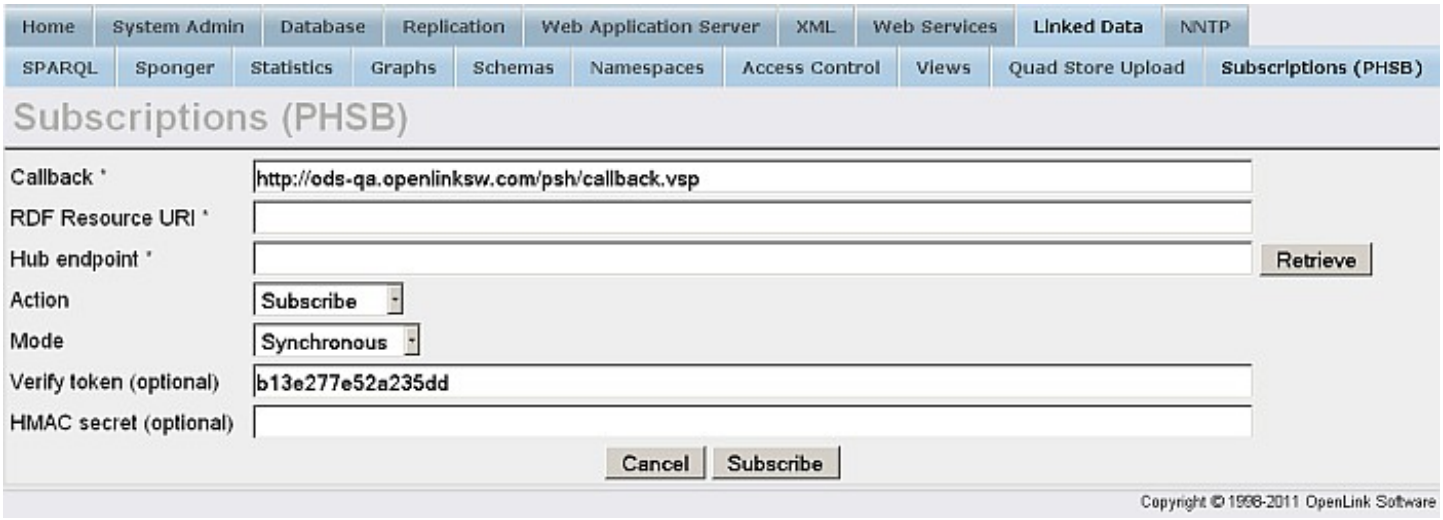


Subscriptions (PHSB)

This tab is available only when the pubsubhub_dav.vad is installed.

Offers Virtuoso's implementation of the PubSubHubBub Protocol to handle Linked Data by subscribing to Linked Data resource URIs.

Figure 6.174. Subscriptions (PHSB)



Note: the Hub endpoint should be defined ACL in System Admin -> Security -> Access Control -> PSH-SLL

See Also:

Example of Linked Data Usage of PubSubHubbub Implementation.

Feed subscription via PubSubHub Protocol.

Setting up PubSubHub in ODS.

Securing your SPARQL Endpoint via WebID.

6.2.11. Conductor News Server Administration

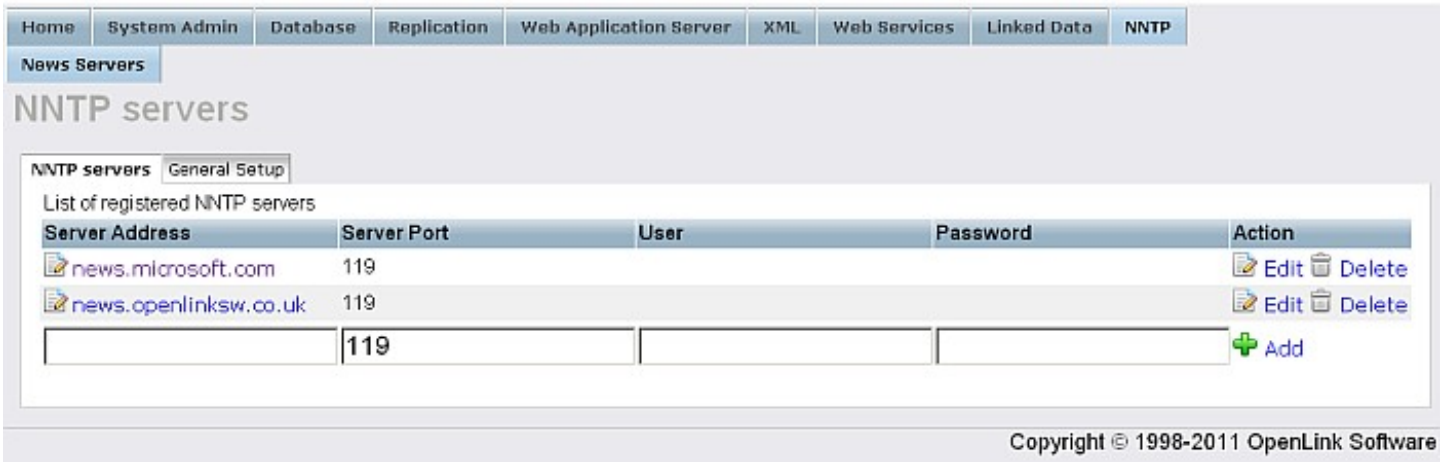
Conductor Newsgroups Administration

From Conductor the NNTP tab allows you to view and configure Newsgroups associated with the Virtuoso News Server. Like Virtuoso's virtual database, Virtuoso can "link" in newsgroups from remote news servers. This interface allows you to view the configuration and control the newsgroups both local and remote.

The Virtuoso News server component will need to be enabled in the Virtuoso.ini file by specifying the NewsServerPort number.

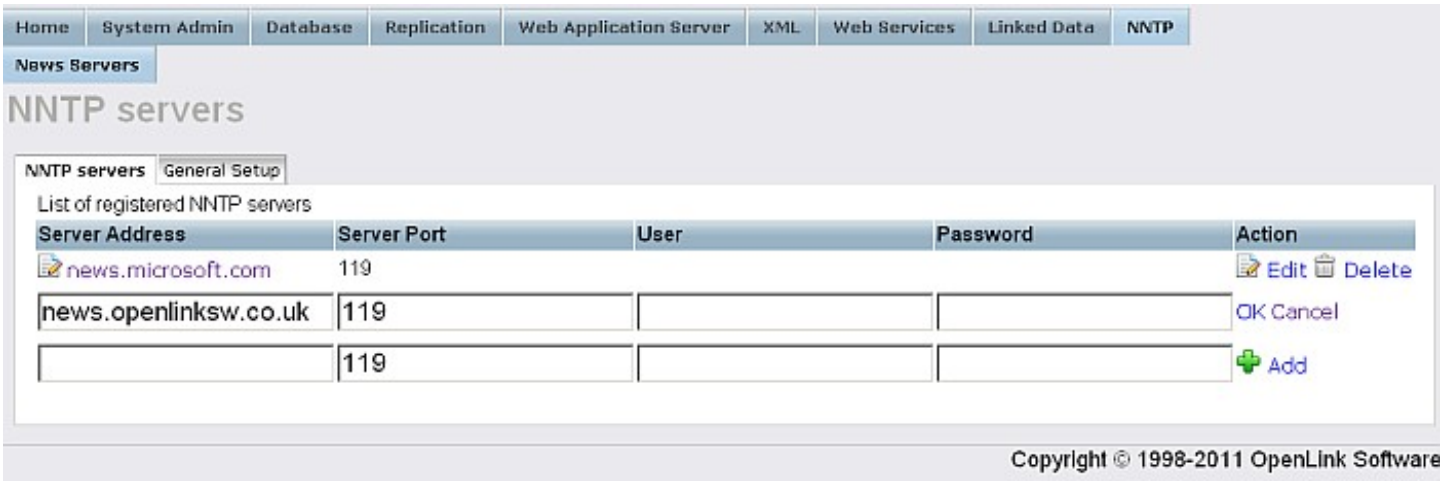
The tab "NNTP Servers" shows list of the registered News servers.

Figure 6.175. News Server Administration



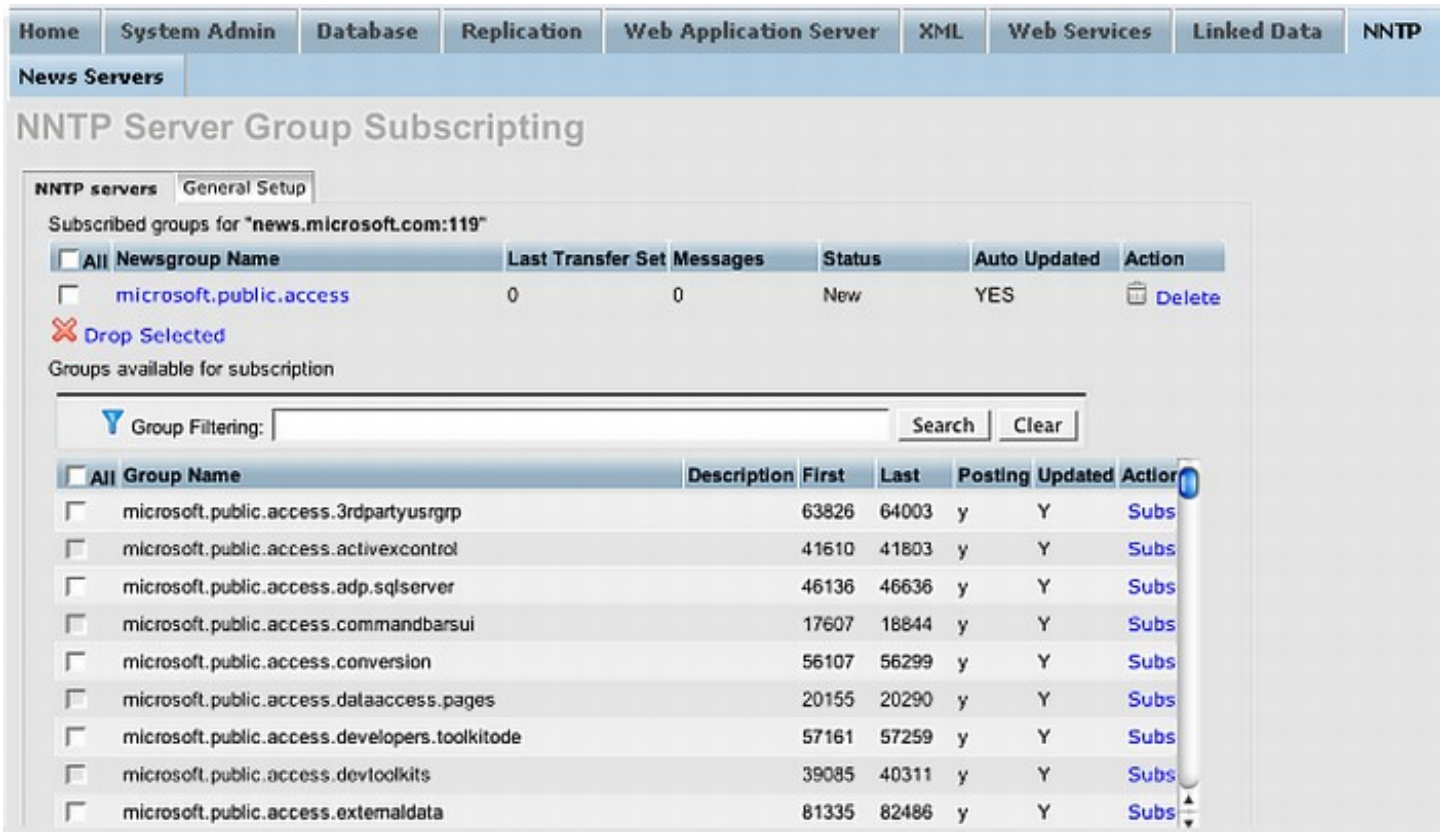
Click the "Edit" link to change the news server settings, or click the "Delete" link to remove the news server.

Figure 6.176. Edit News Server Properties



Click the "Edit Groups" link for a current News Server in order to manage the list of available newsgroups.

Figure 6.177. News Groups List



You can also *update* the cache list of newsgroups. Newsgroup names are (re)fetched in a batch background scheduled task. This is because the list can be very long or the connection to the news server could be slow. The status of the fetch is indicated in the grey bar at the foot of the newsgroup list table.

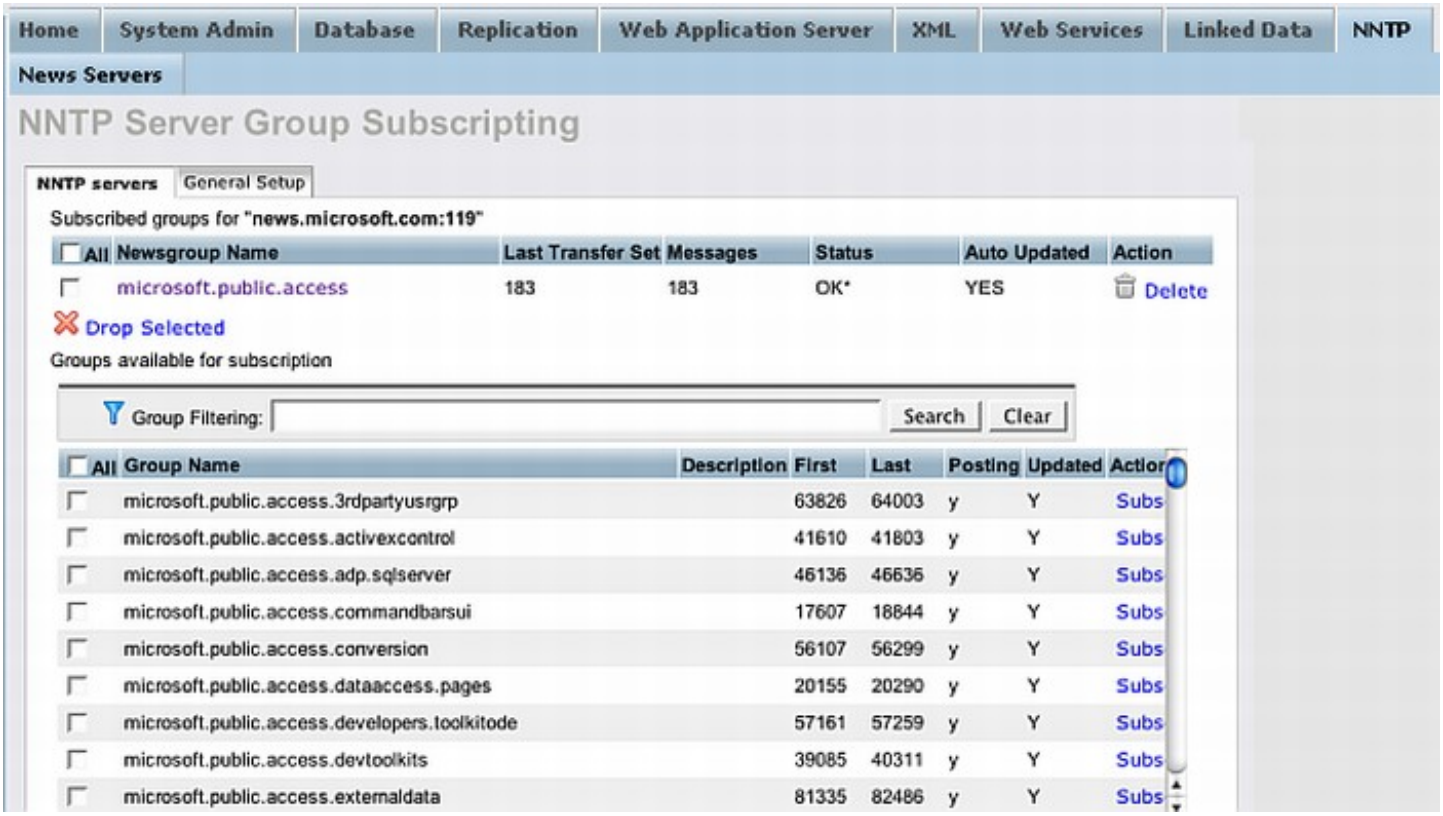
For each newsgroup listed for a news server you can *update* the messages fetched, view and edit the *Properties* of the group, or *Remove* the group from the list. Also you can get previous 500 messages or to get new messages by clicking the links at the bottom of the form shown below.

Figure 6.178. Manage subscribed for Newsgroup

Add Newsgroup

From the "NNTP Servers" tab click on the "Edit Groups" link for a registered News Server. Then click the *Subscribe to newsgroups* link to subscribe to newsgroups on the server.

Figure 6.179. Add Newsgroup



Once groups are selected you can use the *Subscribe Selected* button to add the groups.

Use the *Back to servers list* link to return to the News Servers List page.

Adding New News Server

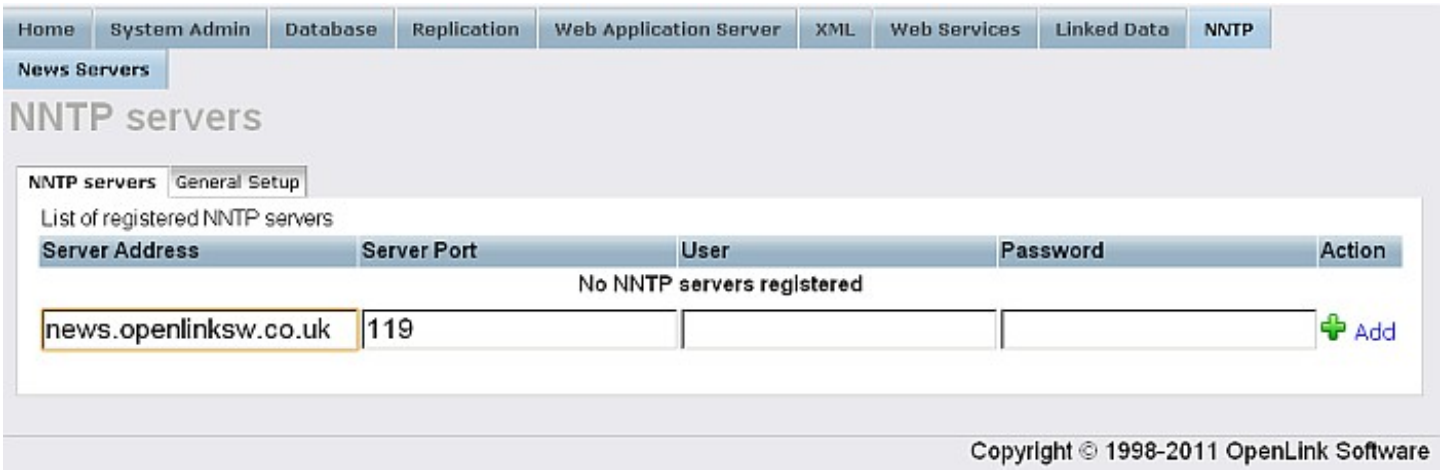
From Conductor NNTP/NNTP Servers you can add remote servers. The values you need to provide in the form are as follows:

Server Address is the IP address or hostname of the external news server.

Server Port is the port number that the news server is listening on. By default news servers listen on port 119.

Username and *Password* allow you to provide login credentials if the server requires them.

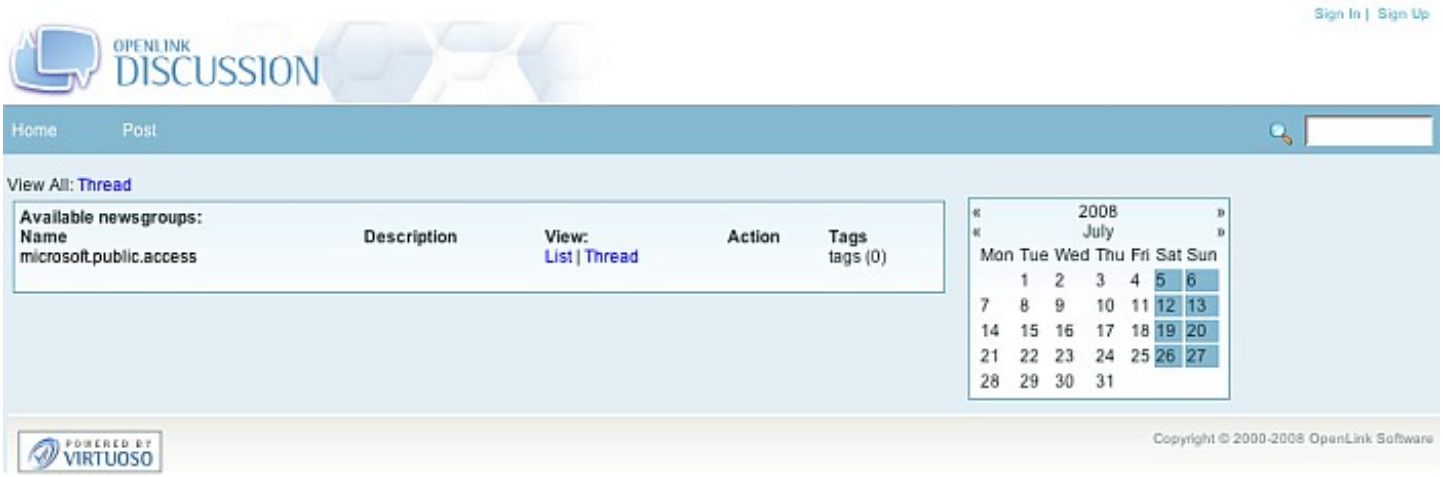
Figure 6.180. News Server Administration



Viewing Newsgroups

You can view the messages of a newsgroup either using the Conductor UI going to NNTP/NNTP Servers/News Server/News Group, or using the ODS Framework UI. In the second case you need to have the ods_framework_dav.vad package and the ods_discussion_dav.vad package installed. Then you can go from ODS to tab "Discussions" where will be shown the list of available newsgroups. Each one has shown as link its name, which leads to page where are listed its messages.

Figure 6.181. View of messages in a newsgroup in ODS Discussions.



News Message Free Text Search

You can perform Free Text Search using the ODS search option.

6.3. Virtuoso Cluster Operation

Abstract

This chapter describes setting up and operating Virtuoso on a cluster of computers. The section on Virtuoso cluster programming documents the SQL extensions specific to cluster application development.

These sections apply to Virtuoso as of version 6.0.

Clustering primarily offers greatly increased scalability for large databases without requiring application changes. The database is divided over a number of servers, of which all provide transparent access to the same data.

6.3.1. General

Virtuoso can be run in cluster mode where one logical database is served by a collection of server processes spread over a cluster of machines.

The cluster's composition is declared in a cluster.ini file which is to be in the starting directory of each of the servers composing the cluster. This file declares the hosts and listening ports of all processes composing the cluster and which of these processes is the local process and which the master.

A cluster has a single master process which is the only one allowed to run DDL operations and which is responsible for distributed deadlock resolution. In all other respects, all server processes of the cluster are interchangeable.

The set of processes declared in the cluster.ini files is called the physical cluster.

Each cluster server process has its own database and log files and is solely responsible for these. All configuration fields in virtuoso.ini and related files apply to the process whose ini file this is and their meaning is not modified by clustering.

Specifically, the SQL client and HTTP and other listening ports of each process are declared as usual and are used as usual. A cluster server process has additionally a cluster listening port that is used for cluster communications. This may not be connected to by anything except other processes of the same physical cluster. The cluster listener ports of all processes are declared in cluster.ini and all processes must specify the same information.

cluster.ini fields

The below is a sample cluster.ini file declaring a physical cluster of 4 processes.

```
[Cluster]
Threads = 100
ThisHost = Host1
Master = Host1
ReqBatchSize = 100
BatchesPerRPC = 4
BatchBufferBytes = 20000
LocalOnly = 2

Host1 = box1:2222
Host2 = box2:2223
Host3 = box3:2224
Host4 = box4:2225

Host1-1 = box1-1:12222
Host2-1 = box2-1:12223
Host3-1 = box3-1:12224
Host4-1 = box4-1:12225
```

The lines Host1 ... Host4 declare the listening ports of each process. The line ThisHost = 1 declares that this process is Host1, hence cluster listener at box1:2222 box1 - box4 and box1-1 - box4-1 are machine names that must be resolvable in the local context. IP numbers can also be used. Mentioning a host several times declares additional interfaces for the host. Any of these interfaces may be used for cluster connection to the Virtuoso server at the host. Thus Host1 = gives the first interface, Host1-1 the second and so on. This is useful since servers most often have multiple network interfaces and Virtuoso balances the traffic among these interfaces if multiple interfaces are provided. Each host will listen at all the host:port numbers mentioned and other hosts will decide which interface to use based on load.

The Threads line gives the maximum number of threads that will be made for serving requests from other hosts of the cluster. This is in addition to any other threads reserved in any other ini files.

The Threads line gives the maximum number of threads that will be made for serving requests from other hosts of the cluster. This is in addition to any other threads reserved in any other ini files.

The other fields should be left at the values shown.

6.3.2. Setting up a Cluster

To make a new empty clustered Virtuoso database, set up the individual instances. The processes must be of exactly the same version, the operating systems, byte orders or word lengths of the participating machines do not have to match.

Each individual database is assigned its database files and other configuration by editing its virtuoso.ini file. No installation besides having the Virtuoso executable and the virtuoso.ini file is needed.

If there is a Virtuoso installation on the machine, use the executable from that and set up an empty directory with the virtuoso.ini file.

Once the individual database directories are set up, write the cluster.ini file as shown above. Copy this into each running directory beside the virtuoso.ini file. Edit each to specify which host the file belongs to. Set ThisHost to point to the host which the this file belongs. Set Master to point to one of the hosts and make sure each file has the same master and a distinct ThisHost.

Start the server processes. This will initialize the empty databases.

Connect to the master host's SQL port with a SQL client to continue. The initial default user and password is dba. To verify that the cluster nodes can connect to each other, do status (''); twice. The second status should have a line beginning with Cluster xx nodes.

6.3.3. Using Clustering with an Existing Database

The procedure for converting a single server database into a clustered one will be specified later. For 6.0, there is no automatic way of doing this.

One can take an existing database and convert it to clustered operation by copying the initial database to each node of the cluster. Set up the database so that each node would run an identical copy. Then make the cluster.ini files.

Start the servers. Connect to the master and run the partitioning statements for all keys of all tables that are to be managed by the cluster.

At this point, each database will also hold rows that are not its responsibility, unless of course all tables are declared as replicated. The rows that are present and do not fall in the partition managed by the host should be deleted.

This can be done with a stored procedure to be supplied later

6.3.4. Partitioning

All databases in a cluster share precisely the same schema. Any DDL operations take effect on all nodes simultaneously.

The tables can be of one of three types:

- Partitioned: A row of a partitioned table resides in exactly one partition, according to the values of its partitioning columns. Each partition is stored at least once in the cluster but may be kept in multiple replicas if so configured. All indices of a partitioned table must specify partitioning but need not be partitioned in the same way. The data tables of an application will typically be partitioned.
- Replicated: The table and its contents exist in identical form on all nodes. Queries are answered from local data and updates go to all nodes. This is the case with schema tables, for example but can be used for application tables also.
- Local: The table exists on all nodes but each node has purely local content and all queries and updates to the table refer only to local content. This is the case for some configuration tables with only local scope, such as specifying web service end points.

Partitioning an index means that different hosts store different parts of the index. For each partitioned index one or more partitioning columns must be declared. Also each partitioned index is always held in its totality in a logical cluster. A logical

cluster is a subset of the machines composing the physical cluster declared in cluster.ini. Usually the logical and physical cluster are the same.

The logical cluster additionally declares how partitions are to be replicated. It is namely possible to declare that a specific partition be stored in multiple identical copies.

There are two predefined logical clusters: REPLICATED and __ALL. If a table's indices specify the REPLICATED cluster in their partitioning declaration, the data will be maintained in identical copies on all nodes.

The __ALL cluster is the default logical cluster for any partitioned table. Using this, each row will go to exactly one place, balanced over the set of nodes declared in the cluster.ini file.

Basic applications do not need to declare their own cluster since the default one is most often applicable.

CREATE CLUSTER Statement

```
CREATE CLUSTER <name> DEFAULT <group>[ [,...]

group: GROUP (<host>[,...])
```

A logical cluster has a single global name and it consists of one or more host groups. Each host group is given a partition of whatever object that is stored in the logical cluster. Each host of a host group replicates the partition assigned to this host group.

One logical cluster is predefined. It is called replicated and it consists of one group which has all the hosts of the physical cluster. The replicated logical cluster is used for storing any schema tables. This causes all schema information to be identically stored on all nodes of the physical cluster.

If a table is created on a clustered Virtuoso and no partitioning is declared, the table exists on all nodes with independent content on each. This is generally not desirable since the same query will return different data depending on which host runs it.

For performance, it is best to replicate any short, seldom changing lookup tables on all hosts.

Example

```
create cluster C2 default group ("Host1"), group ("Host2"), group ("Host3"), group ("Host4") ;
```

This would declare a logical cluster identical to the default __ALL cluster if the cluster.ini specified hosts 1 - 4.

The REPLICATED cluster could be declared as follows:

```
create cluster C3 default group ("Host1", "Host2", "Host3", Host4");
```

ALTER INDEX and CREATE INDEX Statements and Partitioning

The ALTER INDEX statement is used for declaring the partitioning of an index. For a non-primary key index, the corresponding CREATE INDEX can also declare the partitioning. If one index of a table is partitioned, all indices of the table must be partitioned. If no partitioning is declared, the table will exist on all nodes but will have independent content on each. Partitioning of a key must be set when the key is empty. Thus, to create a partitioned table, first create the table and declare partitioning for its primary key.

The name of the primary key index is the same as that of the table. If the table has no explicit primary key, it has an implicit one, named after the table and having the invisible _IDN column as single key part. This may be used as a partitioning key.

If clustering is not enabled, partitioning can still be declared but it will have no effect. Thus a single application DDL file can be used for clustered and single process versions of the application schema.

```
ALTER INDEX <index-name> ON <table-name> PARTITION [CLUSTER <cluster-name>] (<col-spec>[,...])

col-spec : <column-name> <type> [<options>]

type: INT | VARCHAR

options: (<mask>) | (<length>, <mask>)
```

The PARTITION declaration may occur at the end of a create index statement. This causes the index to be created, partitioned and then filled. Otherwise it would not be possible to add indices to non-empty tables.

All or part of a partitioning column's value can be used for calculating a index entry hash which then determines which host group of the logical cluster gets to store the index entry. There are two types of hashing, integer and varchar. Integer applies to integer like types such as integer and bigint and iri id and varchar applies to anything else. Floating point columns or decimals should not be used for partitioning. Large objects or UDT's cannot be used for partitioning.

For an integer partitioning, the mask is a bitmask applied to the number before extracting the part that is used for the hash. A mask of 0hexffff00 will use the second and third least significant bytes for hashing, thus values 0-255 will hash to the same, values 256-512 to the same and so on. The value 0hex1000000 will again hash to the same as 0.

Having consecutive integers hash to the same will cause them to go to the same host group and become physically adjacently which is good for key compression. If no mask is specified 0hexffff is used, meaning that each consecutive number gets a different hash, based on the low 16 bits of the number.

For a varchar partitioned column, the default is to calculate a hash based on all bytes of the string. For purposes of key compression, it may be good to put strings with a common prefix in the same partition.

The option consists of two integers, the length and the mask. If the length is positive, the length first characters are used for the hash. If the string is shorter than length, all characters are used. If the length is negative, we take the absolute value of the length and use all bytes of the string except the length last ones. If the string is shorter than -length, all the bytes are used. A length of -1 means to use all bytes except the last one, a length of 2 means to use the 2 first characters only.

The string's hash value is a large integer. The mask controls how many bits of this hash are used for the hash of the index entry.

Example

```
create table part (id int, code int, str varchar);
alter index part on part partition (id int (0hexffff00));

create index str on part (str) partition (str varchar (-1));
-- for the primary key, hash all values differing in the low byte together.
-- for str, hash all values differing only in the last character together.

create table part_code (code int primary key, description varchar);
alter index part_code on part_code cluster replicated;
```

This declares a lookup table for describing the values of the code column of the part table. This is replicated on all nodes of the cluster. Note that no partitioning columns need be specified since no matter the partition key the row would end up on all nodes regardless.

6.3.5. Transactions

A Virtuoso cluster is fully transactional and supports the 4 isolation levels identically with a single server Virtuoso. Transactions are committed using single to two phase commit as may be appropriate and this is transparent to the application program.

Distributed deadlocks are detected and one of the deadlocking transactions is killed, just as with a single process.

Transactions are logged on the cluster nodes which perform updates pertaining to the transaction.

A transaction has a single owner connection. Each client connection has a distinct transaction. From the application program's viewpoint there is a single thread per transaction. Any parallelization of queries is transparent.

For roll forward recovery, each node is independent. If a transaction is found in the log for which a prepare was received but no final commit or rollback, the recovering node will ask the owner of the transaction whether the transaction did commit. Virtuoso server processes can provide this information during roll forward, hence a simultaneous restart of cluster nodes will not deadlock.

Performance Considerations

A lock wait in a clustered database requires an asynchronous notification to a monitor node. This is done so that a distributed deadlock can be detected. Thus the overhead of waiting is slightly larger than with a single process.

We recommend that read committed be set as the default isolation since this avoids most waiting. A read committed transaction will show the last committed state of rows that have exclusive locks and uncommitted state. This is set as DefaultIsolation = 2.

In the parameters section of each virtuoso.ini file.

Row Autocommit Mode

Virtuoso has a mode where insert/update/delete statements commit after each row. This is called row autocommit mode and is useful for bulk operations that need no transactional semantic.

The row autocommit mode is set by executing log_enable (2) or log_enable (3), for no logging and logging respectively. The setting stays in effect until set again or for the duration of the connection. Do not confuse this with the autocommit mode of SQL client connection.

In a clustered database the row autocommit mode is supported but it will commit at longer intervals in order to save on message latency. Statements are guaranteed to commit at least once, at the end of the statement.

A searched update or delete statement in row autocommit mode processes a few thousand keys between commits, all in a distributed transaction with 2PC. These are liable to deadlock. Since the transaction boundary is not precisely defined for the application, a row autocommit batch update must be such that one can distinguish between updated and non-updated if one must restart after a deadlock. This is of course not an issue if updating several times makes no difference to the application.

Naturally, since a row can be deleted only once, the problem does not occur with deletes. Both updates and deletes in row autocommit mode are guaranteed to keep row integrity, i.e. all index entries of one row will be in the same transaction.

Naturally, since a row can be deleted only once, the problem does not occur with deletes. Both updates and deletes in row autocommit mode are guaranteed to keep row integrity, i.e. all index entries of one row will be in the same transaction.

A row autocommit insert sends all keys of the row at once and each commit independently. Hence, a checkpoint may for example cause a situation where one index of a row is in the checkpoint state and the other is not.

Thus, a row autocommit insert on a non-empty application table with transactional semantic is not recommended. This will be useful for bulk loads into empty tables and the like, though.

6.3.6. Administration

The cl_exec function

All administrative operations other than data definition take effect on the node to which they are issued.

```
cl_exec (in cmd varchar, in params any := NULL, in is_txn := 0)
```

The cl_exec() SQL function can be used for executing things on all nodes of a cluster.

The cmd is a SQL string. If it contains parameter markers (?), the params array is used for assigning values, left to right. If is_txn is 1, the cl_exec makes a distributed transaction and does not automatically commit on locally on each node. Thus cl_exec() can be used as part of a containing distributed transaction.

Example

```
cl_exec ('shutdown')
--will shut all nodes.

cl_exec ('dbg_obj_print (?)', vector ('hello'));
--will print hello to the standard output of all the processes of the cluster.
```

Any recovery, integrity checking, crash dump or similar can be done node by node as with single processes.

See Also:

See Also:

- Cluster Installation and Config
- Backup and Restore Example

Status Display

The status () function has a cluster line right below the database version information. This line shows cluster event counts and statistics between the present and previous calls to status. Calling status ('cluster') will show this line only. Calling status ('cluster_d') shows this line and below it the same line with data on each individual host in the cluster.

If cluster nodes are off-line, the nodes concerned are mentioned above the cluster status line.

The line consists of the following fields.

```
Cluster 4 nodes, 4 s. 9360 m/s 536 KB/s 117% cpu 0% read 44% clw threads 2r 0w 1i buffers 1939 766 d 0 w
Cluster 4 nodes, 4 s. 9360 m/s 536 KB/s
```

This first group gives the network status. The count of nodes online (4), the measurement interval, number of seconds since the last status command (4 seconds). The m/s is the messages per second, i.e. 9360 single messages sent for intra-cluster purposes per second over the last 4s. The KB/s is the aggregate throughput, i.e. the count of bytes sent divided by the length of the measure,measurement interval. This allows calculating an average message length. Only intra-cluster traffic is counted, SQL client server and HTTP connections are not included.

```
117% cpu 0% read 44% clw threads 2r 0w 1i
```

This group gives the process status. The CPU% is at 100% if one thread is running at one full CPU. The maximum CPU% is 100 times the number of CPU's in the cluster. Differences between CPU's are not considered. The read % is the sum of real time spent waiting for disk divided by the time elapsed. The maximum number is 100 times the peak number of threads running during the interval. 500% would mean an average of five threads waiting for disk times during the interval. The clw% is the sum of real time a thread has waited for cluster request responses during the period. The maximum is 100% times the peak number of threads running.

The threads section (2r 0w 1i) is a snapshot of thread state and means that 2 threads are involved with processing, 0 of these is waiting for a lock and 1 is waiting for network I/O.

```
buffers 1939 766 d 0 w 0 pfs
```

This is a snapshot of the database buffers summed over all nodes. 1939 used for disk caching, 766 dirty 0 wired down.

The pfs number is the total number of page faults during the interval summed over the cluster. This provides a warning about swapping and should be 0 or close at ll times.

6.3.7. Cluster Network Diagnostics and Metrics

Proper cluster operation requires that each process in the cluster be capable of initiating a connection to any other process. This may be prevented by firewall settings or the like. If a connection can be initiated from host 1 to host 2, it does not follow that host 2 can initiate a connection to host 1. These situations can lead to intermittent errors. These errors can be difficult to pinpoint since operations from host 2 to host 1 can work for most of the time if there is a connection available that was already established by the other host.

To check point to point connectivity, do the following on each host in turn, with no other activity on the cluster:

Log in to the SQL port of the host.

```
SQL> cl_reset ();
SQL status ('cluster');
SQL status ('cluster');
```

The first status ('cluster') may show no samples if this is the first time it is called. At the second call you should see a status line that does not contain mentions of any host being down.

The cl_reset function disconnects any connections to other cluster hosts from this host. This makes sure that a fresh connection will be started for the status command.

net_meter utility

The net_meter utility is a SQL stored procedure that measures the aggregate throughput of a cluster network with different types of workload.

First load the netmeter.sql file on the master node of the cluster.

```
SQL> load netmeter.sql;
```

Then run

```
SQL> net_meter (1, 1000, 1000, 1);
```

This returns a single result row with two numbers: The count of round trips per second and the throughput in megabytes per second.

```
net_meter ( in n_threads int,
            in n_batches int,
            in bytes int,
            in ops_per_batch int)
```

This SQL procedure runs a network test procedure on every host of the cluster. The network test procedure sends a message to every other host of the cluster and waits for the replies from each host. After the last reply is received the action is repeated. This results in a symmetrical load of the network, all points acting as both clients and servers to all other points.

The parameters have the following meaning:

- *n_threads*

  - The number of network test instances started on each host. A value of 4 on a cluster of 4 hosts would result in a total of 16 network test procedures spread over 4 processes.
- *n_batches*

  - The number of message exchanges done by each network test procedure. A message exchange consists of sending one request to every other host of the cluster and of waiting for all to have replied.
- *bytes*

  - The number of bytes sent to each host in each message exchange. The reply from each host has the same number of bytes.
- *ops_per_batch*

  - This causes each message batch to contain several operations. In practice this is a multiplier on the number of bytes.

cl_ping

```
cl_ping ( in target_host int,
          in n_pings int,
          in bytes_per_ping int)
```

This built-in function measures raw point to point network throughput. Whereas net_meter includes a more complex n to n point traffic pattern and scheduling of functions on multiple threads, cl_ping does not involve anything except a process to process connection and no thread switching, transaction contexts or other overhead.

cl_io_report

This built in function prints out a summary of the cluster connections of the host on which it is run. The output goes to the server process' standard output. This lists the bytes in and out as well as the file descriptor numbers of any connections this host has with any other host.

6.3.8. Elastic Cluster Operations

Initializing

When a set of Virtuoso servers configured as a cluster is first started, the elastic cluster setup is automatically performed if the cluster.ini of each host contains the [ELASTIC] section as described below. This is the normal way in which an elastic cluster is initialized.

The *__elastic* option of the create cluster statement is used for creating an elastic logical cluster.

```
create cluster ELASTIC __elastic 2048 4 group (Host1), group (Host2), group (Host3), group (Host4);
```

or:

```
create cluster ELASTIC __elastic 2048 4 ALL;
```

This creates an elastic cluster called ELASTIC which initially consists of 4 host groups. Each host group has a single copy of the data allocated to the host group. The data is partitioned among the host groups so that each host group has a separate fraction of the data. The second variant defaults the host groups from the cluster.ini, so that each host mentioned is in a host group of its own. The first number after the *__elastic* clause is the maximum number of physical slices in the cluster. The second number is the number of physical slices initially created per host group. In general each host group should have as many slices as there are hardware threads on the CPU (s) running the host(s) in the host group.

The name ELASTIC is specially known. If a logical cluster of this name exists, it is the default logical cluster instead of the non-elastic *__ALL* cluster in create or alter statements that specify partitioning.

Database files for an elastic cluster are created in a directory and striping set that is specified in the cluster.ini file of each host. The cluster.ini file of each participating host must have this section at the time of executing the create cluster statement. The section of the ini file has the name as in the create cluster statement, e.g.:

```
[ELASTIC]
Slices = 16
Segment1 = 1024, /data6/dbs/btc1-el.db = q1
Segment2 = 1024, /data2/dbs/btc1-el.db = q2
Segment3 = 1024, /data8/btc1-1/btc1-el.db = q3

[ELASTIC_1]
Segment1 = 1024, /data2/dbs/btc1-d-el.db = q2
```

Note that an elastic cluster may specify multiple segments. Each segment has an independent striping, as in other segments. When initialized, one segment is created for each of the slices hostted by the host in question. The segment may consist of multiple files if striping is specified. The database files for the segments are named as per the path and file name in the segment clause, with the slice number appended at the end, prefixed with a '.'. The files are initially created in the first segment of the logical cluster. If one wishes to move the files, one can do so between the segments declared for the cluster when the server is not running. At the next startup, it will look in all the segments for slices matching the path name plus .[0-9]*.

Specifying distinct IO queues for distinct devices is useful, as with other file system layouts.

To move slices between servers, more steps are necessary, see following sections.

In the example at hand, we specify two sets of files, one labeled ELASTIC_1. This is useful if one wishes to divide objects among different types of storage, e.g. disk and ssd. If two cluster names differ only by having _[0-9]* tagged at the end or having _[0-9]* with different numbers at the end while the rest of the name is the same, the system will manage these as twins so as to keep the slices belonging to either collocated. So slice movement and splitting will be mirrored between the two. If an If two objects share a partitioning key between such twin clusters operations are assumed to be colocatable.

The Slices setting specifies how many physical slices are initially created per host group. For example if a server has 2 CPU sockets with a 8 core 16 thread CPU on each, and two Virtuoso processes are run on each server, one per NUMA node (CPU socket), it is practical to specify 16 slices per host group. In this way there will be up to 32 threads per query on the machine, leading to full platform utilization.

After creation, the cluster can be used in the cluster clause of create index and alter index for specifying that the object in question

is to be created in the elastic cluster.

```
create table TE ( row_no int,
                  string1 varchar,
                  string2 varchar,
                  primary key (row_no));
alter index TE on TE partition cluster ELASTIC (row_no int (0hexffff00));
```

The table TE is now allocated in the cluster ELASTIC. Due to default behavior, the cluster ELASTIC is redundant since objects preferentially get created in it if it exists. The table is now ready for use. Note that the bit mask in the partitioning clause of the partitioning key row_no must specify enough bits to cover the number of logical slices in the cluster, 2048 in the present case. If this is not the case, some logical slices will never be used, as there will be less distinct partitioning hash values than logical slices.

Adding and Removing Hosts

The number of server processes (hosts) used for managing an elastic cluster may vary during the cluster's lifetime. If the elastic cluster keeps partitions in multiple copies, i.e. host groups in the initial create cluster statement have more than one host, any new host groups that are added must have the same number of hosts. Each new host of a new host group must be identically initialized to contain a copy of the REPLICATED logical cluster of the cluster into which the hosts are added. This is done by simply copying the non-elastic database file(s) of any host of the existing cluster. The copy can be made while the cluster is running.

```
cl_add_host_group (in cl_name varchar, in hosts any array) returns int
```

This function adds a host group to an elastic cluster. The host group must contain the same number of hosts as all the other host groups and the hosts must not be part of any other host group in this cluster.

The new hosts must be assigned IP addresses in the cluster.ini file of all presently running hosts. At the time of being added, the hosts in the new host group must be running and must be initialized to share a copy of the REPLICATED cluster shared by all hosts of the physical cluster.

After successful addition:

```
cl_remove_host_group (in cl_name varchar, in host varchar)
```

6.3.9. Setting CPU Affinity

If one runs multiple Virtuoso processes on a physical machine it may be advantageous to bind the processes to a physical CPU. In this way one may be certain that memory allocated for the processes is local to the CPU on which the process runs. This is specially important for servers with more than 2 CPU sockets, as there can be high variability in memory access latency Even with dual-socket Intel servers there caan be performance gains of up to 20% from setting CPU affinity if the workload is well balanced. On the other hand if the workload is not balanced, binding a process to a CPU may lead to less platform utilization than not specifying affinity.

There are two settings in the virtuoso.ini for affinity:

1. *Affinity*

   is the overall affinity for threads of the server process and is specified by comma-separated CPU numbers or ranges of numbers.
2. *ListenerAffinity*

   is the affinity of the thread handling incoming cluster traffic. In the interest of low latency, it may be advantageous to bind the listener thread to a core of its own, so that it will never be preempted. This can decrease the incidence of blocking on write for other processes since incoming messages will be read at full interface speed all the time.

For example:

```
...
Affinity = 1-7, 16-23
ListenerAffinity = 0
...
```

specifies that the process is bound to CPU 0 of a dual socket 16 core 32 thread system.

See numactl -- hardware for the actual numbers of the cores on each CPU. CPU 0, i.e. first thread of first core is here exclusively allocated to network traffic and canbnot be preempted by any other thread running for the server process.

6.4. Virtuoso Cluster Fault Tolerance

Abstract

This chapter discusses fault tolerance and how to configure it. The following aspects are covered:

- Setting up a fault tolerant logical cluster inside a physical cluster. Creating tables and indices for fault tolerance.
- Interpreting status and error messages and managing failures and recovery.
- Optimizing a schema for fault tolerance: For read-intensive workloads, the work can be profitably split among many copies of the same partition.
- RDF specifics relating to fault tolerance.
- Splitting a cluster so that one copy of the partitions does bulk load while another serves online queries.

6.4.1. Introduction

Virtuoso Cluster supports optional fault tolerance by storing partitions in more than one copy if desired. This is a cluster-only feature, providing for transparent fail-over with fully transactional semantics. This feature is in no way related to the other forms of replication discussed in the Virtuoso documentation. This feature can be used for load balancing of read-intensive workloads and for fault tolerance of arbitrary workloads in a tightly coupled cluster. This feature is not suited for synchronizing geographically distributed copies.

Fault tolerance is enabled at the level of logical cluster. Making the logical cluster __ALL fault tolerant has the effect of making all the normally non fault tolerant database objects into fault tolerant ones.

6.4.2. Sample Configuration

We will take a minimal example of a fault tolerant setup with 4 server processes, grouped in two groups of two mutually mirroring servers. The word host here refers to a single server process. How these are distributed over physical hardware is a separate question. Each host (i.e. server process) has exclusive control over its database files. Two processes may not share files.

```
create cluster DUP default group ("Host1", "Host2"), group ("Host3", "Host4");
```

Each group clause in the statement defines a set of mutually replicating, interchangeable processes. The cluster is operational as long as at least one process for each group is available. If all the processes in one group are down, the tables created in the cluster will not be available in their entirety. Even if some fragment of a table were unavailable, the remaining fragments are still available for transactions that concern only them.

For all tables or indices created in a cluster with fault tolerance, partitioning is applied for determining which group of the groups listed in the create cluster statement gets which individual entry. After this, all the hosts that make up the group are guaranteed to hold a copy of said entry at the commit of each transaction.

Regardless of the definition of logical clusters, there are global functions at the level of the physical cluster which need to be replicated for fault tolerance. For these functions include resolving distributed deadlocks and allocating sequence ranges. See the discussion of sequences in the cluster programming guide for more on this. These global functions are handled by a single process called the master. To keep a standby master that is synchronously kept in sync with the first master, one can define multiple master processes, as follows:

```
Master = Host1
Master2 = Host2
```

These lines in the cluster.ini files of the servers constituting the cluster mean that if Host1 is available, it will perform the functions of the master and if it is not available, these functions go to Host2. If both are available, then Host1 does the work and synchronously updates Host2 before returning the results to the requesting host.

To create a table or index in a specific logical cluster, one uses the cluster option in alter index or create index. For example:

```
create table T1 (row_no int primary key, string1 varchar);
alter index  t1 on t1 partition cluster DUP (row_no int (0hexffff00));
```

```
create index string1 on t1 partition cluster DUP (string1 varchar (5));
```

These statements define that t1 will be kept in duplicate copies spread as declared for logical cluster DUP. Partitioning can be altered only when the table concerned is empty. To make an existing non-replicated table into a replicated one, use alter table rename, as follows:

```
create table T1 (row_no int primary key, string1 varchar);
alter index  t1 on t1 partition (row_no int (0hexffff00));
create index string1 on t1 partition  (string1 varchar (5));
```

The table is created in the default logical cluster, which by default is not replicated. Now fill the table with a large amount of data. Then do the move over to replicated storage with minimum effect on overall server availability, follow the below steps:

```
drop index string2;
alter table t1 rename t1_old;

create table T1 (row_no int primary key, string1 varchar);
alter index  t1 on t1 partition cluster DUP (row_no int (0hexffff00));
create index string1 on t1 partition cluster DUP (string1 varchar (5));

log_enable (2);
```

This turns on row autocommit and disables logging for the session. This is necessary, as otherwise the statements below will abort due to running out of rollback space if the table is large. Disabling logging also saves some extra time.

```
insert into t1 select * from t1_old;
delete from t1_old;
drop table t1_old;
```

First deleting the contents and then dropping the table shortens the global atomic section that corresponds to dropping the table. Otherwise all servers would be unavailable also for the time of deleting the content, which might take long.

Finally:

```
cl_exec (checkpoint');
```

Makes the operation permanent. All the above work would be lost in the event of any failure since it was done without logging.

```
log_enable (1);
```

Restores default transaction and logging behavior to the session.

If T1 were very large, e.g. 100's of G or more, then one could do checkpoint after each step so as not to keep a full copy of all indices of t1 in the old and new versions simultaneously. Dropping an index or deleting rows actually frees the space at the next checkpoint. One could also write a procedure for copying the table in parts and run many such copies in parallel for different parts of the table. This would have obvious advantages for moving terabytes of data.

6.4.3. Transactions

Replication of partitions is entirely synchronous and transactional, with two phase commit. Replicated partitions show serializable semantics insofar the transactions dealing with replicated partitions are in serializable isolation.

These processes are transparent to the developer and administrator. One can program as one would in the case of a non-clustered database. The below has bearing mostly on optimization.

A read with read committed isolation can be done in any copy of a partition. Read committed reads make no locks nor do they wait for locks. For uncommitted updated rows, they show the pre-update state. Thus read committed can be freely load balanced over all copies of a partition.

A read with repeatable or serializable isolation will always be done on the first copy of a partition. This is done so as to have proper serialization of locking. If a row is locked for repeatable read and another transaction wishes to lock it for update, the latter will wait. Thus, transactions with locking resolve locking issues on the first copy of each partition. The first copy is the copy of the partition held by the first process mentioned in the partition's group clause of a create cluster. If the process is not available, the role of the first copy falls on the second, third and so on. If no host of the hosts mentioned in the group clause is available, the

operation cannot be performed. We will later define what we mean by available.

Updates to replicated partitions are performed concurrently on all the copies of the partition. Updates are replicated as they are made, not at time of commit, hence there is almost no added latency arising from keeping replicated partitions: When one is ready to commit, all are.

Distributed deadlock detection is handled by the master host. In the event of its failure, the function is taken over by the first node to be available in the sequence of master succession, as indicated above.

6.4.4. Dividing Virtuoso Hosts Over Physical Machines

This section describes how Virtuoso hosts should be placed on actual machines for optimal balance and fault tolerance.

All hosts (Virtuoso processes) constituting a cluster should be of the same size. This means that most importantly they should have an equal amount of memory allocated in virtuoso.ini.

The situation is simplest if all physical machines are of the same spec. This is not necessary though since a larger machine can host more Virtuoso hosts (processes). For balanced resource use, the machines should however have an equal amount of memory per core.

Naturally, all hosts mentioned in the same group clause in a create cluster statement must reside on different physical hardware. This is also true of the master host list in the cluster.ini file. Putting them on the same machine or a different virtual machine on the same machine defeats the whole point of fault tolerance. If VM's are automatically balanced on a data center network, one should keep the above in mind. It is recommended to use real machines with real network interfaces for database.

In each host group, the first host gets some more load than the other hosts. This is because serializable reading always must go to the first in the group. Thus, the first hosts should be evenly spread over the hardware, so do not put all the group firsts on the same machine.

Supposing 2 machines with 8 cores each, one would have hosts 1-16, 1-8 on the first, and 9-16 on the second. For analytics, have one core per host, for OLTP, one can have two or four cores per host.

In this case, one would write the create cluster as follows:

```
create cluster XX group ("Host1", "Host9"), group ("Host10", "Host2"), group ("Host3", "Host11"), group (
  group ("Host5", "Host13"), group ("Host14", "Host6"), group ("Host7", "Host15"), group ("Host16", "Host
```

Both machines have 4 firsts and four seconds. One could vary the memory allocation per process so as to have maybe 20% more RAM per host for a first of group than for the others in the group. This may optimize the situation when all are online and will not excessively penalize the fallback position.

Varying the amount of memory depending on whether a host is first or second makes sense only for read intensive workloads. Dividing firsts and seconds evenly over the hardware makes sense for all workloads.

6.4.5. Managing Availability

This section concerns prerelease 3117 amd onwards and is not final. Later versions have higher level management features but the primitives discussed here continue to apply.

In its normal state, a cluster has all the constituent processes up and all state is kept synchronous.

When a host unexpectedly disconnects, the following takes place:

- All transactions which have a write affecting this host become uncommittable. The application will see this immediately, as soon as it does anything within the transaction.
- All work proceeding at the request of the failed host on other hosts is aborted.
- All remaining network connections to the failed host are disconnected.

If a query was proceeding and it had state on the failing host, the failure will be reported to the client of the query and the query will be aborted. A subsequent query, if in read committed isolation, will automatically avoid the failed host and use surviving ones. Thus, the application sees a failure as a retryable abort of a transaction or query.

For update transactions, if all copies of a partition are not online, the update cannot be made. In order to allow proceeding with updates even after a failure, the failed host must be declared removed. This means that if it were to come back on, it would not get any updates or queries from the other hosts until it was explicitly admitted back into the cluster.

In version 6.00,3116, enabling updates when all hosts are not online must be done manually. In other words, read only work will proceed uninterrupted but updates will be prohibited if all hosts are not online. Read balancing and re-enabling updates when all hosts have rejoined the cluster is done automatically.

In order to declare that a host has for the time being left the cluster or has returned to the cluster after having left it, one uses the function cl_host_enable ().

For example, suppose a hardware failure that takes multiple processes (hosts) offline. As long as for each there is at least one surviving host of the same group (as per create cluster), read operations proceed normally. But to re-enable writes for the time the failed hardware is replaced, the operator must inform the cluster that the failed hosts are not expected to return immediately and that no further reference to them should be made, specifically, the rest should not attempt to keep them up to date.

This is done with cl_host_enable. This is a SQL stored procedure. Log in as dba on a surviving master host and do:

```
SQL> cl_host_enable (1, 0);
```

This will abort all the transactions pending at the time and declare host 1 to be off limits to the rest of the cluster. If Host1 was playing the role of the master, the master role is automatically transferred to the next one in the succession.

The succession of master hosts is declared in the cluster.ini with the settings of Master, Master2, Master3 and so on. All cluster.ini files must agree.

After this, even though Host1 is now acknowledged offline, updates can proceed.

To rejoin a recovered host into a cluster, so as to again have an additional copy of the formerly incompletely replicated partition, one can do

```
SQL>cl_host_enable (1, 1);
```

This states that Host1 is again part of the cluster. This statement must be executed on an online master node of the cluster, thus not on Host1 itself.

Supposing that the database files of Host1 have been lost in the failure and that Host1 and Host2 were in the same group. The restore would go by taking the cluster offline, copying the database files of Host1 to Host1 and starting the database again. Then the dba would issue cl_host_enable (1) and Host1 would again be available.

To do this without downtime, one may do the following:

- Disable checkpoints on Host2: checkpoint_interval (0); Operations continue. Copy the database files of host2 to host1.
- Start host1.
- Put host2 and all hosts with which host2 occurs in the same group in read-only mode: cl_read_only (2, 1)
- copy the transaction file of host2 to host1 and replay it with replay ().
- rejoin host1 to the cluster with cl_host_enable (1, 1);
- Re-enable updates with cl_read_only (2, 0);
- re-enable checkpoint on host2 with checkpoint_interval (), setting it to its previous value. See virtuoso.ini.

Further versions perform these operations automatically. The above procedure is error prone. Do not try it unless you understand exactly why each step is made and what its effects are supposed to be.

6.4.6. Optimizing Schema for Fault Tolerance

Having the working set in memory is the single most important factor of database performance. When storing partitions in duplicate, one in principle also requires double the memory to keep adequate working set during write operations.

However, most web and data warehouse workloads are read-intensive. In this situation, the reading load can be balanced over the replicated copies. If this balancing were done at random or round robin, all copies would eventually maintain the same working set. In other words, 64G of RAM spread over two machines would behave like 32G. If the data volune is larger than memory, it

makes sense to have the different replicas cache different parts of the partition they share.

Consider, using the example of cluster DUP mentioned above:

```
create table customer (c_id int primary key, c_name varchar, c_state varchar);
alter index customer on customer partition cluster DUP (c_id int (0hexffff0000));

create table orders (o_id int primary key, o_c_id  int, o_date datetime, o_value numeric);
alter index orders on orders partition cluster DUP (o_id int (0hexffff0000));
create index o_c_id on orders (o_c_id) partition cluster DUP (o_c_id (0hexffff0000));
```

This has the effect of saying that the 16 low bits of c_id or o_id do not participate in the partition hash. The hash is made from bits 32-16. Thus c_id 0-64K will be in one partition, 64K-128K in another, 128K-192K in a third and so on, these partitions are then spread by hash over the host groups listed in the create cluster.

Now, doing the join

```
select sum (o_value) from customer, orders where c_state = 'MA' and c_id = o_c_id;
```

will take o_c_id's 0-32K from the first copy of the first partition, id's 32K-64K from the second copy of the first partition, c_o_id's 64K-96K from the first copy of the second partition and so forth.

The load is split by applying range partition on the low bits of id's, so that a system with 64G split over two replicas behaves like 64G RAM for read committed reading but as 32G of RAM for writing. This is enabled by leaving low bits of id's outside of the partition hash by specifying a mask, as shown above.

6.4.7. Interpreting Status Messages

There are special error codes and status reports dealing with cluster failures.

The status function with an argument of 'cluster_d' shows a host by host report of the cluster:

```
SQL> status ('cluster_d');
```

If all is normal, the message is as described in the cluster administration section. If some hosts are down, meaning that they do not accept network connections at the cluster port, these are first listed as being down. Then follows the summary status line and a status line for all the hosts that can be connected to.

A host being contactable over the cluster protocol does not mean that it is online or in sync with the rest.

If a physical cluster has no logical clusters that are in duplicate, there is no redundancy, except for the built in redundancy of schema.

If a host is not in the online state, an extra line in the cluster status report describes the state in more detail. The state can be one of:

- roll forward - The host is recovering from log. The count of transactions replayed to date is shown after this.
- removed - The host is considered down and no attempt is made to contact it until it explicitly rejoins the cluster. This is controlled with the cl_host_enable function.

It is possible that the host considers itself in one state and the host showing the report thinks that it is in some other state. If this is so, the status report mentions it.

Applications see cluster failures through the following SQL states:

- 08C01 - A host cannot be contacted or can be contacted but is not in the online state
- 08C02 - An operation that previously had a connection to a host finds that it no longer has the connection.
- 08C03 - A master only operation was tried on a non-master. Indicates possibility of divergent understanding of master succession. This is expected to reset itself.
- 08C04 - A write was attempted on a partition that is flagged read only, as per cl_read_only.
- 08C05 - A request was refused because the host serving the request thinks the requesting host is not admitted to cluster as per cl_host_enable, i.e. was removed and not reintroduced.

- 08C06 - A cluster operation was not made because the host thinks it is not joined to the cluster either because it has not finished roll forward or because it is marked removed by ch_host_enable.

6.4.8. Administration API

This section documents the SQL procedures used for managing failures and setting hosts on and off line. These are stored procedures for DBA group users only, all in the DB.DBA. qualifier/owner. The dba will typically not call these directly. These are intended for use by management scripts and internal functions.

```
cl_host_enable (in host_no int, in flag int)
```

The host_no is the number as in the cluster.ini Host<nn> entries. The flag is 1 for joined and 0 for removed.

This controls whether a host is excluded from operations. Only a previously excluded host can be rejoined to the cluster with this function. This is not for adding new hosts. A host will be excluded from operations if it is long term unavailable,e.g. is running on failed hardware. If the unavailability is only for the time of a restart, removing the host is not generally practical.

If a host is rejoined to its cluster, then the caller of this function asserts that the joining host is up to date. If it were not up to date one could get discrepancy between copies of partitions, which is a loss of integrity and can be hard to detect. Being up to date means, for all objects of all replicated logical clusters where this host participates, having the exact same logical content in the host's (i.e. server process') database files as in the databases of the hosts which are in the same group as the rejoining host.

This function must be called on a master host that is itself in the online state. The setting is recorded on all master hosts. All non-master hosts update their copy of this setting from the first available master in the master succession. There is always at least one master node that is in the online state. If they all are offline, then the cluster in general is unavailable and the last one cannot be removed from online state with cl_host_enable.

This function aborts all pending transactions, so that the whole cluster has no uncommitted state. This puts all the hosts that can be reached into an atomic state where they only accept operations from the master who initiated the atomic state. If the atomic state cannot be obtained within a timeout, the operation fails and can be retried. This is possible if two hosts attempt to get an atomic state and deadlock or if rollback of pending state takes longer than the timeout. Once in atomic state, all masters record the change in cluster join status and all non-masters get a notification of the change. Once all these are acknowledge and logged, the atomic section ends.

An application whose transaction was aborted in this manner will see this as a deadlock, with the 40001 SQL state and a message mentioning global atomic state.

```
cl_read_only (in host_no int, in flag int)
```

This sets the partitions of which host host_no has a copy into read only state. A flag of 1 means read only, 0 means read write. This does not abort transactions but will prevent any new updates touching the partition. Transactions with existing uncommitted state in the partition can commit. To abort all transactions first, use this with __atomic (1) first and then __atomic (0) to finish the atomic state. This is a volatile state and does not survive server restart. This is intended for use in bringing copies of partitions up to date, which is a process that would have to be retried anyway if interrupted by failure.

```
__atomic (in flag int)
```

This places the cluster in global atomic mode. A flag of 1 starts this and a flag of 0 finishes this. Row autocommit is also implicitly enabled. During such time, no host of the cluster accepts connection through web or SQL ports and only serves requests made by the transaction which started the atomic section with __atomic (1). When an atomic section starts, all transactions are aborted and are guaranteed to all be rolled back upon successful completion of __atomic (1). New transactions will also not start until __atomic (0) is called. Starting an atomic section may fail by timeout if rollback takes too long or if two competing __atomic(1) requests deadlock with each other.

```
cl_control (in host_no int, in op varchar, in new_value any := null)
```

This returns the value associated with cluster related settings. If a new value is specified the old value is returned and the new value is set.

The op specifies the setting. It is one of:

- cl_master_list - succession of master hosts as an array of host numbers. Read only.
- cl_host_list- Array of all host numbers. Read only.
- ch_group_list - List of host numbers of hosts which occur in the same group with host no. These are the hosts which share a partition with host no according to at least one create cluster statement. Read only.
- cl_host_map - String with a character per each host number up to the maximum existing host number. The character is represents the status as known by the host on which thus function is called. The new_value argument can be specified for changing this setting.
- ch_status - Returns/sets the status of host no as known by this host.
- cl_master_host - Return/set the host number used by this host for master only requests.

The status of a host is one of:

- 0 - online
- 1 - removed
- 2 - temporarily offline
- 4 - pending roll forward
- 7 - host number is not used

Cluster Control Utility

A Virtuoso cluster does not per se require a separate administration utility. It will be online and usable when all the constituent servers are online. In the event of using fault tolerance, the *clctl utility* should however be used for managing servers going offline and rejoining the cluster. This is needed in order to ensure that the operation sequences for marking a server process as failed and for bringing a failed server process back online are correctly carried out. In the following section, we use the word host to mean an individual database server process. Several of these may exist on a single computer.

clctl has a configuration file named *cl.cfg* , which it reads from its working directory. This file describes the layout of the Virtuoso cluster being managed.

This file can be filled in manually after the cluster has been set up. It is only used by *clctl* .

Before using clctl, the cl.cfg file must be created, see the cl.cfg Configuration File section below. Note that ssh is used for executing commands on remote machines. This requires ssh to be set up so as not to require manual login.

The database file names for the database files of two hosts that are each others' online replicas must be the same and there must be an equal number of stripes.

Note : *clctl* can presently be used only if the database consists of a single file plus transaction log file in the same directory. These must be named *virtuoso.db* and *virtuoso.trx* . The hostname and directory is given separately for each host in *cl.cfg* .

clctl has the following subcommands:

```
clctl start
```

Starts all the hosts of the cluster.

```
clctl start <host>
```

Starts all a given node of the cluster.

```
clctl stop
```

Shuts down all the hosts of the cluster.

```
clctl stop <host>
```

Shuts down a given node of the cluster.

```
clctl status
```

Displays a one line status summary for each host.

```
clctl ro <host>
```

Sets a node of the cluster in a read-only mode

```
clctl rw <host>
```

Sets a node of the cluster in a read-write mode

```
clctl disable <host>
```

This notifies the cluster that <host> identified by its number in the cl.cfg file has failed and that the cluster will continue normal operation without this host.

The host will have to be rebuilt based on the state of its replica before it can rejoin the cluster. This is done with the clctl rebuild command.

If there is a transient failure like an unplugged network cable or a reboot, disabling the temporarily failed host does not always make sense because rebuilding it may involve copying hundreds of GB of data which can take significant time.

After the failure, the cluster is available for reading but is not available for writing until the failed host is either back online or disabled with this command.

Example:

```
clctl disable 3
```

Means that Host3 as defined in cl.cfg is removed and will have to be rejoined to the cluster with:

```
clctl rebuild 3

clctl rebuild <host>
```

Once a host has been removed from the cluster with clctl disable, it can be brought back online with this command. The host is identified by its number in the cl.cfg file. This command automates the following operations:

- Make a log checkpoint on the online replica of the host being rebuilt. This allows the database files to be copied to serve as a basis of the database of the recovering host. During this copy the replica will not do log checkpoints, so that the database file copy is clean. During this time the cluster is available for writing.
- After the database files are copied, the recovering host is started. At this point, there is a read only window while the recovering host replays the transactions its online replica has accumulated while its database was being copied. When the replay of the replica's transaction log is complete, the recovering host is marked to be a part of the cluster and writing is re-enabled.
- To ensure integrity of the recovered cluster, these operations have to be performed in this exact sequence, hence the clctl utility should be used for this.

```
clctl enable <host>
```

Enable a node of the cluster.

The cl.cfg Configuration File

The *cl.cfg* file contains a description of the layout of the cluster. It specifies a remote exec command for starting servers and gives the locations of database files used by these servers. The database file locations are needed for copying the database for restoring a failed host.

```
[Global]
Exec     = ssh
# command for remote execution. ssh must be set up so that the machine
# from which clctl is run can connect without login prompt to the
# machines hosting the databases.

Copy     = rsync -zvP
# command for copying files between the hosts. This is executed on the
# host receiving the copy.
```

```
BinDir   = /usr/bin
# Directory for the isql and other Virtuoso commands on the host
# running clctl

Password = dba
# The Virtuoso password for the dba account. All hosts of the cluster
# have the same dba password since all login and user account information
# is replicated on all hosts. This is a Virtuoso account and no operating
# system account corresponds to it.

Master   = Host1
# The cluster master information as given in each cluster.ini file

Master2  = Host2
# Fallback master as given in each cluster.ini file.
```

For each host the following fields are given:

```
[Host1]
Host      = localhost
# DNS Name of the machine hosting  the Virtuoso host in question

DBDir     = /home/v6/binsrc/tests/suite/clflt/cl1
# The absolute path of the directory containing the database files
# of this host

BinDir    = /home/v6/bin
# Absolute path of the Virtuoso binaries

SpareHost = 2
# Online replica of this host. The rebuild command uses this
# information to decide which database files to copy when
# rebuilding this host after failure.

SqlPort   = 1111
# Port for SQL listener of this host. Corresponds to the
# ServrPort setting in this host's virtuoso.ini file.
```

Below is a sample of a full cl.cfg file for a cluster of 4 hosts all on the same machine (localhost):

Example:

```
[Global]
Exec     = ssh
Copy     = rsync -zvP
BinDir   = /usr/bin
Password = dba
Master   = Host1
Master2  = Host2

[Host1]
Host      = localhost
DBDir     = /home/v6/binsrc/tests/suite/clflt/cl1
BinDir    = /home/v6/bin
SpareHost = 2
SqlPort   = 1111

[Host2]
Host      = localhost
DBDir     = /home/v6/binsrc/tests/suite/clflt/cl2
BinDir    = /home/v6/bin
SpareHost = 1
SqlPort   = 1311

[Host3]
Host      = localhost
DBDir     = /home/v6/binsrc/tests/suite/clflt/cl3
BinDir    = /home/v6/bin
SpareHost = 4
SqlPort   = 1312
```

```
[Host4]
Host      = localhost
DBDir     = /home/v6/binsrc/tests/suite/clflt/cl4
BinDir    = /home/v6/bin
SpareHost = 3
SqlPort   = 1313
```

6.4.9. RDF Specifics

To set up fault tolerant RDF storage, one can use the template provided in the clrdfdup.sql file in the distribution.

The below applies to testing with prerelease 06.00.3116.The fault tolerance function in 6.00.3116 is provided as a demonstration of concept exclusively and is not intended for production use and has not been tested in production. The below steps will demonstrate the basic capability but one should not try things not explicitly mentioned.

The test starts with an empty database. Edit the create cluster statement in the clrdfdup.sql file to correspond to the setup at hand. Then load the file:

```
SQL> load clrdfdup.sql;
SQL> cl_exec ('checkpoint');
```

Now load data. The load will be non-transactional but now will keep two copies of each partition. Use log_enable (2) and ttlp or rdf_load_db..rdfxml as described in the relevant documentation.

After some data is loaded, do another checkpoint.

```
SQL> cl_exec ('checkpoint');
```

You may query the data. Now shut down one of the servers. Querying should remain possible as long as one host in each group is online. Start the previously stopped host and stop of another from the same group. Querying remains possible.

Do not try non-transactional loading when all hosts are not online. This produces incorrect results. Do not stop hosts during non-transactional loading. This also produces inconsistent results.

Transactional loading is safe for stopping servers during loading but will stop the loading with an error. The removed host must be either brought back online or removed with cl_host_enable for the loading to proceed.

Do not test transactional RDF loading with 6.00.3116. This version is tested for duplicate partitions with SQL data but incompletely with RDF.

6.4.10. Fault Tolerance Programming

This section describes aspects of fault tolerance in writing cluster aware SQL applications. Specifically, partitioned functions, which are a way of explicitly dividing procedural execution among hosts of a cluster, have issues and features that are specific to fault tolerance and must be treated separately.

In using a daq or dpipe, one can specify whether the function is to be partitioned like:

1. Read committed read - low bits not used for partition can be used for intra partition balancing, as described in the schema optimization section.
2. The function is called on all replicas, as an update.
3. The function is called on the first replica, like a read for update
4. The function is called on all but the first copy of the partition.

Combinations of 3 and 4 can be used if the function, for example, allocates sequence numbers which must be then replicated over the remaining copies. Thus the function that allocates the new sequence number is called with in mode 3 and another function that uses this number is called in mode 4.

For a daq_call call, these options are specified in the 5th argument, flags.

- 0 - read committed

- 1 - write all
- 2 - write first
- 3 - write all but first

For dpipes, this is stated in the dpipe_define call's flags argument. The values to be or'ed over the flags are:

- 0 - read committed
- 1 - update all
- 4 - update first copy
- 8 - update all but first copy.

Chapter 7. Data Access Interfaces

Abstract

This chapter covers installation, configuration, and utilization of the data access drivers / providers (ODBC, JDBC, OLE-DB, and ADO.NET) that comprise the Virtuoso Client Connectivity kit.

7.1. ADO.Net Data Provider

7.1.1. Introduction

Virtuoso includes an ADO.NET 2.x & 3.x data providers compatible with Visual Studio 2008 and Entity Frameworks, that provides access to native Virtuoso data (SQL, XML, and RDF) in addition to any Virtuoso Linked Tables from external ODBC

and JDBC accessible data sources. Known-compatible external data sources include Oracle (versions 7.x to 11.x) , Microsoft SQL Server (6.x to 2005) , IBM DB2 , Sybase (4.2 to 15.x) , IBM Informix (5.x to 11.x) , Ingres (6.4 to 9.x) , Progress (7.x to 10.x) , MySQL , PostgreSQL and Firebird .

Benefits

This provider equips Microsoft .NET based applications, development environments, and programming languages with conceptual entity-based access to native and heterogeneous data sources.

Features

- ADO.NET 3.5 compliance
- .NET Entity Frameworks compatibility
- Full integration with Visual Studio 2008
- ADO.NET Data Services compatibility
- LINQ to Entities compatibility
- High-Performance & Scalability
- High Security
- Support for Native and 3rd party ODBC and JDBC accessible RDBMS engines and Web Services
- Tested against all major .NET applications and development environments
- Support for Microsoft Siliverlight versions 2 and 3
- Support for .Net RIA Services

7.1.2. Installation & Configuration

There are two installation types to consider:

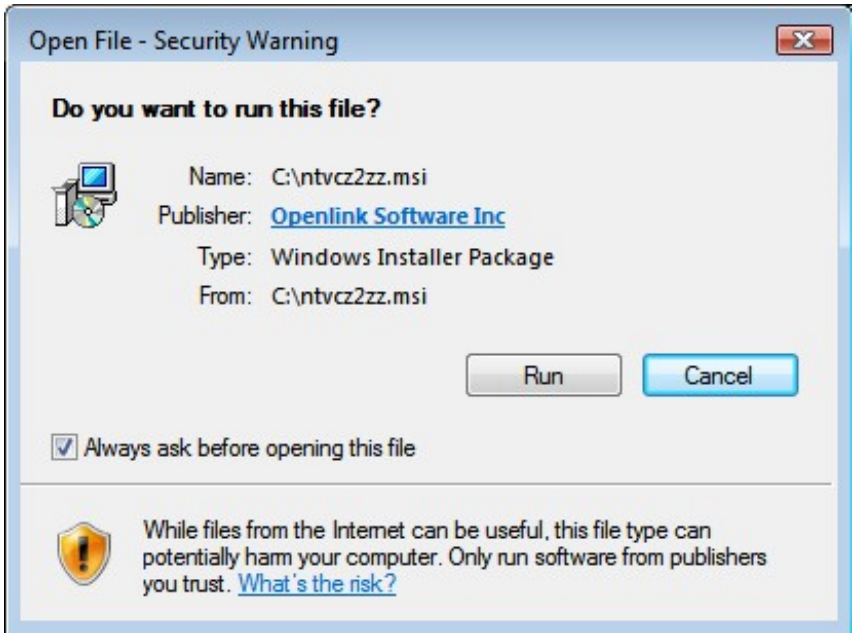
- If you have an existing Virtuoso instance, or will install Virtuoso on a different host, you only need to install of the ADO.Net Provider Client on the Visual Studio host.
- If you have no existing Virtuoso instance, you can install both the ADO.Net Provider Client and the Virtuoso Universal Server on the Visual Studio host.

Installation of the ADO.Net Provider Client on Windows

The Virtuoso ADO.Net Provider is part of the Virtuoso Client Connectivity Kit installer, and if the target Virtuoso Server installation already exists on the network this is the only component that needs to be installed.

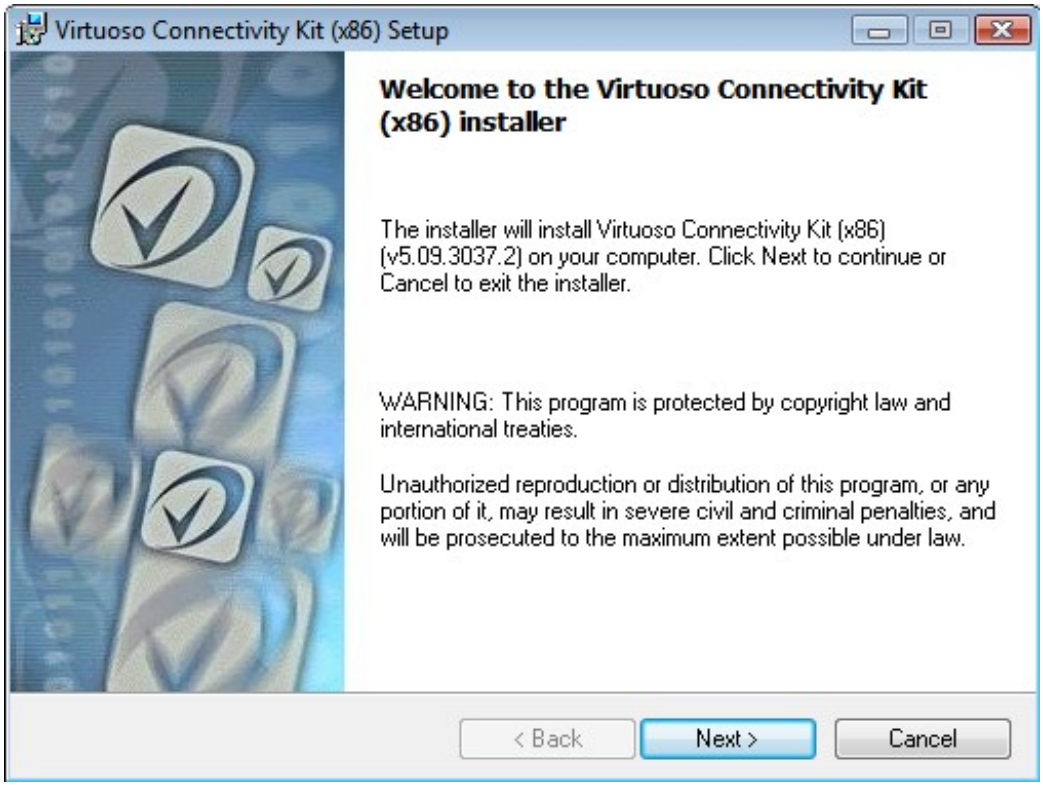
1. Download and run the Virtuoso Client Connectivity Kit installer for your target Windows OS:
   - Windows 98/NT/2000/XP/2003/Vista/2008 (32 Bit) (x86)
   - Windows XP/2003/Vista/2008 (64 Bit) (x86_64)
2. Choose the "run" button to allow the signed "Virtuoso Client Connectivity kit" installer to start.

   Figure 7.1. Run



3. Choose the "Next" button to begin the installation process.

Figure 7.2. Installation: Next

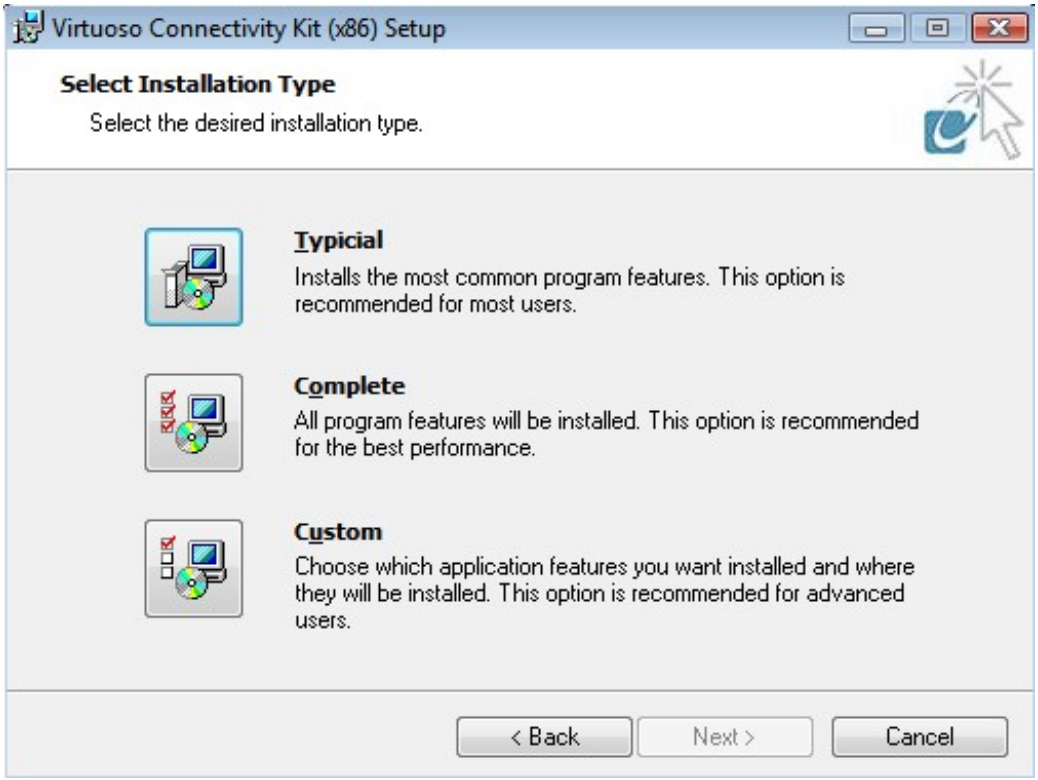


4. Check the "I agree with license agreement" checkbox and choose the "Next" button.

Figure 7.3. Installation: Agree license

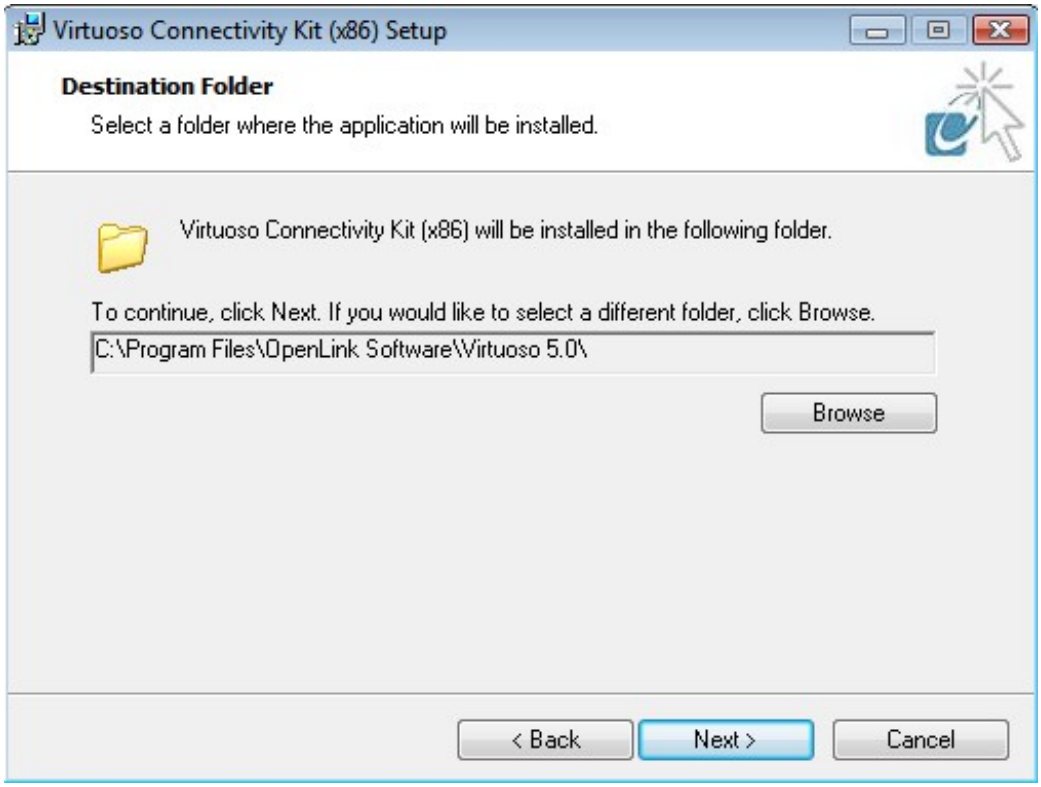
5. Choose the "Custom" option to manually select what components are installed and in which location, or choose the "Typical" or "Complete" options as recommended and skip to step 7.

Figure 7.4. Custom Install



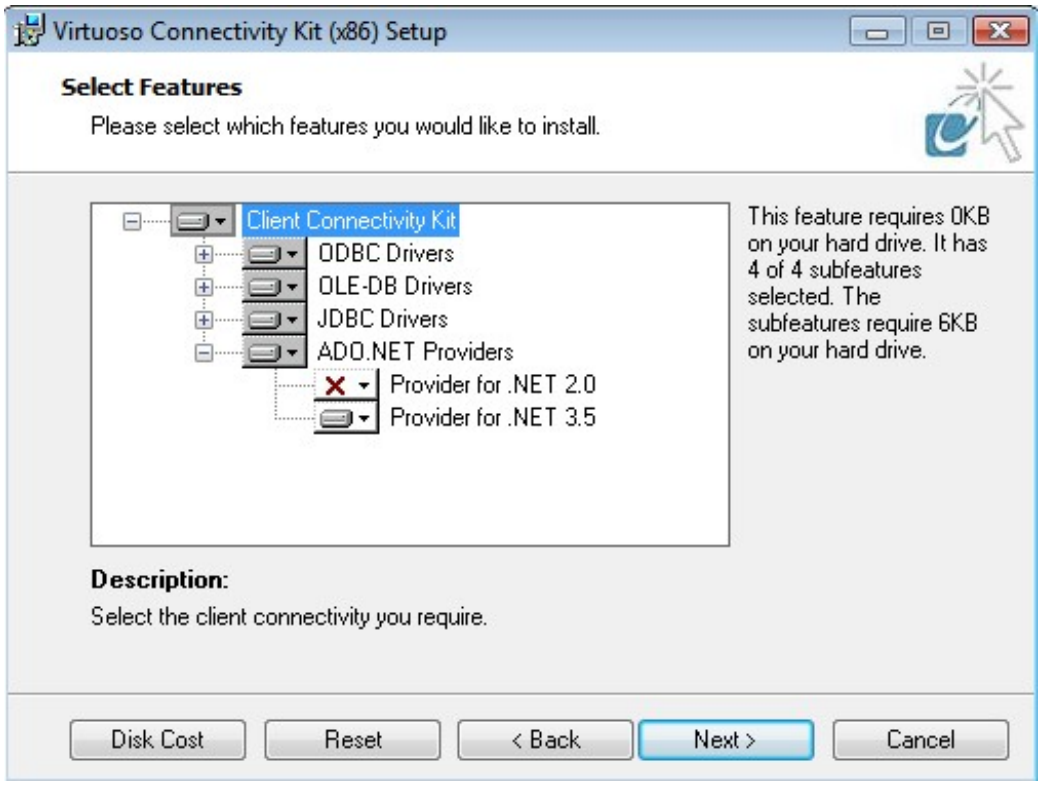
6. If the "Custom" option what chosen, select the installation directory or leave default and choose the "Next" button to continue.

Figure 7.5. Installation



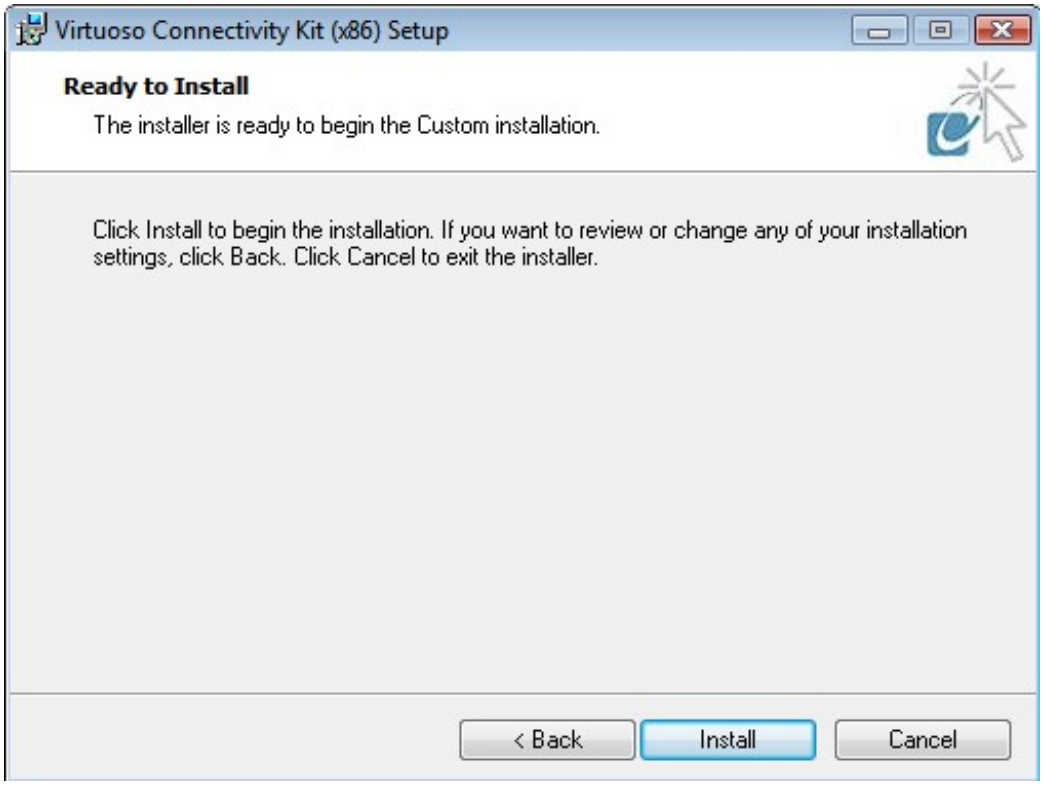
7. Select which of the ADO.NET Providers are to be installed and choose "Next" to continue.

Figure 7.6. ADO.NET Providers



8. The installer is now ready to begin the installation, choose the "Install" button to commence the process.

Figure 7.7. Install

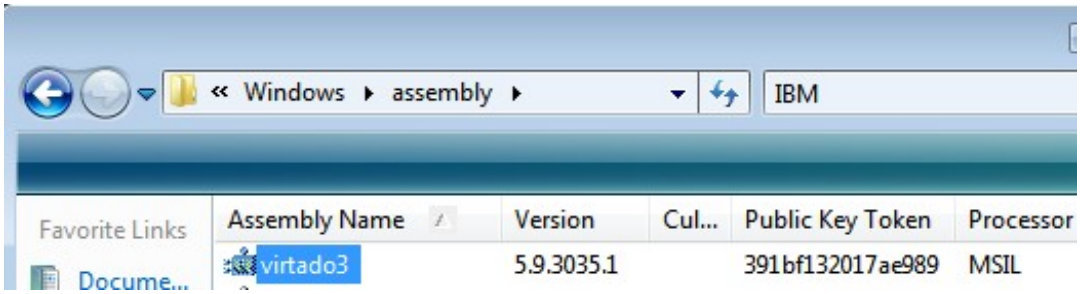


9. The installation was successfully completed.

Figure 7.8. Completed installation

10. The installation of the Provider can be verified by checking the %WINDOWS%\assembly folder and checking that the Provider(s) chosen for installation is(are) listed - "virtado3" and/or "virtado2".

Figure 7.9. Provider

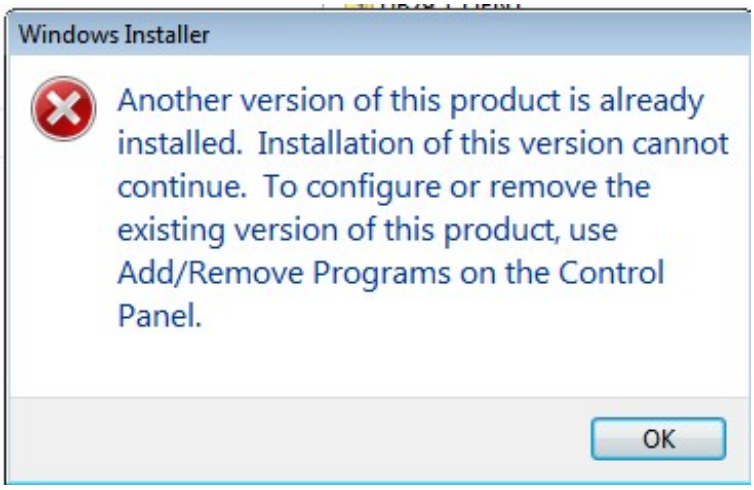


The installation of the ADO.Net Provider Client is complete.

Uninstallation

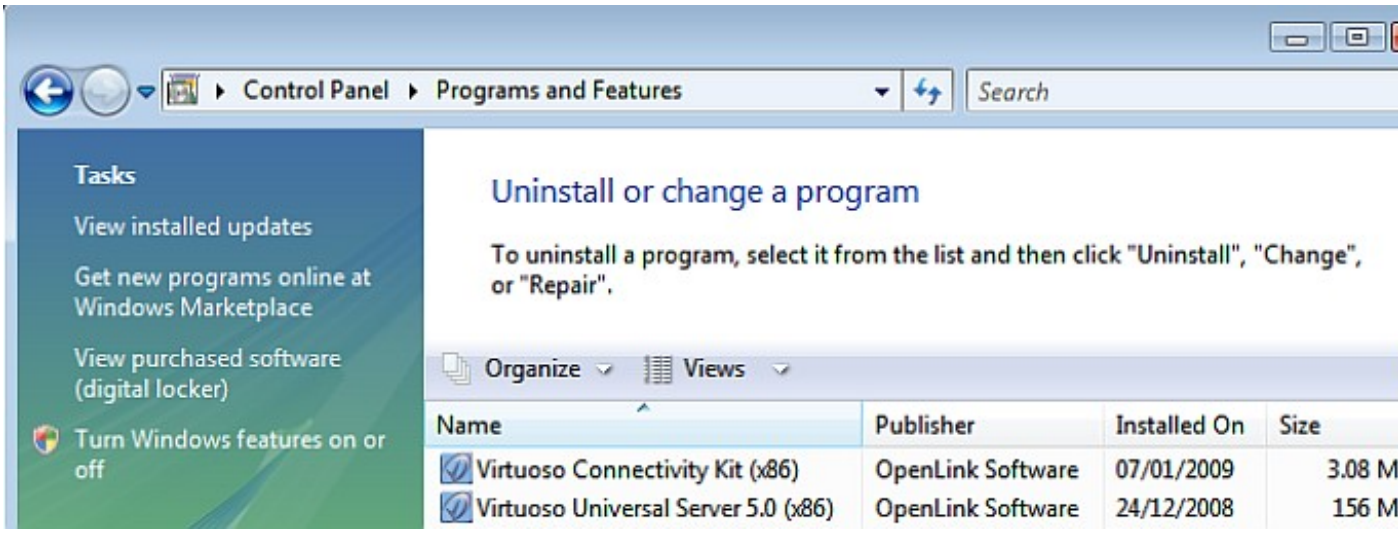
1. If you attempt to run the installer again, the following message will be displayed.

   Figure 7.10. Installer message

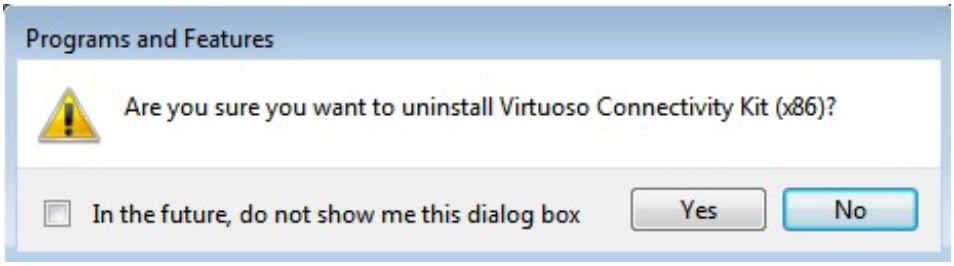
2. The Provider can be uninstalled from the "Control Panel" -> "Programs and Features" menu, by selecting the "Virtuoso Client Connectivity kit" from the list of installed components and clicking the "unistall" option presented.

   Figure 7.11. Uninstall

3. Choose "Yes" to uninstall the Virtuoso ADO.Net Provider.

   Figure 7.12. Uninstall



Installation of the ADO.Net Provider Client and Virtuoso Universal Server on Windows

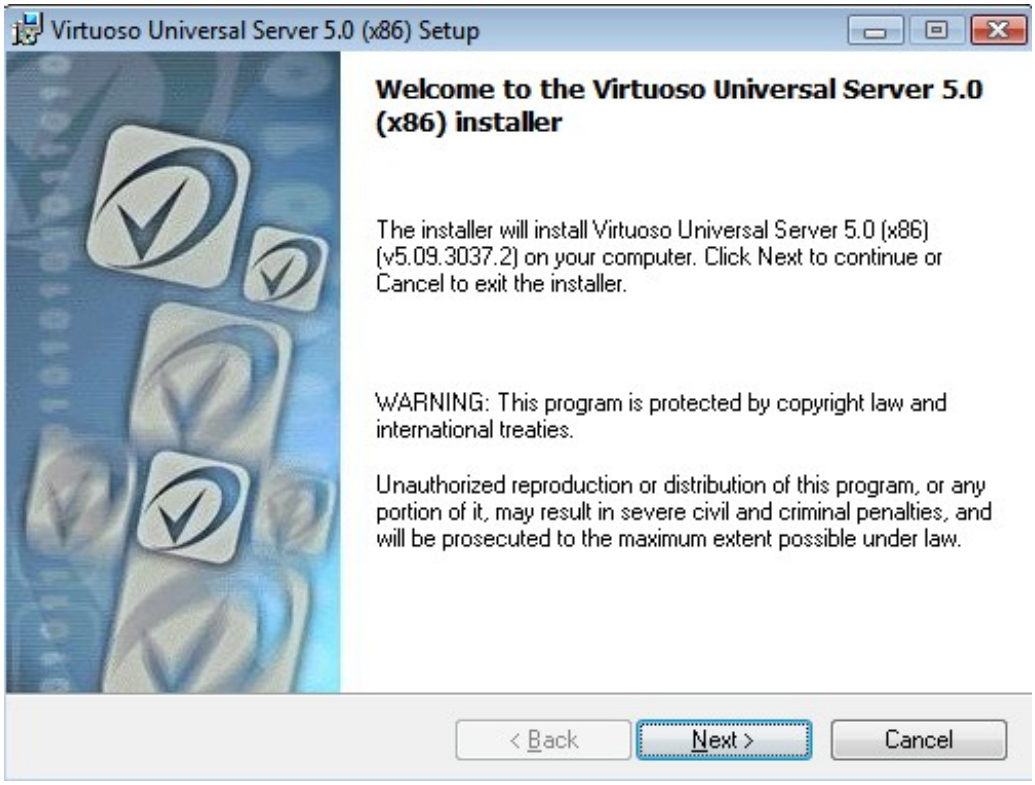
If both the ADO.Net Provider client and Virtuoso Server are to be installed on the same Windows machine, then the Virtuoso Universal Server installer can be used as it includes all the Virtuoso client and server components available in a single installer. An instance in which this particular installation type is require would be when using the Provider to provide Entity Framework connectivity to the remote database schema objects.

1. Download and run the Virtuoso Universal Server installer for your target Windows OS:
   - Windows 98/NT/2000/XP/2003/Vista/2008 (32 Bit) (x86)
   - Windows XP/2003/Vista/2008 (64 Bit) (x86_64)
2. Choose the "run" button to allow the signed "Virtuoso Universal Server" installer to start.

Figure 7.13. Run

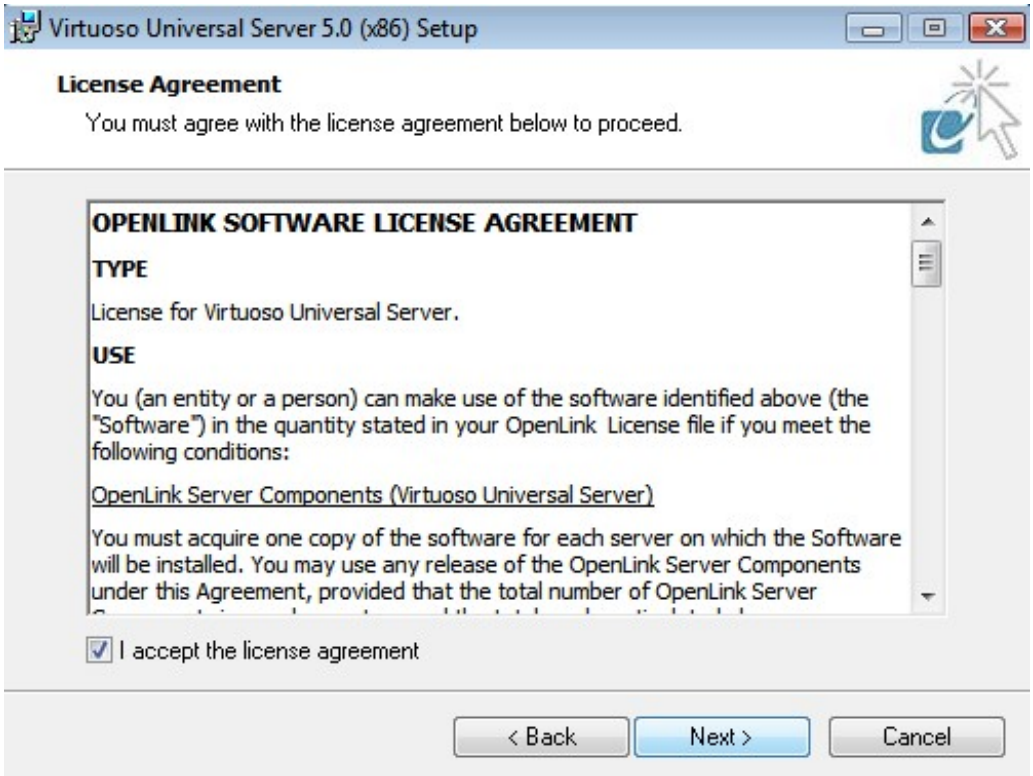
3. Choose the "Next" button to begin the installation process.

Figure 7.14. Next



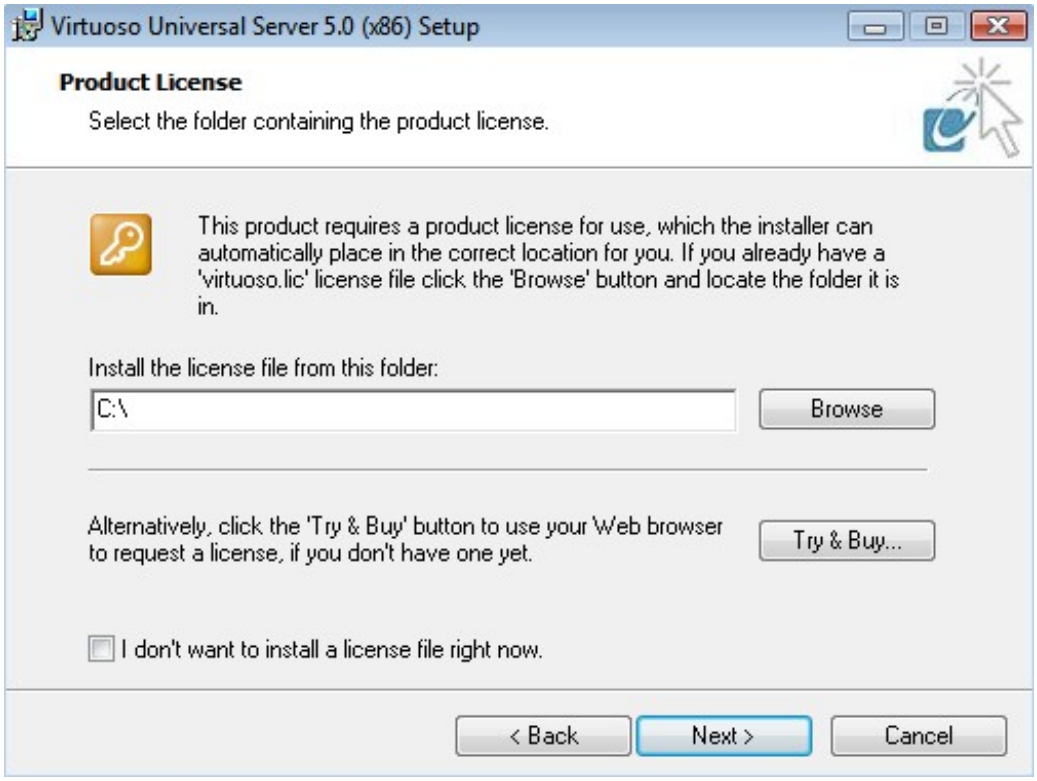
4. Check the "I agree with license agreement" checkbox and choose the "Next" button.

Figure 7.15. Agree license



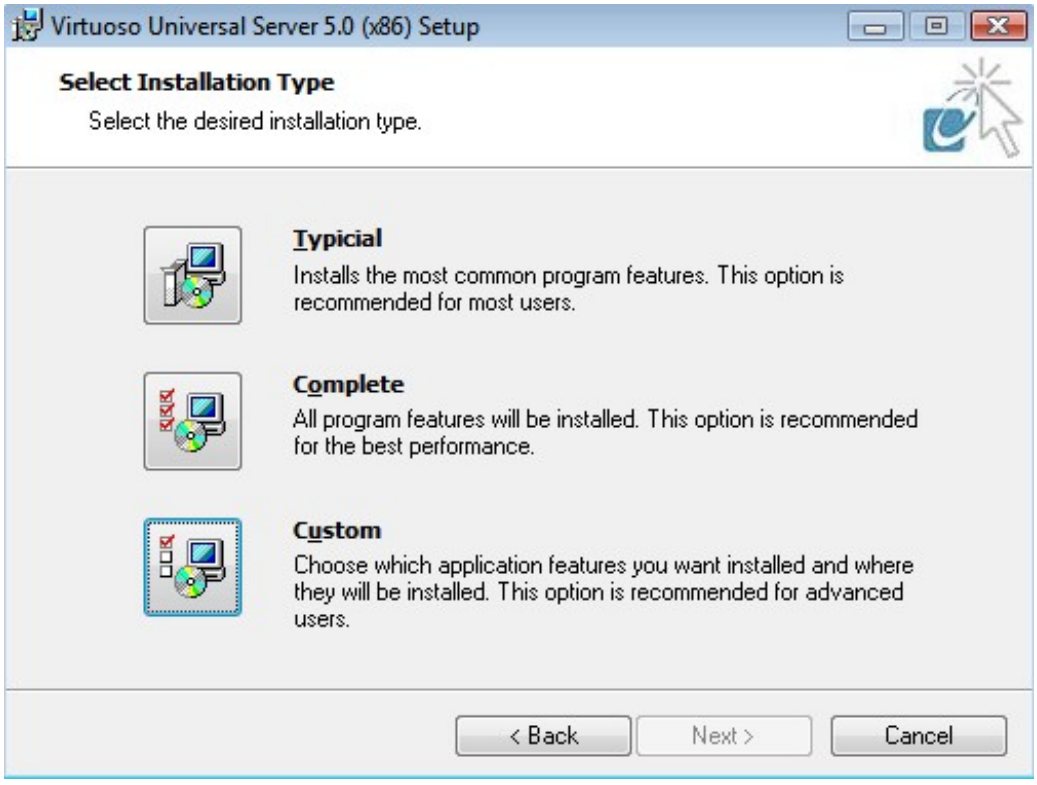
5. Choose the "Custom" option to manually select what components are installed and in which location, or choose the "Typical" or "Complete" options as recommended and skip to step 7.

Figure 7.16. Custom option



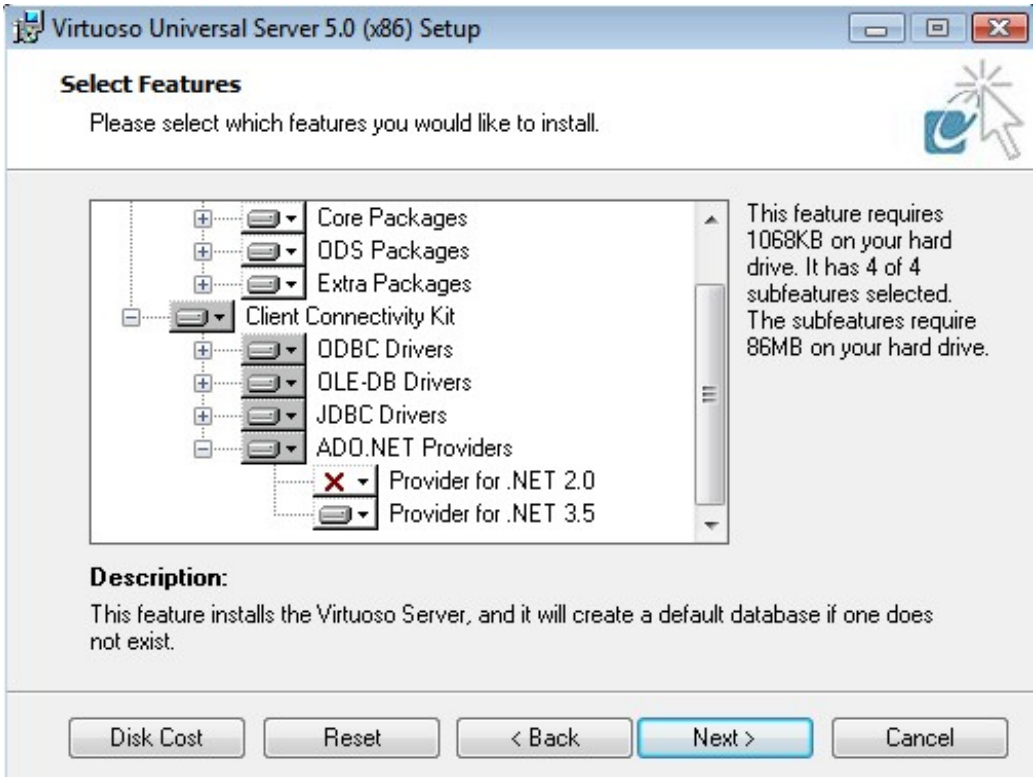
6. If the "Custom" option what chosen, select the installation directory or leave default and choose the "Next" button to continue.

Figure 7.17. Installation directory



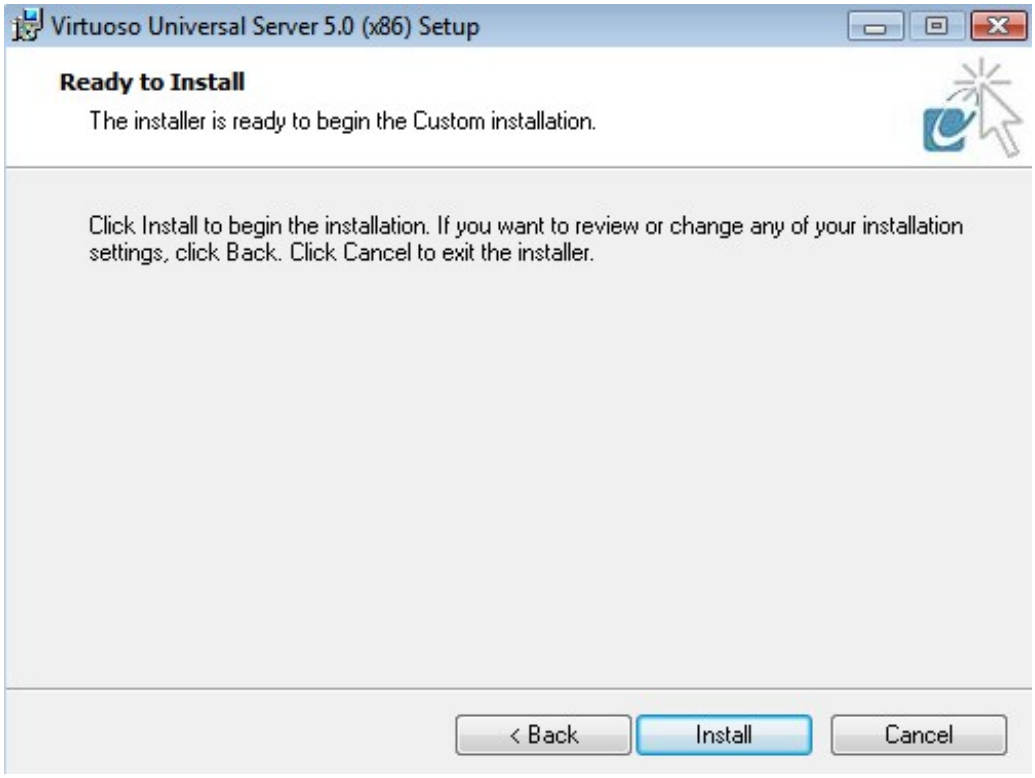
7. Select which of the ADO.NET Providers are to be installed and choose "Next" to continue.

Figure 7.18. ADO.NET Providers



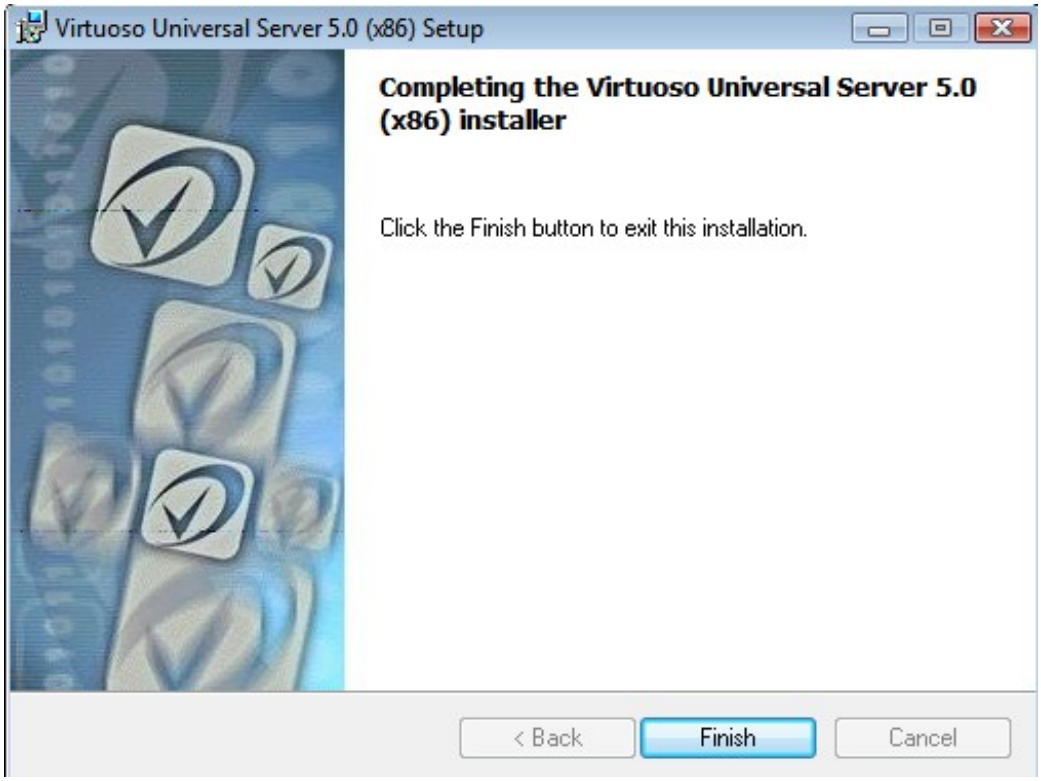
8. The installer is now ready to begin the installation, choose the "Install" button to commence the process.

Figure 7.19. Begin installation



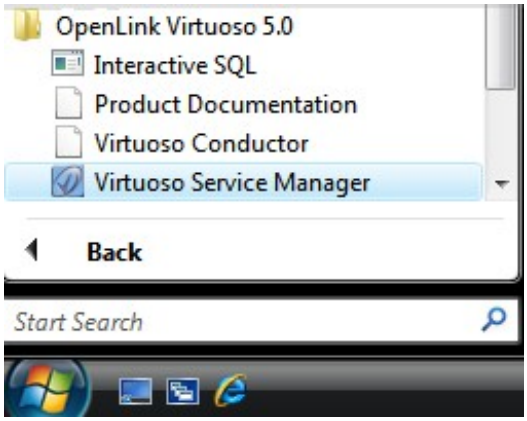
9. The installation was successfully completed.

Figure 7.20. Installation completed



10. Post installation the Virtuoso Server can be started by running the "Virtuoso Service Manager" which gets added to the Windows "Icons Tray" in the status bar.

Figure 7.21. Post installation



11. Double click on the "Virtuoso Service Manager" icon in the "Icons Tray" to load the application, select the "OpenLink Virtuoso Server" instance and press the "Start" button to run the Service.

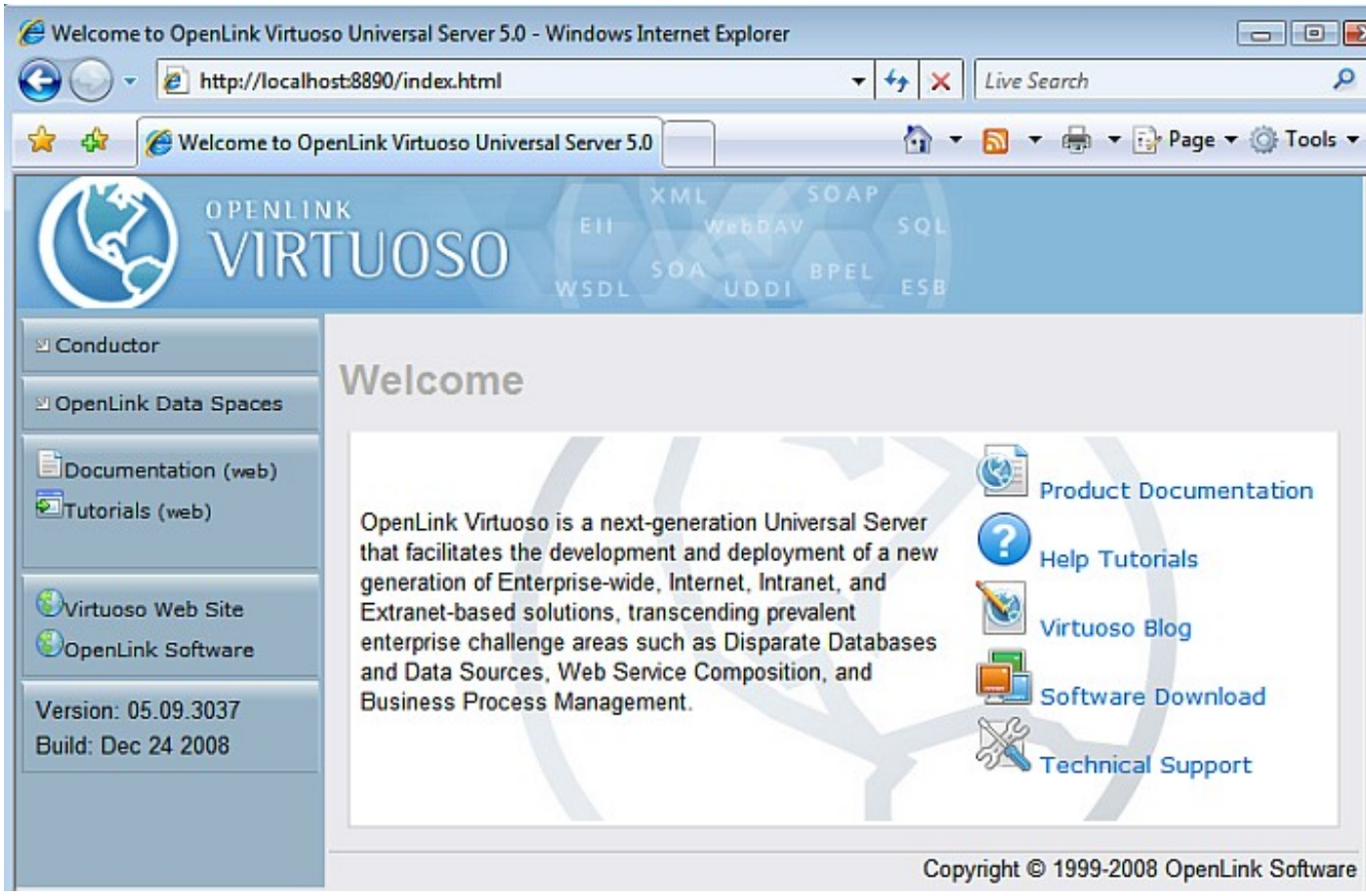
Figure 7.22. Virtuoso Service Manager

12. Once the Virtuoso service has successfully started, it is *recommended* that the default "dba" and "dav" passwords be changed to secure values by loading the URL "http://example.com/install" in a Web Browser which loads the following page for making these changes.

Figure 7.23. Defaults

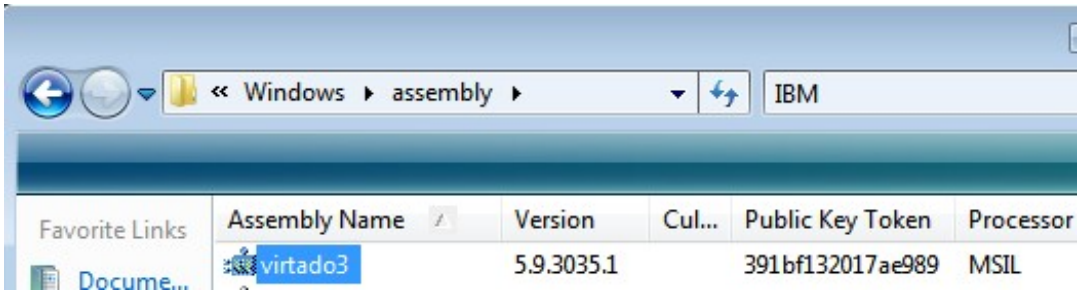
13. The Virtuoso Server is now available for use.

Figure 7.24. Virtuoso Server



14. The installation of the ADO.Net Provider can be verified by checking the %WINDOWS%\assembly folder and checking that the Provider(s) chosen for installation is(are) listed - "virtado3" and/or "virtado2".

Figure 7.25. Installation verification

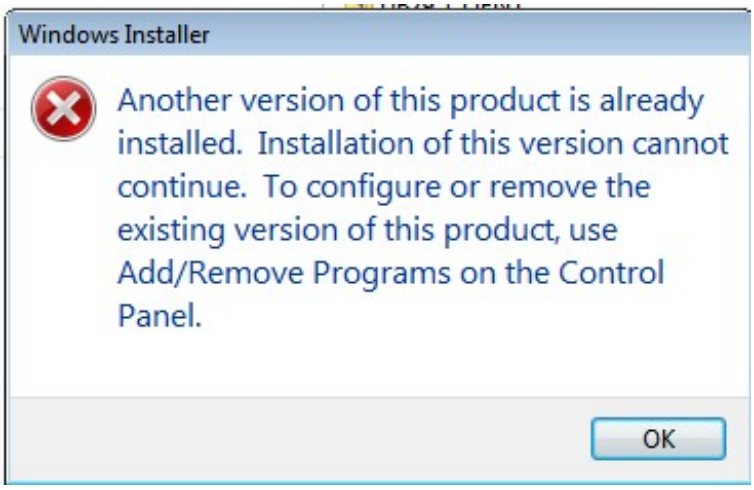


The installation of the ADO.Net Provider Client and Virtuoso Universal Server is complete.

Uninstallation

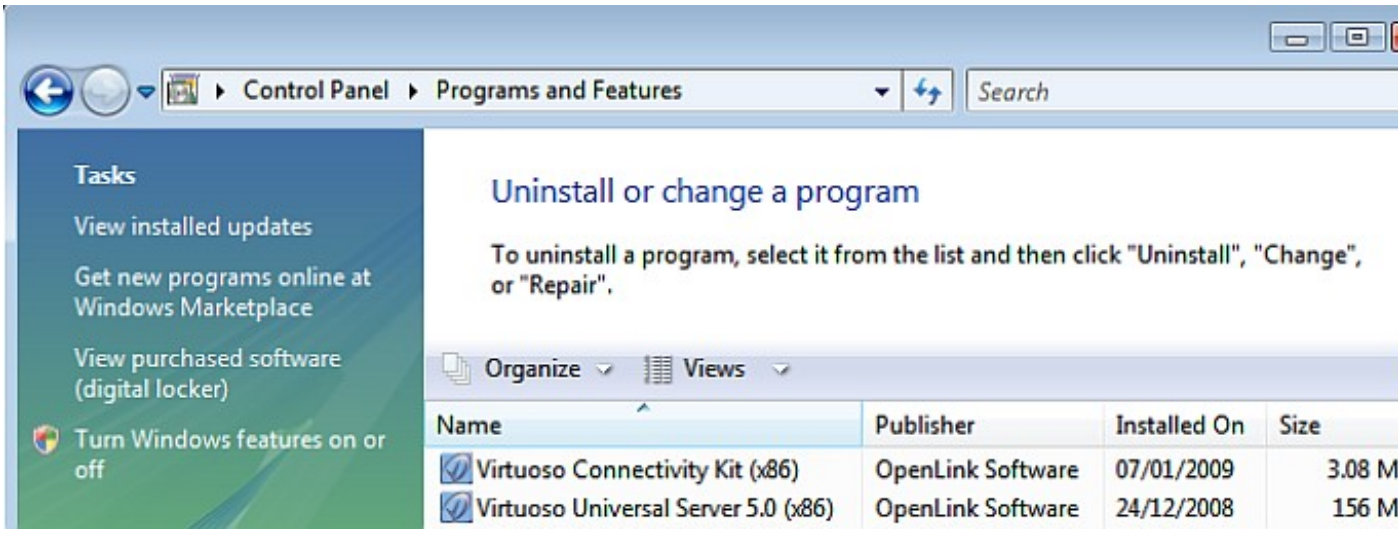
1. If you attempt to run the installer again, the following message will be displayed.

Figure 7.26. Installer message



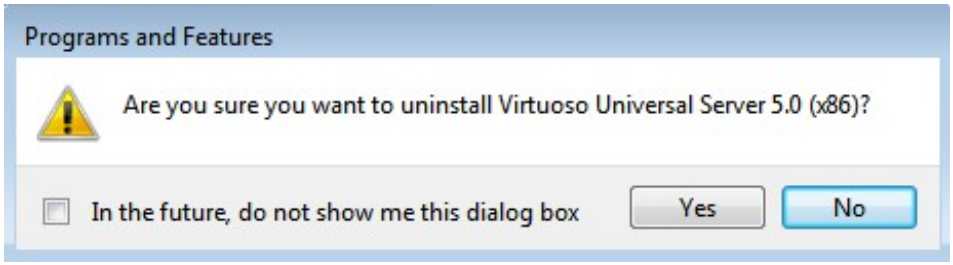
2. The Provider can be uninstalled from the "Control Panel" -> "Programs and Features" menu, by selecting the "Virtuoso Client Connectivity kit" from the list of installed components and clicking the "unistall" option presented.

Figure 7.27. Uninstall



3. Choose "Yes" to uninstall the Virtuoso ADO.Net Provider.

Figure 7.28. Uninstall



7.1.3. Programmers Guide

Sample Walk through Applications

- Using Visual Studio 2008 to Build a Data Grid Form Application
- Using Visual Studio 2008 to Build an Entity Frameworks based Windows Form Application
- Using Visual Studio 2008 to Build an ADO.NET Data Services based Application
- Windows Form Application for accessing Virtuoso RDF data via SPASQL using the Virtuoso ADO.Net Provider
- Web Data Service Application for accessing Virtuoso RDF data via SPASQL using the Virtuoso ADO.Net Provider
- Creating a Silverlight Application to consume the service
- Creating A Simple .NET RIA Services Application to Display Data from Virtuoso
- Creating a .Net RIA Services Application that will Update Virtuoso Data

Using Entity Frameworks, ADO.NET, the Virtuoso Virtual Database Engine

Virtuoso's in-built virtual database engine for ODBC and JDBC accessible databases enables it to act as bridge between Entity Frameworks & ADO.NET based client applications. Thus, you simply link external databases into Virtuoso using the browser based Conductor UI or programmatically using SQL extensions. Once the external tables a linked/attached, they inherit the functionality prowess of Virtuoso, and this particular use case scenario, complete compatibility with Entity Frameworks and ADO.NET 3.5.

- Using Microsoft Entity Frameworks to Access Oracle Schema Objects with Virtuoso
- Using Microsoft Entity Frameworks to Access Progress Schema Objects with Virtuoso
- Using Microsoft Entity Frameworks to Access Ingres Schema Objects with Virtuoso
- Using Microsoft Entity Frameworks to Access Informix Schema Objects with Virtuoso
- Using Microsoft Entity Frameworks to Access DB2 Schema Objects with Virtuoso
- Using Microsoft Entity Frameworks to Access Sybase Schema Objects with Virtuoso
- Using Microsoft Entity Frameworks to Access MySQL Schema Objects with Virtuoso
- Using Microsoft Entity Frameworks to Access PostgreSQL Schema Objects with Virtuoso
- Using Microsoft Entity Frameworks to Access JDBC Schema Objects with Virtuoso
- Using Microsoft Entity Frameworks to Access ODBC Schema Objects with Virtuoso
- Using Microsoft Entity Frameworks to Access Firebird Schema Objects with Virtuoso
- Using Microsoft Entity Frameworks to Access Microsoft SQL Server Schema Objects with Virtuoso

VirtDbType Enumeration

Specifies Virtuoso data types.

Members

Table 7.1.

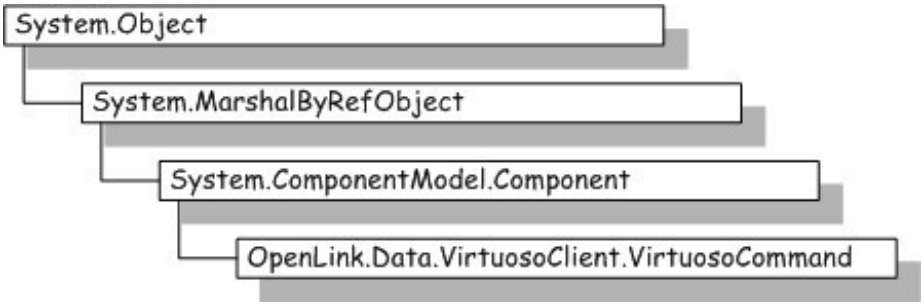
Member name	Description
`Binary`	BINARY data. This maps to an Array of type Byte .
`Char`	CHAR data. This maps to String .

Member name	Description
Date	DATE data. This maps to DateTime .
DateTime	DATETIME data. This maps to DateTime .
Decimal	This is equal to Numeric .
Double	DOUBLE PRECISION data. This maps to Double .
Float	This is equal to Double.
Integer	INTEGER data. This maps to Int32
LongBinary	LONG BINARY data. This maps to an Array of type Byte .
LongNVarChar	LONG NVARCHAR data. This maps to String .
LongVarChar	LONG VARCHAR data. This maps to String .
NChar	NChar data. This maps to String .
Numeric	NUMERIC data. This maps to Decimal
NVarChar	NVARCHAR data. This maps to String .
Real	REAL data, This maps to Single .
SmallInt	SMALLINT data. This maps to Int16
Time	TIME data. This maps to TimeSpan
TimeStamp	TIMESTAMP data. This maps to an Array of type Byte .
VarBinary	VARBINARY data. This maps to an Array of type Byte .
VarChar	VARCHAR data. This maps to String .

VirtuosoCommand Class

Represents an SQL statement or stored procedure to execute against a Virtuoso database. This class cannot be inherited.

Figure 7.29. Virtuoso .Net Provider VirtuosoCommand Class




```
                public
                sealed
                class
                VirtuosoCommand
               extends,
                Component

    implements,
                ICloneable
              ,
                IDbCommand
               {
}
```

Thread Safety

Any public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Any instance members are not guaranteed to be thread safe.

Remarks

The VirtuosoCommand class provides the following methods for executing commands against a Virtuoso database:

Table 7.2.

Item	Description
ExecuteReader	Executes commands that return rows.
ExecuteNonQuery	Executes commands such as SQL INSERT, DELETE, UPDATE, and SET statements.
ExecuteScalar	Retrieves a single value (for example, an aggregate value) from a database.

VirtuosoCommand Constructor

Initializes a new instance of the VirtuosoCommand class.

VirtuosoCommand Constructor ()

Initializes a new instance of the VirtuosoCommand class.

```
public VirtuosoCommand();
```

Remarks

The base constructor initializes all fields to their default values. The following table shows initial property values for an instance of VirtuosoCommand.

Table 7.3.

Properties	Initial Value
CommandText	empty string ("")
CommandTimeout	30
CommandType	CommandType.Text
Connection	null

You can change the value for any of these properties through a separate call to the property.

VirtuosoCommand Constructor (string)

Initializes a new instance of the VirtuosoCommand class with the text of the query.

```
public VirtuosoCommand( string  cmdText );
```

Parameters

cmdText
: The text of the query.

VirtuosoCommand Constructor (string, VirtuosoConnection)

Initializes a new instance of the VirtuosoCommand class with the text of the query and an VirtuosoConnection object.

```
public VirtuosoCommand( string  cmdText ,
                        VirtuosoConnection  connection );
```

Parameters

cmdText
: The text of the query.

connection
: A VirtuosoConnection object that represents the connection to a Virtuoso database.

ConnectionString Property

The Virtuoso ADO.NET Provider ConnectionString property implements the IDbConnection.ConnectionString property to get or set the string used to open a Virtuoso database connection, and includes the source database name and other parameters needed to establish the initial connection. The default value is an empty string.

```
public  string  ConnectionString  ;
```

Property Value

Includes the source database name and other parameters needed to establish the initial connection. The default value is an empty string.

ConnectionString has the following syntax: Each connection string is a sequence of settings Individual settings are separated by semicolons. Each setting is a pair of name and value delimited by the equal sign. Whitespace is ignored on either side of both names and values. Names are case insensitive. The value part can be quoted by either single or double quote characters or remain unquoted at all. However if it includes a semicolon, single quote, or double quote characters, it must be enclosed in either type of quotes. To embed the same character that is used for enclosing the value the character within the value must be doubled.

The following table lists the valid names for values within the ConnectionString:

Table 7.4.

Name	Default	Description
Connect Timeout or Connection Timeout	15	The number of seconds to wait for a connection to the server before terminating the attempt and generating an error.
Connection Lifetime	0	When a connection is returned to the pool, its creation time is compared with the current time, and the connection is destroyed if that time span (in seconds) exceeds the value specified by connection lifetime. Useful in clustered configurations to force load balancing between a running server and a server just brought on-line.
Charset	utf-16	Specifies the character set to be used by the provider when passing string values to and from the database. Must be set to utf-8 to handle Unicode strings passed in SPARQL/SPASQL queries of RDF data.
Data Source or Server or Address or Network Address or Host		The name or network address of the instance of Virtuoso server to which to connect. Can take comma delimited list of instances for connection fail over.
Encrypt	false	Specifies if the connection must be SSL encrypted. Currently encryption only works with an ODBC-based provider.
Enlist	true	When true, the pooler automatically enlists the connection in the creation thread's current transaction context.
Initial Catalog or Database		The name of the database.
Max Pool Size	100	The maximum number of connections allowed in the pool.
Min Pool Size	0	The minimum number of connections allowed in the pool.
Password or Pwd		The password for the Virtuoso account logging on.
Persist Security Info	false	When set to 'false', security-sensitive information, such as the password, is not returned as part of the connection if the connection is open or has ever been in an open State. Resetting the connection string resets all connection string values including the password.
Pooling	true	When true, the VirtuosoConnection object is drawn from the appropriate pool, or if necessary, is created and added to the appropriate pool.
RoundRobin	false	Enables load balancing in which case the server for the connection is chosen at random from the comma delimited provided as for a Failover connection.
User ID or Uid		The Virtuoso login name.

Implements

IDbCommand.ConnectionString

Remarks

The *ConnectionString* is similar to an OLE DB connection string, but is not identical. Unlike OLE DB or ADO, the connection string that is returned is the same as the user set *ConnectionString* minus security information if the *Persist Security Info* value is set to *false* (default). The Virtuoso ADO.NET Data Provider neither persists nor returns the password in a connection string unless you set *Persist Security Info* to true.

The *ConnectionString* property can be set only when the connection is closed. Many of the connection string values have corresponding read-only properties. When the connection string is set, all of these properties are updated, except when an error is detected; in this case, none of the properties are updated. *VirtuosoConnection* properties return only those settings contained in the *ConnectionString* .

Resetting the *ConnectionString* on a closed connection resets all connection string values (and related properties) including the password. For example, if you set a connection string that includes "*Database=Demo* ", and then reset the connection string to "*Data Source=myserver;User ID=dba;Password=dba* ", the Database property is no longer set to *Demo* .

The connection string is parsed immediately after being set. If errors in syntax are found when parsing, a runtime exception (e.g., *ArgumentException*) is generated. Other errors can be found only when an attempt is made to open the connection.

VirtuosoCommand Constructor (string, VirtuosoConnection, VirtuosoTransaction)

Initializes a new instance of the VirtuosoCommand class with the text of the query, an VirtuosoConnection object, and the VirtuosoTransaction.

```
public VirtuosoCommand( string  cmdText ,
                        VirtuosoConnection  connection ,
                        VirtuosoTransaction  transaction );
```

Parameters

cmdText
: The text of the query.

connection
: A VirtuosoConnection object that represents the connection to a Virtuoso database.

transaction
: The VirtuosoTransaction in which the VirtuosoCommand executes.

Properties

CommandText Property

Gets or sets the SQL statement or stored procedure to execute against the Virtuoso database.

```
public  string  CommandText  ;
```

Property Value

The SQL statement or stored procedure to execute. The default value is an empty string ("").

Implements

IDbCommand.CommandText

Remarks

When the CommandType property is set to StoredProcedure, the CommandText property should be set using standard ODBC stored procedure escape sequences. Setting the CommandText to the name of the stored procedure does not function as it does for some other .NET data providers.

The Virtuoso .NET Data Provider does not support named parameters for passing parameters to a SQL statement or a stored procedure called by an OleDbCommand when CommandType is set to Text. In this case, the question mark (?) placeholder must be used. For example:

```
SELECT * FROM Customers WHERE CustomerID = ?
```

As a result, the order in which VirtuosoParameter objects are added to the VirtuosoParameterCollection must directly correspond to the position of the question mark placeholder for the parameter.

CommandTimeout Property

Gets or sets the wait time before terminating an attempt to execute a command and generating an error.

```
public int CommandTimeout ;
```

Property Value

The time (in seconds) to wait for the command to execute. The default is 30 seconds.

Implements

IDbCommand.CommandTimeout

Remarks

A value of zero (0) specifies no limit to the wait time, rather than no wait time, and therefore should be avoided.

CommandType Property

Gets or sets a value indicating how the CommandText property is interpreted.

```
public CommandType CommandType ;
```

Property Value

One of the System.Data.CommandType values. The default is Text.

Implements

IDbCommand.CommandType

Remarks

When the CommandType property is set to StoredProcedure, you should set the CommandText property to the full ODBC call syntax. The command then executes this stored procedure when you call one of the Execute methods (for example, ExecuteReader or ExecuteNonQuery).

The Connection, CommandType and CommandText properties cannot be set if the current connection is performing an execute or fetch operation.

The Virtuoso .NET Data Provider does not support passing named parameters to an SQL statement or to a stored procedure called by a VirtuosoCommand. In either of these cases, use the question mark (?) placeholder. For example:

```
SELECT * FROM Customers WHERE CustomerID = ?
```

The order in which VirtuosoParameter objects are added to the VirtuosoParameterCollection must directly correspond to the position of the question mark placeholder for the parameter.

Connection Property

Gets or sets the VirtuosoConnection used by this instance of the VirtuosoCommand.

```
public VirtuosoConnection Connection ;
```

Property Value

The connection to a Virtuoso database. The default is a null value.

Parameters Property

Gets the VirtuosoParameterCollection.

```
public  VirtuosoParameterCollection  Connection  ;
```

Property Value

The parameters of the SQL statement or stored procedure. The default is an empty collection.

Remarks

When CommandType is set to Text, the Virtuoso .NET Data Provider does not support passing named parameters to an SQL statement or to a stored procedure called by an VirtuosoCommand. In either of these cases, use the question mark (?) placeholder. For example:

```
SELECT * FROM Customers WHERE CustomerID = ?
```

The order in which VirtuosoParameter objects are added to the VirtuosoParameterCollection must directly correspond to the position of the question mark placeholder for the parameter in the command text.

Note:
If the parameters in the collection do not match the requirements of the query to be executed, an error may result.

Transaction Property

Gets or sets the VirtuosoTransaction within which the VirtuosoCommand executes.

```
public  VirtuosoTransaction  Transaction  ;
```

Property Value

A VirtuosoTransaction. The default is a null value.

Remarks

You cannot set the Transaction property if it is already set to a specific value, and the command is in the process of executing. If you set the transaction property to an VirtuosoTransaction object that is not connected to the same VirtuosoConnection as the VirtuosoCommand object, an exception will be thrown the next time you attempt to execute a statement.

UpdatedRowSource Property

Gets or sets how command results are applied to the DataRow when used by the Update method of the DbDataAdapter.

```
public  UpdateRowSource  UpdatedRowSource  ;
```

Property Value

One of the System.Data.UpdateRowSource values.

Implements

IDbCommand.UpdatedRowSource

Methods

Cancel Method

Attempts to cancel the execution of an VirtuosoCommand.

```
public void Cancel();
```

Implements

IDbCommand.Cancel

Remarks

If there is nothing to cancel, nothing happens. However, if there is a command in process, and the attempt to cancel fails, no exception is generated.

CreateParameter Method

Creates a new instance of a VirtuosoParameter object.

```
public VirtuosoParameter CreateParameter();
```

Return Value

A VirtuosoParameter object.

Remarks

The CreateParameter method is a strongly-typed version of IDbCommand.CreateParameter.

ExecuteNonQuery Method

Executes an SQL statement against the Connection and returns the number of rows affected.

```
public int ExecuteNonQuery();
```

Return Value

For UPDATE, INSERT, and DELETE statements, the return value is the number of rows affected by the command. For all other types of statements, the return value is -1.

Implements

IDbCommand.ExecuteNonQuery

Remarks

You can use ExecuteNonQuery to perform catalog operations (for example, querying the structure of a database or creating database objects such as tables); or to change the data in a database, without using a DataSet, by executing UPDATE, INSERT, or DELETE statements.

Although ExecuteNonQuery does not return any rows, any output parameters or return values mapped to parameters are populated with data.

ExecuteReader Method

Sends the CommandText to the Connection and builds a VirtuosoDataReader.

ExecuteReader Method ()

Sends the CommandText to the Connection and builds a VirtuosoDataReader.

```
public VirtuosoDataReader ExecuteReader();
```

Return Value

A VirtuosoDataReader object.

ExecuteReader Method (CommandBehavior)

Sends the CommandText to the Connection and builds a VirtuosoDataReader using one of the CommandBehavior values.

```
public VirtuosoDataReader ExecuteReader( CommandBehavior  behavior );
```

Parameters

behavior
: One of the System.Data.CommandBehavior values.

Return Value

A VirtuosoDataReader object.

ExecuteScalar Method

Executes the query, and returns the first column of the first row in the resultset returned by the query. Extra columns or rows are ignored.

```
public object ExecuteScalar();
```

Return Value

The first column of the first row in the resultset.

Implements

IDbCommand.ExecuteScalar

Remarks

Use the ExecuteScalar method to retrieve a single value (for example, an aggregate value) from a database. This requires less code than using the ExecuteReader method, and then performing the operations necessary to generate the single value using the data returned by a VirtuosoDataReader.

A typical ExecuteScalar query can be formatted as in the following C# example:

```
command.CommandText = "select count(*) from foobar";
int count = (int) command.ExecuteScalar();
```

ICloneable.Clone Method

This member supports the .NET Framework infrastructure and is not intended to be used directly from your code.

```
object ICloneable.Clone();
```

IDbCommand.CreateParameter Method

This member supports the .NET Framework infrastructure and is not intended to be used directly from your code.

```
IDbDataParameter IDbCommand.CreateParameter();
```

IDbCommand.ExecuteReader Method

This member supports the .NET Framework infrastructure and is not intended to be used directly from your code.

IDbCommand.ExecuteReader Method ()

This member supports the .NET Framework infrastructure and is not intended to be used directly from your code.

```
IDataReader IDbCommand.ExecuteReader();
```

IDbCommand.ExecuteReader Method (CommandBehavior)

This member supports the .NET Framework infrastructure and is not intended to be used directly from your code.

```
IDataReader IDbCommand.ExecuteReader( CommandBehavior  behavior );
```

Prepare Method

Creates a prepared (or compiled) version of the command at the Virtuoso server.

```
public void Prepare();
```

Implements

IDbCommand.Prepare

VirtuosoCommandBuilder Class

Provides a means of automatically generating single-table commands used to reconcile changes made to a DataSet with the associated Virtuoso database. This class cannot be inherited.

System.Object
System.MarshalByRefObject
System.ComponentModel.Component
VirtuosoCommandBuilder

```
                    public
                    sealed
                    class
                    VirtuosoCommandBuilder
                   extends,
                    Component
                   {
      }
```

Thread Safety

Any public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Any instance members are not guaranteed to be thread safe.

Remarks

The VirtuosoDataAdapter does not automatically generate the SQL statements required to reconcile changes made to a DataSet associated with the data source. However, you can create a VirtuosoCommandBuilder object that generates SQL statements for single-table updates if you set the SelectCommand property of the VirtuosoDataAdapter. Then, the VirtuosoCommandBuilder generates any additional SQL statements that you do not set.

The relationship between a VirtuosoDataAdapter and its corresponding VirtuosoCommandBuilder is always one-to-one. To create this correspondence, you set the DataAdapter property of the VirtuosoCommandBuilder object. This causes the VirtuosoCommandBuilder to register itself as a listener for RowUpdating events on the specified VirtuosoDataAdapter object.

To generate INSERT, UPDATE, or DELETE statements, the VirtuosoCommandBuilder uses the SelectCommand property to retrieve a required set of metadata. If you change the value of SelectCommand after the metadata has been retrieved (for example, after the first update), you then should call the RefreshSchema method to update the metadata.

The VirtuosoCommandBuilder also uses the Connection, CommandTimeout, and Transaction properties referenced by the SelectCommand. The user should call RefreshSchema if any of these properties are modified, or if the value of the SelectCommand property itself is changed. Otherwise the InsertCommand, UpdateCommand, and DeleteCommand properties retain their previous values.

If you call Dispose, the VirtuosoCommandBuilder is disassociated from the VirtuosoDataAdapter, and the generated commands are no longer used.

VirtuosoCommandBuilder Constructor

VirtuosoCommandBuilder Constructor ()

Initializes a new instance of the VirtuosoCommandBuilder class.

```
public VirtuosoCommandBuilder();
```

VirtuosoCommandBuilder Constructor (VirtuosoDataAdapter)

Initializes a new instance of the VirtuosoCommandBuilder class with the associated VirtuosoDataAdapter object.

```
public VirtuosoCommandBuilder( VirtuosoDataAdapter  adapter );
```

Parameters

adapter
: A VirtuosoDataAdapter object to associate with this VirtuosoCommandBuilder.

Remarks

The VirtuosoCommandBuilder registers itself as a listener for RowUpdating events that are generated by the VirtuosoDataAdapter specified in this property.

Properties

DataAdapter Property

Gets or sets a VirtuosoDataAdapter object for which SQL statements are automatically generated.

```
public  VirtuosoDataAdapter  DataAdapter  ;
```

Property Value

A VirtuosoDataAdapter object.

Remarks

The VirtuosoCommandBuilder registers itself as a listener for RowUpdating events that are generated by the VirtuosoDataAdapter specified in this property.

QuotePrefix Property

Gets or sets the beginning character or characters to use when working with database objects (for example, tables or columns) whose names contain characters such as spaces.

```
public  string  QuotePrefix  ;
```

Property Value

The beginning character or characters to use. The default is an empty string.

Remarks

Database objects in Virtuoso servers can contain any valid characters, including spaces, commas, and semicolons. To accommodate this capability, use the QuotePrefix and QuoteSuffix properties to specify delimiters such as a left bracket and a right bracket to encapsulate the object name.

QuoteSuffix Property

Gets or sets the ending character or characters to use when working with database objects (for example, tables or columns) whose names contain characters such as spaces.

```
public  string  QuotePrefix  ;
```

Property Value

The ending character or characters to use. The default is an empty string.

Remarks

Database objects in Virtuoso servers can contain any valid characters, including spaces, commas, and semicolons. To accommodate this capability, use the QuotePrefix and QuoteSuffix properties to specify delimiters such as a left bracket and a right bracket to encapsulate the object name.

Methods

DeriveParameters Method

Retrieves parameter information from the stored procedure specified in the VirtuosoCommand and populates the Parameters collection of the specified VirtuosoCommand object.

```
public static void DeriveParameters( VirtuosoCommand  command );
```

Parameters

command
: The VirtuosoCommand referencing the stored procedure from which the parameter information is to be derived. The derived parameters are added to the Parameters collection of the VirtuosoCommand.

Remarks

`DeriveParameters` overwrites any existing parameter information for the VirtuosoCommand. `DeriveParameters` requires an extra call to the data server to obtain the information. If the parameter information is known in advance, it is more efficient to populate the parameters collection by setting the information explicitly.

For more information, see "Using Stored Procedures with a Command" in the Microsoft® .NET Framework SDK documentation.

Dispose Method

Releases the unmanaged and, optionally, the managed resources used by the VirtuosoCommandBuilder.

```
protected override void Dispose( bool  disposing );
```

Parameters

disposing
: `true` to release both managed and unmanaged resources; `false` to release only unmanaged resources.

Remarks

This method is called by the public Dispose method and the Finalize method. Dispose() invokes the protected Dispose(Boolean) method with the disposing parameter set to true. Finalize invokes Dispose with disposing set to false.

When the disposing parameter is true, the method releases all resources held by any managed objects that this VirtuosoCommand references. It does this by invoking the Dispose() method of each referenced object.

For more information about Dispose and Finalize, see "Cleaning Up Unmanaged Resources," and "Overriding the Finalize Method," in the .NET Framework SDK documentation.

GetDeleteCommand Method

Gets the automatically generated VirtuosoCommand object required to perform deletions on the Virtuoso database.

```
public VirtuosoCommand GetDeleteCommand();
```

Return Value

The automatically generated VirtuosoCommand object required to perform deletions.

Remarks

You can use the GetDeleteCommand method for informational or troubleshooting purposes because it returns the VirtuosoCommand object to be executed.

You can also use GetDeleteCommand as the basis of a modified command. For example, you might call GetDeleteCommand and modify the CommandTimeout value, and then explicitly set that on the VirtuosoDataAdapter.

After the SQL statement is first generated, you must explicitly call RefreshSchema if you change the statement in any way. Otherwise, the GetDeleteCommand still will be using information from the previous statement, which might not be correct. The SQL statements are first generated when the application calls either Update or GetDeleteCommand.

GetInsertCommand Method

Gets the automatically generated VirtuosoCommand object required to perform insertions on the Virtuoso database.

```
public VirtuosoCommand GetInsertCommand();
```

Return Value

The automatically generated VirtuosoCommand object required to perform insertions.

Remarks

You can use the GetInsertCommand method for informational or troubleshooting purposes because it returns the VirtuosoCommand object to be executed.

You can also use GetInsertCommand as the basis of a modified command. For example, you might call GetInsertCommand and modify the CommandTimeout value, and then explicitly set that on the VirtuosoDataAdapter.

After the SQL statement is first generated, you must explicitly call RefreshSchema if you change the statement in any way. Otherwise, the GetInsertCommand still will be using information from the previous statement, which might not be correct. The SQL statements are first generated when the application calls either Update or GetInsertCommand.

GetUpdateCommand Method

Gets the automatically generated VirtuosoCommand object required to perform updates on the Virtuoso database.

```
public VirtuosoCommand GetUpdateCommand();
```

Return Value

The automatically generated VirtuosoCommand object required to perform updates.

Remarks

You can use the GetUpdateCommand method for informational or troubleshooting purposes because it returns the VirtuosoCommand object to be executed.

You can also use GetUpdateCommand as the basis of a modified command. For example, you might call GetUpdateCommand and modify the CommandTimeout value, and then explicitly set that on the VirtuosoDataAdapter.

After the SQL statement is first generated, you must explicitly call RefreshSchema if you change the statement in any way. Otherwise, the GetUpdateCommand still will be using information from the previous statement, which might not be correct. The SQL statements are first generated when the application calls either Update or GetUpdateCommand.

RefreshSchema Method

Refreshes the database schema information used to generate INSERT, UPDATE, or DELETE statements.

```
public void RefreshSchema();
```

Remarks

You should call RefreshSchema whenever the SelectCommand value of the VirtuosoDataAdapter changes.

VirtuosoConnection Class

Represents an open connection to a Virtuoso database. This class cannot be inherited.

System.Object
System.MarshalByRefObject
System.ComponentModel.Component
VirtuosoConnection

```
                    public
                    sealed
                    class
                    VirtuosoConnection
                   extends,
                    Component

          implements,
                    ICloneable
                  ,
                    IDbConnection
                   {
       }
```

Thread Safety

Any public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Any instance members are not guaranteed to be thread safe.

Remarks

A VirtuosoConnection object represents a unique session to a Virtuoso database server.

The VirtuosoConnection object uses native resources such as network connection handles. You should always explicitly close any open VirtuosoConnection objects by calling Close or Dispose before the VirtuosoConnection object goes out of scope. Not doing so leaves the freeing of these native resources to garbage collection, which may not free them immediately. This, in turn, may eventually cause the underlying driver to run out of resources or reach a maximum limit, resulting in sporadic failures. For example, you might encounter Maximum Connections-related errors while a number of connections are waiting to be deleted by the garbage collector. Explicitly closing the connections by calling Close or Dispose allows a more efficient use of native resources, enhancing scalability and improving overall application performance.

VirtuosoConnection Constructor

Initializes a new instance of the VirtuosoConnection class.

VirtuosoConnection Constructor ()

Initializes a new instance of the VirtuosoConnection class.

```
public VirtuosoConnection();
```

Remarks

When a new instance of VirtuosoConnection is created, the read/write properties are set to the following initial values unless they are specifically set using their associated keywords in the ConnectionString property.

Table 7.5.

Properties	Initial Value
ConnectionString	empty string ("")

Properties	Initial Value
ConnectionTimeout	15
Database	empty string ("")

You can change the value for these properties only by using the ConnectionString property

VirtuosoConnection Constructor (string)

Initializes a new instance of the VirtuosoConnection class with the specified connection string.

```
public VirtuosoConnection( string  connectionString );
```

Parameters

connectionString
: The connection used to open the Virtuoso database.

Remarks

When a new instance of VirtuosoConnection is created, the read/write properties are set to the following initial values unless they are specifically set using their associated keywords in the ConnectionString property.

Table 7.6.

Properties	Initial Value
ConnectionString	connectionString
ConnectionTimeout	15
Database	empty string ("")

You can change the value for these properties only by using the ConnectionString property

Properties

ConnectionString Property

Gets or sets the string used to open a Virtuoso database.

```
public  string  ConnectionString  ;
```

Property Value

The connection string that includes the source database name, and other parameters needed to establish the initial connection. The default value is an empty string.

Implements

IDbConnection.ConnectionString

Remarks

The ConnectionString is similar to an OLE DB connection string, but is not identical. Unlike OLE DB or ADO, the connection string that is returned is the same as the user-set ConnectionString minus security information if the `Persist Security` Info value is set to `false` (default). The Virtuoso .NET Data Provider does not persist or return the password in a connection string unless you set `Persist Security Info` to true.

The ConnectionString property can be set only when the connection is closed. Many of the connection string values have corresponding read-only properties. When the connection string is set, all of these properties are updated, except when an error is detected. In this case, none of the properties are updated. VirtuosoConnection properties return only those settings contained in the ConnectionString.

Resetting the ConnectionString on a closed connection resets all connection string values (and related properties) including the password. For example, if you set a connection string that includes "Database=Demo", and then reset the connection string to "Data Source=myserver;User ID=dba;Password=dba", the Database property is no longer set to northwind.

The connection string is parsed immediately after being set. If errors in syntax are found when parsing, a runtime exception, such as ArgumentException, is generated. Other errors can be found only when an attempt is made to Open the connection.

Connection string has the following syntax. Each connection string is a sequence of settings Individual settings are separated by semicolons. Each setting is a pair of name and value delimited by the equal sign. Whitespace is ignored on either side of both names and values. Names are case insensitive. The value part can be quoted by either single or double quote characters or remain unquoted at all. However if it includes a semicolon, single quote, or double quote characters, it must be enclosed in either type of quotes. To embed the same character that is used for enclosing the value the character within the value must be doubled.

The following table lists the valid names for values within the ConnectionString.

Table 7.7.

Name	Default	Description
Connect Timeout -or- Connection Timeout	15	The length of time (in seconds) to wait for a connection to the server before terminating the attempt and generating an error.
Connection Lifetime	0	When a connection is returned to the pool, its creation time is compared with the current time, and the connection is destroyed if that time span (in seconds) exceeds the value specified by connection lifetime. Useful in clustered configurations to force load balancing between a running server and a server just brought on-line.
Data Source -or- Server -or- Address -or- Network Address		The name or network address of the instance of Virtuoso server to which to connect.
Encrypt	false	Specifies if the connection must be SSL encrypted. Currently encryption only works with an ODBC-based provider.
Enlist	true	When true, the pooler automatically enlists the connection in the creation thread's current transaction context.
Initial Catalog -or- Database		The name of the database.
Max Pool Size	100	The maximum number of connections allowed in the pool.
Min Pool Size	0	The minimum number of connections allowed in the pool.
Password		The password for the Virtuoso account logging on.

Name	Default	Description
-or- Pwd		
Persist Security Info	false	When set to 'false', security-sensitive information, such as the password, is not returned as part of the connection if the connection is open or has ever been in an open State. Resetting the connection string resets all connection string values including the password
Pooling	true	When true, the VirtuosoConnection object is drawn from the appropriate pool, or if necessary, is created and added to the appropriate pool.
User ID -or- Uid		The Virtuoso login name.

ConnectionTimeout Property

Gets or sets the time to wait while trying to establish a connection before terminating the attempt and generating an error.

```
public  int  ConnectionTimeout  ;
```

Property Value

The time (in seconds) to wait for a connection to open. The default value is 15 seconds.

Implements

IDbConnection.ConnectionTimeout

Remarks

A value of 0 indicates no limit, and should be avoided in a ConnectionString because an attempt to connect will wait indefinitely.

Database Property

Gets the name of the current database or the database to be used after a connection is opened.

```
public  string  Database  ;
```

Property Value

The name of the current database or the name of the database to be used once a connection is open. The default value is an empty string.

Implements

IDbConnection.Database

Remarks

Initially, the Database property is set in the connection string. The Database property can be updated by using the ChangeDatabase method.

State Property

Gets the current state of the connection.

```
public  ConnectionState  State  ;
```

Property Value

A bitwise combination of the ConnectionState values. The default is Closed.

Remarks

The allowed state changes are:

- From Closed to Open, using the Open method of the connection object.
- From Open to Closed, using either the Close method or the Dispose method of the connection object.

Methods

BeginTransaction Method

Begins a database transaction.

BeginTransaction Method ()

Begins a database transaction.

```
public VirtuosoTransaction BeginTransaction();
```

Return Value

An object representing the new transaction.

Remarks

To commit or roll back the transaction, you must explicitly use the Commit or Rollback methods.

To ensure that the Virtuoso .NET Data Provider transaction management model performs correctly, avoid using other transaction management models, such as those provided by the data source.

BeginTransaction Method (IsolationLevel)

Begins a database transaction with the specified isolation level.

```
public VirtuosoTransaction BeginTransaction( IsolationLevel  isoLevel );
```

Parameters

isoLevel
: The isolation level under which the transaction should run.

Return Value

An object representing the new transaction.

Remarks

To commit or roll back the transaction, you must explicitly use the Commit or Rollback methods.

To ensure that the Virtuoso .NET Data Provider transaction management model performs correctly, avoid using other transaction management models, such as those provided by the data source.

ChangeDatabase Method

Changes the current database for an open VirtuosoConnection.

```
public void ChangeDatabase( string  database );
```

Parameters

database
: The name of the database to use in place of the current database.

Implements

IDbConnection.ChangeDatabase

Remarks

The value parameter must contain a valid database name, and cannot contain a null value, an empty string (""), or a string with only blank characters.

Close Method

Closes the connection to the database. This is the preferred method of closing any open connection.

```
public void Close();
```

Implements

IDbConnection.Close

Remarks

The Close method rolls back any pending transactions. It then releases the connection to the connection pool, or closes the connection if connection pooling is disabled.

An application can call Close more than one time. No exception is generated.

CreateCommand Method

Creates and returns a VirtuosoCommand object associated with the VirtuosoConnection.

```
public VirtuosoCommand CreateCommand();
```

Return Value

A VirtuosoCommand object.

EnlistDistributedTransaction Method

Enlists in the specified transaction as a distributed transaction.

```
public void EnlistDistributedTransaction( ITransaction  transaction );
```

Parameters

transaction
: A reference to an existing transaction in which to enlist or null to unenlist.

Remarks

You can enlist in an existing distributed transaction using the EnlistDistributedTransaction method if auto-enlistment is disabled. Enlisting in an existing distributed transaction ensures that, if the transaction is committed or rolled back, modifications made by the code at the data source are also committed or rolled back.

ICloneable.Clone Method

This member supports the Microsoft® .NET Framework infrastructure and is not intended to be used directly from your code.

```
object ICloneable.Clone();
```

IDbConnection.BeginTransaction Method

This member supports the Microsoft® .NET Framework infrastructure and is not intended to be used directly from your code.

IDbConnection.BeginTransaction Method ()

This member supports the Microsoft® .NET Framework infrastructure and is not intended to be used directly from your code.

```
IDbTransaction IDbConnection.BeginTransaction();
```

IDbConnection.BeginTransaction Method (IsolationLevel)

This member supports the Microsoft® .NET Framework infrastructure and is not intended to be used directly from your code.

```
IDbTransaction IDbConnection.BeginTransaction( IsolationLevel  isoLevel );
```

IDbConnection.CreateCommand Method

This member supports the Microsoft® .NET Framework infrastructure and is not intended to be used directly from your code.

```
IDbCommand IDbConnection.CreateCommand();
```

Open Method

Opens a database connection with the property settings specified by the ConnectionString.

```
public void Open();
```

Implements

IDbConnection.Open

Remarks

The VirtuosoConnection draws an open connection from the connection pool if one is available. Otherwise, it establishes a new connection to an instance of Virtuoso server.

Note:
If the VirtuosoConnection goes out of scope, it is not closed. Therefore, you must explicitly close the connection by calling Close or Dispose.

VirtuosoDataAdapter Class

Represents a set of data commands and a connection to a data source that are used to fill the DataSet and update the data source. This class cannot be inherited.

System.Object
System.MarshalByRefObject
System.ComponentModel.Component
System.Data.Common.DataAdapter
System.Data.Common.DbDataAdapter
VirtuosoDataAdapter

```
                    public
                    sealed
                    class
                    VirtuosoDataAdapter
                   extends,
                    DbDataAdapter

        implements,
                    IDbDataAdapter
                   {
    }
```

Thread Safety

Any public static (Shared in Microsoft® Visual Basic®) members of this type are safe for multithreaded operations. Any instance members are not guaranteed to be thread safe.

Remarks

The VirtuosoDataAdapter serves as a bridge between a DataSet and data source for retrieving and saving data. The VirtuosoDataAdapter provides this bridge by using Fill to load data from the data source into the DataSet, and using Update to send changes made in the DataSet back to the data source.

The VirtuosoDataAdapter also includes the SelectCommand, InsertCommand, DeleteCommand, UpdateCommand, and TableMappings properties to facilitate loading and updating of data.

VirtuosoDataAdapter Constructor

VirtuosoDataAdapter Constructor ()

Initializes a new instance of the VirtuosoDataAdapter class.

```
public VirtuosoDataAdapter();
```

Remarks

When you create an instance of VirtuosoDataAdapter, the following read/write properties are set to their default values, as shown in the table.

Table 7.8.

Properties	Default Value
MissingMappingAction	MissingMappingAction.Passthrough
MissingSchemaAction	MissingSchemaAction.Add

You can change the value of any of these properties through a separate call to the property.

VirtuosoDataAdapter Constructor (VirtuosoCommand)

Initializes a new instance of the VirtuosoDataAdapter class with the specified VirtuosoCommand as the SelectCommand property.

```
public VirtuosoDataAdapter( VirtuosoCommand  selectCommand );
```

Parameters

selectCommand
: A VirtuosoCommand that is an SQL SELECT statement or stored procedure, and is set as the SelectCommand property of the VirtuosoDataAdapter.

Remarks

This implementation of the VirtuosoDataAdapter constructor sets the SelectCommand property to the value specified in the selectCommand parameter.

When you create an instance of VirtuosoDataAdapter, the following read/write properties are set to their default values, as shown in the table.

Table 7.9.

Properties	Default Value
MissingMappingAction	MissingMappingAction.Passthrough
MissingSchemaAction	MissingSchemaAction.Add

You can change the value of any of these properties through a separate call to the property.

VirtuosoDataAdapter Constructor (string, VirtuosoConnection)

Initializes a new instance of the VirtuosoDataAdapter class with an SQL SELECT statement and a VirtuosoConnection.

```
public VirtuosoDataAdapter( string  selectCommandText ,
                            VirtuosoConnection  selectConnection );
```

Parameters

selectCommandText
: A string that is a SQL SELECT statement or stored procedure to be used by the SelectCommand property of the VirtuosoDataAdapter.

selectConnection
: A VirtuosoConnection that represents the connection.

VirtuosoDataAdapter Constructor (string, string)

Initializes a new instance of the VirtuosoDataAdapter class with an SQL SELECT statement and a connection string.

```
public VirtuosoDataAdapter( string  selectCommandText ,
                            string  selectConnectionString );
```

Parameters

selectCommandText
: A string that is a SQL SELECT statement or stored procedure to be used by the SelectCommand property of the VirtuosoDataAdapter.

selectConnectionString
: The connection string.

Properties

DeleteCommand

Gets or sets an SQL statement or stored procedure used to delete records in the data source.

```
public  VirtuosoCommand  DeleteCommand  ;
```

Property Value

A VirtuosoCommand used during an update operation to delete records in the database that correspond to deleted rows in the DataSet.

Remarks

When DeleteCommand is assigned to a previously created VirtuosoCommand, the VirtuosoCommand is not cloned. The DeleteCommand maintains a reference to the previously created VirtuosoCommand object.

InsertCommand

Gets or sets an SQL statement or stored procedure used to insert new records in the data source.

```
public  VirtuosoCommand  InsertCommand  ;
```

Property Value

A VirtuosoCommand used during an update operation to insert records in the database that correspond to new rows in the DataSet.

Remarks

When InsertCommand is assigned to a previously created VirtuosoCommand, the VirtuosoCommand is not cloned. The InsertCommand maintains a reference to the previously created VirtuosoCommand object.

SelectCommand

Gets or sets an SQL statement or stored procedure used to select records in the data source.

```
public  VirtuosoCommand  SelectCommand  ;
```

Property Value

A VirtuosoCommand used during Fill to select records from the database for placement in the DataSet.

Remarks

When SelectCommand is assigned to a previously created VirtuosoCommand, the VirtuosoCommand is not cloned. The SelectCommand maintains a reference to the previously created VirtuosoCommand object.

If the SelectCommand does not return any rows, no tables are added to the DataSet, and no exception is raised.

UpdateCommand

Gets or sets an SQL statement or stored procedure used to update records in the data source.

```
public  VirtuosoCommand  UpdateCommand  ;
```

Property Value

A VirtuosoCommand used during an update operation to update records in the database that correspond to modified rows in the DataSet.

Remarks

When UpdateCommand is assigned to a previously created VirtuosoCommand, the VirtuosoCommand is not cloned. The UpdateCommand maintains a reference to the previously created VirtuosoCommand object.

Methods

CreateRowUpdatedEvent

This member overrides DbDataAdapter.CreateRowUpdatedEvent.

```
protected override RowUpdatedEventArgs CreateRowUpdatedEvent( DataRow  dataRow ,
                                                              IDbCommand  command ,
                                                              StatementType
statementType ,
                                                              DataTableMapping
tableMapping );
```

CreateRowUpdatingEvent

This member overrides DbDataAdapter.CreateRowUpdatingEvent.

```
protected override RowUpdatedEventArgs CreateRowUpdatingEvent( DataRow  dataRow ,
                                                               IDbCommand  command ,
                                                               StatementType
statementType ,
                                                               DataTableMapping
tableMapping );
```

OnRowUpdated

Raises the RowUpdated event.

```
protected override void OnRowUpdated( RowUpdatedEventArgs  value );
```

Parameters

value
: A System.Data.Common.RowUpdatedEventArgs object that contains the event data.

Remarks

Raising an event invokes the event handler through a delegate. For an overview, see "Raising an Event" in the Microsoft® .NET Framework SDK documentation.

OnRowUpdating

Raises the RowUpdating event.

```
protected override void OnRowUpdating( RowUpdatingEventArgs  value );
```

Parameters

value
: A System.Data.Common.RowUpdatingEventArgs object that contains the event data.

Remarks

Raising an event invokes the event handler through a delegate. For an overview, see "Raising an Event" in the Microsoft® .NET Framework SDK documentation.

Events

RowUpdated

Occurs during an Update operation after a command is executed against the data source.

```
public  event  VirtuosoRowUpdatedEventHandler  RowUpdated  ;
```

Event Data

The event handler receives an argument of type VirtuosoRowUpdatedEventArgs containing data related to this event. The following VirtuosoRowUpdatedEventArgs properties provide information specific to this event.

Table 7.10.

Property	Description
Command	Gets the VirtuosoCommand executed when Update is called.
Errors (inherited from RowUpdatedEventArgs)	Gets any errors generated by the .NET data provider when the Command was executed.
RecordsAffected (inherited from RowUpdatedEventArgs)	Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.
Row (inherited from RowUpdatedEventArgs)	Gets the DataRow sent through an Update.
StatementType (inherited from RowUpdatedEventArgs)	Gets the type of SQL statement executed.
Status (inherited from RowUpdatedEventArgs)	Gets the UpdateStatus of the Command.
TableMapping (inherited from RowUpdatedEventArgs)	Gets the DataTableMapping sent through an Update.

Remarks

When using the Update method, there are two events that occur per data row updated. The order of execution is as follows:

1. The values in the DataRow are moved to the parameter values.
2. The OnRowUpdating event is raised.
3. The command executes.
4. If the UpdateRowSource enumeration is set to FirstReturnedRecord, the first returned result is placed in the DataRow.
5. If there are output parameters, they are placed in the DataRow.
6. The OnRowUpdated event is raised.
7. AcceptChanges is called.

RowUpdating

Occurs during an Update operation before a command is executed against the data source.

```
public event VirtuosoRowUpdatingEventHandler RowUpdating ;
```

Event Data

The event handler receives an argument of type VirtuosoRowUpdatingEventArgs containing data related to this event. The following VirtuosoRowUpdatingEventArgs properties provide information specific to this event.

Table 7.11.

Property	Description
Command	Gets or sets the VirtuosoCommand to execute when Update is called.
Errors (inherited from RowUpdatingEventArgs)	Gets any errors generated by the .NET data provider when the Command executes.
Row (inherited from RowUpdatingEventArgs)	Gets the DataRow to send through an Update.
StatementType (inherited from RowUpdatingEventArgs)	Gets the type of SQL statement to execute.
Status (inherited from RowUpdatingEventArgs)	Gets the UpdateStatus of the Command.
TableMapping (inherited from RowUpdatingEventArgs)	Gets the DataTableMapping to send through the Update.

Remarks

When using the Update method, there are two events that occur per data row updated. The order of execution is as follows:

1. The values in the DataRow are moved to the parameter values.
2. The OnRowUpdating event is raised.
3. The command executes.
4. If the UpdateRowSource enumeration is set to FirstReturnedRecord, the first returned result is placed in the DataRow.
5. If there are output parameters, they are placed in the DataRow.
6. The OnRowUpdated event is raised.
7. AcceptChanges is called.

VirtuosoDataReader Class

Provides a means of reading a forward-only stream of rows from a Virtuoso database. This class cannot be inherited.

System.Object

System.MarshalByRefObject

VirtuosoDataReader

```
public
sealed
class
VirtuosoDataReader
extends,
```

```
                MarshalByRefObject

        implements,
                IDataReader
              ,
                IDataRecord
              ,
                IDisposable
              ,
                IEnumerable
               {
    }
```

Thread Safety

Any public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Any instance members are not guaranteed to be thread safe.

Remarks

To create a VirtuosoDataReader, you must call the ExecuteReader method of the VirtuosoCommand object, rather than directly using a constructor.

Changes made to a resultset by another process or thread while data is being read may be visible to the user of the VirtuosoDataReader. However, the precise behavior is both driver and timing dependent.

IsClosed and RecordsAffected are the only properties that you can call after the VirtuosoDataReader is closed. In some cases, you must call Close before you can call RecordsAffected.

Properties

Depth Property

Gets a value indicating the depth of nesting for the current row.

```
public  int  Depth  ;
```

Property Value

The depth of nesting for the current row.

Implements

IDataReader.Depth

Remarks

The outermost table has a depth of zero. The Virtuoso .NET Data Provider does not support nesting and always returns zero.

FieldCount Property

Gets the number of columns in the current row.

```
public  int  FieldCount  ;
```

Property Value

When not positioned in a valid record set, 0; otherwise the number of columns in the current record. The default is -1.

Implements

IDataRecord.FieldCount

Remarks

After executing a query that does not return rows, FieldCount returns 0.

IsClosed Property

Gets a value indicating whether the data reader is closed.

```
public  bool  IsClosed  ;
```

Property Value

`true` if the VirtuosoDataReader is closed; otherwise, `false` .

Implements

IDataReader.IsClosed

Remarks

IsClosed and RecordsAffected are the only properties that you can call after the VirtuosoDataReader is closed.

Item Property

Item Property (int)

Gets the value of the specified column in its native format given the column ordinal.

```
public  object  this[int i]  ;
```

Parameters

i
: The zero-based column ordinal.

Property Value

The value of the specified column in its native format.

Implements

IDataRecord.Item

Item Property (string)

Gets the value of the specified column in its native format given the column name.

```
public  object  this[string name]  ;
```

Parameters

name
: The column name.

Property Value

The value of the specified column in its native format.

Implements

IDataRecord.Item

RecordsAffected Property

Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.

```
public  int  RecordsAffected  ;
```

Property Value

The number of rows changed, inserted, or deleted; 0 if no rows were affected or the statement failed; and -1 for SELECT statements.

Implements

IDataReader.RecordsAffected

Remarks

IsClosed and RecordsAffected are the only properties that you can call after the VirtuosoDataReader is closed.

Methods

Close Method

Closes the VirtuosoDataReader object.

```
public void Close();
```

Implements

IDataReader.Close

Remarks

You must explicitly call the Close method when you are through using the VirtuosoDataReader to use the associated VirtuosoConnection for any other purpose.

The Close method fills in the values for output parameters, return values and RecordsAffected.

GetBoolean Method

Gets the value of the specified column as a Boolean.

```
public bool GetBoolean( int  i );
```

Parameters

i
: The zero-based column ordinal.

Return Value

A Boolean that is the value of the column.

Implements

IDataRecord.GetBoolean

Remarks

Call IsDBNull to check for null values before calling this method.

GetByte Method

```
public byte GetByte( int  i );
```

Parameters

i
: The zero-based column ordinal.

Return Value

The value of the specified column as a byte.

Implements

IDataRecord.GetByte

Remarks

Call IsDBNull to check for null values before calling this method.

GetBytes Method

Reads a stream of bytes from the specified column offset into the buffer as an array, starting at the given buffer offset.

```
public long GetBytes( int  i ,
                      long  fieldOffset ,
                      byte[]  buffer ,
                      int  bufferOffset ,
                      int  length );
```

Parameters

i
: The zero-based column ordinal.

fieldOffset
: The index within the field from which to begin the read operation.

buffer
: The buffer into which to read the stream of bytes.

bufferOffset
: The index for buffer to begin the read operation.

length
: The number of bytes to read.

Return Value

The actual number of bytes read.

Implements

IDataRecord.GetBytes

Remarks

GetBytes returns the number of available bytes in the field. In most cases this is the exact length of the field. However, the number returned may be less than the true length of the field if GetBytes has already been used to obtain bytes from the field. This may be the case, for example, if the VirtuosoDataReader is reading a large data structure into a buffer. For more information, see the SequentialAccess setting of System.Data.CommandBehavior in the Microsoft® .NET Framework SDK documentation.

If you pass a buffer that is a null value, GetBytes returns the length of the field in bytes.

GetChar Method

```
public char GetChar( int  i );
```

Parameters

i
: The zero-based column ordinal.

Return Value

The value of the specified column as a character.

Implements

IDataRecord.GetChar

Remarks

Call IsDBNull to check for null values before calling this method.

GetChars Method

Reads a stream of characters from the specified column offset into the buffer as an array, starting at the given buffer offset.

```
public long GetChars( int  i ,
                      long  fieldOffset ,
                      char[]  buffer ,
                      int  bufferOffset ,
                      int  length );
```

Parameters

i
: The zero-based column ordinal.

fieldOffset
: The index within the field from which to begin the read operation.

buffer
: The buffer into which to copy data..

bufferOffset
: The index for buffer to begin the read operation.

length
: The number of characters to read.

Return Value

The actual number of characters read.

Implements

IDataRecord.GetChars

Remarks

GetChars returns the number of available characters in the field. In most cases this is the exact length of the field. However, the number returned may be less than the true length of the field if GetChars has already been used to obtain characters from the field. This may be the case, for example, if the VirtuosoDataReader is reading a large data structure into a buffer. For more information, see the SequentialAccess setting of System.Data.CommandBehavior in the Microsoft® .NET Framework SDK documentation.

If you pass a buffer that is a null value. GetChars returns the length of the field in characters.

GetData Method

Not currently supported.

```
public IDataReader GetData( int  i );
```

Implements

IDataRecord.GetData

GetDataTypeName Method

Gets the name of the source data type.

```
public string GetDataTypeName( int  i );
```

Parameters

i
The zero-based column ordinal.

Return Value

The name of the source data type.

Implements

IDataRecord.GetDataTypeName

GetDateTime Method

Gets the value of the specified column as a DateTime object.

```
public DateTime GetDateTime( int  i );
```

Parameters

i
The zero-based column ordinal.

Return Value

The value of the specified column as a DateTime object.

Implements

IDataRecord.GetDateTime

Remarks

Call IsDBNull to check for null values before calling this method.

GetDecimal Method

Gets the value of the specified column as a Decimal object.

```
public decimal GetDecimal( int  i );
```

Parameters

i
The zero-based column ordinal.

Return Value

The value of the specified column as a Decimal object.

Implements

IDataRecord.GetDecimal

Remarks

Call IsDBNull to check for null values before calling this method.

GetDouble Method

```
public double GetDouble( int  i );
```

Parameters

i
: The zero-based column ordinal.

Return Value

The value of the specified column as a double-precision floating point number.

Implements

IDataRecord.GetDouble

Remarks

Call IsDBNull to check for null values before calling this method.

GetFieldType Method

Gets the Type that is the data type of the object.

```
public Type GetFieldType( int  i );
```

Parameters

i
: The zero-based column ordinal.

Return Value

The Type that is the data type of the object.

Implements

IDataRecord.GetFieldType

GetFloat Method

```
public float GetFloat( int  i );
```

Parameters

i
: The zero-based column ordinal.

Return Value

The value of the specified column as a single-precision floating-point number.

Implements

IDataRecord.GetFloat

Remarks

Call IsDBNull to check for null values before calling this method.

GetGuid Method

Gets the value of the specified column as a globally-unique identifier (GUID).

```
public Guid GetGuid( int  i );
```

Parameters

i
: The zero-based column ordinal.

Return Value

The value of the specified column as a GUID.

Implements

IDataRecord.GetGuid

Remarks

Call IsDBNull to check for null values before calling this method.

GetInt16 Method

Gets the value of the specified column as a 16-bit signed integer.

```
public short GetInt16( int  i );
```

Parameters

i
: The zero-based column ordinal.

Return Value

The value of the specified column as a 16-bit signed integer.

Implements

IDataRecord.GetInt16

Remarks

Call IsDBNull to check for null values before calling this method.

GetInt32 Method

Gets the value of the specified column as a 32-bit signed integer

```
public int GetInt32( int  i );
```

Parameters

i
: The zero-based column ordinal.

Return Value

The value of the specified column as a 32-bit signed integer.

Implements

IDataRecord.GetInt32

Remarks

Call IsDBNull to check for null values before calling this method.

GetInt64 Method

Gets the value of the specified column as a 64-bit signed integer.

```
public long GetInt64( int  i );
```

Parameters

i
: The zero-based column ordinal.

Return Value

The value of the specified column as a 64-bit signed integer.

Implements

IDataRecord.GetInt64

Remarks

Call IsDBNull to check for null values before calling this method.

GetName Method

Gets the name of the specified column.

```
public string GetName( int  i );
```

Parameters

i
: The zero-based column ordinal.

Return Value

A string that is the name of the specified column.

Implements

IDataRecord.GetName

GetOrdinal Method

Gets the column ordinal, given the name of the column.

```
public int GetOrdinal( string  name );
```

Parameters

name
: The name of the column.

Return Value

The zero-based column ordinal.

Implements

IDataRecord.GetOrdinal

GetSchemaTable Method

Returns a DataTable that describes the column metadata of the VirtuosoDataReader.

```
public DataTable GetSchemaTable();
```

Return Value

A DataTable that describes the column metadata.

Implements

IDataReader.GetSchemaTable

Remarks

For the GetSchemaTable method returns metadata about each column in the following order:

Table 7.12.

DataReader Column	Description
ColumnName	The name of the column; this might not be unique. If the column name cannot be determined, a null value is returned. This name always reflects the most recent naming of the column in the current view or command text.
ColumnOrdinal	The ordinal of the column. This is zero for the bookmark column of the row, if any. Other columns are numbered starting with one. This column cannot contain a null value.
ColumnSize	The maximum possible length of a value in the column. For columns that use a fixed-length data type, this is the size of the data type.
NumericPrecision	If ProviderType is a numeric data type, this is the maximum precision of the column. The precision depends on the definition of the column. If ProviderType is not a numeric data type, this is a null value.
NumericScale	If ProviderType is decimal, the number of digits to the right of the decimal point. Otherwise, this is a null value.
DataType	Maps to the .Net Framework type of the column.
ProviderType	The indicator of the column's data type. If the data type of the column varies from row to row, this must be Object. This column cannot contain a null value.
IsLong	Set if the column contains a Binary Long Object (BLOB) that contains very long data.
AllowDBNull	Set if the consumer can set the column to a null value or if the provider cannot determine whether or not the consumer can set the column to a null value. Otherwise, not set. A column may contain null values, even if it cannot be set to a null value.
IsReadOnly	`true` if the column can be modified; otherwise `false` .
IsRowVersion	
IsUnique	`true` : No two rows in the base table-the table returned in BaseTableName-can have the same value in this column. IsUnique is guaranteed to be true if the column constitutes a key by itself or if there is a constraint of type UNIQUE that applies only to this column. `false` : The column can contain duplicate values in the base table. The default of this column is `false` .
IsKey	`true` : The column is one of a set of columns in the rowset that, taken together, uniquely identify the row. The set of columns with IsKey set to `true` must uniquely identify a row in the rowset. There is no requirement that this set of columns is a minimal set of columns. This set of columns may be generated from a base table primary key, a unique constraint or a unique index. `false` : The column is not required to uniquely identify the row.
IsAutoIncrement	`true` : The column assigns values to new rows in fixed increments. `false` : The column does not assign values to new rows in fixed increments. The default of this column is `false` .
BaseSchemaName	The name of the schema in the data store that contains the column. A null value if the base schema name cannot be determined. The default of this column is a null value.
BaseCatalogName	The name of the catalog in the data store that contains the column. NULL if the base catalog name cannot be determined. The default of this column is a null value.
BaseTableName	

DataReader Column	Description
	The name of the table or view in the data store that contains the column. A null value if the base table name cannot be determined. The default of this column is a null value.
BaseColumnName	The name of the column in the data store. This might be different than the column name returned in the ColumnName column if an alias was used. A null value if the base column name cannot be determined or if the rowset column is derived, but not identical to, a column in the data store. The default of this column is a null value.

GetString Method

Gets the value of the specified column as a string.

```
public string GetString( int  i );
```

Parameters

i
: The zero-based column ordinal.

Return Value

The value of the specified column as a string.

Implements

IDataRecord.GetString

Remarks

Call IsDBNull to check for null values before calling this method.

GetValue Method

Gets the value of the column at the specified ordinal in its native format.

```
public object GetValue( int  i );
```

Parameters

i
: The zero-based column ordinal.

Return Value

The value to return.

Implements

IDataRecord.GetValue

Remarks

This method returns DBNull for null database columns.

GetValues Method

Gets all the attribute columns in the current row.

```
public int GetValues( object[]  values );
```

Parameters

values
: An array of type Object into which to copy the attribute columns.

Return Value

The number of instances of Object in the array.

Implements

IDataRecord.GetValues

Remarks

For most applications, the GetValues method provides an efficient means for retrieving all columns, rather than retrieving each column individually.

You can pass an Object array that contains fewer than the number of columns contained in the resulting row. Only the amount of data the Object array holds is copied to the array. You can also pass an Object array whose length is more than the number of columns contained in the resulting row.

This method returns DBNull for null database columns.

IDisposable.Dispose Method

This member supports the Microsoft® .NET Framework infrastructure and is not intended to be used directly from your code.

```
void IDisposable.Dispose();
```

IEnumerable.GetEnumerator Method

This member supports the Microsoft® .NET Framework infrastructure and is not intended to be used directly from your code.

```
IEnumerable IEnumerable.GetEnumerator();
```

IsDBNull Method

Gets a value indicating whether the column contains non-existent or missing values.

```
public bool IsDBNull( int  i );
```

Parameters

i
: The zero-based column ordinal.

Return Value

`true` if the specified column value is equivalent to DBNull; otherwise, `false` .

Implements

IDataRecord.IsDBNull

Remarks

To avoid raising an error, call this method to check for null column values before calling the typed Get methods (for example, GetByte, GetChar, and so on).

NextResult Method

Advances the VirtuosoDataReader to the next result, when reading the results of batch SQL statements.

```
public bool NextResult();
```

Return Value

`true` if there are more result sets; otherwise, `false` .

Implements

IDataReader.NextResult

Remarks

Used to process multiple results, which can be generated by executing batch SQL statements.

By default, the VirtuosoDataReader is positioned on the first result.

Read Method

Advances the VirtuosoDataReader to the next record.

```
public bool Read();
```

Return Value

`true` if there are more rows; otherwise, `false` .

Implements

IDataReader.Read

Remarks

The default position of the VirtuosoDataReader is prior to the first record. Therefore, you must call Read to begin accessing any data.

VirtuosoError Class

Collects information relevant to a warning or error returned by Virtuoso server. This class cannot be inherited.

System.Object
VirtuosoError

```
                    public
                    sealed
                    class
                    VirtuosoError
                   {
        }
```

Thread Safety

Any public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Any instance members are not guaranteed to be thread safe.

Remarks

This class is created by the Virtuoso .NET Data Provider when an error occurs. An instance of VirtuosoError is created and managed by the VirtuosoErrorCollection, which in turn is created by the VirtuosoException class.

Properties

Message

Gets a short description of the error.

```
public  string  Message  ;
```

Property Value

A description of the error.

SQLState

Gets the five-character error code that follows the ANSI SQL standard for the database.

```
public  string  SQLState  ;
```

Property Value

The five-character error code, which identifies the source of the error.

VirtuosoErrorCollection Class

Collects all errors generated by the Virtuoso .NET Data Provider. This class cannot be inherited.

System.Object
VirtuosoErrorCollection

```
                  public
                  sealed
                  class
                  VirtuosoErrorCollection
                implements,
                  ICollection
                 {
      }
```

Thread Safety

Any public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Any instance members are not guaranteed to be thread safe.

Remarks

This class is created by VirtuosoException to collect instances of the VirtuosoError class. VirtuosoErrorCollection always contains at least one instance of the VirtuosoError class.

Properties

Count Property

Gets the number of errors in the collection.

```
public  int  Count  ;
```

Property Value

The total number of errors in the collection.

Implements

ICollection.Count

Item Property

Gets the error at the specified index.

```
public  VirtuosoError  this[int i]  ;
```

Parameters

i
: The zero-based index of the error to retrieve.

Property Value

A VirtuosoError that contains the error at the specified index.

Methods

CopyTo Method

Copies the elements of the VirtuosoErrorCollection into an array, starting at the given index within the array.

```
public void CopyTo( Array  array ,
                    int  i );
```

Parameters

array
: The array into which to copy the elements.

i
: The starting index of array.

Implements

ICollection.CopyTo

GetEnumerator Method

This member supports the Microsoft® .NET Framework infrastructure and is not intended to be used directly from your code.

```
IEnumerable IEnumerable.GetEnumerator();
```

VirtuosoException Class

The exception that is thrown when Virtuoso server returns a warning or error. This class cannot be inherited.

System.Object
System.Exception
System.SystemException
VirtuosoException

```
                    public
                    sealed
                    class
                    VirtuosoException
                   extends,
                    SystemException
                   {
}
```

Thread Safety

Any public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Any instance members are not guaranteed to be thread safe.

Remarks

This class is created whenever the Virtuoso .NET Data Provider encounters an error generated by the server (Client-side errors are raised as standard common language runtime exceptions.). It always contains at least one instance of VirtuosoError.

Properties

Errors Property

Gets a collection of one or more VirtuosoError objects that give detailed information about exceptions generated by the Virtuoso .NET Data Provider.

```
public  VirtuosoErrorCollection  Errors  ;
```

Property Value

The collected instances of the VirtuosoError class.

Remarks

This property is a wrapper for the VirtuosoErrorCollection.

Message Property

Gets the text describing the error.

```
public  string  Message  ;
```

Property Value

The text describing the error.

Remarks

This is a wrapper for the Message property of the first VirtuosoError in the Errors property.

VirtuosoInfoMessageEventArgs Class

Provides data for the InfoMessage event. This class cannot be inherited.

System.Object
System.EventArgs
VirtuosoInfoMessageEventArgs

```
                    public
                    sealed
                    class
                    VirtuosoInfoMessageEventArgs
                   extends,
                    EventArgs
                   {
      }
```

Thread Safety

Any public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Any instance members are not guaranteed to be thread safe.

Remarks

The InfoMessage event contains a VirtuosoErrorCollection collection with warnings sent from the Virtuoso srver.

Properties

Errors Property

Gets the collection of warnings sent from the Virtuoso server.

```
public  VirtuosoErrorCollection  Errors  ;
```

Property Value

The collection of warnings sent from the server.

VirtuosoInfoMessageEventHandler Delegate

Represents the method that will handle the InfoMessage event of a VirtuosoConnection.

```
public delegate void VirtuosoInfoMessageEventHandler( object  sender ,
                                                      VirtuosoInfoMessageEventArgs  e
);
```

Parameters

The declaration of your event handler must have the same parameters as the VirtuosoInfoMessageEventHandler delegate declaration.

sender
: The source of the event.

e
: A VirtuosoInfoMessageEventArgs object that contains the event data.

Remarks

When you create a VirtuosoInfoMessageEventArgs delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, unless you remove the delegate. For more information about event handler delegates, see "Events and Delegates" in the .NET Framework SDK documentation.

VirtuosoParameter Class

Represents a parameter to an VirtuosoCommand and optionally, its mapping to a DataColumn. This class cannot be inherited.

System.Object
System.MarshalByRefObject
VirtuosoParameter

```
                  public
                  sealed
                  class
                  VirtuosoParameter
                 extends,
                  MarshalByRefObject

        implements,
                  IDbDataParameter
                ,
                  IDataParameter
                ,
                  ICloneable
                 {
    }
```

Thread Safety

Any public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Any instance members are not guaranteed to be thread safe.

VirtuosoParameter Constructor

VirtuosoParameter Constructor ()

Initializes a new instance of the VirtuosoParameter class.

```
public VirtuosoParameter();
```

VirtuosoParameter Constructor (string, object)

Initializes a new instance of the VirtuosoParameter class with the parameter name and value.

```
public VirtuosoParameter( string  parameterName ,
                          object  object );
```

Parameters

parameterName
: The name of the parameter to map.

value
: An Object that is the value of the VirtuosoParameter.

Remarks

When you specify an Object in the value parameter, the VirtDbType is inferred from the .NET Framework type of the Object.

VirtuosoParameter Constructor (string, VirtDbType)

Initializes a new instance of the VirtuosoParameter class with the parameter name and the data type.

```
public VirtuosoParameter( string  parameterName ,
                          VirtDbType  dbType );
```

Parameters

parameterName
: The name of the parameter to map.

dbType
: One of the VirtDbType values.

Remarks

The data type, and if appropriate, Size and Precision are inferred from the value of the dbType parameter.

VirtuosoParameter Constructor (string, VirtDbType, int)

Initializes a new instance of the VirtuosoParameter class with the parameter name, the VirtDbType, and the size.

```
public VirtuosoParameter( string  parameterName ,
                          VirtDbType  dbType ,
                          int  size );
```

Parameters

parameterName
: The name of the parameter to map.

dbType
: One of the VirtDbType values.

size
: The width of the parameter.

Remarks

The Size is inferred from the value of the dbType parameter if it is not explicitly set in the size parameter.

VirtuosoParameter Constructor (string, VirtDbType, int, string)

Initializes a new instance of the VirtuosoParameter class with the parameter name, VirtDbType, size, and source column name.

```
public VirtuosoParameter( string  parameterName ,
                          VirtDbType  dbType ,
```

```
                        int  size ,
                        string  sourceColumn );
```

Parameters

parameterName
: The name of the parameter to map.

dbType
: One of the VirtDbType values.

size
: The width of the parameter.

sourceColumn
: The name of the source column.

Remarks

The Size is inferred from the value of the dbType parameter if it is not explicitly set in the size parameter.

VirtuosoParameter Constructor (string, VirtDbType, int, ParameterDirection, Boolean, Byte, Byte, String, DataRowVersion, Object)

Initializes a new instance of the VirtuosoParameter class with the parameter name, the type of the parameter, the size of the parameter, a ParameterDirection, the precision of the parameter, the scale of the parameter, the source column, a DataRowVersion to use, and the value of the parameter.

```
public VirtuosoParameter( string  parameterName ,
                          VirtDbType  dbType ,
                          int  size ,
                          ParameterDirection  direction ,
                          bool  isNullable ,
                          byte  precision ,
                          byte  scale ,
                          string  sourceColumn ,
                          DataRowVersion  sourceVersion ,
                          object  value );
```

Parameters

parameterName
: The name of the parameter to map.

dbType
: One of the VirtDbType values.

size
: The width of the parameter.

direction
: One of the System.Data.ParameterDirection values.

isNullable
: `true` if the value of the field can be null, otherwise `false` .

precision
: The total number of digits to the left and right of the decimal point to which Value is resolved.

scale
: The total number of decimal places to which Value is resolved.

sourceColumn
: The name of the source column.

sourceVersion
: One of the System.Data.DataRowVersion values.

value
: An Object that is the value of the VirtuosoParameter.

Remarks

The Size and Precision are inferred from the value of the dbType parameter if they are not explicitly set in the size and precision parameters.

Properties

DbType Property

Gets or sets the DbType of the parameter.

```
public  DbType  DbType  ;
```

Property Value

One of the System.Data.DbType values. The default is String.

Implements

IDataParameter.DbType

Remarks

The VirtDbType and DbType are linked. Therefore, setting the DbType changes the VirtDbType to a supporting VirtDbType.

For a list of the supported data types, see the appropriate VirtDbType member.

Direction Property

Gets or sets a value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter.

```
public  ParameterDirection  Direction  ;
```

Property Value

One of the System.Data.ParameterDirection values. The default is Input.

Implements

IDataParameter.Direction

Remarks

If the ParameterDirection is Output, and execution of the associated VirtuosoCommand does not return a value, the VirtuosoParameter will contain a null value. Null values are handled using the DBNull class. After the last row from the last resultset is read, the Output, InputOut, and ReturnValue parameters are updated.

IsNullable Property

Gets or sets a value indicating whether the parameter accepts null values.

```
public  bool  IsNullable  ;
```

Property Value

`true` if null values are accepted; otherwise, `false` . The default is `false` .

Implements

IDataParameter.IsNullable

Remarks

Null values are handled using the System.DBNull class.

ParameterName Property

Gets or sets the name of the VirtuosoParameter.

```
public  string  ParameterName  ;
```

Property Value

The name of the VirtuosoParameter. The default is an empty string ("").

Implements

IDataParameter.ParameterName

Remarks

Instead of named parameters, the Virtuoso .NET Data Provider uses positional parameters that are marked with a question mark (?) in the syntax of the command text. Parameter objects in the VirtuosoParameterCollection and the actual parameters accepted by the stored procedure or parameterized SQL statement correspond to each other based on the order in which the VirtuosoParameter objects are inserted into the collection rather than by parameter name. Parameter names can be supplied, but will be ignored during parameter object binding.

Precision Property

Gets or sets the maximum number of digits used to represent the Value property.

```
public  byte  Precision  ;
```

Property Value

The maximum number of digits used to represent the Value property. The default value is 0.

Implements

IDbDataParameter.Precision

Remarks

The Precision property is used only for decimal and numeric input parameters.

Scale Property

Gets or sets the number of decimal places to which Value is resolved.

```
public  byte  Scale  ;
```

Property Value

The number of decimal places to which Value is resolved. The default is 0.

Implements

IDbDataParameter.Scale

Remarks

The Scale property is used only for decimal and numeric input parameters.

Size Property

Gets or sets the maximum size, in bytes, of the data within the column.

```
public  int  Size  ;
```

Property Value

The maximum size, in bytes, of the data within the column. The default value is inferred from the parameter value.

Implements

IDbDataParameter.Size

Remarks

The Size property is used for binary and string types.

For variable-length data types, the Size property describes the maximum amount of data to transmit to the server. For example, for a string value, the Size property could be used to limit the amount of data sent to the server to the first one hundred bytes.

For nonstring data types and ANSI string data, the Size property refers to the number of bytes. For Unicode string data, the Size property refers to the number of characters. The count for strings does not include the terminating character.

If not explicitly set, the value of Size is inferred from the actual size of the specified parameter value.

For fixed-width data types, the value of Size is ignored. It can be retrieved for informational purposes, and returns the maximum amount of bytes the provider uses when transmitting the value of the parameter to the server.

SourceColumn Property

Gets or sets the name of the source column mapped to the DataSet and used for loading or returning the Value.

```
public  string  SourceColumn  ;
```

Property Value

The name of the source column that will be used to set the value of this parameter. The default is an empty string ("").

Implements

IDataParameter.SourceColumn

Remarks

When SourceColumn is set to anything other than an empty string, the value of the parameter is retrieved from the column with the SourceColumn name. If Direction is set to Input, the value is taken from the DataSet. If Direction is set to Output, the value is taken from the data source. A Direction of InputOutput is a combination of both.

SourceVersion Property

Gets or sets the DataRowVersion to use when loading Value.

```
public  DataRowVersion  DataRowVersion  ;
```

Property Value

One of the System.Data.DataRowVersion values. The default is Current.

Implements

IDataParameter.SourceVersion

Remarks

The SourceVersion is used by UpdateCommand during an Update operation to determine whether the parameter value is set to Current or Original. This allows primary keys to be updated. This property is ignored by InsertCommand and DeleteCommand.

This property is set to the version of the DataRow used by either the Item property (DataRow indexer), or the GetChildRows method of the DataRow object.

Value Property

Gets or sets the value of the parameter.

```
public  object  Value  ;
```

Property Value

An Object that is the value of the parameter. The default value is null.

Implements

IDataParameter.Value

Remarks

For input parameters, the value is bound to the VirtuosoCommand that is sent to the server. For output and return-value parameters, the value is set on completion of the VirtuosoCommand and after the VirtuosoDataReader is closed.

When sending a null parameter value to the server, the user must specify DBNull, not null. A null value in the system is an empty object that has no value. DBNull is used to represent null values.

If the application specifies the database type, the bound value is converted to that type when the provider sends the data to the server. The provider attempts to convert any type of value if it supports the IConvertible interface. Conversion errors may result if the specified type is not compatible with the value.

Both the DbType and VirtDbType properties can be inferred by setting Value. If applicable, the size, precision and scale will also be inferred from Value.

The Value property is overwritten by the Update method.

VirtDbType Property

Gets or sets the VirtDbType of the parameter.

```
public  VirtDbType  VirtDbType  ;
```

Property Value

One of the VirtDbType values. The default is NVarChar.

Remarks

The VirtDbType and DbType are linked. Therefore, setting the DbType changes the VirtDbType to a supporting VirtDbType.

For a list of the supported data types, see the appropriate VirtDbType member. For more information, see "Using Parameters with a DataAdapter" in the Microsoft® .NET Framework SDK documentation.

Methods

ICloneable.Clone Method

This member supports the Microsoft® .NET Framework infrastructure and is not intended to be used directly from your code.

```
object ICloneable.Clone();
```

ToString Method

Gets a string containing the ParameterName.

```
public override string ToString();
```

Return Value

A string containing the ParameterName.

VirtuosoParameterCollection Class

Represents a collection of parameters relevant to a VirtuosoCommand as well as their respective mappings to columns in a DataSet. This class cannot be inherited.

System.Object
System.MarshalByRefObject
VirtuosoParameterCollection

```
                    public
                    sealed
                    class
                    VirtuosoParameterCollection
                   extends,
                    MarshalByRefObject

          implements,
                    IDataParameterCollection
                  ,
                    ICollection
                  ,
                    IEnumerable
                  ,
                    IList
                   {
       }
```

Thread Safety

Any public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Any instance members are not guaranteed to be thread safe.

Properties

Count Property

Gets the number of VirtuosoParameter objects in the collection.

```
public  int  Count  ;
```

Property Value

The number of VirtuosoParameter objects in the collection.

Implements

ICollection.Count

Item Property

Item Property (int)

Gets or sets the VirtuosoParameter at the specified index.

```
public  object  this[int i]  ;
```

Parameters

index
: The zero-based index of the parameter to retrieve.

Property Value

The VirtuosoParameter at the specified index.

Item Property (string)

Gets or sets the VirtuosoParameter with the specified name.

```
public  object  this[string parameterName]  ;
```

Parameters

parameterName
: The name of the parameter to retrieve.

Property Value

The VirtuosoParameter with the specified name.

Methods

Add Method

Add Method (object)

Adds the specified VirtuosoParameter object to the VirtuosoParameterCollection.

```
public int Add( object  value );
```

Parameters

value
: The VirtuosoParameter to add to the collection.

Return Value

The index in the collection of the new VirtuosoParameter object.

Implements

IList.Add

Add Method (VirtuosoParameter)

Adds the specified VirtuosoParameter object to the VirtuosoParameterCollection.

```
public VirtuosoParameter Add( VirtuosoParameter  value );
```

Parameters

value
: The VirtuosoParameter to add to the collection.

Return Value

A reference to the new VirtuosoParameter object.

Add Method (string, object)

Adds a VirtuosoParameter to the VirtuosoParameterCollection with the specified parameter name and value.

```
public VirtuosoParameter Add( string  parameterName ,
                              object  value );
```

Parameters

parameterName
: The name of the parameter.

value
: The Value of the VirtuosoParameter to add to the collection.

Return Value

The new VirtuosoParameter object.

Add Method (string, VirtDbType)

Adds a VirtuosoParameter to the VirtuosoParameterCollection with the specified parameter name and data type.

```
public VirtuosoParameter Add( string  parameterName ,
                              VirtDbType  dbType );
```

Parameters

parameterName
: The name of the parameter.

value
: One of the VirtDbType values.

Return Value

The new VirtuosoParameter object.

Add Method (string, VirtDbType, int)

Adds a VirtuosoParameter to the VirtuosoParameterCollection with the specified parameter name, data type, and parameter size.

```
public VirtuosoParameter Add( string  parameterName ,
                              VirtDbType  dbType ,
                              int  size );
```

Parameters

parameterName
: The name of the parameter.

value
: One of the VirtDbType values.

size
: The size of the parameter (width of the column).

Return Value

The new VirtuosoParameter object.

Add Method (string, VirtDbType, int, string)

Adds a VirtuosoParameter to the VirtuosoParameterCollection with the specified parameter name, data type, parameter size, and source column name.

```
public VirtuosoParameter Add( string  parameterName ,
                              VirtDbType  dbType ,
                              int  size ,
                              string  sourceColumn );
```

Parameters

parameterName
: The name of the parameter.

value
: One of the VirtDbType values.

size
: The size of the parameter (width of the column).

sourceColumn
: The name of the source column.

Return Value

The new VirtuosoParameter object.

Clear Method

Removes all items from the collection.

```
public void Clear();
```

Implements

IList.Clear

Contains Method

Contains Method (object)

Gets a value indicating whether a VirtuosoParameter object exists in the collection.

```
public bool Contains( object  value );
```

Parameters

value
: The value of the VirtuosoParameter object to find.

Return Value

`true` if the collection contains the VirtuosoParameter; otherwise, `false` .

Implements

IList.Contains

Contains Method (string)

Gets a value indicating whether a VirtuosoParameter object with the specified parameter name exists in the collection.

```
public bool Contains( string  parameterName );
```

Parameters

parameterName
: The name of the VirtuosoParameter object to find.

Return Value

`true` if the collection contains the VirtuosoParameter; otherwise, `false` .

Implements

IDataParameterCollection.Contains

CopyTo Method

Copies VirtuosoParameter objects from the VirtuosoParameterCollection to the specified array.

```
public void CopyTo( Array  array ,
                    int  index );
```

Parameters

array
: The array into which to copy the VirtuosoParameter objects.

index
: The starting index of the array.

Implements

ICollection.CopyTo

GetEnumerator Method

This member supports the Microsoft® .NET Framework infrastructure and is not intended to be used directly from your code.

```
public IEnumerator GetEnumerator();
```

IndexOf Method

IndexOf Method (object)

Gets the location in the collection of a VirtuosoParameter object.

```
public int IndexOf( object  value );
```

Parameters

value
: The VirtuosoParameter object to find.

Return Value

The zero-based location of the VirtuosoParameter in the collection.

Implements

IList.IndexOf

IndexOf Method (string)

Gets the location in the collection of the VirtuosoParameter object with the specified parameter name.

```
public int IndexOf( string  parameterName );
```

Parameters

parameterName
: The name of the VirtuosoParameter object to find.

Return Value

The zero-based location of the VirtuosoParameter in the collection.

Implements

IDataParameterCollection.IndexOf

Insert Method

Inserts a VirtuosoParameter into the collection at the specified index.

```
public void Insert( int  index ,
                    object  value );
```

Parameters

index
: The zero-based index where the parameter is to be inserted within the collection.

value
: The VirtuosoParameter to add to the collection.

Implements

IList.Insert

Remove Method

Removes the specified VirtuosoParameter from the collection.

```
public void Remove( object  value );
```

Parameters

value
: The VirtuosoParameter object to remove from the collection.

Implements

IList.Remove

RemoveAt Method

RemoveAt Method (int)

Removes the VirtuosoParameter at the specified index from the collection.

```
public void RemoveAt( int  index );
```

Parameters

index
: The zero-based index of the parameter to remove.

Implements

IList.RemoveAt

RemoveAt Method (string)

Removes the VirtuosoParameter with the specified name from the collection.

```
public void RemoveAt( string  parameterName );
```

Parameters

parameterName
: The name of the parameter to remove.

Implements

IDataParameterCollection.RemoveAt

VirtuosoPermission Class

Provides the capability for the Virtuoso .NET Data Provider to ensure that a user has a security level adequate to access a data source. This class cannot be inherited.

System.Object

System.Security.CodeAccessPermission

VirtuosoPermission

```
                    public
                    sealed
                    class
                    VirtuosoPermission
                   extends,
                    CodeAccessPermission

          implements,
                    IUnrestrictedPermission
                   {
      }
```

Thread Safety

Any public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Any instance members are not guaranteed to be thread safe.

VirtuosoPermission Constructor

VirtuosoPermission Constructor ()

Initializes a new instance of the VirtuosoPermission class.

```
public VirtuosoPermission();
```

VirtuosoPermission Constructor (PermissionState)

Initializes a new instance of the VirtuosoPermission class.

```
public VirtuosoPermission( PermissionState  state );
```

Parameters

state
: One of the System.Security.Permissions.PermissionState values.

Methods

Copy Method

Creates and returns an identical copy of the current permission object.

```
public override IPermission Copy();
```

Return Value

A copy of the current permission object.

Implements

IPermission.Copy

Remarks

A copy of a permission object represents the same access to resources as the original permission object.

FromXml Method

Reconstructs a security object with a specified state from an XML encoding.

```
public override void FromXml( SecurityElement  securityElement );
```

Parameters

securityElement
: The XML encoding to use to reconstruct the security object.

Implements

ISecurityEncodable.FromXml

Intersect Method

Returns a new permission object representing the intersection of the current permission object and the specified permission object.

```
public override IPermission Intersect( IPermission  target );
```

Parameters

target
: A permission object to intersect with the current permission object. It must be of the same type as the current permission object.

Return Value

A new permission object that represents the intersection of the current permission object and the specified permission object. This new permission object is a null reference (Nothing in Visual Basic) if the intersection is empty.

Implements

IPermission.Intersect

Remarks

The intersection of two permissions is a permission that describes the set of operations they both describe in common. Only a demand that passes both original permissions will pass the intersection.

IsSubsetOf Method

Returns a value indicating whether the current permission object is a subset of the specified permission object.

```
public override bool IsSubsetOf( IPermission  target );
```

Parameters

target
: A permission object that is to be tested for the subset relationship. This object must be of the same type as the current permission object.

Return Value

`true` if the current permission object is a subset of the specified permission object; otherwise `false` .

Implements

IPermission.IsSubsetOf

Remarks

The current permission object is a subset of the specified permission object if the current permission object specifies a set of operations that is wholly contained by the specified permission object. For example, a permission that represents access to C:\example.txt is a subset of a permission that represents access to C:\. If this method returns true, the current permission object represents no more access to the protected resource than does the specified permission object.

IsUnrestricted Method

Returns a value indicating whether the permission can be represented as unrestricted without any knowledge of the permission semantics.

```
public bool IsUnrestricted();
```

Return Value

`true` if the VirtuosoPermission instance was created with PermissionState.Unrestricted; otherwise, `false` .

Implements

IUnrestrictedPermission.IsUnrestricted

ToXml Method

Creates an XML encoding of the security object and its current state.

```
public override SecurityElement ToXml();
```

Return Value

An XML encoding of the security object, including any state information.

Implements

ISecurityEncodable.ToXml

Union Method

Creates a permission that is the union of the permission and the specified permission.

```
public override IPermission Union( IPermission  target );
```

Parameters

target
: A permission to combine with the current permission. It must be of the same type as the current permission.

Return Value

A new permission that represents the union of the current permission and the specified permission.

Remarks

The result of a call to Union is a permission that represents all the operations represented by both the current permission and the specified permission. Any demand that passes either permission passes their union.

VirtuosoPermissionAttribute

Allows security actions for VirtuosoPermission to be applied to code using declarative security. This class cannot be inherited.

System.Object
System.Attribute
System.Security.Permissions.SecurityAttribute
System.Security.Permissions.CodeAccessSecurityAttribute
VirtuosoPermissionAttribute

```
public
sealed
class
VirtuosoPermissionAttribute
extends,
```

```
                    CodeAccessSecurityAttribute
                   {
        }
```

Thread Safety

Any public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Any instance members are not guaranteed to be thread safe.

VirtuosoPermissionAttribute Constructor

Initializes a new instance of the VirtuosoPermissionAttribute class.

```
public VirtuosoPermissionAttribute();
```

Parameters

action
: One of the SecurityAction values representing an action that can be performed using declarative security.

Methods

CreatePermission Method

Returns a VirtuosoPermission object that is configured according to the attribute properties.

```
public override IPermission CreatePermission();
```

Return Value

A VirtuosoPermission object.

VirtuosoRowUpdatedEventArgs Class

Provides data for the RowUpdated event. This class cannot be inherited.

System.Object
System.EventArgs
System.Data.Common.RowUpdatedEventArgs
VirtuosoRowUpdatedEventArgs

```
                    public
                    sealed
                    class
                    VirtuosoRowUpdatedEventArgs
                   extends,
                    RowUpdatedEventArgs
                   {
        }
```

Thread Safety

Any public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Any instance members are not guaranteed to be thread safe.

VirtuosoRowUpdatedEventArgs Constructor

Initializes a new instance of the VirtuosoRowUpdatedEventArgs class.

```
public VirtuosoRowUpdatedEventArgs( DataRow  row ,
                                    IDbCommand  command ,
                                    StatementType  statementType ,
                                    DataTableMapping  tableMapping );
```

Parameters

row
: The DataRow sent through an Update.

command
: The IDbCommand executed when Update is called.

statementType
: One of the StatementType values that specifies the type of query executed.

tableMapping
: The DataTableMapping sent through an Update.

Properties

Command Property

Gets the VirtuosoCommand executed when Update is called.

```
public  new  VirtuosoCommand  Command  ;
```

Property Value

The VirtuosoCommand executed when Update is called.

VirtuosoRowUpdatedEventHandler Delegate

Represents the method that will handle the RowUpdated event of a VirtuosoDataAdapter.

```
public delegate void VirtuosoRowUpdatedEventHandler( object  sender ,
                                                     VirtuosoRowUpdatedEventArgs  e );
```

Parameters

The declaration of your event handler must have the same parameters as the VirtuosoRowUpdatedEventHandler delegate declaration.

sender
: The source of the event.

e
: A VirtuosoRowUpdatedEventArgs object that contains the event data.

Remarks

The handler is not required perform any action, and your code should avoid generating exceptions or allowing exceptions to propagate to the calling method. Any exceptions that do reach the caller are ignored. When you create a VirtuosoRowUpdatedEventHandler delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, unless you remove the delegate. For more information about event handler delegates, see "Events and Delegates" in the .NET Framework SDK documentation.

VirtuosoRowUpdatingEventArgs Class

Provides data for the RowUpdating event. This class cannot be inherited.

System.Object

System.EventArgs

System.Data.Common.RowUpdatingEventArgs

VirtuosoRowUpdatingEventArgs

```
public
sealed
class
VirtuosoRowUpdatingEventArgs
extends,
```

```
                    RowUpdatingEventArgs
                   {
        }
```

Thread Safety

Any public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Any instance members are not guaranteed to be thread safe.

VirtuosoRowUpdatingEventArgs Constructor

Initializes a new instance of the VirtuosoRowUpdatingEventArgs class.

```
public VirtuosoRowUpdatingEventArgs( DataRow  row ,
                                     IDbCommand  command ,
                                     StatementType  statementType ,
                                     DataTableMapping  tableMapping );
```

Parameters

row
: The DataRow sent through an Update.

command
: The IDbCommand executed when Update is called.

statementType
: One of the StatementType values that specifies the type of query executed.

tableMapping
: The DataTableMapping sent through an Update.

Properties

Command Property

Gets or sets the VirtuosoCommand executed when Update is called.

```
public  new  VirtuosoCommand  Command  ;
```

Property Value

The VirtuosoCommand executed when Update is called.

VirtuosoRowUpdatingEventHandler Delegate

Represents the method that will handle the RowUpdating event of a VirtuosoDataAdapter.

```
public delegate void VirtuosoRowUpdatingEventHandler( object  sender ,
                                                      VirtuosoRowUpdatingEventArgs  e
);
```

Parameters

The declaration of your event handler must have the same parameters as the VirtuosoRowUpdatingEventHandler delegate declaration.

sender
: The source of the event.

e
: A VirtuosoRowUpdatingEventArgs object that contains the event data.

Remarks

The handler is not required perform any action, and your code should avoid generating exceptions or allowing exceptions to propagate to the calling method. Any exceptions that do reach the caller are ignored. When you create a

VirtuosoRowUpdatingEventHandler delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, unless you remove the delegate. For more information about event handler delegates, see "Events and Delegates" in the .NET Framework SDK documentation.

VirtuosoTransaction Class

Represents a transaction to be made at a Virtuoso database. This class cannot be inherited.

System.Object
System.MarshalByRefObject
VirtuosoTransaction

```
                    public
                    sealed
                    class
                    VirtuosoTransaction
                   extends,
                    MarshalByRefObject

          implements,
                    IDbTransaction
                  ,
                    IDisposable
                   {
       }
```

Thread Safety

Any public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Any instance members are not guaranteed to be thread safe.

Remarks

The application creates a VirtuosoTransaction object by calling BeginTransaction on the VirtuosoConnection object. All subsequent operations associated with the transaction (for example, committing or aborting the transaction), are performed on the VirtuosoTransaction object.

Properties

Connection

Gets the VirtuosoConnection object associated with the transaction.

```
public  VirtuosoConnection  Connection  ;
```

Property Value

The VirtuosoConnection object to associate with the transaction.

Remarks

A single application may have multiple database connections, each with zero or more transactions. This property enables you to determine the connection object associated with a particular transaction.

IsolationLevel

Specifies the IsolationLevel for this transaction

```
public  IsolationLevel  IsolationLevel  ;
```

Property Value

The IsolationLevel for this transaction.

Implements

IDbTransaction.IsolationLevel

Remarks

Parallel transactions are not supported. Therefore, the IsolationLevel applies to the entire transaction.

Methods

Commit Method

Commits the database transaction.

```
public void Commit();
```

Dispose Method

Releases the unmanaged resources used by the VirtuosoTransaction and optionally releases the managed resources.

```
public void Dispose();
```

Implements

IDisposable.Dispose

Rollback Method

Rolls back a transaction from a pending state.

```
public void Rollback();
```

Implements

IDbTransaction.Rollback

Remarks

The transaction can be rolled back only from a pending state (after BeginTransaction has been called, but before Commit is called).

7.2. Interactive SQL Utility

ISQL/ISQLO/ISQLU/isql-iodbc/isql-udbc

The Virtuoso ISQL utility allows users to execute queries & scripts against the Virtuoso or other SQL servers (depending on the suffix). It also supports a large number of options and it's own commands.

ISQL parses it's input for CREATE (PROCEDURE|TRIGGER|FUNCTION), '{', '}', ';', double and single quotes and comments to detect where the current command starts and finishes. If it's in a PROCEDURE|TRIGGER|FUNCTION declaration it changes it's command line prompt to '<procedure_name>(<curly_brace_nesting_level>)' to help the user enter the correct command. Otherwise it considers semicolon (;) as the statement end and executes the statement.

ISQL also has a form of script language by itself.

7.2.1. Invoking ISQL

ISQL is invoked from the command line of the operating system. It is a text-mode application with support of readline library (where available).

Specify the --help or the -? option to view the usage:

```
bash$ ./bin/isql --help
OpenLink Interactive SQL (Virtuoso), version 0.9849b.

   Usage :
isql <HOST>[:<PORT>] <UID> <PWD> file1 file2 ...

isql -H <server_IP> [-S <server_port>] [-U <UID>] [-P <PWD>]
     [-E] [-X <pkcs12_file>] [-K]
     [-u <name>=<val>]* [-i <param1> <param2>]

     isql -?
Connection options:

  -?                  - This help message
  -U username         - Specifies the login user ID
  -P password         - Specifies the login password
  -H server_addr      - Specifies the Server address (IP)
  -S server port      - Specifies the TCP port to connect to
  -E                  - Specifies that encryption will be used.
                        If no PK is specified, then anonymous connection is established.
  -C                  - Specifies that password will be sent in cleartext
  -X pkcs12_file      - Specifies that encryption & X509 certificates will
                        be used i.e the client PK <pksc12 file>
  -T server_cert      - Specifies the CA certificate(s) file to be used i.e. the list
                        of trusted issuers: a PEM (base64) file of all the X509 certificates
                        of the Certification Authorities (CA) which the client will use for
                        verifying the server's certificate. This is used on client side to
                        verify the server certificate.
  -b size             - Specifies the size of the command buffer to be used (in KBytes)
  -K                  - Shuts down the virtuoso on connecting to it

Parameter passing options:

  -u name1=val1... - Everything after -u is stored to associative array U,
                        until -i is encountered. If no equal sign then value
                        is NULL
  -i                  - Ignore everything after the -i option, after which
                        comes arbitrary input parameter(s) for isql procedure,
                        which can be referenced with $ARGV[$I] by the
                        ISQL-commands.
  <OPT>=<value>       - Sets the ISQL options

  Note that if none of the above matches then the non-options go as
  <HOST>[:<PORT>] <UID> <PWD> file1 file2 ...
```

If the command line option is not one of the above it is considered a "non-option" parameter. If the non-option parameter contains the equal sign ('=') then it's considered an ISQL option value assignment. For example 'MAXROWS=10' limits the number of rows returned for a resultset to 10 instead of the default 0.

Otherwise ISQL treats the "non-option" parameters (identified by their position) as follows :

Position 1 - <Host> or <port> or <Host:port> (native), DSN (ISQLO/ISQLU/isql-iodbc/isql-udbc)

Position 2 - <user_id>

Position 3 - <password>

Position 4 and later - script file name to load

Let us consider the following command line:

```
isql 1111 dba dba VERBOSE=OFF 'EXEC=status()' test.sql test2.sql -i arg1 arg2
```

The '1111' is the first "non-option", so it represents the connection host:port (This is all digits so the virtuoso CLI prepends this with 'localhost', i.e. this is an shortcut for 'localhost:1111').

The first 'dba' is the second "non-option", so it is used as User ID.

The second 'dba' is the third "non-option", so it is used as Password.

'VERBOSE=OFF' is a "non-option", but it has an equal sign in it, so this becomes an ISQL option SET statement. In this particular case this disables the output of "Done xxx msec" messages after each command.

'EXEC=status()' is a "non-option", but again has again an equal sign in it. EXEC is a special option. Setting it to a value means executing that value as an SQL command. The result of this is that status() gets executed and it's results shown.

'test.sql' is the fourth "non-option" and it specifies a file name to load a script from and execute.

'test2.sql' is the fifth "non-option" and it again specifies a file name to load a script from and execute.

'-i' specifies that the script argument list starts. So every option after the -i is filled in the ISQL $ARGV[] array.

The order of execution is:

1. Connect to 'localhost:1111' using UserID/Password dba/dba.
2. Set the VERBOSE ISQL flag to OFF.
3. Set the $ARGV[0]=arg1 and $ARGV[1]=arg2.
4. Execute status() and display the results.
5. Execute the script in test.sql and display the results.
6. Execute the script in test2.sql and display the results.
7. Disconnect and exit the ISQL

7.2.2. ISQL Commands

```
<SQL_command> &
```

Spawns a new copy of ISQL as a background OS process to execute the command. The new copy will make its own connection and will terminate after the command completes. The isql instance which received the command prompts for the next command without waiting for the spawned isql instance

```
! <command> (;|&)
```

Executes an OS command

```
SET <ISQL_OPTION>( |=)<VALUE>
```

sets the ISQL variable or associative array U value, if it is recognized as such. Otherwise passed to the server.

Example 7.1. Examples

```
SET U{test} 5; sets the associative array U's 'test' to have value of 5
```

```
SHOW [<ISQL_OPTION>]
```

shows the value of an ISQL option, if it is recognized as such. Otherwise passed to the server.

```
NOP
```

no-operation. Useful sometimes in an $IF command.

```
(ECHO|ECHOLN) [STDOUT|STDERR|ERROR_STREAM|BOTH] string1 string2 ...
```

echoes it's arguments into the specifies output stream (defaults to STDOUT if not specified) If ECHOLN is used it appends a new line after the last character printed.

```
WAIT_FOR_CHILDREN
```

Waits for background ISQL subcommands spawned by specifying an & instead of ; at the end of the statement.

```
LOAD <file_name>
```

Executes each statement of the specified file. Statements end in a semicolon or at the closing curly brace for procedure or trigger definitions.

```
HELP [<isql-command>]
```

Prints general or command specific help texts.

```
(EXIT|QUIT) [NOT]
```

Exits ISQL

EXIT NOT is a NO-OP, allowing statements like: EXIT $IF $EQU $ARGV[0] 10 $ARGV[0] NOT; which exits with exit code 10 if $ARGV[0] is ten (presumably keeping a some-kind of failure counter) but otherwise does nothing special, and continues from the next statement.

```
SLEEP [<seconds>]
```

sleeps for the specified number of seconds or until an OS signal arrives. If seconds is not specified it's considered 0 and the behavior is OS dependent (the OS sleep function on Unixes, Sleep() in the Windows API).

```
CONNECT [<conn_str>]
```

Without arguments forces the connection to be made with an ordinary way with SQLConnect and so-far specified connection details. With an argument uses SQLDriverConnect, giving that argument as a connection string (e.g. "DSN=Virtuoso;UID=DBA,PWD=DBA") If we are already connected, then is just NO-OP which is silently ignored.

```
TableQualifiers
```

Returns all defined table qualifiers.

```
TableOwners
```

Returns existing table owners list.

```
TableTypes
```

Returns list of current table types.

```
RECONNECT [<User_ID>]
```

If the ISQL is already connected then disconnects. After that it tries connecting to the same data source, but with using its argument as a user name. If none specified it gets the user name/password from the UID and PWD ISQL variables.

```
FOREACH [LINE|INTEGER|BLOB|TIMESTAMP|DATE|TIME]
        [FOLLOWING|IN (-|-b|<file_name>)|BETWEEN <n1> <n2>]
        <statement_with_params> <statement_with_params> ... [END]
```

This command executes the <statement_with_params> in a loop, after binding the parameters specified in it to values specified in the command.

Parameters are specified as follows :

? - an ordinary parameter bound to the foreach value

?C - the count of successful operations

LINE processes the input line by line

INTEGER converts the input into integer and then processes it.

BLOB reads the entire input and sets it as an blob parameter using SQLPutData

TIMESTAMP is the same as LINE, bit binds to a TIMESTAMP SQL Type

DATE is the same as LINE, but binds to a DATE SQL Type

TIME is the same as LINE, bit binds to a TIME SQL Type

IN specifies a set of items terminated by an END

BETWEEN specifies integer set starting from n1 and ending on n2

FOLLOWING as as IN but skips the input till the END keyword

Example 7.2. Examples:

```
foreach integer between 1 10 insert into num_tb values (?);
```

Inserts 10 rows from 1 to 10 into the num_tb

```
foreach line in my_text_file.txt insert into str_tb (line_no, data) values (?C, ?);
```

Inserts each line of the text file my_text_file.txt into str_tb's data column and puts line number in line_no column

```
foreach blob in my_text_file.txt insert into file_tb values (?);
```

inserts a row into the file_tb with the contents of the my_text_file.txt as a single blob value.

```
SHUTDOWN
```

Passes this through to the server as a command

```
CALL <SQL statement>
```

Tries to bind & print the SQL_RETURN_VALUE and display multiple resultsets (if any) after executing the statement.

```
COLUMNS [<table_mask>][/<column_mask>]
```

Calls SQLColumns ODBC

```
TABLES [<table_mask>]
```

Calls SQLTables ODBC

```
PRIMARYKEYS [<table_mask>]
```

Calls SQLPrimaryKeys ODBC

```
COLUMNPRIVILEGES [<table_mask>]/[<procedure_column_mask>]
```

Calls SQLColumnPrivileges ODBC

```
PROCEDURES [<procedure_mask>]
```

Calls SQLProcedures ODBC

```
PROCEDURECOLUMNS [<procedure_mask>]/[<procedure_column_mask>]
```

Calls SQLProcedureColumns ODBC

```
TABLEPRIVILEGES [<table_mask>]
```

Calls SQLTablePrivileges ODBC

```
GETTYPEINFO
```

Calls SQLGetTypeInfo (SQL_ALL_TYPES) ODBC

```
STATISTICS [<table_mask>] [/U]
```

Calls SQLStatistics ODBC. U means show only unique indexes otherwise all

```
SPECIALCOLUMNS [<table_mask>] [/(B?S?T?N?)]
```

Calls SQLSpecialColumns ODBC.

B means BEST_ROWID, otherwise ROWVER.

S means Session scope, T means transaction scope, otherwise current row.

N means No Nulls, otherwise Nullable.

7.2.3. ISQL Macro Substitution

Before executing commands ISQL expands macros in statement text. The following macros and macro related commands are available:

$LOADEXPR - the current script startup command

$LINENO - the current line number

$YYYYMMDD - the current date in the same format

$HHMMSS - the current time in the same format

```
$IF <cond> <THEN_RESULT> [<ELSE_RESULT>]
```

If the condition evaluates to non-empty and is not the literal 0 the value of the $IF is the <THEN_RESULT> else the <ELSE_RESULT>.

```
$(+|-) <arg1> <arg2>
```

result is the addition or substitution of the arg1 and arg2

```
$LAST[<n>]
```

an array of the last resultset row

```
$ARGV[<n>]
```

an array of the command line arguments

```
$U[<name>]
```

prints the user associative array U value for the name <name>

```
$<ISQL_Variable>
```

returns the value of that variable

```
$LT <v1> <v2>
```

returns 1 if the <v1> is lower that <v2>, empty otherwise

```
$GT <v1> <v2>
```

returns 1 if the <v1> is greater than <v2>, empty otherwise

```
$GTE <v1> <v2>
```

returns 1 if the <v1> is greater or equal than <v2>, empty otherwise

```
$LTE <v1> <v2>
```

returns 1 if the <v1> is lower or equal than <v2>, empty otherwise

```
$EQU <v1> <v2>
```

returns 1 if the <v1> is equal to <v2>, empty otherwise

```
$NEQ <v1> <v2>
```

returns 1 if the <v1> is not equal to <v2>, empty otherwise

Example 7.3. Examples:

```
ECHO $IF $EQU 1 2 "True" "False";
```

prints "False" to the standard output

```
EXIT $IF $NEQ 1 2 NOT;
```

never exits

```
ECHO $IF $EQU $ARGV[1] 1 "One" $IF $EQU $ARGV[1] "Two" "Unknown;
```

prints the "one" on 1 as arg 1, "two" on 2 as arg 1 otherwise "Unknown"

7.2.4. ISQL Variables

These variables are sometimes set as a side effect of executing statements, e.g. rowcnt and sometimes should be explicitly set by the user to control the operation of isql, e.g. maxrows.

The variable names are case insensitive. Like in UNIX shells, the variable name in an expression must be prefixed by a $ to return its value. Variables do not have an explicit data type. Like Unix shell variables they have string values which are sometimes interpreted as numbers and sometimes as booleans. Generally an on/off choice is represented by the values ON/OFF. The ON/OFF values are case insensitive.

Example 7.4. Example:

```
SQL> SET AUTOCOMMIT ON;
SQL> echo $autocommit;
ON
SQL> set U{var} foo;
SQL> echo $u{VAR};
foo
```

Note that the $u{<var>} notation is a general purpose text substitution macro without arguments. The expansion takes place regardless of SQL syntactic context.

Example 7.5. Example:

```
set u{table} SYS_KEYS;
select count (*) from $u{table};
```

Variable Reference

```
RETVAL
```

Function called return value

```
RETVALLEN
```

Function called return value length

```
ROWCNT
```

Number of rows in the last result set. If the statement was an insert, update or delete this is the number of rows affected.

```
COLCNT
```

Number of columns in the last resultset

```
ARGC
```

Number of ISQL command line arguments

```
I
```

Number of script arguments (after -i)

```
LIF
```

Result of the last $IF

```
INPUTLINE
```

Current executed line

```
STATE
```

SQL State (defaults to OK)

```
SQLSTATE
```

Same as state.

```
MESSAGE
```

the SQL error message from the last operation.

```
DRIVER
```

The Driver name

```
LWE
```

Last ECHO output

```
DSN
```

The DSN or host address used/to be used in connecting to the server. Use this with uid and pwd before the connect or reconnect command to change the connected server. This is useful for scripts which access multiple servers.

```
UID
```

The User ID used/to be used in connecting to the server

```
PWD
```

The Password used/to be used in connecting to the server

```
ERRORS
```

The current stream for errors (STDIN, STDOUT, STDERR or a file name)

```
PROMPT
```

Prints the PROMPT or not

```
EMPTY
```

The current empty string value

```
VERSION
```

ISQL version

```
BLOBS
```

When ON binds prints the BLOB resultset values when printing the resultset. Otherwise prints 'BLOB x chars'

```
ECHO
```

When ON prints the commands to the standard output before executing

```
BANNER
```

When ON prints the column names & types banner when printing the resultset

```
TYPES
```

When ON prints the Type information in the resultset's banner

```
VERBOSE
```

When ON Prints the Timing row after the resultset output

```
TIMESTOSTRINGS
```

When ON binds the SQL_DATE, SQL_TIME & SQL_TIMESTAMP columns to strings

```
TRAILING_NEWLINES
```

When ON prints new line after the row's end

```
DEADLOCK_RETRIES
```

How many times to retry the statement if deadlock occurred

```
MACRO_SUBSTITUTION
```

When ON ISQL does understand & process $ macros

```
IGNORE_PARAMS
```

When ON passes the ? through to the server instead of returning an error for unbound ?

```
BIND_RETURN_VALUES
```

When ON binds a buffer to SQL_RETURN_VALUE

```
AUTOCOMMIT
```

When ON ISQL executes statements in autocommit mode. By default isql executes statements in manual commit mode, following each execution with a SQLTransact call to commit. The difference is not visible most of the time.

```
ACCESSMODE
```

Sets the ODBC access mode (RW, RO)

```
TIMEOUT
```

ODBC Query timeout

```
MAXROWS
```

Print only the first so many rows. If 0 - unlimited.

```
CURRENT_QUALIFIER
```

Prints the current ODBC SQLGetInfo client Qualifier

```
INFO_DATABASE_NAME
```

Prints the current ODBC SQLGetInfo Database name

```
INFO_USER_NAME
```

Prints the current ODBC SQLGetInfo user name

```
INFO_GETDATA_EXTENSTIONS
```

Prints the current ODBC SQLGetInfo SQLGetData extensions flag

```
COMMAND_TEXT_ON_ERROR
```

When OFF ISQL does not print the text of the command in the error messages

7.2.5. Using isql as a General Purpose Test Driver

isql can be used to automatically run SQL scripts which execute statements, perform simple tests on the results and generate a report. This can be used to automate tests of stored procedures or to benchmark them.

Also note the use of & at the end of a command to spawn multiple isql instances on the background. This is useful for automatically creating concurrency situations for testing.

Consider the script:

```
drop table tt;
create table tt (id int identity not null primary key, ctr int);
create procedure tt_fill (in n int)
{
  declare ctr int;
  ctr := 0;
  while (ctr < n){
    insert into tt (ctr) values (ctr);
    ctr := ctr + 1;
  }
}

tt_fill (10000) &
tt_fill (10000) &
tt_fill (10000) &
tt_fill (10000) &

wait_for_children;
select count (*), count (distinct ctr)  from tt;

echo both $if $equ $last[1] 40000 "PASSED" "***FAILED";
echo both " Inserted " $last[1] " rows\n";
echo both $if $equ $last[2] 10000 "PASSED" "***FAILED";
echo both " Inserted " $last[2] " distinct ctr values\n";
```

Now suppose the above text were in the file test.sql. The command

```
isql 1111 errors=stdout <test.sql >test.out
```

would print the diagnostics to the standard error and the full trace to test.out. Note the errors=stdout would direct the error message for no table in the initial drop table to the text.out file, so the console would just print:

```
PASSED Inserted 40000 rows
PASSED Inserted 10000 distinct ctr values
```

7.3. Virtuoso Driver for ODBC

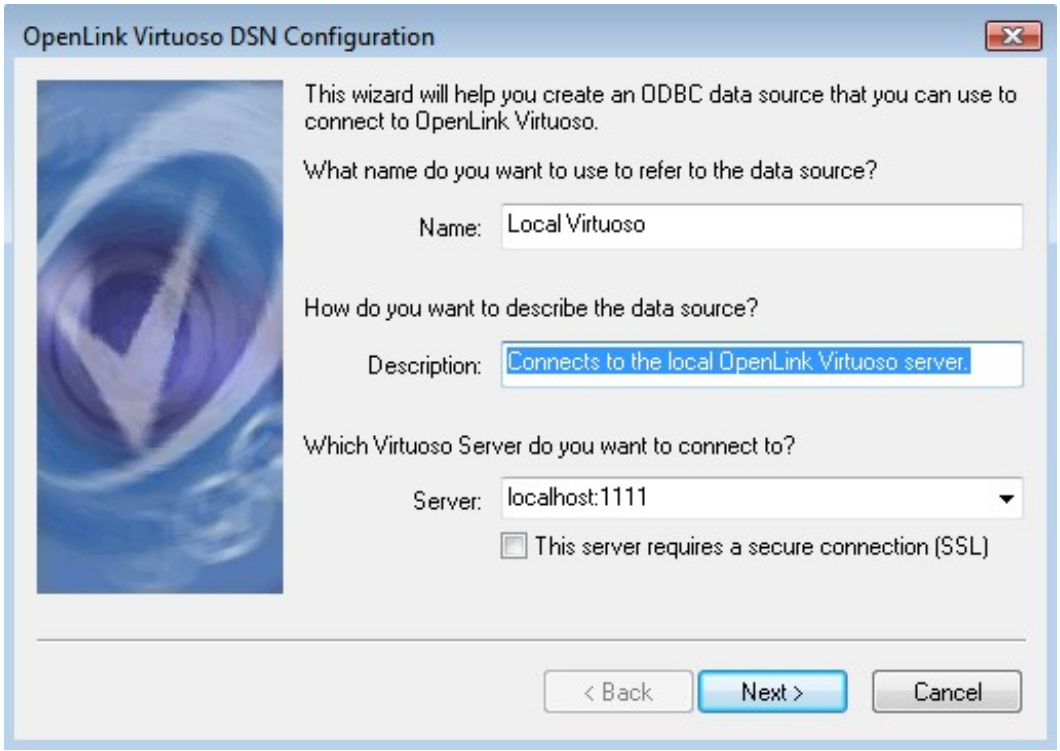
7.3.1. Windows ODBC Driver Configuration

At installation time two ODBC data source names (DSN's) are created with default values by the Virtuoso installer, the first DSN named "Local Virtuoso" provides a link to a local default Virtuoso database server instance, while the other named "Local Virtuoso Demo" provides a link to a local Virtuoso server for the Virtuoso demonstration database.

The process of creating additional ODBC DSN's for you Virtuoso drivers for ODBC is explained in the steps that follow:

1. Go to the Windows *Control panel* .
2. Double click on the *ODBC Administrator* applet. On Windows 2000 / XP the ODBC Administrator applet may be called *Data Source (ODBC)* and may be found under Administrative tools icon of the Control Panel.
3. Once the ODBC Administrator has been loaded choose by selecting the appropriate panel whether you want a new User or System Data Source. User Data Sources will only be available to the user that created them. System Data Sources will be available to all users and applications on the system.
4. Click on the *Add* Data Source Name button
5. Select the Driver named *OpenLink Virtuoso Driver*
6. Enter values into the fields presented by the Virtuoso Driver's DSN configuration dialog:

Figure 7.30. Virtuoso ODBC Driver Setup Dialogue for Windows



Name: provide a name that will act as a logical reference to the Virtuoso database server that you will be connecting to. Subsequent references to this database will be made to this value when ODBC compliant applications interact with your Virtuoso driver.

Description: allows you to provide a short description about the nature of the connection. This is optional.

Server: enter the hostname or IP address of the machine hosting your Virtuoso server and enter the port number that Virtuoso is listening at. This is configured in the Virtuoso ini file on the server.

7. Press the *Next* button to configure more details about the connection

Figure 7.31. Virtuoso ODBC Driver Setup Dialogue for Windows

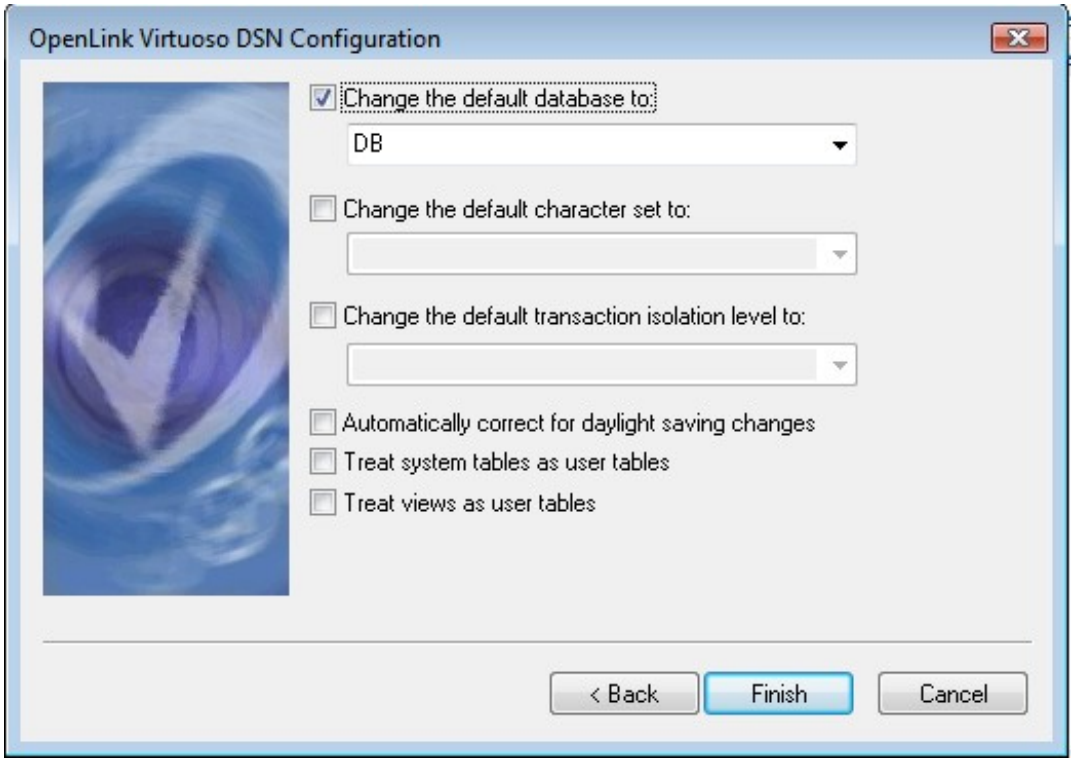
Connect to the Virtuoso Servet to obtain default settings for the additional configuration options: allows you to specify the default username and password for the connection.

Press the *Next* button

Database: allows you to select the default database for the connection. You will need to check the box above and supply a password to refresh this list. Objects created or selected without an explicit catalogue/database qualifier will automatically be sought from this select database.

Charset: lets you choose the default character set for the connection.

Figure 7.32. Virtuoso ODBC Driver Setup Dialogue for Windows



8. When the configuration is complete, and indeed at any time you are satisfied with the configuration press the *Finish* button to save the DSN.

7.3.2. Using X509 Certificates With ODBC Connection

Virtuoso can be configured to authenticate ODBC logins with a single X.509 certificate. The certificate must be registered server side for this purpose and may contain an additional custom attribute for the users SQL account name. In this way all login information is held in the possibly password protected certificate. The user specifies the certificate path in the place of the user name and the certificate encryption password as the password. This works with the login dialog of the ODBC driver or within a SQLDriverConnect login string.

It is also possible to set up the ODBC client to check for the server's X.509 certificate by specifying a client side CA (Certificate Authority) list.

The X509 certificate can be used for authentication with the Virtuoso server via SSL. In this case the ODBC client will use an X.509 certificate which contains a SQL username as an extension. This extension is added when the certificate is issued. Since an object ID (OID) of the username attribute can be used any valid object identifier that does not conflict with existing OIDs (for example 2.16.840.1.NNNNNN). It will be best to have one's own registered OID for that purpose.

WebID Protocol ODBC Login

Virtuoso supports WebID Protocol ODBC Login. Thus, the INI parameter X509ClientVerify can accept the following values:

- 0 - do not require
- 1 - ask for trusted certificates
- 2 - optionally ask, if certificate is given will be verified
- 3 - accept optionally any certificate even self-signed

If certificate contains WebID, the odbc login will use it. The WebID can be setup to particular account via Conductor's users interface.

A typical [Parameters] INI section should contain:

```
SSLServerPort         = 1113
SSLCertificate          = keys/server.crt
SSLPrivateKey         =  keys/server.key
X509ClientVerify      = 3
```

The client could connect in following way:

```
isql 1113 "" -X client.p12 -T server.crt
```

Note: The client certificate client.p12 contains WebID which is registered to some sql user account via Conductor.

ODBC Client Certificate Generation Using "openssl" Tool

To generate X509 certificates one can use the "openssl" tool obtained freely from www.openssl.org . The `openssl.cnf` configuration file must be edited to include the new extension for sqlUserName, but first we have to find the hexadecimal representation of the SQL Username. Then you can proceed the use the openssl to create and confirm the certificate. Follow the steps below:

1. *Find the HEX representation of the SQL Username*

   You can either work this out from ASCII codes or use a hexdump utility found on most Unix platforms. The following command would be sufficient, replacing <user_name> with the actual username:

   ```
   $ echo -n "<user_name>" | hexdump -e '":" 1/1 "%02X"'
   ```

   For example, the username `'dba'` would be `:64:62:61` .
2. *Edit the OpenSSL config file: openssl.cnf*

   Open the file in a text editor of your choosing and add the following:

   ```
   [ new_oids ]
   sqlUserName=2.16.840.1.NNNNNN.1

   ...
   [ usr_cert ]
   ...
   sqlUserName = DER:NN:NN:NN:NN:NN.....
   ```

   replacing `:NN:NN.....` with the hexadecimal representation of the username we discovered in the previous step.
3. *Make New Certificate Request*

   Make new certificate request using the command:

   ```
   $ openssl req -new
   ```

   The tool will ask for certain details. Once completed it will generate a private key also.
4. *Generate Certificate*

   Assuming the role of CA, generate a certificate using the request from the previous step, using the following command:

   ```
   $ openssl ca -in <req_file>
   ```

   The tool will print the details of request and will ask you to sign and commit the certificate into the CA repository.

5. *Verify New Certificate*

   Check that the certificate contains the right SQL account name; use the following command to obtain a text dump of the certificate:

   ```
   openssl x509 -in <certfile> -text -noout
   ```

   Now we can scan the contents of the output for the extension entries we added earlier:

   ```
   >>>>>
   X509v3 extensions:
   ....
   2.16.840.1.NNNNNNN.1:
   <SqlAccountName>
   ^^^^^^ this should match the new OID ^^^^^
   >>>>>
   ```

Virtuoso Server Configuration

The Virtuoso server can work in two modes based on an SSL connection:

basic - this is when only connection is secured, no certificate verification. The client is not required to have its own trusted certificate.

trusted - additional mechanisms are enabled to check client's certificate. in this case the client is required to have a trusted certificate.

To allow Virtuoso to check the client's certificates, the virtuoso.ini file must contain the following entries in [Parameters] section:

```
; Basic session encryption only parameters
SSLServerPort           = 1113
SSLCertificate          = ./srv.cert.pem ; server's certificate
SSLPrivateKey           = ./srv.key.pem  ; server's private key, must match the certificate
; Trusted operation parameters
X509ClientVerify        = 1
X509ClientVerifyCAFile  = ./ca.pem         ; CA list; file containing certificates of acceptable CA
X509ClientVerifyDepth   = 1                ; dependent of type of certificate checking can be &gt;1
X509ExtensionOID          = 2.16.840.1.NNNNNNN.1 ; the OID value, same as that used to make client's cert
```

All certificate/key files need to be in PEM format.

The server needs an "SSLCertificate" and "SSLPrivateKey" to begin listening on the "SSLServerPort". These are essential for the secure operations. Furthermore the certificate must match the private key; non-matching certificate and private keys will prevent server startup and an error will be logged. The private key is required to guarantee that the certificate's claim is true. The server certificate is used by the client to identify the server. The client can retrieve and verify this key and choose whether the server an be trusted depending on circumstances.

Note:

basic operation (SSL/TLS) encryption only cannot be used to identify a client with certificate.

The "X509ClientVerifyCAFile" is a file containing a CA's (Certificate Authority) certificates that the server can use to verify client certificates. The client certificate verification in general depends on the "X509ClientVerify" flag, which enables or disable this feature. The 'X509ClientVerifyDepth" parameter is used to verify to what degree server will trust the client's certificate. The lower the value, the higher the restriction levels, with a minimum value of 1. This means that the server will look in CA's certificates to find who has issued the client certificate. If there is no matching CA entry the connection will be rejected; If there is a matching entry then verify the issuer chain; If issuer chain is greater than "X509ClientVerifyDepth", the connection will be rejected.

All file paths above must be either absolute or relative to the server working directory.

Note:

If primary key file is encrypted then the server must be started in foreground mode so that a password can be supplied in order to open the file.

Virtuoso Client Configuration

The following connection options control the client's behavior regarding SSL:

Encrypt - specifies type of secure connection to be used.
ServerCert - (optional) to specify which certificate(s) are to be used to verify server certificates.
The *Encrypt* option can be set to '1' to specify a basic secure connection; no server identity verification will be performed. Note that this is only possible when the server is also set to make basic SSL connections.

To ensure server's identity the *Encrypt* option must be set to full or relative (to application working directory) path to the file containing client's certificate and private key. This file can be encoded in PKCS#12 or PEM format. The certificate and private key contained may be generated using the steps outlined above.

When the Encrypt option is set to point to a certificate file, the *ServerCert* option must be set to the full or relative path to file containing list of CA certificates in PEM format. The content of this file will be used to verify server's certificate.

When using these options the UID connection-option must be set to an empty string to enable certificate authorization. The PWD option will be used to open the private key.

Here is an example of an ODBC connect-string:

```
...
connectString =
"HOST=localhost:1113;UID=;PWD=keysecret;ENCRYPT=c:\certs\mycertwithkey.p12;SERVERCERT=c:\certs\ca.pem"
...
```

If client's certificate does not contains user name for SQL login then the server will try matching the certificate fingerprint against registered certificates. If any SQL account has such a certificate registered it will be used for login. Otherwise login will be rejected.

Registering Keys with Virtuoso

To enable a client certificate to be used for authorization, the DBA must register it's MD5 fingerprint (checksum) in the database. Registered certificates can be removed from system. Two functions exist for the purposes: `USER_CERT_REGISTER()` , and `USER_CERT_UNREGISTER()` . Both functions rely on the MD5 checksum of the certificates being registered or un-registered, therefor you have the option of supplying these functions with the certificate file or the MD5 checksum directly.

The functions are:

```
USER_CERT_REGISTER (user_name , certificate , password , type );

USER_CERT_UNREGISTER (user_name , certificate , password , type );
```

The registered certificate's fingerprints are kept in the DB.DBA.SYS_USERS table as vectors of strings under the 'LOGIN_CERTIFICATES' user option value (U_OPTS column). The data stored there can be retrieved using a third function:

```
USER_GET_OPTION()
```

For example, one might invoke:

```
USER_CERT_REGISTER ('DBA', 'file:/dba.pem', '', 'PEM');
```

Note that 'file:' URL is needed to designate certificate is in a file on file system.

The above action can be performed also without certificate supplied if the MD5 fingerprint is known:

```
USER_CERT_REGISTER ('DBA', 'D9:6D:47:D7:67:47:D7:3C:2C:E0:89:91:F3:BC:E7:59');
```

and

```
USER_CERT_UNREGISTER ('DBA', 'D9:6D:47:D7:67:47:D7:3C:2C:E0:89:91:F3:BC:E7:59');
```

7.3.3. Manually configuring a Virtuoso ODBC DSN on Unix

If you have iODBC installed, you can configure Virtuoso data sources by adding the following entry into the relevant .odbc.ini file. Usually it is the value of the ODBCINI environment variable or $HOME/.odbc.ini:

Sample DSN:

```
[LocalVirt]
Driver=/usr/local/lib/virtodbc_32.so
# absolute path to the shared object
Address=localhost:1111
# host and port of the Virtuoso server
```

If the application that will load the ODBC driver is multithreaded, use the virtodbc32_r.so driver instead.

Linking Client Applications

The isql and other utilities are linked directly with the Virtuoso client code. See the Makefiles for the libraries used. These are identical in function with the ODBC driver but accept a host:port in the place of a data-source name to be resolved from the odbc ini file.

Generally applications should pass via ODBC. Directly linking with the ODBC driver shared object is also possible.

JDBC

If you specified:

```
--with-jdbc3=<path of JDK>
```

to the configure in the installation root directory, running make will produce the files libsrc/JDBCDriverType4/virtjdbc3.jar and virtjdbc3ssl.jar. These can be placed on the Java class path. See Virtuoso JDBC Documentation for URL formats etc.

7.3.4. ODBC Compliance

The Virtuoso Driver for ODBC conforms to both the ODBC 1.x,2.x,and 3.x versions of the ODBC specification, it implements Core, Level 1, Level 2, and Extensions functionality. It also has a native support for the wide versions of the ODBC API (e.g. SQLColumnsW) in Windows. This driver enables you to communicate with local or remote Virtuoso servers across any combination of platforms supported by Virtuoso.

ODBC API implementation details

SQLAllocHandle

Virtuoso ODBC driver does not allow allocation and usage of explicitly allocated descriptor handles. That is why the SQLAllocHandle (SQL_HANDLE_DESC) will return an error.

SQLBulkOperations

Only the SQL_ADD operation is supported.

SQLColAttributes

The virtuoso ODBC driver does not return information for the following attributes:

- **SQL_COLUMN_TABLE_NAME.** Returns an empty string instead
- **SQL_COLUMN_OWNER_NAME.** Returns an empty string instead
- **SQL_COLUMN_QUALIFIER_NAME.** Returns an empty string instead
- **SQL_COLUMN_CASE_SENSITIVE.** Returns 1 instead
- **SQL_COLUMN_AUTO_INCREMENT.** Returns 0 instead
- **SQL_COLUMN_MONEY.** Returns 0 instead
- **SQL_COLUMN_UNSIGNED.** Returns 0 instead

SQLDriverConnect

The Virtuoso ODBC driver recognizes the following SQLDriverConnect connection string keywords:

- **DSN.** The data source name.
- **HOST.** The virtuoso server host specification (in the form : [<hostname>[:]][<portnumber>]
- **UID.** The virtuoso user ID used to connect
- **PWD.** The login password used to connect
- **DATABASE.** The qualifier to use when connected (overrides the user's default qualifier)
- **CHARSET.** The name of the character set to use for wide/narrow conversions
- **DAYLIGHT.** Boolean parameter (1/0). When ON (1) it takes into account the client OS daylight savings settings.
- **ENCRYPT.** String parameter (file name or "1"). Specifies how the ODBC connection will be encrypted. see the Using SSL For Secure ODBC connections for details.
- **PWDCLEAR.** Integer parameter (default to 0). Specifies how the password will be sent over the wire. 0 - send digest of the password, 1 - send password in cleartext, 2 - send password garbaled.
- **SERVERCERT.** String parameter (file name). Specifies the path for the CA list used to verify the server's certificate (in PEM format). see the Using SSL For Secure ODBC connections for details.
- **FORCE_DBMS_NAME.** String parameter (default "OpenLink Virtuoso"). When set it alters the result of SQLGetInfo (SQL_DBMS_NAME).
- **NoSystemTables.** Boolean parameter (1/0) (default 0). When set it alters the result of SQLTables () so that it won't find or return tables of type SYSTEM TABLE.
- **IsolationLevel.** String parameter ("Read Uncommitted"/"Read Committed"/"Repeatable Read"/"Serializable"). When set it specifies the initial transaction isolation mode for that connection.

SQLGetEnvAttr

The SQL_ATTR_OUTPUT_NTS does not have any effect on the Virtuoso driver. It allows the value to be set and retrieved but with no further effect.

SQLNativeSql

All ODBC syntax is parsed server side. The native SQL syntax for Virtuoso is the ODBC syntax.

Not Supported ODBC API functions

Virtuoso ODBC driver does not support the following ODBC API functions:

- SQLCopyDesc

7.3.5. Virtuoso Scrollable Cursor Engine

Virtuoso implements server side scrollable cursors. ODBC 2.0, ODBC 3.5 and JDBC 2.0 API's are supported.

Cursor types include:

- **Forward only.** This is the default, non-scrollable cursor.
- **Static.** The cursor's evaluation is computed when the cursor statement is first executed. Positioned operations are possible but their effect will not show nor will changes be detected.
- **Keyset.** When the cursor is opened a keyset is built. Rows within the keyset can be fetched and modified by positioned operations. Changes will show when refreshing data and changes by other transactions will be detected for update and delete. Inserts by the same or different transaction will not appear. A keyset cursor may have a finite keyset size. When scrolling outside of the given keyset the keyset will shift to cover the new rows.
- **Dynamic.** A dynamic cursor will reflect all changes by the same and different transactions. The dynamic cursor's evaluation is constructed as needed, hence it generally has less overhead than other types of cursors.
- **Mixed.** A mixed cursor is a combination of a keyset-driven cursor and a dynamic cursor. It is used when the result set is too large to reasonably generate a keyset for the entire result set. Mixed cursors use a keyset smaller than the entire result set but larger than the rowset.

  While the application is scrolling within the keyset, the behavior is keyset-driven. When the application scrolls beyond the keyset, the behavior becomes dynamic to fetch the requested rows and generate a new keyset. The behavior then reverts back to keyset-driven within that keyset, as before.

Forward Only Cursors

A forward only cursor is substantially more efficient than a scrollable cursor. It however does not allow positioned operations (the WHERE CURRENT OF SQL phrase), or SQLSetPos. The SQLExtendedFetch function is supported but only the SQL_FETCH_NEXT fetch type is then allowed.

Cursor Operations

Virtuoso supports all ODBC scrollable cursor operations. These include

SQLExtendedFetch / SQLScrollFetch fetch type.

- SQL_FETCH_FIRST
- SQL_FETCH_LAST
- SQL_FETCH_NEXT
- SQL_FETCH_PRIOR
- SQL_FETCH_RELATIVE
- SQL_FETCH_ABSOLUTE
- SQL_FETCH_BOOKMARK

SQLSetPos operations.

- SQL_POSITION
- SQL_REFRESH
- SQL_ADD
- SQL_UPDATE
- SQL_DELETE

Positioned SQL statements, i.e. the WHERE CURRENT OF clause, is supported for scrollable cursors.

Cursor Options

The cursor options

- SQL_CURSOR_TYPE
- SQL_CONCURRENCY
- SQL_KEYSET_SIZE
- Cursor name (SQLSetCursorName)

have to be set before a SQLPrepare or SQLExecDirect.

The SQL_ROWSET_SIZE can be varied while a cursor is open.

Cursors and Transactions

All forward only or scrollable cursors survive committing or rolling back transactions. A cursor maintains its position over a transaction's end. Hence the next fetch operation will resume from the correct place. If a dynamic cursor's current row / rowset is deleted, the cursor will continue from the next greater / lesser row in the order of the cursor's ordering columns. This also applies to mixed mode (keyset with finite keyset size) cursors scrolling outside of the keyset bounds. A forward only cursor will retain its logical position across commit/rollback.

The SQL_CONCURRENCY values of SQL_CONCUR_READ_ONLY and SQL_CONCUR_LOCK cause Virtuoso to lock the rows in the keyset / rowset in shared or exclusive mode, respectively.

Optimistic Concurrency Control

A scrollable cursor may have a SQL_CONCURRENCY setting of SQL_CONCUR_VALUES. This enables optimistic concurrency control. This is a mechanism which will reflect an update or delete of a row if the row has been modified by a third party after the time the application last read the row.

A 'updated meanwhile' condition detected in this manner will prevent the operation and return a SQL state of 01001 with SQL_SUCCESS_WITH_INFO from SQLSetPos.

The updates and deletes made through SQLSetPos are still subject to being committed or rolled back by normal transaction control. The 01001 state does not prevent the current transaction from committing.

The 01001 state is only detected if the update or delete is made by SQLSetPos and the row at hand has been changed by any statement of any transaction. If the update of updated operation is carried out by any other operation than SQLSetPos of the statement that last read the value the condition cannot be detected.

Note that the time between the last read and the SQLSetPos update can be long and can span multiple transactions.

Cursor Information

Table 7.13. Cursor Support

	Static	Keyset	Dynamic
SQLRowCount	x	x	
SQL_BOOKMARK	x	x	x
SQL_ROW_NUMBER	x	x	
reflect update		x	x
reflect delete			x
reflect inx			x
Notice update	x	x	x
Notice delete	x	x	x

SQLRowCount is set after the initial SQLExecute. This is the actual row count or the keyset size for a mixed mode cursor. A dynamic cursor does not know this since it builds the evaluation as needed.

All cursors support bookmarks.

The SQL_ROW_NUMBER is the ordinal number of the current row in the cursor's evaluation. A dynamic cursor cannot know this, as the scrolling may start from end and rows may appear on either side of the current row while the cursor is open.

'reflect' means that the new values or added / deleted rows appear when re-scrolling over the rows. A deletion is reflected by omitting the row.

'notice' means that the row is flagged by SQL_UPDATE, SQL_DELETED or SQL_ADDED in the SQLExtendedFetch status array when the cursor re-scrolls over the rows in question.

Cursors and Virtual Database

The cursor subsystem modifies the cursor's defining select statement to make various backward and forward read statements, update and delete statements etc. These transformations can be seen through the explain function.

Since cursors are implemented by generating SQL statements these work transparently against remote database, independently of their possible native cursor support.

Cursor Special Cases

SQL SELECT statements fall in two categories: Simple, which consist of one or more tables, an arbitrary WHERE clause and an optional ORDER BY, and Complex, which includes all other SELECT's, e.g. ones with GROUP BY, derived tables, full outer joins, UNION's etc.

A simple statement can easily be modified to read backwards or forwards from a given point. Also, each result row of a simple statement has a physical counterpart, or many physical counterparts for a join. Having a physical counterpart makes it possible to define the meaning of a positioned update or delete. A GROUP BY is a counter-example.

All complex SQL statements occurring as cursors are processed as static cursors regardless of the declared cursor type.

Cursors and Performance

If a cursor's ordering corresponds to an ordering in an index, dynamic cursors will generally perform best. This is understandable since the engine can quickly locate the current row based on an index and then just read backward or forward on that index. On the other hand, if the result set is very sparsely spread over the table or if there is no ordering index dynamic cursors will have a substantial penalty.

If used as a forward only cursor in a single transaction, a dynamic cursor is only some 30% slower than a forward only cursor.

A static or keyset cursor has a relatively long execute time since the initial execute will make the entire keyset. The initial execute of a dynamic cursor is instantaneous since the fetch operations will do the work as needed.

With most off the shelf applications, e.g. Microsoft ADO, static and keyset cursors are preferable because the applications may rely on row counts and row numbers which are not known for dynamic cursors.

Positioned operations are not affected by cursor type.

7.3.6. Effect of Connection & Statement Options

Connection Options

SQL_AUTOCOMMIT

The autocommit option is set at the connection level and affects every statement executed after it being set. Setting the option does not communicate itself with the server and is therefore fast.

Autocommit is on by default. Autocommitting SELECT statements are executed with read committed isolation. This is appropriate since any update based on the autocommitting read would be in a different transaction and hence would block to wait for access to the selected row. Also re-evaluating a select in autocommit mode would read the data in a different transaction. Hence there is no point in repeatable read isolation for autocommitting cursors. Cursors inside procedures have the normal repeatable read isolation regardless of whether the procedure was called in autocommit mode.

When an autocommitting statement starts it is executed in the transaction That is the connection's current transaction when it is received. Before starting the autocommitting statement sets the connection's current transaction to a new one. In this manner a client can issue multiple asynchronous autocommitting statements at the same time and the statements will execute concurrently, each in its own transaction.

If array parameters are used in a statement on an autocommitting connection each parameter row will be processed in its own transaction in sequential order. Multiple Asynchronous statements must be used to execute one client's statements in parallel.

To commit or roll back a transaction in manual-commit mode, an application should call SQLEndTran. Applications should not attempt to commit or roll back transactions by executing COMMIT or ROLLBACK statements with SQLExecute or SQLExecDirect. The effects of doing this are undefined.

SQL_TXN_ISOLATION

This option allows all the values defined in ODBC,

The isolation of an operation is the property of the operation rather than of the transaction within which it takes place. Once an operation has started, e.g. a cursor has been opened, its isolation cannot be changed.

The value of this option will affect any subsequently executed statement. Note that setting this option to different values during a transaction will work, thus a transaction can have cursors with different isolations although that is presumably not the intention of the ODBC specification.

See the transaction model for a definition of the different isolation levels.

A statement in autocommit mode executes in the same transaction as the previous statement. The transaction is committed when the statement successfully completes. The next statement in the connection will execute in the fresh transaction that was

associated to the connection at the start of the previous autocommitting statement.

As a consequence of this multiple concurrent autocommitting transactions may execute on the same connection at the same time.

Statements executed with array parameters execute each set of parameters as a separate transaction if the connection is in autocommit mode.

SQL_ACCESS_MODE

This has the effect of reversing any SQL_CONCUR_LOCK concurrency to SQL_CONCUR_READ_ONLY. The statement option's value is not changed though.

SQL_CURRENT_QUALIFIER

This sets or gets the current qualifier. The initial value is obtained from the server at connect time. The values reflect the effects of any USE statements.

SQL_NO_CHAR_C_ESCAPE (=5002)

This has the same effect as the NO_CHAR_C_ESCAPE option in the SET statement. It takes boolean int values (0/non-0)

SQL_CHARSET (=5003)

This has the same effect as the CHARSET option in SET statement. It takes string values (the name of the character set to use).

SQL_ENCRYPT_CONNECTION (=5004)

Usable only with the Virtuoso CLI (because the ODBC/iODBC driver manager does not pass-through the custom options to the driver on SQLConnect/SQLDriverConnect). When set to the string "1" means use SSL but no X509 certificates. Setting it to a NULL (default) means no encryption of the ODBC connection. Any other string is treated as a file name of one PKCS12 package to get the data from for establishing an encrypted SSL connection using X509 certificates (see the -E/-X ISQL options).

SQL_SHUTDOWN_ON_CONNECT (=5005)

Usable only with the Virtuoso CLI (because the ODBC/iODBC driver manager does not pass through the custom options to the driver on SQLConnect/SQLDriverConnect). Shuts down the server on connection after authenticating the DBA user (see the -K ISQL option).

SQLSetStmtOption Statement Options

Virtuoso supports all ODBC 2.x and ODBC 3.x statement options. The following options are mentioned below due to implementation specific details.

SQL_CONCURRENCY

The supported values are SQL_CONCUR_READ_ONLY, SQL_CONCUR_LOCK and SQL_CONCUR_VALUES, the last option is only available for scrollable cursors. A select statement with SQL_CONCUR_READ_ONLY will make shared locks when locking for repeatable read or serializable transactions. The SQL_CONCUR_LOCK for a select statement will cause it to make exclusive locks, as if it had the FOR UPDATE clause specified.

See the section on scrollable cursors for the effect of SQL_CONCUR_VALUES. For all statements except scrollable cursors this value reverts to SQL_CONCUR_READ_ONLY.

Any searched update or delete statements will make exclusive locks in all cases.

SQL_MAX_ROWS

This option only affects static cursors.

SQL_TXN_TIMEOUT

This is an extension that allows setting a time limit on the current transaction. The time limit starts at the execution of the statement specifying this option. The transaction will terminate the indicated number of seconds after the execute whether the statement has completed or not.

SQL_PREFETCH_SIZE

This is an extension option that controls how many rows of a forward only cursor are prefetched by the execute and fetch calls. A high value is advantageous for long consecutive reads since it cuts down on the number of client server messages exchanged. On the other hand a large value can result in unnecessary data transfer and locking if only the first few rows of a cursor are fetched. A value of -1 will cause the entire rowset to be fetched at the execute, so that no matter the select size, only one message is exchanged. The default value is 20. This can also be set in the virtuoso.ini file.

A select with array parameters will always work as with a SQL_PREFETCH_SIZE of -1, meaning that all the result sets are computed and sent to the client by the SQLExecute call that opens the cursor.

SQL_CURSOR_TYPE

SQL_KEYSET_SIZE
SQL_CONCURRENCY
The cursor type options should be set before preparing a statement. Other options may be set at any time. The rowset and prefetch options should not be modified after executing a SELECT.

SQL_GETLASTSERIAL

This is an extension that returns the last assigned identity column value. The return buffer pvParam is of type SQLINTEGER. For this to be meaningful, the statement in question must be an INSERT into a table that has an identity column. Note that if there are more than one identity columns or if triggers make inserts with identity columns the value will be undefined.

7.3.7. Efficient Use of API

DO NOT USE SQLExecDirect. If a statement is executed more than once it is much faster to prepare the statement with SQLPrepare and then use SQLExecute repeatedly. The system only compiles the statement once, therefore there is no parsing overhead for repeatedly compiling the same text.

Array parameters for insert, update or single row selects are about twice as fast as the same operations on a single parameter set.

The OR connective in SQL can result in slow queries with extensive locking.

Autocommit should be used when possible, i.e. make the last statement of a transaction autocommitting to avoid having to commit the transaction as a separate operation.

7.3.8. Executing SQL from Python script

In order to execute SQL from Python script, you need to add the following lines to the /etc/odbc.ini file:

```
[Local Virtuoso]
Driver  = /PREFIX/lib/virtodbc_r.so
Address = localhost:1111
```

where PREFIX is replaced by the full path where Virtuoso is installed and also assuming that is used let's say port 1111 in virtuoso.ini (which is set by default).

Then you should be able to connect with:

```
c = pyodbc.connect('DSN=Local Virtuoso;UID=dba;PWD=dba')
```

7.3.9. Extensions

Virtuoso ODBC RDF Extensions for SPASQL

The Virtuoso ODBC Driver adds a number of defines to the ODBC API to allow an ODBC compliant application to query meta information on SPASQL queries.

If the application uses the iODBC Driver Manager V3.52.7 or higher, it can simply include the iodbcext.h header file, which contains information on extensions of many ODBC drivers like DB2, SQL Server and Virtuoso.

If however the application is compiled against another Driver Manager, like the Microsoft Driver Manager on Windows, the following construction should to be used:

```
#ifdef WIN32
# include <windows.h>
#endif

#include <sql.h>
#include <sqlext.h>

#if defined (HAVE_IODBC)
#include <iodbcext.h>
#endif

/*
 *  Include Virtuoso ODBC extensions for SPASQL result set
 */
#if !defined (SQL_DESC_COL_DV_TYPE)

/*
 *  ODBC extensions for SQLGetDescField
 */
# define SQL_DESC_COL_DV_TYPE               1057L
# define SQL_DESC_COL_DT_DT_TYPE            1058L
# define SQL_DESC_COL_LITERAL_ATTR          1059L
# define SQL_DESC_COL_BOX_FLAGS             1060L
# define SQL_DESC_COL_LITERAL_LANG          1061L
# define SQL_DESC_COL_LITERAL_TYPE          1062L

/*
 *  Virtuoso - ODBC SQL_DESC_COL_DV_TYPE
 */
# define VIRTUOSO_DV_DATE                   129
# define VIRTUOSO_DV_DATETIME               211
# define VIRTUOSO_DV_DOUBLE_FLOAT           191
# define VIRTUOSO_DV_IRI_ID                 243
# define VIRTUOSO_DV_LONG_INT               189
# define VIRTUOSO_DV_NUMERIC                219
# define VIRTUOSO_DV_RDF                    246
# define VIRTUOSO_DV_SINGLE_FLOAT           190
# define VIRTUOSO_DV_STRING                 182
# define VIRTUOSO_DV_TIME                   210
# define VIRTUOSO_DV_TIMESTAMP              128
# define VIRTUOSO_DV_TIMESTAMP_OBJ          208

/*
 *  Virtuoso - ODBC SQL_DESC_COL_DT_DT_TYPE
 */
# define VIRTUOSO_DT_TYPE_DATETIME          1
# define VIRTUOSO_DT_TYPE_DATE              2
# define VIRTUOSO_DT_TYPE_TIME              3

/*
 *  Virtuoso - ODBC SQL_DESC_COL_BOX_FLAGS
 */
#define VIRTUOSO_BF_IRI                     0x1
#define VIRTUOSO_BF_UTF8                    0x2
#define VIRTUOSO_BF_DEFAULT_ENC             0x4

#endif
```

API

SQLGetDescField

Before the application can retrieve the column meta data using `SQLGetDescField`, it first needs to retrieve the correct descriptor handle attached to the statement handle:

```
SQLHDESC hdesc = NULL;
SQLRETURN rc;

rc = SQLGetStmtAttr (hstmt, SQL_ATTR_IMP_ROW_DESC, &hdesc, SQL_IS_POINTER, NULL);
if (!SQL_SUCCEEDED(rc))
  {
    /* Handle error */
  }
```

SQLGetDescField - SQL_DESC_COL_DV_TYPE. Retrieves the datatype of a field.

```
SQLINTEGER dvtype;
SQLRETURN rc;

rc = SQLGetDescField (hdesc, colNum, SQL_DESC_COL_DV_TYPE, &dvtype, SQL_IS_INTEGER, NULL);

#define VIRTUOSO_DV_DATE                    129
#define VIRTUOSO_DV_DATETIME                211
#define VIRTUOSO_DV_DOUBLE_FLOAT            191
#define VIRTUOSO_DV_IRI_ID                  243
#define VIRTUOSO_DV_LONG_INT                189
#define VIRTUOSO_DV_NUMERIC                 219
#define VIRTUOSO_DV_RDF                     246
#define VIRTUOSO_DV_SINGLE_FLOAT            190
#define VIRTUOSO_DV_STRING                  182
#define VIRTUOSO_DV_TIME                    210
#define VIRTUOSO_DV_TIMESTAMP               128
#define VIRTUOSO_DV_TIMESTAMP_OBJ           208
```

If this call returns `SQL_SUCCESS` or `SQL_SUCCESS_WITH_INFO`, the dvtype variable will contain the underlying Virtuoso datatype as indicated in the following table:

SQLGetDescField - SQL_DESC_COL_DT_DT_TYPE. Retrieves the date subtype of a field.

```
SQLINTEGER dv_dt_type;
SQLRETURN rc;

rc = SQLGetDescField (hdesc, colNum, SQL_DESC_COL_DT_DT_TYPE, &dv_dt_type, SQL_IS_INTEGER, NULL);

#define VIRTUOSO_DT_TYPE_DATETIME           1
#define VIRTUOSO_DT_TYPE_DATE               2
#define VIRTUOSO_DT_TYPE_TIME               3
```

If this call returns `SQL_SUCCESS` or `SQL_SUCCESS_WITH_INFO`, the dttype variable will contain the underlying Virtuoso date subtype as indicated in the following table:

SQLGetDescField - SQL_DESC_COL_LITERAL_ATTR. Retrieves the literal attributes associated with the field.

```
SQLINTEGER littype;
SQLINTEGER lang, type;
SQLRETURN rc;

rc = SQLGetDescField (hdesc, colNum, SQL_DESC_COL_DT_DT_TYPE, &littype, SQL_IS_INTEGER, NULL);

lang = (littype >> 16) & 0xFFFF;
type = littype & 0xFFFF;

select RL_ID from DB.DBA.RDF_LANGUAGE where RL_TWOBYTE = ?

select RDT_QNAME from DB.DBA.RDF_DATATYPE where RDT_TWOBYTE = ?
```

Note:
This call is deprecated in favor of using the SQL_DESC_COL_LITERAL_LANG and SQL_DESC_LITERAL_TYPE options of SQLGetDescField which caches these lookups to speed up describe operations.

If this call returns SQL_SUCCESS or SQL_SUCCESS_WITH_INFO , the littype variable will contain the encoded language and rdf type information of the field.

These numbers are uniq to the database the client has connected to, and correspond to information in the DB.DBA.RDF_LANGUAGE and DB.DBA.RDF_DATATYPE tables:

and

SQLGetDescField - SQL_DESC_COL_BOX_FLAGS. Retrieves the flags associated with the field:

```
SQLINTEGER flags;
SQLRETURN rc;

rc = SQLGetDescField (hdesc, colNum, SQL_DESC_COL_BOX_FLAGS, &flags, SQL_IS_INTEGER, NULL);

#define VIRTUOSO_BF_IRI                     0x1
#define VIRTUOSO_BF_UTF8                    0x2
#define VIRTUOSO_BF_DEFAULT_ENC             0x4

for example:

flags    description

0        field contains a Latin1 encoded literal string
1        field contains an IRI (always UTF-8 encoded)
2        field contains an UTF-8 encoded literal string
3        field contains an UTF-8 encoded IRI (should not happen)
```

If this call returns SQL_SUCCESS or SQL_SUCCESS_WITH_INFO , the following bitmasks can be used to determine the representation of the field:

SQLGetDescField - SQL_DESC_COL_LITERAL_LANG. Retrieves the language string for this field:

```
SQLCHAR langbuf[100];
SQLINTEGER len1;
SQLRETURN rc;

rc = SQLGetDescField (hdesc, colNum, SQL_DESC_COL_LITERAL_LANG, langbuf, sizeof (langbuf), &len1);
```

If this call returns SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, , the langbuf variable will contain the language of the field.

SQLGetDescField - SQL_DESC_COL_LITERAL_TYPE. Retrieves the data type string for this field:

```
SQLCHAR typebuf[100];
SQLINTEGER len2;
SQLRETURN rc;

rc = SQLGetDescField (hdesc, colNum, SQL_DESC_COL_LITERAL_TYPE, typebuf, sizeof (typebuf), &len2);
```

If this call returns SQL_SUCCESS or SQL_SUCCESS_WITH_INFO , the typebuf variable will contain the rdf type of the field.

Evaluating Nodes

The following pseudo code evaluates the various type and flag information retrieved using the above API calls and shows what kind of node a field is.

```
switch (dvtype)
  {
  case VIRTUOSO_DV_STRING:
    {
      if (flag == 1)
```

```
      {
        if (strncmp ((char *) data, "_:", 2) == 0)
          {
            /* node is a Turtle style named BNode */
          }
        else
          {
            /* node is an URI string */
          }
      }
    else
      {
        if (strncmp ((char *) data, "nodeID://", 9) == 0)
          {
            /* node is a BNode */
          }
        else
          {
            /* node is a string literal */
          }
      }
    break;
  }

case VIRTUOSO_DV_RDF:
  /* node is a typed literal with possible lang and type */
  break;

case VIRTUOSO_DV_LONG_INT:
  /* node is a literal http://www.w3.org/2001/XMLSchema#integer */
  break;

case VIRTUOSO_DV_SINGLE_FLOAT:
  /* node is a literal http://www.w3.org/2001/XMLSchema#float */
  break;

case VIRTUOSO_DV_DOUBLE_FLOAT:
  /* node is a literal http://www.w3.org/2001/XMLSchema#double */
  break;

case VIRTUOSO_DV_NUMERIC:
  /* node is a literal http://www.w3.org/2001/XMLSchema#decimal */
  break;

case VIRTUOSO_DV_TIMESTAMP:
case VIRTUOSO_DV_DATE:
case VIRTUOSO_DV_TIME:
case VIRTUOSO_DV_DATETIME:
  switch (dv_dt_type)
    {
    case VIRTUOSO_DT_TYPE_DATE:
      /* node is a literal http://www.w3.org/2001/XMLSchema#date */
      break;
    case VIRTUOSO_DT_TYPE_TIME:
      /* node is a literal http://www.w3.org/2001/XMLSchema#time */
      break;
    default:
      /* node is a literal http://www.w3.org/2001/XMLSchema#dateTime */
      break;
    }
  break;

case VIRTUOSO_DV_IRI_ID:
  /*
   * node is an IRI ID
   *
   * This type is only returned in output:valmode "LONG"
   * It needs to be translated into a literal string using the
   * ID_TO_IRI() function as the value is database specific.
   */
  break;

default:
```

```
    /* unhandled type */
    return NULL;
  }
```

Examples

The following program performs a SPARQL query against a Virtuoso Database using SPASQL. Note that the connection parameters and the sparql query are compiled into the executable.

To compile it on Linux against iODBC the following command can be used:

```
gcc -o odbc_iri -I/usr/local/iODBC odbc_iri.c -L/usr/local/iODBC/lib -liodbc -ldl
```

It can then be called as:

```
./odbc_iri
```

which will print out the first 50 triples from the database in N3 format.

Here it is the source code:

```
/*
 *  odbc_iri.c
 *
 *  This file is part of the OpenLink Software Virtuoso Open-Source (VOS)
 *  project.
 *
 *  Copyright (C) 1998-2024 OpenLink Software
 *
 *  This project is free software; you can redistribute it and/or modify it
 *  under the terms of the GNU General Public License as published by the
 *  Free Software Foundation; only version 2 of the License, dated June 1991.
 *
 *  This program is distributed in the hope that it will be useful, but
 *  WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 *  General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License along
 *  with this program; if not, write to the Free Software Foundation, Inc.,
 *  51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
 *
 */

#include <stdio.h>
#include <string.h>

#ifdef WIN32
# include <windows.h>
#endif

#include <sql.h>
#include <sqlext.h>

#if defined (HAVE_IODBC)
#include <iodbcext.h>
#endif

/*
 *  Include Virtuoso ODBC extensions for SPASQL result set
 */
#if !defined (SQL_DESC_COL_DV_TYPE)

/*
 *  ODBC extensions for SQLGetDescField
 */
# define SQL_DESC_COL_DV_TYPE               1057L
# define SQL_DESC_COL_DT_DT_TYPE            1058L
# define SQL_DESC_COL_LITERAL_ATTR          1059L
# define SQL_DESC_COL_BOX_FLAGS             1060L
```

```
# define SQL_DESC_COL_LITERAL_LANG          1061L
# define SQL_DESC_COL_LITERAL_TYPE          1062L

/*
 *  Virtuoso - ODBC SQL_DESC_COL_DV_TYPE
 */
# define VIRTUOSO_DV_DATE                   129
# define VIRTUOSO_DV_DATETIME               211
# define VIRTUOSO_DV_DOUBLE_FLOAT           191
# define VIRTUOSO_DV_IRI_ID                 243
# define VIRTUOSO_DV_LONG_INT               189
# define VIRTUOSO_DV_NUMERIC                219
# define VIRTUOSO_DV_RDF                    246
# define VIRTUOSO_DV_SINGLE_FLOAT           190
# define VIRTUOSO_DV_STRING                 182
# define VIRTUOSO_DV_TIME                   210
# define VIRTUOSO_DV_TIMESTAMP              128
# define VIRTUOSO_DV_TIMESTAMP_OBJ          208

/*
 *  Virtuoso - ODBC SQL_DESC_COL_DT_DT_TYPE
 */
# define VIRTUOSO_DT_TYPE_DATETIME          1
# define VIRTUOSO_DT_TYPE_DATE              2
# define VIRTUOSO_DT_TYPE_TIME              3

/*
 *  Virtuoso - ODBC SQL_DESC_COL_BOX_FLAGS
 */
#define VIRTUOSO_BF_IRI                     0x1
#define VIRTUOSO_BF_UTF8                    0x2
#define VIRTUOSO_BF_DEFAULT_ENC             0x4

#endif

SQLHANDLE henv = SQL_NULL_HANDLE;
SQLHANDLE hdbc = SQL_NULL_HANDLE;
SQLHANDLE hstmt = SQL_NULL_HANDLE;

#define MAXCOLS                             25

int
ODBC_Errors (char *where)
{
  unsigned char buf[250];
  unsigned char sqlstate[15];

  /*
   *  Get statement errors
   */
  while (SQLError (henv, hdbc, hstmt, sqlstate, NULL, buf, sizeof (buf), NULL) == SQL_SUCCESS)
    {
      fprintf (stdout, "STMT: %s || %s, SQLSTATE=%s\n", where, buf, sqlstate);
    }

  /*
   *  Get connection errors
   */
  while (SQLError (henv, hdbc, SQL_NULL_HSTMT, sqlstate, NULL, buf, sizeof (buf), NULL) == SQL_SUCCES
    {
      fprintf (stdout, "CONN:%s || %s, SQLSTATE=%s\n", where, buf, sqlstate);
    }

  /*
   *  Get environment errors
   */
  while (SQLError (henv, SQL_NULL_HDBC, SQL_NULL_HSTMT, sqlstate, NULL, buf, sizeof (buf), NULL) == S
    {
      fprintf (stdout, "ENV:%s || %s, SQLSTATE=%s\n", where, buf, sqlstate);
    }

  return -1;
}
```

```
int
ODBC_Disconnect (void)
{
  if (hstmt)
    SQLFreeHandle (SQL_HANDLE_STMT, hstmt);
  hstmt = SQL_NULL_HANDLE;

  if (hdbc)
    SQLDisconnect (hdbc);

  if (hdbc)
    SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
  hdbc = SQL_NULL_HANDLE;

  if (henv)
    SQLFreeHandle (SQL_HANDLE_ENV, henv);
  henv = SQL_NULL_HANDLE;

  return 0;
}

int
ODBC_Connect (char *dsn, char *usr, char *pwd)
{
  SQLRETURN rc;

  /* Allocate environment handle */
  rc = SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
  if (!SQL_SUCCEEDED (rc))
    goto error;

  /* Set the ODBC version environment attribute */
  rc = SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION, (void *) SQL_OV_ODBC3, 0);
  if (!SQL_SUCCEEDED (rc))
    goto error;

  /* Allocate connection handle */
  rc = SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
  if (!SQL_SUCCEEDED (rc))
    goto error;

  /* Connect to data source */
  rc = SQLConnect (hdbc, (SQLCHAR *) dsn, SQL_NTS, (SQLCHAR *) usr, SQL_NTS, (SQLCHAR *) pwd, SQL_NTS
  if (!SQL_SUCCEEDED (rc))
    goto error;

  /* Allocate statement handle */
  rc = SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt);
  if (!SQL_SUCCEEDED (rc))
    goto error;

  /* Successful connection */
  return 0;

error:
  /* Failed connection */
  ODBC_Errors ("ODBC_Connect");

  ODBC_Disconnect ();

  return -1;
}

int
ODBC_PrintResult ()
{
  char fetchBuffer[1000];
  short numCols = 0;
  short colNum;
  SDWORD colIndicator;
  UDWORD totalRows;
  UDWORD totalSets;
```

```
SQLHANDLE hdesc = SQL_NULL_HANDLE;
SQLRETURN rc;

totalSets = 0;
do
  {
    /*
     *  Get the number of result columns for this cursor.
     *  If it is 0, then the statement was probably not a select
     */
    rc = SQLNumResultCols (hstmt, &numCols);
    if (!SQL_SUCCEEDED (rc))
      {
        ODBC_Errors ("SQLNumResultCols");
        goto endCursor;
      }
    if (numCols == 0)
      {
        printf ("Statement executed.\n");
        goto endCursor;
      }
    if (numCols > MAXCOLS)
      numCols = MAXCOLS;

    /*
     *  Print all the fields
     */
    totalRows = 0;
    while (1)
      {
        /*
         *  Fetch next record
         */
        rc = SQLFetch (hstmt);
        if (rc == SQL_NO_DATA_FOUND)
          break;
        if (!SQL_SUCCEEDED (rc))
          {
            ODBC_Errors ("Fetch");
            break;
          }

        for (colNum = 1; colNum <= numCols; colNum++)
          {
            char buf[1000];
            SQLINTEGER len;
            int flag, dvtype;

            /*
             *  Fetch this column as character
             */
            rc = SQLGetData (hstmt, colNum, SQL_C_CHAR, fetchBuffer, sizeof (fetchBuffer), &colIndi
            if (!SQL_SUCCEEDED (rc))
              {
                ODBC_Errors ("SQLGetData");
                goto endCursor;
              }

            /*
             *  Get descriptor handle for this statement
             */
            rc = SQLGetStmtAttr (hstmt, SQL_ATTR_IMP_ROW_DESC, &hdesc, SQL_IS_POINTER, NULL);
            if (!SQL_SUCCEEDED (rc))
              {
                ODBC_Errors ("SQLGetStmtAttr");
                goto endCursor;
              }

            /*
             *  Get data type of column
             */
            rc = SQLGetDescField (hdesc, colNum, SQL_DESC_COL_DV_TYPE, &dvtype, SQL_IS_INTEGER, NUL
            if (!SQL_SUCCEEDED (rc))
```

```
                {
                  ODBC_Errors ("SQLGetDescField");
                  goto endCursor;
                }

              /*
               *  Get flags
               */
              rc = SQLGetDescField (hdesc, colNum, SQL_DESC_COL_BOX_FLAGS, &flag, SQL_IS_INTEGER, NUL
              if (!SQL_SUCCEEDED (rc))
                {
                  ODBC_Errors ("SQLGetDescField");
                  goto endCursor;
                }

              /*
               *  Show NULL fields as ****
               */
              if (colIndicator == SQL_NULL_DATA)
                {
                  printf ("NULL");
                }
              else
                {
                  if (flag & VIRTUOSO_BF_IRI)
                    printf ("<%s>", fetchBuffer); /* IRI */
                  else if (dvtype == VIRTUOSO_DV_STRING || dvtype == VIRTUOSO_DV_RDF)
                    printf ("\"%s\"", fetchBuffer); /* literal string */
                  else
                    printf ("%s", fetchBuffer); /* value */

                  if (dvtype == VIRTUOSO_DV_RDF)
                    {
                      rc = SQLGetDescField (hdesc, colNum, SQL_DESC_COL_LITERAL_LANG, buf, sizeof (bu
                      if (!SQL_SUCCEEDED (rc))
                        {
                          ODBC_Errors ("SQLGetDescField");
                          goto endCursor;
                        }
                      if (len)
                        printf ("@%.*s", (int) len, buf);

                      rc = SQLGetDescField (hdesc, colNum, SQL_DESC_COL_LITERAL_TYPE, buf, sizeof (bu
                      if (!SQL_SUCCEEDED (rc))
                        {
                          ODBC_Errors ("SQLGetDescField");
                          goto endCursor;
                        }
                      if (len)
                        printf ("^^<%.*s>", (int) len, buf);
                    }

                  if (colNum < numCols)
                    putchar (' ');
                }
            }
          printf (" .\n");
          totalRows++;
        }

      printf ("\n\nStatement returned %lu rows.\n", totalRows);
      totalSets++;
    }
  while (SQLMoreResults (hstmt) == SQL_SUCCESS);

endCursor:
  SQLCloseCursor (hstmt);

  return 0;
}

int
ODBC_Execute (char *qr)
```

```
{
  int rc;
  SQLCHAR *Statement = (SQLCHAR *) qr;

  if ((rc = SQLExecDirect (hstmt, Statement, SQL_NTS)) != SQL_SUCCESS)
    {
      ODBC_Errors ("ODBC_Execute");
      if (rc != SQL_SUCCESS_WITH_INFO)
        return -1;
    }
  return 0;
}

char dsn[] = "Local Virtuoso";
char uid[] = "dba";
char pwd[] = "dba";
char txt[] = "sparql SELECT * WHERE {?s ?p ?o} LIMIT 50";

int
main (int argc, char *argv[])
{
  if (ODBC_Connect (dsn, uid, pwd))
    exit (1);

  if (ODBC_Execute (txt) == 0)
    ODBC_PrintResult ();

  ODBC_Disconnect ();

  exit (0);
}
```

See Also:

- RDF Data Access and Data Management

7.3.10. Examples

- C++ Demo
- ODBC Bench Test 32

7.4. Virtuoso Driver for JDBC

The Virtuoso Drivers for JDBC are available in "jar" file formats for JDBC 1.x, JDBC 2.x and JDBC 3.x specifications. These are Type 4 Drivers implying that utilization is simply a case of adding the relevant "jar" file to your CLASSPATH and then providing an appropriate JDBC URL format in order to establish a JDBC session with a local or remote Virtuoso server. It is important to note that when you make a JDBC connection to a Virtuoso Server, you do also have access to Native and External Virtuoso tables. Thus, you actually have a type 4 JDBC Driver for any number of different database types that have been linked into Virtuoso.

The JDBC 2 and JDBC 3 drivers also incorporate SSL encryption to enable very secure connections to the Virtuoso database.

7.4.1. Virtuoso Drivers for JDBC Packaging

These drivers are installed alongside the Virtuoso Server or as part of a Virtuoso Client components only installation. They are packaged as follows:

Table 7.14. Features Comparison

Driver Name	Java Package	"jar" File Archive	Default Location	Java Version
virtuoso.jdbc.Driver	virtuoso.jdbc	virtjdbc.jar	<virtuoso installation directory>\jdk11	Java 1.1.x
virtuoso.jdbc2.Driver	virtuoso.jdbc2	virtjdbc2.jar		Java 1.2/1.3

Driver Name	Java Package	"jar" File Archive	Default Location	Java Version
			<virtuoso installation directory>\jdk12	
virtuoso.jdbc3.Driver used for Java 1.4 and Java 1.5	virtuoso.jdbc3	virtjdbc3.jar	<virtuoso installation directory>\jdk13	Java 1.4
virtuoso.jdbc4.Driver used for Java 1.6	virtuoso.jdbc4	virtjdbc4.jar	<virtuoso installation directory>\jdk14	Java 1.6

7.4.2. Virtuoso Driver For JDBC URL Format

JDBC compliant applications and applets connect to JDBC Drivers using JDBC Uniform Resource Locators (URLs). Although there are two Virtuoso Drivers for JDBC, both share the same JDBC URL format.

The Virtuoso Driver for JDBC URL format takes the following form:

```
jdbc:virtuoso://<Hostname>:<Port#>/DATABASE=<dbname>/UID=<user name>/PWD=<password>/
  CERT=<certificate_alias>/KPATH=<keystore_path>/PASS=<keystore_password>/
  PROVIDER=<ssl_provider_classname>/SSL/CHARSET=<character set>/
  TIMEOUT=<timeout_secs>/PWDTYPE=<authentication_type>/log_enable=<integer>
```

Also is supported Host:Port list in connection string:

```
jdbc:virtuoso://<Hostname>:<Port#>,<Hostname1>:<Port1#>,<Hostname2>:<Port2#>/
```

If Port is omitted, the default port 1111 is used.

Each part of the URL is explained below:

Protocol Identifiers. this is a constant value of "jdbc" since JDBC is the protocol in question

Sub Protocol Identifier. this is a constant value that identifies "virtuoso" as a sub protocol of JDBC

Hostname. this identifies the machine hosting a server process that speaks the "virtuoso" sub dialect of the "jdbc" protocol

Port Number. this identifies the port number on the machine from which the server which speaks the "virtuoso" sub dialect of "jdbc" listening for incoming client connections The default port number for a Virtuoso server is "1111".

/DATABASE. this identifies the database (Qualifier or Catalog) that you are connecting to via a Virtuoso server

/UID. a valid user name for the Virtuoso database that you are connecting to via JDBC

/PWD. a valid password for the user name

/CERT=<certificate _alias>. name of the certificate to use for the SSL connection stored in the keystore. This is a required option for an SSL authenticated connection

/KPATH=<keystore_p ath>. This optional parameter lets you specify the keystore file name (default: $HOME/.keystore). The path separator is \, and which is then replaced during the connection by the right platform path separator.

/PASS=<keystore_pa ssword>. password required for accessing the keystore file. This is required for the SSL authenticated connection.

/PROVIDER=<ssl_pro vider_classname>. The class name of the SSL Provider (e.g. com.sun.ssl.net.internal.ssl.Provider) to use for the SSL cryptography. This parameter is required for SSL connections.

/SSL. The SSL option is used only for SSL connection without user authentication

/CHARSET=<characte r set>. This allows the client to specify a character set for data encoding. When this option is set then all Java strings, natively Unicode, are converted to the character set specified here.

/TIMEOUT=<timeout_ secs>. Specifies the maximum amount of time (in seconds) that the driver will wait for a response to a query. When this time is exceeded a time-out error will be reported and the network connection closed, assumed to be broken.

/PWDCLEAR=<authent ication_type>. Specifies the authentication mode; how the user credentials may be transmitted to the server. This option can be one of the following 3 types: cleartext, encrypt, digest. The default is digest.

"cleartext" will transfer the password to the server in cleartext

"encrypt" will transfer the password to the server using Virtuoso's symmetric encryption technique.

"digest" will calculate an MD5 digest of the password (and some additional session variables) that will be sent to the server to be compared with the value calculated server-side.

/log_enable=<integ er>. Set log_enable=2 in order to auto commit on every changed row. Out of memory cannot be caused as with this setting there is no image in the memory for rollback.

/roundrobin. boolean attribute 1 - for use RoundRobin; 0 - do not use . (used only if more than one host:post values are in connection string)

/fbs. integer attribute, prefetch buffer size (default is 100)

/sendbs. integer attribute, socket send buffer size (default is 32768 bytes)

/recvbs. integer attribute, socket receive buffer size (default is 32768 bytes)

/usepstmtpool. boolean attribute 1 - for use PreparedStatement pool; 0 - do not use (Only for Java 1.6 and above)

/pstmtpoolsize. integer attribute, PreparedStatement pool size (default is 25)

Note:
Since JSSE has only incorporated SSL support for JDK 1.2 and above, SSL has only been implemented for the JDBC 2.x, JDBC 3.x and JDBC 4.x drivers for Virtuoso.

7.4.3. Virtuoso Driver JDBC 3.0 features

Virtuoso Driver For JDBC 3.0 javax.sql.DataSource

JDBC 3.0 compliant applications and applets may connect to a JDBC data source using JDBC `javax.sql.DataSource` instances. The Virtuoso JDBC 3.0 driver provides an implementation of the `javax.sql.DataSource` interface in the `virtuoso.jdbc3.VirtuosoDataSource` class, supporting the following properties:

Table 7.15. JDBC 3.0 VirtuosoDataSource properties

Name	Type	URL Option Equivalent	Description	
dataSourceName	java.lang.String		used in connection pooling	
description	java.lang.String		string to describe the data source (free form)	
serverName	java.lang.String		The host name of the remote host to connect to	
portNumber	int		The port on the remote host to connect to	
user	java.lang.String	/UID	username to use for the session	
password	java.lang.String	/PWD	password to use for the session	
databaseName	java.lang.String	/DATABASE	Initial catalog qualifier for the session	
charset	java.lang.String	/CHARSET	Charset used in wide<->narrow translations	
pwdClear	java.lang.String	/PWDTYPE	authentication method	

Virtuoso Driver For JDBC 3.0 & Connection Pooling

The Virtuoso JDBC 3.0 driver supports connection pooling.

The virtuoso.jdbc3.VirtuosoDataSource implements the `javax.sql.ConnectionPoolDataSource` interface. In order to use the connection pooling the administrator must deploy one instance of the `virtuoso.jdbc3.VirtuosoDriver` in the JNDI repository and set all of it's properties except `dataSourceName` . This is the "main" connection pooling data source. Then the administrator should deploy a second instance of the `virtuoso.jdbc3.VirtuosoDataSource` class and set only it's `dataSourceName` property.

Applications will use the second `virtuoso.jdbc3.VirtuosoDataSource` instance to get a connection. It will in turn call the first one to obtain all connect info and return the `java.sql.Connection` instance.

Virtuoso Driver For JDBC 3.0 & Distributed Transactions

Virtuoso supports the industry standard XA specification for distributed transaction processing. The XA specification defines an interface between the transaction manager (TM) and resource manager (RM) in a distributed transaction system. This is a generic interface and it does not directly address the use of distributed transactions from Java. The Java mapping of the XA interface is defined in Sun Microsystems Java Transaction API (JTA) and JDBC 3.0 specifications. The Virtuoso JDBC 3.0 driver supports the JTA architecture by providing the implementation of JTA resource manager interfaces.

The Virtuoso JDBC 3.0 driver provides the `virtuoso.java3.VirtuosoXid` , `virtuoso.java3.VirtuosoXADataSource` , `virtuoso.java3.VirtuosoXAConnection` , and `virtuoso.java3.VirtuosoXAResource` classes which implement the interfaces `javax.transaction.xa.Xid` , `javax.transaction.xa.XADataSource` , `javax.sql.XAConnection` , and `javax.sql.XAResource` respectively. The use if these interfaces is usually transparent for applications and the application developer shouldn't bother with them. They are used only by the JTS transaction manager which normally runs as a part of the J2EE server.

The task of the J2EE server administrator is to setup the necessary Virtuoso XA datasources. The exact procedure of this depends on the J2EE server in use (such as BEA WebLogic, IBM WebSphere, etc). Generally, this includes two steps:

1. Include the JDBC driver's jar file into J2EE server's class path.
2. Deploy an instance of `javax.transaction.xa.XADataSource` with appropriately set properties into the J2EE server's JNDI repository.

The `virtuoso.java3.VirtuosoXADataSource` class is derived from `virtuoso.java3.VirtuosoDataSource` and inherits all of its properties. These properties has to be set as described in the section Virtuoso Driver For JDBC 3.0 javax.sql.DataSource .

For example, the following has to be done in case of Sun's J2EE Reference Implementation.

1. Add the path of virtjdbc3.jar to the J2EE_CLASSPATH variable in the file $(J2EE_HOME)/bin/userconfig.bat on Windows or $(J2EE_HOME)/bin/userconfig.sh on Unix/Linux:

   ```
   set J2EE_CLASSPATH=C:/Virtuoso/lib/virtjdbc3.jar
   ```

   or

   ```
   J2EE_CLASSPATH=/home/login/virtuoso/lib/virtjdbc3.jar
   export J2EE_CLASSPATH
   ```
2. Using the following command add the XA datasource with JNDI name "jdbc/Virtuoso" which refers to the Virtuoso server running on the same computer on port 1111:

   ```
   j2eeadmin -addJdbcXADatasource jdbc/Virtuoso virtuoso.jdbc3.VirtuosoXADataSource -props server
   ```

JDBC 3.0 javax.sql.RowSet Driver Implementation

The Virtuoso JDBC 3.0 driver has two implementations of the `javax.sql.RowSet` interface - `virtuoso.javax.OPLCachedRowSet` and `virtuoso.javax.OPLJdbcRowSet` .

The `virtuoso.javax.OPLCachedRowSet` class implements a totally disconnected, memory cached rowset and the `virtuoso.javax.OPLJdbcRowset` class spans the rest of the JDBC API to implement it's methods.

Extension datatype for RDF

The IRIs and RDF literals, kept in the Virtuoso RDF store are represented by a strings and structures. Thus accessing RDF objects needs special datatypes in order to distinguish strings from IRIs and to get language and datatype of the RDF literals.

Therefore Virtuoso JDBC driver provides following classes for accessing RDF store: `virtuoso.jdbc3.VirtuosoExtendedString` for IRIs and `virtuoso.jdbc3.VirtuosoRdfBox` for RDF literal objects.

The class `virtuoso.jdbc3.VirtuosoExtendedString` will be returned when a string representing an IRI is returned to the client. It has two members "str" and "iriType", the "str" member keeps string representation of the IRI, "iriType" denote regular IRI or blank node with enum values VirtuosoExtendedString.IRI and VirtuosoExtendedString.BNODE.

If the RDF literal object have language or datatype specified then `virtuoso.jdbc3.VirtuosoRdfBox` will be returned. The following methods could be used :

```
String toString () : returns string representation of the literal
String getType () : returns string containing the datatype  of the literal
String getLang () : returns language code for the literal
```

The following code snippet demonstrates how to use extension datatypes for RDF

```
... initialization etc. skipped for brevity
boolean more = stmt.execute("sparql select * from <gr> where { ?x ?y ?z }");
ResultSetMetaData data = stmt.getResultSet().getMetaData();
while(more)
{
    rs = stmt.getResultSet();
    while(rs.next())
    {
        for(int i = 1;i <= data.getColumnCount();i++)
        {
            String s = rs.getString(i);
            Object o = ((VirtuosoResultSet)rs).getObject(i);
            if (o instanceof VirtuosoExtendedString) // String representing an IRI
              {
                VirtuosoExtendedString vs = (VirtuosoExtendedString) o;
                if (vs.iriType == VirtuosoExtendedString.IRI && (vs.strType & 0x01) == 0x01)
                    System.out.println ("<" + vs.str +">");
                else if (vs.iriType == VirtuosoExtendedString.BNODE)
                    System.out.println ("<" + vs.str +">");
              }
            else if (o instanceof VirtuosoRdfBox) // Typed literal
              {
                VirtuosoRdfBox rb = (VirtuosoRdfBox) o;
                System.out.println (rb.rb_box + " lang=" + rb.getLang() + " type=" + rb.getType());

              }
            else if(stmt.getResultSet().wasNull())
                System.out.println("NULL");
            else //
            {
                System.out.println(s);
            }

        }
    }
    more = stmt.getMoreResults();
}
...
```

7.4.4. Virtuoso Driver JDBC 4.0 features

Virtuoso Driver For JDBC 4.0 javax.sql.DataSource

JDBC 4.0 compliant applications and applets may connect to a JDBC data source using JDBC `javax.sql.DataSource` instances. The Virtuoso JDBC 4.0 driver provides an implementation of the `javax.sql.DataSource` interface in the `virtuoso.jdbc4.VirtuosoDataSource` class, supporting the following properties:

Table 7.16. JDBC 4.0 VirtuosoDataSource properties

Name	Type	URL Option Equivalent	Description	
dataSourceName	java.lang.String		used in connection pooling	
description	java.lang.String		string to describe the data source (free form)	
serverName	java.lang.String		The host name of the remote host to connect to	
portNumber	int		The port on the remote host to connect to	
user	java.lang.String	/UID	username to use for the session	
password	java.lang.String	/PWD	password to use for the session	
databaseName	java.lang.String	/DATABASE	Initial catalog qualifier for the session	
charset	java.lang.String	/CHARSET	Charset used in wide<->narrow translations	
pwdClear	java.lang.String	/PWDTYPE	authentication method	

Additionally, the following attributres are supported:

```
--- for SSL enabled ---
   public void setCertificate (String value);
   public String getCertificate ();

   public void setKeystorepass (String value);
   public String getKeystorepass ();

   public void setKeystorepath (String value);
   public String getKeystorepath ();

   public void setProvider (String value);
   public String getProvider ();

----------------------

   public void setFbs (int value);
   public int getFbs ();

   public void setSendbs (int value);
   public int getSendbs ();

   public void setRecvbs (int value);
   public int getRecvbs ();

   public void setRoundrobin (boolean value);
   public boolean getRoundrobin ();

-- For Java 1.6 and above
   public void setUsepstmtpool (boolean value);
   public boolean getUsepstmtpool ();

   public void setPstmtpoolsize (int value);
   public int getPstmtpoolsize ();
```

Virtuoso Driver For JDBC 4.0 & Connection Pooling

The Virtuoso JDBC 4.0 driver supports connection pooling.

The virtuoso.jdbc4.VirtuosoDataSource implements the `javax.sql.ConnectionPoolDataSource` interface. In order to use the connection pooling the administrator must deploy one instance of the `virtuoso.jdbc4.VirtuosoDriver` in the

JNDI repository and set all of it's properties except `dataSourceName` . This is the "main" connection pooling data source. Then the administrator should deploy a second instance of the `virtuoso.jdbc4.VirtuosoDataSource` class and set only it's `dataSourceName` property.

Applications will use the second `virtuoso.jdbc4.VirtuosoDataSource` instance to get a connection. It will in turn call the first one to obtain all connect info and return the `java.sql.Connection` instance.

VirtuosoConnectionPoolDataSource.class has the following connection pooling attributes:

```
/**
 * Get the minimum number of physical connections
 * the pool will keep available at all times. Zero ( 0 ) indicates that
 * connections will be created as needed.
 *
 * @return   the minimum number of physical connections
 *
**/
public int getMinPoolSize();

/**
 * Set the number of physical connections the pool should keep available
 * at all times. Zero ( 0 ) indicates that connections should be created
 * as needed
 * The default value is 0 .
 *
 * @param   parm a minimum number of physical connections
 *
 * @exception  java.sql.SQLException if an error occurs
 *
**/

public void setMinPoolSize(int parm);

/**
 * Get the maximum number of physical connections
 * the pool will be able contain. Zero ( 0 ) indicates no maximum size.
 *
 * @return   the maximum number of physical connections
 *
**/
public int getMaxPoolSize();

/**
 * Set the maximum number of physical conections that the pool should contain.
 * Zero ( 0 ) indicates no maximum size.
 * The default value is 0 .
 *
 * @param   parm a maximum number of physical connections
 *
 * @exception  java.sql.SQLException if an error occurs
 *
**/
public void setMaxPoolSize(int parm);

/**
 * Get the number of physical connections the pool
 * will contain when it is created
 *
 * @return   the number of physical connections
 *
**/
public int getInitialPoolSize();

/**
 * Set the number of physical connections the pool
 * should contain when it is created
 *
 * @param   parm a number of physical connections
 *
 * @exception  java.sql.SQLException if an error occurs
 *
**/
```

```
public void setInitialPoolSize(int parm);

/**
 * Get the number of seconds that a physical connection
 * will remain unused in the pool before the
 * connection is closed. Zero ( 0 ) indicates no limit.
 *
 * @return   the number of seconds
**/
public int getMaxIdleTime();

/**
 * Set the number of seconds that a physical connection
 * should remain unused in the pool before the
 * connection is closed. Zero ( 0 ) indicates no limit.
 *
 * @param  parm a number of seconds
 *
 * @exception  java.sql.SQLException if an error occurs
 *
**/
public void setMaxIdleTime(int parm);

/**
 * Get the interval, in seconds, that the pool will wait
 * before enforcing the current policy defined by the
 * values of the above connection pool properties
 *
 * @return  the interval (in seconds)
**/
public int getPropertyCycle();

/**
 * Set the interval, in seconds, that the pool should wait
 * before enforcing the current policy defined by the
 * values of the above connection pool properties
 *
 * @param  parm an interval (in seconds)
**/
public void setPropertyCycle(int parm);

/**
 * Get the total number of statements that the pool will
 * keep open. Zero ( 0 ) indicates that caching of
 * statements is disabled.
 *
 * @return  the total number of statements
**/
public int getMaxStatements();

/**
 * Set the total number of statements that the pool should
 * keep open. Zero ( 0 ) indicates that caching of
 * statements is disabled.
 *
 * @param  parm a total number of statements
 *
 * @exception  java.sql.SQLException if an error occurs
 *
**/
public void setMaxStatements(int parm);
```

Virtuoso Driver For JDBC 4.0 & Distributed Transactions

Virtuoso supports the industry standard XA specification for distributed transaction processing. The XA specification defines an interface between the transaction manager (TM) and resource manager (RM) in a distributed transaction system. This is a generic interface and it does not directly address the use of distributed transactions from Java. The Java mapping of the XA interface is defined in Sun Microsystems Java Transaction API (JTA) and JDBC 4.0 specifications. The Virtuoso JDBC 4.0 driver supports the JTA architecture by providing the implementation of JTA resource manager interfaces.

The Virtuoso JDBC 4.0 driver provides the `virtuoso.java3.VirtuosoXid`, `virtuoso.java3.VirtuosoXADataSource`, `virtuoso.java3.VirtuosoXAConnection`, and `virtuoso.java3.VirtuosoXAResource` classes which implement the interfaces `javax.transaction.xa.Xid`, `javax.transaction.xa.XADataSource`, `javax.sql.XAConnection`, and `javax.sql.XAResource` respectively. The use if these interfaces is usually transparent for applications and the application developer shouldn't bother with them. They are used only by the JTS transaction manager which normally runs as a part of the J2EE server.

The task of the J2EE server administrator is to setup the necessary Virtuoso XA datasources. The exact procedure of this depends on the J2EE server in use (such as BEA WebLogic, IBM WebSphere, etc). Generally, this includes two steps:

1. Include the JDBC driver's jar file into J2EE server's class path.
2. Deploy an instance of `javax.transaction.xa.XADataSource` with appropriately set properties into the J2EE server's JNDI repository.

The `virtuoso.java3.VirtuosoXADataSource` class is derived from `virtuoso.java3.VirtuosoDataSource` and inherits all of its properties. These properties has to be set as described in the section Virtuoso Driver For JDBC 4.0 javax.sql.DataSource .

For example, the following has to be done in case of Sun's J2EE Reference Implementation.

1. Add the path of virtjdbc4.jar to the J2EE_CLASSPATH variable in the file $(J2EE_HOME)/bin/userconfig.bat on Windows or $(J2EE_HOME)/bin/userconfig.sh on Unix/Linux:

   ```
   set J2EE_CLASSPATH=C:/Virtuoso/lib/virtjdbc4.jar
   ```

   or

   ```
   J2EE_CLASSPATH=/home/login/virtuoso/lib/virtjdbc4.jar
   export J2EE_CLASSPATH
   ```
2. Using the following command add the XA datasource with JNDI name "jdbc/Virtuoso" which refers to the Virtuoso server running on the same computer on port 1111:

   ```
   j2eeadmin -addJdbcXADatasource jdbc/Virtuoso virtuoso.jdbc4.VirtuosoXADataSource -props server
   ```

JDBC 4.0 javax.sql.RowSet Driver Implementation

The Virtuoso JDBC 4.0 driver has two implementations of the `javax.sql.RowSet` interface - `virtuoso.javax.OPLCachedRowSet` and `virtuoso.javax.OPLJdbcRowSet` .

The `virtuoso.javax.OPLCachedRowSet` class implements a totally disconnected, memory cached rowset and the `virtuoso.javax.OPLJdbcRowset` class spans the rest of the JDBC API to implement it's methods.

Extension datatype for RDF

The IRIs and RDF literals, kept in the Virtuoso RDF store are represented by a strings and structures. Thus accessing RDF objects needs special datatypes in order to distinguish strings from IRIs and to get language and datatype of the RDF literals.

Therefore Virtuoso JDBC driver provides following classes for accessing RDF store: `virtuoso.jdbc4.VirtuosoExtendedString` for IRIs and `virtuoso.jdbc4.VirtuosoRdfBox` for RDF literal objects.

The class `virtuoso.jdbc4.VirtuosoExtendedString` will be returned when a string representing an IRI is returned to the client. It has two members "str" and "iriType", the "str" member keeps string representation of the IRI, "iriType" denote regular IRI or blank node with enum values VirtuosoExtendedString.IRI and VirtuosoExtendedString.BNODE.

If the RDF literal object have language or datatype specified then `virtuoso.jdbc4.VirtuosoRdfBox` will be returned. The following methods could be used :

```
String toString () : returns string representation of the literal
String getType () : returns string containing the datatype  of the literal
String getLang () : returns language code for the literal
```

The following code snippet demonstrates how to use extension datatypes for RDF

```
... initialization etc. skipped for brevity
boolean more = stmt.execute("sparql select * from <gr> where { ?x ?y ?z }");
ResultSetMetaData data = stmt.getResultSet().getMetaData();
while(more)
{
    rs = stmt.getResultSet();
    while(rs.next())
    {
        for(int i = 1;i <= data.getColumnCount();i++)
        {
            String s = rs.getString(i);
            Object o = ((VirtuosoResultSet)rs).getObject(i);
            if (o instanceof VirtuosoExtendedString) // String representing an IRI
              {
                VirtuosoExtendedString vs = (VirtuosoExtendedString) o;
                if (vs.iriType == VirtuosoExtendedString.IRI && (vs.strType & 0x01) == 0x01)
                    System.out.println ("<" + vs.str +">");
                else if (vs.iriType == VirtuosoExtendedString.BNODE)
                    System.out.println ("<" + vs.str +">");
              }
            else if (o instanceof VirtuosoRdfBox) // Typed literal
              {
                VirtuosoRdfBox rb = (VirtuosoRdfBox) o;
                System.out.println (rb.rb_box + " lang=" + rb.getLang() + " type=" + rb.getType());

              }
            else if(stmt.getResultSet().wasNull())
                System.out.println("NULL");
            else //
            {
                System.out.println(s);
            }

        }
    }
    more = stmt.getMoreResults();
}
...
```

7.4.5. Installation & Configuration Steps

Perform the following steps in order to make use of your Virtuoso Drivers for JDBC:

Note:
You only have to perform these steps if a first attempt to use the Virtuoso Drivers for JDBC fails, the Virtuoso installer will attempt to configure these settings for you at installation time.

1. Ensure your PATH environment variable is pointing to a version of the Java Virtual Machine (JVM) that is compatible with the version of the JDBC Driver Manager installed on your machine. Consult the section above to double check. You can also type the following command to verify Java versions:

   ```
   java -version
   ```
2. Add the appropriate Virtuoso for JDBC "jar" file to your CLASSPATH environment variable.
3. Attempt to make a connection using one of the sample JDBC Applications or Applets provided with your Virtuoso installation.

Note:
If you have problems using the virtuoso JDBC driver despite your CLASSPATH being defined correctly, you may force the Java Virtual Machine to load a specific JDBC driver using: `-D` on the java command line: e.g:

```
-Djdbc.drivers=virtuoso.jdbc.Driver.
```

You can check the Virtuoso JDBC driver version from the command line using: `java virtuoso.jdbc.Driver`

7.4.6. Virtuoso JDBC Driver Hibernate Support

Introduction

What

Hibernate is a powerful, open source, high performance object/relational persistence and query service. Hibernate lets you develop persistent classes following object-oriented idiom - including association, inheritance, polymorphism, composition, and collections. Hibernate allows you to express queries in its own portable SQL extension (HQL), as well as in native SQL, or with an object-oriented Criteria and Example API.

Why

Hibernate employs very aggressive, and very intelligent first and second level caching strategy, providing a high performance and scalable development framework for Java. Greater cross portability and productivity can also be achieve using hibernate as the same techniques can be employed across multiple databases.

How

Hibernate uses JDBC for accessing databases and may require a given database has a custom SQL dialect file that informs Hibernate what SQL dialects are to be use for performing certain operations against the target database. Although not strictly required, it should be used to ensure Hibernate Query Language (HQL) statements are correctly converted into the proper SQL dialect for the underlying database. Virtuoso includes a new jar file called virt_dialect.jar containing the SQL dialect mappings required between hibernate and Virtuoso and is used in conjunction to the Virtuoso JDBC Drivers (virtjdbc3.jar or virtjdbc4.jar).

Setup and Testing

Three sample programs are provided to test Virtuoso hibernate support. Extract the contents of the zip file to a location of choice.

Requirements

- Hibernate 3.3 or higher
- JDK 5.0 or higher on any operating system
- Ant 1.6
- Virtuoso SQL Dialect jar file (virt_dialect.jar)
- Virtuoso JDBC Driver (virtjdbc4.jar)

Building and running the example

The following Ant targets are available:

```
clean         Clean the build directory
compile       Build example
run           Build and run example
```

Hibernate connection string

Edit the file `hibernate.cfg.xml` in the "bin" and "src" directories of the hibernate application directory with the correct connection attributes for the Virtuoso Server instance:

```
$ more hibernate.cfg.xml
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

    <session-factory>

        <!-- Database connection settings -->

        <property name="connection.driver_class">virtuoso.jdbc4.Driver</property>
        <property name="connection.url">jdbc:virtuoso://localhost:1111/</property>
```

```
        <property name="connection.username">dba</property>
        <property name="connection.password">dba</property>

        <!-- JDBC connection pool (use the built-in) -->

        <property name="connection.pool_size">1</property>

        <!-- SQL dialect -->

        <property name="dialect">virtuoso.hibernate.VirtuosoDialect</property>

        <!-- Enable Hibernate's current session context -->

        <property name="current_session_context_class">thread</property>

        <!-- Disable the second-level cache  -->

        <property name="cache.provider_class">org.hibernate.cache.NoCacheProvider</property>

        <!-- Echo all executed SQL to stdout -->

        <property name="show_sql">true</property>

        <!-- Drop and re-create the database schema on startup -->

        <property name="hbm2ddl.auto">create</property>

        <mapping resource="events/Event.hbm.xml"/>

    </session-factory>

</hibernate-configuration>
```

The key attributes being

- *connection.driver_class*

  - Virtuoso JDBC Driver class name, typically virtuoso.jdbc4.Driver
- *connection.url* - Virtuoso JDBC Driver connect string for target Virtuoso server instance, of the form `jdbc:virtuoso://hostname:portno`
- *connection.username*

  - Virtuoso Server username
- *connection.password*

  - Virtuoso Server password

7.4.7. Examples

To assist you further during your utilization or evaluation of OpenLink's Drivers for JDBC a number of demonstration JDBC compliant Applets and Applications are bundled with your OpenLink Driver for JDBC installation, our demo programs are provided in both binary and source code format for your free use.

Example 1

This sample performs a simple insert and retrieval of data against the Virtuoso database.

1. Open your command shell and change to the "ex1" directory
2. The following required files need to be placed in the ./lib directory:

```
antlr-2.7.6.jar
commons-collections-3.1.jar
commons-logging-1.0.4.jar
dom4j-1.6.1.jar
hibernate3.jar
javassist-3.4.GA.jar
jta-1.1.jar
lib.lst
```

```
log4j-1.2.15.jar
slf4j-api-1.5.10.jar
slf4j-api-1.5.2.jar
slf4j-jcl-1.5.10.jar
virtjdbc4.jar
virt_dialect.jar
```

3. Run the example with "ant run" and read the log output on the console:

```
$ ant run
Buildfile: build.xml

clean:
   [delete] Deleting directory /Users/hughwilliams/hibernate/ex1/bin
    [mkdir] Created dir: /Users/hughwilliams/hibernate/ex1/bin

copy-resources:
     [copy] Copying 3 files to /Users/hughwilliams/hibernate/ex1/bin
     [copy] Copied 2 empty directories to 1 empty directory under /Users/hughwilliams/hibernate/ex

compile:
    [javac] Compiling 3 source files to /Users/hughwilliams/hibernate/ex1/bin

run:
     [java] Hibernate: insert into Event (EVENT_DATE, title) values (?, ?)
     [java] Hibernate: select identity_value()
     [java] Hibernate: insert into Event (EVENT_DATE, title) values (?, ?)
     [java] Hibernate: select identity_value()
     [java] Hibernate: select event0_.EVENT_ID as EVENT1_0_, event0_.EVENT_DATE as EVENT2_0_, even
     [java] Event: My Event1 Time: 2010-03-14 03:27:51.0
     [java] Event: My Event2 Time: 2010-03-14 03:27:53.0

BUILD SUCCESSFUL
Total time: 3 seconds
```

4. Hibernate logging levels can be varied by changing the settings in the src/hibernate.cfg.xml and src/log4j.properties files.

Example 2

Hello World with Java Persistence

1. Open your command shell and change to the "ex2" directory
2. The following required files need to be placed in the ./lib directory:

```
antlr-2.7.6.jar
asm-attrs.jar
asm.jar
c3p0-0.9.0.jar
cglib-2.1.3.jar
commons-collections-2.1.1.jar
commons-logging-1.0.4.jar
dom4j-1.6.1.jar
ejb3-persistence.jar
freemarker.jar
hibernate-annotations.jar
hibernate-commons-annotations.jar
hibernate-entitymanager.jar
hibernate-tools.jar
hibernate3.jar
javassist.jar
jboss-archive-browsing.jar
jta.jar
log4j-1.2.13.jar
virtjdbc4.jar
virt_dialect.jar
```

3. Use "ant schemaexport" to export a database schema automatically to the database. Ignore any errors about failing ALTER TABLE statements, they are thrown because there are no tables when you run this for the first time.

```
$ ant schemaexport
Buildfile: build.xml

compile:
```

```
copymetafiles:

schemaexport:
[hibernatetool] Executing Hibernate Tool with a JPA Configuration
[hibernatetool] 1. task: hbm2ddl (Generates database schema)
[hibernatetool]
[hibernatetool]     drop table MESSAGES;
[hibernatetool]
[hibernatetool]     create table MESSAGES (
[hibernatetool]         MESSAGE_ID decimal(20,0)  identity,
[hibernatetool]         MESSAGE_TEXT varchar(255) null,
[hibernatetool]         NEXT_MESSAGE_ID decimal(20,0) null,
[hibernatetool]         primary key (MESSAGE_ID)
[hibernatetool]     );
[hibernatetool]
[hibernatetool]     alter table MESSAGES
[hibernatetool]         add
[hibernatetool]         foreign key (NEXT_MESSAGE_ID)
[hibernatetool]         references MESSAGES;

BUILD SUCCESSFUL
Total time: 2 seconds
```

4. Check the DDL that was also exported to the file helloworld-ddl.sql

```
$ more helloworld-ddl.sql

    drop table MESSAGES;

    create table MESSAGES (
        MESSAGE_ID decimal(20,0)  identity,
        MESSAGE_TEXT varchar(255) null,
        NEXT_MESSAGE_ID decimal(20,0) null,
        primary key (MESSAGE_ID)
    );

    alter table MESSAGES
        add
        foreign key (NEXT_MESSAGE_ID)
        references MESSAGES;
```

5. Run the example with "ant run" and read the log output on the console

```
$ ant run
Buildfile: build.xml

compile:

copymetafiles:

run:
     [java] 1 message(s) found:
     [java] Hello World with JPA

BUILD SUCCESSFUL
Total time: 2 seconds
```

6. Call ant run again

```
$ ant run
Buildfile: build.xml

compile:

copymetafiles:

run:
     [java] 2 message(s) found:
     [java] Hello World with JPA
     [java] Hello World with JPA

BUILD SUCCESSFUL
Total time: 2 seconds
```

7. If you call ant schemaexport again, all tables will be re-created

8. Hibernate logging levels can be varied by changing the settings in the src/etc/log4j.properties and src/etc/META-INF/persistence.xml files

Example 3

This sample performs more complex insert and retrieval of data against the Virtuoso database.

1. Open your command shell and change to the "ex3" directory
2. The following required files need to be placed in the ./lib directory:

   antlr-2.7.6.jar commons-collections-3.1.jar commons-logging-1.0.4.jar dom4j-1.6.1.jar hibernate3.jar javassist-3.4.GA.jar jta-1.1.jar libs.lst log4j-1.2.15.jar slf4j-api-1.5.10.jar slf4j-api-1.5.2.jar slf4j-jcl-1.5.10.jar virtjdbc4.jar virt_dialect.jar
3. Run the example with "ant run" and read the log output on the console:

```
$ ant run
Buildfile: build.xml

clean:
   [delete] Deleting directory /Users/hughwilliams/hibernate/ex3/bin
    [mkdir] Created dir: /Users/hughwilliams/hibernate/ex3/bin

copy-resources:
     [copy] Copying 4 files to /Users/hughwilliams/hibernate/ex3/bin
     [copy] Copied 2 empty directories to 1 empty directory under /Users/hughwilliams/hibernate/ex

compile:
    [javac] Compiling 4 source files to /Users/hughwilliams/hibernate/ex3/bin
    [javac] Note: Some input files use unchecked or unsafe operations.
    [javac] Note: Recompile with -Xlint:unchecked for details.

run:
     [java] Hibernate: insert into EVENTS (EVENT_DATE, title) values (?, ?)
     [java] Hibernate: select identity_value()
     [java] Hibernate: insert into EVENTS (EVENT_DATE, title) values (?, ?)
     [java] Hibernate: select identity_value()
     [java] Hibernate: insert into PERSON (age, firstname, lastname) values (?, ?, ?)
     [java] Hibernate: select identity_value()
     [java] Hibernate: select person0_.PERSON_ID as PERSON1_2_0_, event2_.EVENT_ID as EVENT1_0_1_,
     [java] Hibernate: select event0_.EVENT_ID as EVENT1_0_0_, event0_.EVENT_DATE as EVENT2_0_0_,
     [java] Hibernate: select participan0_.EVENT_ID as EVENT1_1_, participan0_.PERSON_ID as PERSON
     [java] Hibernate: insert into PERSON_EVENT (PERSON_ID, EVENT_ID) values (?, ?)
     [java] Hibernate: update PERSON set age=?, firstname=?, lastname=? where PERSON_ID=?
     [java] Added person 1 to event 2
     [java] Hibernate: insert into PERSON (age, firstname, lastname) values (?, ?, ?)
     [java] Hibernate: select identity_value()
     [java] Hibernate: select person0_.PERSON_ID as PERSON1_2_0_, person0_.age as age2_0_, person0
     [java] Hibernate: select emailaddre0_.PERSON_ID as PERSON1_0_, emailaddre0_.EMAIL_ADDR as EMA
     [java] Hibernate: insert into PERSON_EMAIL_ADDR (PERSON_ID, EMAIL_ADDR) values (?, ?)
     [java] Hibernate: select person0_.PERSON_ID as PERSON1_2_0_, person0_.age as age2_0_, person0
     [java] Hibernate: select emailaddre0_.PERSON_ID as PERSON1_0_, emailaddre0_.EMAIL_ADDR as EMA
     [java] Hibernate: insert into PERSON_EMAIL_ADDR (PERSON_ID, EMAIL_ADDR) values (?, ?)
     [java] Added two email addresses (value typed objects) to person entity : 1
     [java] Hibernate: select event0_.EVENT_ID as EVENT1_0_, event0_.EVENT_DATE as EVENT2_0_, even
     [java] Event: My Event Time: 2010-03-14 03:30:02.0
     [java] Event: My Event Time: 2010-03-14 03:30:06.0

BUILD SUCCESSFUL
Total time: 5 seconds
```

4. Hibernate logging levels can be varied by changing the settings in the src/hibernate.cfg.xml and src/log4j.properties files.

Example 4

This example demonstrates how to make Multi Thread Virtuoso connection using JDBC.

It starts 3 threads: one thread executes SPARQL select and two threads execute SPARQL inserts:

```
import java.util.*;
```

```
import java.sql.*;
import java.math.*;

public class MTtest extends Thread {

  int mode = 0;
  int startId = 0;

  static  String urlDB = "jdbc:virtuoso://localhost:1111";

 public MTtest(int _mode, int _init) {
   mode = _mode;
   startId = _init;
 }

 void prnRs(ResultSet rs)
 {
   try {
     ResultSetMetaData rsmd;

     System.out.println(">>>>>>>>>");
     rsmd = rs.getMetaData();
     int cnt = rsmd.getColumnCount();

       while(rs.next()) {
         Object o;

         System.out.print("Thread:"+Thread.currentThread().getId()+"  ");
         for (int i = 1; i <= cnt; i++) {
           o = rs.getObject(i);
           if (rs.wasNull())
             System.out.print("<NULL> ");
           else
             System.out.print("["+ o + "] ");
         }
         System.out.println();
       }

   } catch (Exception e) {
     System.out.println(e);
     e.printStackTrace();
   }
   System.out.println(">>>>>>>>>");
 }

 public static void main(String argv[])
 {
  try {

    Class.forName("virtuoso.jdbc3.Driver");

    Connection conn = DriverManager.getConnection(urlDB,"dba","dba");
    Statement st = conn.createStatement();

    st.execute("sparql clear graph <mttest>");
    st.execute("sparql insert into graph <mttest> { <xxx> <P01> \"test1\" }");
    st.execute("sparql insert into graph <mttest> { <xxx> <P01> \"test2\" }");
    st.execute("sparql insert into graph <mttest> { <xxx> <P01> \"test3\" }");
    st.execute("sparql insert into graph <mttest> { <xxx> <P01> \"test4\" }");
    st.execute("sparql insert into graph <mttest> { <xxx> <P01> \"test5\" }");
    conn.close();

    int init = 0;

    for(int i=0; i < 2; i++) {
      MTtest thr1 = new MTtest(1, init);
      init+=10;

      MTtest thr2 = new MTtest(0, init);
      init+=10;

      MTtest thr3 = new MTtest(0, init);
      init+=10;
```

```
      thr1.start();
      thr2.start();
      thr3.start();

      thr1.join();
      thr2.join();
      thr3.join();
    }

    System.out.println("===End===");

  } catch (Exception e) {
    System.out.print(e);
    e.printStackTrace();
  }
 }

 public void run( )
 {
  try {

    Connection conn = DriverManager.getConnection(urlDB,"dba","dba");
    Statement st;

    st = conn.createStatement();

    if (mode == 1)
    {
      String query = "sparql SELECT * from <mttest> WHERE {?s ?p ?o}";
      ResultSet rs = st.executeQuery(query);
      prnRs(rs);
    }
    else
    {
      long id = Thread.currentThread().getId();
      for (int i =0; i < 5; i++)
        st.execute("sparql insert into graph <mttest> { <xxx"+startId+"> <P"+id+"> \"test"+i+"\" }");

      System.out.println("\nThread:"+Thread.currentThread().getId()+"  ===Rows Inserted===");
    }

    conn.close();

  } catch (SQLException e) {
        System.out.println("============================================================");
        System.out.println(">>["+e.getMessage()+"]");
        System.out.println(">>["+e.getErrorCode()+"]");
        System.out.println(">>["+e.getSQLState()+"]");
        System.out.println(e);
        System.out.println("============================================================");
        e.printStackTrace();
        System.exit(-1);
  } catch (Exception e) {
        System.out.println("============================================================");
        System.out.println(e);
        System.out.println("============================================================");
        e.printStackTrace();
        System.exit(-1);
  }
 } // run( )

}
```

7.5. OLE DB Provider for Virtuoso

OLE DB is an open data access technology developed and promoted by Microsoft. It allows for uniform data access across diverse data sources including but not limited to conventional RDBMSes. Based on the COM architecture it provides very flexible and exhaustive set of interfaces any database application developer might ever need. At the same time, OLE DB is somewhat overcomplicated and therefore is rarely used directly. In the majority of cases people use ADO(+) instead, ADO is another, somewhat simpler, of Microsoft's data access technologies. However, ADO itself is based upon OLE DB, thus those who use

ADO make indirect use of OLE DB as well.

With the advent of ADO.NET, OLE DB is no longer as an much essential part of the overall Microsoft data access architecture as it used to be. However it still remains important and useful working together with the Microsoft OLE DB .NET Data Provider.

The OLE DB Provider for Virtuoso (VIRTOLEDB) gives OLE DB and ADO applications access to the Virtuoso databases. In principle, the same thing is possible through the conjunction of the Virtuoso ODBC driver and Microsoft OLE DB Provider for ODBC. However, VIRTOLEDB provides native OLE DB access which is more complete and more efficient. Therefore it is preferable for this purpose.

7.5.1. Using the OLE DB Provider for Virtuoso

Being a COM in-process server VIRTOLEDB has to be installed on the client computer. When VIRTOLEDB is properly installed it can be used by a host of existing applications thanks to the support of standard OLE DB interfaces.

System Requirements

VIRTOLEDB requires the following software:

- Windows 98, ME, NT4.0, 2000, or XP.
- Virtuoso Server running on the same or remote computer.
- Redistributable files provided with Microsoft Data Access Components (MDAC) 2.5 or higher. Note that this version of MDAC comes along with Windows 2000. It is also available (as well as other versions of MDAC) at the Microsoft Web site: http://www.microsoft.com/data/

Installing the Provider

VIRTOLEDB is installed as a part of Virtuoso installation on a Windows platform. The following files pertain to the OLE DB Provider for Virtuoso.

Table 7.17. OLE DB Provider for Virtuoso Files

File	Description
virtoledb.dll	DLL that implements the VIRTOLEDB provider.
virtoledb.h	Header file for development of VIRTOLEDB consumers.

The installation procedure, in addition to placing VIRTOLEDB files on a client computer, also registers VIRTOLEDB in the system to make it known as a COM server. This also makes it available through the OLE DB root enumerator object and the Data Links component.

Invoking the Provider

Applications that utilize OLE DB Data Links component can use VIRTOLEDB without any specific measures. Applications that need to directly invoke VIRTOLEDB should follow examples provided in this section.

Using Data Link User Interface

Data Links is a user interface component for connecting to OLE DB data sources and constructing ADO-style connection strings. It belongs to OLE DB core components and is a part of MDAC. It is used my many applications including development environments like MS Visual Studio.

VIRTOLEDB uses the extension mechanism of the Data Link API and provides a customized version of the Data Link connection page.

Figure 7.33. Data Link Provider Page

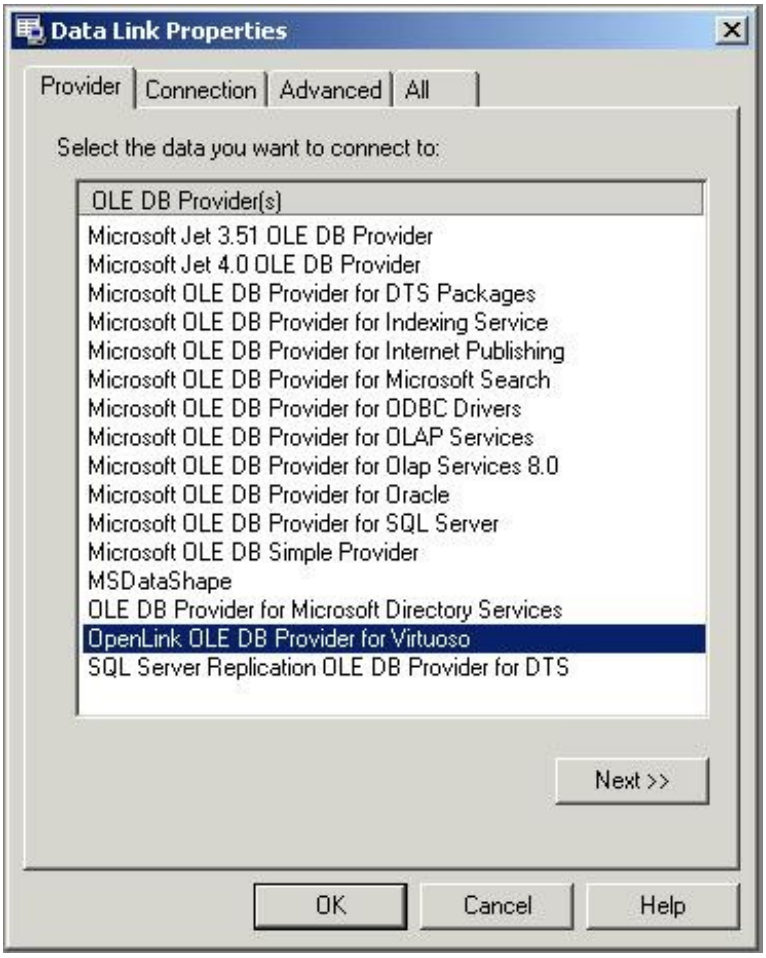
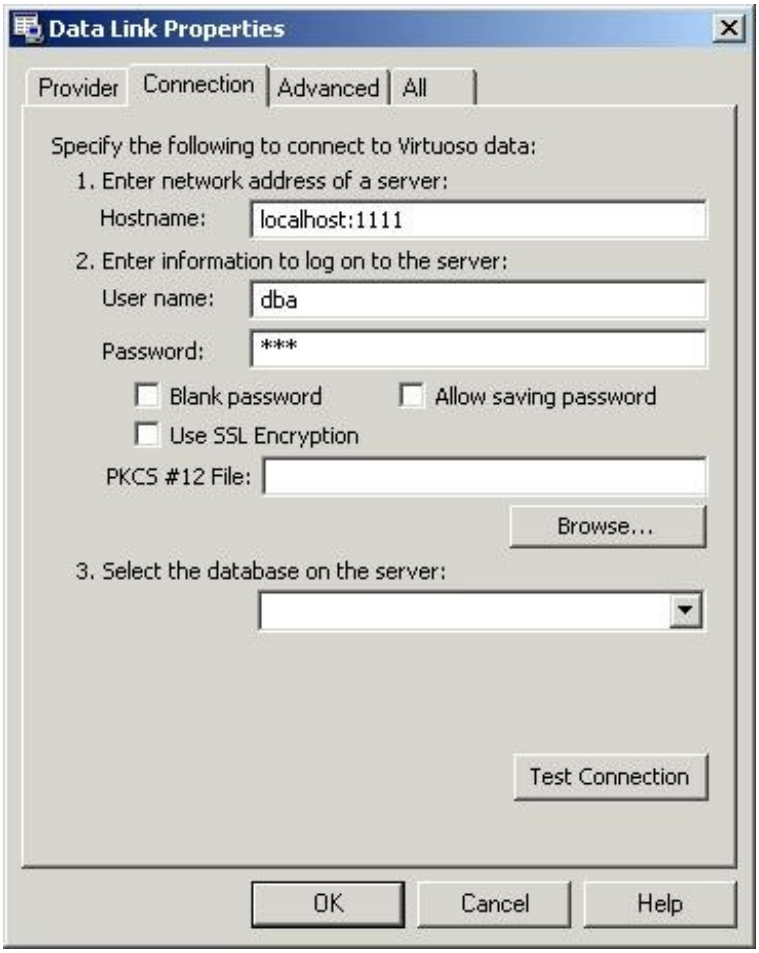


Figure 7.34. Data Link Connection Page



Using COM and OLE DB Interfaces Directly

Applications that utilize OLE DB Data Links component can use VIRTOLEDB without any specific measures. Applications that need to directly invoke VIRTOLEDB should follow examples provided in this section.

```
#define INITGUID
#include "virtoledb.h"

..

  IDBInitialize* pIDBInitialize = NULL;
  HRESULT hr = CoCreateInstance(CLSID_VIRTOLEDB, NULL, CLSCTX_INPROC_SERVER,
                                IID_IDBInitialize, (void**) &pIDBInitialize);
  if (FAILED(hr))
    goto EXIT;
```

ADO Applications

```
Dim strConn As String
Dim objConn As ADODB.Connection

strConn = "Provider=VIRTOLEDB;Data Source=localhost:1111;User Id=dba;Password=dba;Initial Catalog=Demo;Pr

Set objConn = New ADODB.Connection
objConn.CursorLocation = adUseServer
objConn.Open strConn
```

7.5.2. Known Limitations

- Per-column properties are not supported.
- Reference accessors are not supported.
- Optimized accessors are not supported.
- Asynchronous execution is not supported.

7.5.3. Data Types

Data Type Mappings in Rowsets and Parameters

The methods that provide information about rowset columns and command parameters (IColumnsInfo::GetColumnInfo, ICommandWithParameters::GetParameterInfo) use the mapping of Virtuoso data types into OLE DB data types shown in the following table.

Table 7.18. Data Type Mappings

Virtuoso Type	OLE DB Type
CHAR	DBTYPE_STR
VARCHAR	DBTYPE_STR
LONG VARCHAR	DBTYPE_STR
NCHAR	DBTYPE_WSTR
NVARCHAR	DBTYPE_WSTR
LONG NVARCHAR	DBTYPE_WSTR
NUMERIC	DBTYPE_NUMERIC
DECIMAL	DBTYPE_NUMERIC
SMALLINT	DBTYPE_I2
INTEGER	DBTYPE_I4
FLOAT	DBTYPE_R8
DOUBLE	DBTYPE_R8
REAL	DBTYPE_R4
VARBINARY	DBTYPE_BYTES
LONG VARBINARY	DBTYPE_BYTES
DATE	DBTYPE_DBDATE
TIME	DBTYPE_DBTIME
DATETIME	DBTYPE_DBTIMESTAMP
TIMESTAMP	BINARY

Data Type Conversions

Applications can bind column and parameter values using data types different from those described in the section called “Data Type Mappings in Rowsets and Parameters” . In such cases VIRTOLEDB uses OLE DB Data Conversion Library. See MDAC documentation for the list of supported data type conversions.

Long Data Types

Long data types include LONG VARCHAR, LONG NVARCHAR, and LONG VARBINARY. A long value can only be bound to a buffer that have one of these OLE DB types:

- DBTYPE_IUNKNOWN
- DBTYPE_STR
- DBTYPE_STR | DBTYPE_BYREF
- DBTYPE_WSTR
- DBTYPE_WSTR | DBTYPE_BYREF
- DBTYPE_BYTES
- DBTYPE_BYTES | DBTYPE_BYREF

Other type conversions are not supported.

If a long data is bound to a DBTYPE_IUNKNOWN type, this implies the use of the ISequentialStream interface. VIRTOLEDB supports the ISequentialStream::Read method both when getting and setting data. The ISequentialStream::Write method is never supported.

7.5.4. Metadata

Schema Rowsets

VIRTOLEDB supports schema rowsets listed in the following table.

Table 7.19. Supported Schema Rowsets

Schema Rowset	Supported Restrictions
DBSCHEMA_CATALOGS	All (CATALOG_NAME)
DBSCHEMA_COLUMN_PRIVILEGES	All (TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME, GRANTOR, GRANTEE)
DBSCHEMA_COLUMNS	All (TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME)
DBSCHEMA_FOREIGN_KEYS	All (PK_TABLE_CATALOG, PK_TABLE_SCHEMA, PK_TABLE_NAME, FK_TABLE_CATALOG, FK_TABLE_SCHEMA, FK_TABLE_NAME)
DBSCHEMA_INDEXES	1, 2, 3, and 5 (TABLE_CATALOG, TABLE_SCHEMA, INDEX_NAME, TABLE_NAME)
DBSCHEMA_PRIMARY_KEYS	All (TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME)
DBSCHEMA_PROCEDURE_PARAMETERS	All (PROCEDURE_CATALOG, PROCEDURE_SCHEMA, PROCEDURE_NAME, PARAMETER_NAME)
DBSCHEMA_PROCEDURES	All (PROCEDURE_CATALOG, PROCEDURE_SCHEMA, PROCEDURE_NAME, PROCEDURE_TYPE)
DBSCHEMA_PROVIDER_TYPES	All (DATA_TYPE, BEST_MATCH)
DBSCHEMA_SCHEMATA	All (CATALOG_NAME, SCHEMA_NAME, SCHEMA_OWNER)
DBSCHEMA_TABLE_PRIVILEGES	All (TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, GRANTOR, GRANTEE)
DBSCHEMA_TABLES	All (TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, TABLE_TYPE)

Columns Rowset

VIRTOLEDB supports the following columns in the columns rowset.

- DBCOLUMN_IDNAME
- DBCOLUMN_GUID
- DBCOLUMN_PROPID
- DBCOLUMN_NAME
- DBCOLUMN_NUMBER
- DBCOLUMN_TYPE
- DBCOLUMN_TYPEINFO
- DBCOLUMN_COLUMNSIZE
- DBCOLUMN_PRECISION
- DBCOLUMN_SCALE
- DBCOLUMN_FLAGS
- DBCOLUMN_BASECATALOGNAME
- DBCOLUMN_BASECOLUMNNAME
- DBCOLUMN_BASESCHEMANAME
- DBCOLUMN_BASETABLENAME
- DBCOLUMN_COMPUTEMODE
- DBCOLUMN_DATETIMEPRECISION
- DBCOLUMN_ISCASESENSITIVE
- DBCOLUMN_ISSEARCHABLE
- DBCOLUMN_OCTETLENGTH
- DBCOLUMN_KEYCOLUMN

7.5.5. Supported Interfaces

VIRTOLEDB supports the interfaces listed in the following table.

Table 7.20. Supported Interfaces

Object	Interface	Notes
Data Source	IDBCreateSession	
	IDBInitialize	
	IDBProperties	
	IPersist	
	IDBInfo	
	IPersistFile	
	ISupportErrorInfo	
Session	IGetDataSource	
	IOpenRowset	
	ISessionProperties	
	IDBCreateCommand	
	IDBSchemaRowset	
	ISupportErrorInfo	
	ITransaction	
	ITransactionJoin	
	ITransactionLocal	
Command	IAccessor	
	IColumnsInfo	
	ICommand	
	ICommandProperties	
	ICommandText	
	IConvertType	
	IColumnsRowset	
	ICommandPrepare	
	ICommandWithParameters	
	ISupportErrorInfo	
Multiple Results	IMultipleResults	
	ISupportErrorInfo	
Rowset	IAccessor	
	IColumnsInfo	
	IConvertType	
	IRowset	
	IRowsetInfo	
	IColumnsRowset	
	IConnectionPointContainer	For IRowsetNotify interface.
	IRowsetChange	
	IRowsetIdentity	
	IRowsetLocate	
	IRowsetRefresh	
	IRowsetResynch	
	IRowsetScroll	
	IRowsetUpdate	
	ISupportErrorInfo	

Object	Interface	Notes
Error Lookup	IErrorLookup	

7.5.6. Data Source Objects

Initialization and Authorization Properties

VIRTOLEDB supports the following properties in the initialization property group.

Table 7.21. Supported Initialization and Authorization Properties

Property	R/W	Default	Notes
DBPROP_AUTH_PASSWORD	Read/Write		General Info: Password
DBPROP_AUTH_USERID	Read/Write		User ID
DBPROP_AUTH_PERSIST_SENSITIVE_AUTHINFO	Read/Write	VARIANT_FALSE	Persist Security Info
DBPROP_INIT_DATASOURCE	Read/Write		Data Source
DBPROP_INIT_CATALOG	Read/Write		Initial Catalog
DBPROP_INIT_PROVIDERSTRING	Read/Write		Extended Properties
DBPROP_INIT_HWND	Read/Write		Window Handle
DBPROP_INIT_PROMPT	Read/Write	DBPROMPT_COMPLETE	Prompt
DBPROP_INIT_TIMEOUT	Read/Write	0	Connect Timeout

In addition, VIRTOLEDB implements a provider-specific property set DBPROPSET_VIRTUOSODBINIT with the following properties

Table 7.22. Provider-Specific Initialization and Authorization Properties

Property	R/W	Default	Notes
VIRTPROP_INIT_ENCRYPT	Read/Write	VARIANT_FALSE	Type: VT_BOOL Description: Encrypt Connection If this property is set to VARIANT_TRUE the provider uses SSL encrypted connections.
VIRTPROP_AUTH_PKCS12FILE	Read/Write		Type: VT_BSTR Description: PKCS #12 File If this property is set to a non-empty string then it is used as a name of a PKCS #12 file that authenticates the client. This also implies that the VIRTPROP_INIT_ENCRYPT property is set to VARIANT_TRUE

Data Source Properties

VIRTOLEDB supports the following properties in the data source property group.

Table 7.23. Supported Data Source Properties

Property	R/W	Default	Notes
DBPROP_CURRENTCATALOG	Read/Write		Current Catalog
DBPROP_MULTIPLECONNECTIONS	Read/Write		Multiple Connections

Data Source Information Properties

VIRTOLEDB supports the following properties in the data source information property group.

Table 7.24. Supported Data Source Information Properties

Property	R/W	Default	Notes
DBPROP_ACTIVESESSIONS	Read-Only		The number of connections the Virtuoso server is licensed to.
DBPROP_ASYNCTXNABORT	Read-Only	VARIANT_FALSE	VIRTOLEDB cannot abort transactions asynchronously.
DBPROP_ASYNCTXNCOMMIT	Read-Only	VARIANT_FALSE	VIRTOLEDB cannot commit transactions asynchronously.
DBPROP_BYREFACCESSORS	Read-Only	VARIANT_FALSE	VIRTOLEDB does not support reference accessors.
DBPROP_CATALOGLOCATION	Read-Only	DBPROPVAL_CL_START	
DBPROP_CATALOGTERM	Read-Only	"qualifier"	
DBPROP_CATALOGUSAGE	Read-Only	DBPROPVAL_CU_DML_STATEMENTS | DBPROPVAL_CU_TABLE_DEFINITION | DBPROPVAL_CU_INDEX_DEFINITION | DBPROPVAL_CU_PRIVILEGE_DEFINITION	
DBPROP_COLUMNDEFINITION	Read-Only	DBPROPVAL_CD_NOTNULL	
DBPROP_COMSERVICES	Read-Only	0	
DBPROP_CONCATNULLBEHAVIOR	Read-Only	DBPROPVAL_CB_NULL	
DBPROP_CONNECTIONSTATUS	Read-Only	DBPROPVAL_CS_INITIALIZED	
DBPROP_DATASOURCENAME	Read-Only	N/A	
DBPROP_DATASOURCEREADONLY	Read-Only	VARIANT_FALSE	
DBPROP_DATASOURCE_TYPE	Read-Only	DBPROPVAL_DST_TDP	
DBPROP_DBMSNAME	Read-Only	"Virtuoso"	
DBPROP_DBMSVER	Read-Only		
DBPROP_DSOTHREADMODEL	Read-Only	DBPROPVAL_RT_FREETHREAD	
DBPROP_GROUPBY	Read-Only	DBPROPVAL_GB_NO_RELATION	
DBPROP_HETEROGENEOUSTABLES	Read-Only	DBPROPVAL_HT_DIFFERENT_CATALOGS	
DBPROP_IDENTIFIERCASE	Read-Only	DBPROPVAL_IC_SENSITIVE	
DBPROP_MAXINDEXSIZE	Read-Only	2000	
DBPROP_MAXOPENCHAPTERS	Read-Only	0	
DBPROP_MAXROWSIZE	Read-Only	2000	
DBPROP_MAXROWSIZEINCLUDESBLOB	Read-Only	VARIANT_FALSE	
DBPROP_MAXTABLESINSELECT	Read-Only	0	
DBPROP_MULTIPLEPARAMSETS	Read-Only	VARIANT_TRUE	
DBPROP_MULTIPLERESULTS	Read-Only	DBPROPVAL_MR_SUPPORTED	
DBPROP_MULTIPLESTORAGEOBJECTS	Read-Only	VARIANT_FALSE	
DBPROP_MULTITABLEUPDATE	Read-Only	VARIANT_TRUE	
DBPROP_NULLCOLLATION	Read-Only	DBPROPVAL_NC_HIGH	
DBPROP_OLEOBJECTS	Read-Only	DBPROPVAL_OO_BLOB	

Property	R/W	Default	Notes
DBPROP_OPENROWSETSUPPORT	Read-Only	DBPROPVAL_ORS_TABLE	
DBPROP_ORDERBYCOLUMNSINSELECT	Read-Only	VARIANT_FALSE	
DBPROP_OUTPUTPARAMETERAVAILABILITY	Read-Only	DBPROPVAL_OA_ATROWRELEASE	
DBPROP_PERSISTENTIDTYPE	Read-Only	DBPROPVAL_PT_NAME	
DBPROP_PREPAREABORTBEHAVIOR	Read-Only	DBPROPVAL_CB_PRESERVE	
DBPROP_PREPARECOMMITBEHAVIOR	Read-Only	DBPROPVAL_CB_PRESERVE	
DBPROP_PROCEDURETERM	Read-Only	"procedure"	
DBPROP_PROVIDERFRIENDLYNAME	Read-Only	"OpenLink OLE DB Provider for Virtuoso"	
DBPROP_PROVIDERMEMORY	Read-Only	VARIANT_TRUE	
DBPROP_PROVIDERFILENAME	Read-Only	"virtoledb.dll"	
DBPROP_PROVIDEROLEDBVER	Read-Only	"02.60"	
DBPROP_PROVIDERVER	Read-Only		
DBPROP_QUOTEDIDENTIFIERCASE	Read-Only	DBPROPVAL_IC_SENSITIVE	
DBPROP_ROWSETCONVERSIONSONCOMMAND	Read-Only	VARIANT_TRUE	
DBPROP_SCHEMATERM	Read-Only	"owner"	
DBPROP_SCHEMAUSAGE	Read-Only	DBPROPVAL_SU_DML_STATEMENTS | DBPROPVAL_SU_TABLE_DEFINITION | DBPROPVAL_SU_INDEX_DEFINITION | DBPROPVAL_SU_PRIVILEGE_DEFINITION	
DBPROP_SERVERNAME	Read-Only		
DBPROP_SQLSUPPORT	Read-Only	DBPROPVAL_SQL_ODBC_MINIMUM | DBPROPVAL_SQL_ODBC_CORE | DBPROPVAL_SQL_ANSI89_IEF | DBPROPVAL_SQL_ESCAPECLAUSES | DBPROPVAL_SQL_ANSI92_ENTRY	
DBPROP_STRUCTUREDSTORAGE	Read-Only	DBPROPVAL_SS_ISEQUENTIALSTREAM	
DBPROP_SUBQUERIES	Read-Only	DBPROPVAL_SQ_CORRELATEDSUBQUERIES | DBPROPVAL_SQ_COMPARISON | DBPROPVAL_SQ_EXISTS | DBPROPVAL_SQ_IN | DBPROPVAL_SQ_QUANTIFIED | DBPROPVAL_SQ_TABLE	
DBPROP_SUPPORTEDTXNDDL	Read-Only	DBPROPVAL_TC_DML	
DBPROP_SUPPORTEDTXNISOLEVELS	Read-Only	DBPROPVAL_TI_READUNCOMMITTED | DBPROPVAL_TI_READCOMMITTED | DBPROPVAL_TI_REPEATABLEREAD | DBPROPVAL_TI_SERIALIZABLE	
DBPROP_SUPPORTEDTXNISORETAIN	Read-Only	DBPROPVAL_TR_DONTCARE	
DBPROP_TABLESTATISTICS	Read-Only	0	
DBPROP_TABLETERM	Read-Only	"table"	
DBPROP_USERNAME	Read-Only	N/A	

7.5.7. Sessions

Session Properties

VIRTOLEDB supports the following properties in the session property group.

Table 7.25. Supported Session Properties

Property	R/W	Default	Notes
DBPROP_SESS_AUTOCOMMITISOLEVELS	Read/Write	DBPROPVAL_TI_REPEATABLEREAD	Isolation level in

Property	R/W	Default	Notes
			auto-commit mode.

7.5.8. Rowsets

Properties

Table 7.26. Supported Rowset Properties

Property	R/W	Default	Notes
DBPROP_ABORTPRESERVE	Read/Write		
DBPROP_ACCESSORDER	Read-Only	DBPROPVAL_AO_RANDOM	
DBPROP_BLOCKINGSTORAGEOBJECTS	Read-Only	VARIANT_TRUE	
DBPROP_BOOKMARKINFO	Read-Only	DBPROPVAL_BI_CROSSROWSET	
DBPROP_BOOKMARKS	Read/Write	VARIANT_FALSE	
DBPROP_BOOKMARKSKIPPED	Read-Only	VARIANT_FALSE	
DBPROP_BOOKMARKTYPE	Read-Only	DBPROPVAL_BMK_NUMERIC	
DBPROP_CACHEDEFERRED	Read-Only	VARIANT_FALSE	
DBPROP_CANFETCHBACKWARDS	Read/Write	VARIANT_FALSE	
DBPROP_CANHOLDROWS	Read/Write	VARIANT_FALSE	
DBPROP_CANSCROLLBACKWARDS	Read/Write	VARIANT_FALSE	
DBPROP_CHANGEINSERTEDROWS	Read-Only	VARIANT_FALSE	
DBPROP_COLUMNRESTRICT	Read-Only	VARIANT_FALSE	
DBPROP_COMMANDTIMEOUT	Read/Write		
DBPROP_COMMITPRESERVE	Read/Write	VARIANT_FALSE	
DBPROP_DEFERRED	Read-Only	VARIANT_FALSE or VARIANT_TRUE	
DBPROP_DELAYSTORAGEOBJECTS	Read/Write	VARIANT_FALSE	
DBPROP_FINDCOMPAREOPS	Read-Only	0	
DBPROP_HIDDENCOLUMNS	Read-Only		
DBPROP_IAccessor	Read-Only	VARIANT_TRUE	
DBPROP_IColumnsInfo	Read-Only	VARIANT_TRUE	
DBPROP_IColumnsRowset	Read/Write	VARIANT_FALSE	
DBPROP_IConnectionPointContainer	Read/Write	VARIANT_FALSE	
DBPROP_IConvertType	Read-Only	VARIANT_TRUE	
DBPROP_IMMOBILEROWS	Read/Write	VARIANT_FALSE	
DBPROP_IMultipleResults	Read/Write	VARIANT_FALSE	
DBPROP_IRowset	Read-Only	VARIANT_TRUE	
DBPROP_IRowsetChange	Read/Write	VARIANT_FALSE	
DBPROP_IRowsetIdentity	Read/Write	VARIANT_FALSE	
DBPROP_IRowsetInfo	Read-Only	VARIANT_TRUE	
DBPROP_IRowsetLocate	Read/Write	VARIANT_FALSE	
DBPROP_IRowsetRefresh	Read/Write	VARIANT_FALSE	
DBPROP_IRowsetResynch	Read/Write	VARIANT_FALSE	
DBPROP_IRowsetScroll	Read/Write	VARIANT_FALSE	
DBPROP_IRowsetUpdate	Read/Write	VARIANT_FALSE	
DBPROP_ISequentialStream	Read-Only	VARIANT_TRUE	
DBPROP_ISupportErrorInfo	Read/Write	VARIANT_FALSE	
DBPROP_LITERALBOOKMARKS	Read-Only	VARIANT_FALSE	
DBPROP_LITERALIDENTITY	Read-Only	VARIANT_TRUE	

Property	R/W	Default	Notes
DBPROP_LOCKMODE	Read/Write	DBPROPVAL_LM_NONE	
DBPROP_MAXOPENROWS	Read-Only	0	
DBPROP_MAXPENDINGROWS	Read-Only	0	
DBPROP_MAXROWS	Read/Write	0	
DBPROP_NOTIFICATIONGRANULARITY	Read-Only	DBPROPVAL_NT_SINGLEROW	
DBPROP_NOTIFICATIONPHASES	Read-Only	DBPROPVAL_NP_OKTODO | DBPROPVAL_NP_ABOUTTODO | DBPROPVAL_NP_SYNCHAFTER | DBPROPVAL_NP_FAILEDTODO | DBPROPVAL_NP_DIDEVENT	
DBPROP_NOTIFYCOLUMNSET DBPROP_NOTIFYROWDELETE DBPROP_NOTIFYROWFIRSTCHANGE DBPROP_NOTIFYROWINSERT DBPROP_NOTIFYROWRESYNCH DBPROP_NOTIFYROWSETCHANGED DBPROP_NOTIFYROWSETFETCHPOSITIONCHANGE DBPROP_NOTIFYROWSETRELEASE DBPROP_NOTIFYROWUNDOCHANGE DBPROP_NOTIFYROWUNDODELETE DBPROP_NOTIFYROWUNDOINSERT DBPROP_NOTIFYROWUPDATE	Read-Only	DBPROPVAL_NP_OKTODO | DBPROPVAL_NP_ABOUTTODO | DBPROPVAL_NP_SYNCHAFTER	
DBPROP_ORDEREDBOOKMARKS	Read/Write	VARIANT_FALSE	
DBPROP_OTHERINSERT	Read/Write	VARIANT_FALSE	
DBPROP_OTHERUPDATEDELETE	Read/Write	VARIANT_FALSE	
DBPROP_OWNINSERT	Read/Write	VARIANT_TRUE	
DBPROP_OWNUPDATEDELETE	Read/Write	VARIANT_TRUE	
DBPROP_QUICKRESTART	Read/Write	VARIANT_FALSE	
DBPROP_REENTRANTEVENTS	Read-Only	VARIANT_FALSE	
DBPROP_REMOVEDELETED	Read/Write	VARIANT_FALSE	
DBPROP_REPORTMULTIPLECHANGES	Read-Onle	VARIANT_FALSE	
DBPROP_RETURNPENDINGINSERTS	Read-Only	VARIANT_FALSE	
DBPROP_ROWRESTRICT	Read-Only	VARIANT_FALSE	
DBPROP_ROWSET_ASYNCH	Read-Only	VARIANT_FALSE	
DBPROP_ROWTHREADMODEL	Read-Only	DBPROPVAL_RT_FREETHREAD	
DBPROP_SERVERCURSOR	Read-Only	VARIANT_TRUE	
DBPROP_SERVERDATAONINSERT	Read-Only	VARIANT_FALSE	
DBPROP_SKIPROWCOUNTRESULTS	Read-Only	VARIANT_TRUE	
DBPROP_STRONGIDENTITY	Read-Only	VARIANT_FALSE	
DBPROP_TRANSACTEDOBJECT	Reaq-Only	VARIANT_FALSE	
DBPROP_UNIQUEROWS	Read/Write	VARIANT_FALSE	
DBPROP_UPDATABILITY	Read/Write	0	

Interfaces

VIRTOLEDB supports the following rowset interfaces.

Table 7.27. Supported Rowset Interfaces

Interface	Description
IAccessor	
IColumnsInfo	

Interface	Description
IConvertType	
IRowset	
IRowsetChange	
IRowsetIdentity	
IRowsetInfo	
IRowsetLocate	
IRowsetScroll	
ISupportErrorInfo	

7.6. Virtuoso In-Process Client

The in-process client is an efficient mechanism used by hosted applications to access the database functionality of the hosting Virtuoso server.

Normally applications access Virtuoso using the remote procedure call mechanism hidden behind some sort of higher-level API such as ADO.NET, ODBC, and so on. The RPC based mechanism is the only available interface for non-hosted applications. For hosted applications, when the application and the DBMS server reside in the same OS process, Virtuoso provides a mechanism based on direct in-process procedure calls.

The in-process call mechanism by itself is not exposed to the applications. It is hidden behind the same APIs as the RPC one: ODBC, JDBC, ADO.NET. So in both cases applications deal with the same APIs and the application logic does not change. The application chooses which mechanism to use when it opens a database connection.

To make an in-process connection to a Virtuoso server from within an application hosted by the same server it is necessary to use a special inprocess address syntax distinguishing it from the regular network "HOST:PORT" address syntax. The In-process address has the following form:

```
:in-process:PORT
```

Note that it begins with a colon, then specifies the "`in-process` " keyword, followed by another colon, and finally the port number on which the server listens for TCP connections.

This port is only used for establishing the in-process connection, after which the connection proceeds exclusively through the in-process procedure call mechanism.

The in-process connections are presently only supported by the Virtuoso ODBC driver and the ODBC-based .NET Data Provider (OpenLink.Data.VirtuosoClient). The managed .NET Data Provider (OpenLink.Data.Virtuoso) does not yet support in-process connections.

In-process ODBC connection are defined in either a DSN or a connect string directly. This is an example section of the odbc.ini file for unix/linux:

```
[InProcess]
Driver  = /opt/virtuoso/lib/virtodbc32_r.so
Address = :in-process:1111
```

To open an ADO.NET connection the in-process address should be specified in the connection string.

Here is an example of using the OpenLink.Data.VirtuosoClient adapter specifying the details on the connect string:

```
VirtuosoConnection = new VirtuosoConnection ("HOST=:in-process:1111");
```

Every in-process connection originates from the existing connection. For instance, it might be initiated by a stored procedure called from an ISQL connection, or it can be called from an ASPX session.

In-process connections always inherit the user ID from its parent connection. There is no way to set or change the connecting user ID. Therefore the user ID and password are not specified when the in-process connection is opened.

In-process connections also limit database transactions. The autocommit mode is never in effect for in-process connections. The setting of autocommit mode is silently ignored without reporting any errors.

7.7. Unix Domain Socket Connections

Client connections to Virtuoso servers running on the same Unix or Linux server host can benefit from faster connections utilizing Unix Domain Sockets. This does not apply to Windows platforms.

By default Virtuoso will open a Unix Domain listen socket in addition to the TCP listen socket. The name of the UD socket will be:

```
/tmp/virt-<tcp-listen-port>
```

When a client attempts to connect to the Virtuoso server using the specific address `localhost` it will first try connecting to the UD socket, failing that it will silently revert to the TCP socket.

Unix Domain Socket connections only work if `localhost` is explicitly specified or the host is unspecified which defaults to a localhost connection. UD socket connections will *not* work to any other address such as:

```
virt.mydomain.com:1111
127.0.0.1:1111
```

regardless of whether that is the localhost or not.

Unix Domain sockets can be disabled using the `DisableUnixSocket` parameter in the Parameters section of the Virtuoso INI file.

The `sys_connected_server_address()` function can be used to check the connection type. It will return

```
/tmp/virt-<tcp-listen-port>
```

for connections using UD sockets.

7.8. Virtuoso Data Access Clients Connection Fail over and Load Balancing Support

The Virtuoso Data Access Clients ODBC, JDBC, ADO.Net, OLE DB, Sesame, Jena and Redland as of Release 6.1 and above support round robin connections to Virtuoso server instances enabling server fail over, load balancing and fault tolerant connections to be performed across multiple server instances configured in a cluster or as separate server instances.

Fail over connections are enabled by specifying a comma delimited list of servers to failover to, in the "Host" or 'Server" connect string attribute, with the list being worked through in the order presented to determine which Virtuoso Server instance is used. A Round robin (load balanced) connection can be configured by adding the connect string attribute "RoundRobin" = [True | Yes | False | No], in which case the server for the connection is chosen at random from the comma delimited provided as for a Failover connection above.

Example connect strings for Virtuoso ODBC, JDBC, ADO.Net and OLE DB driver/providers are provided below. The Virtuoso Sesame and Jena providers which make use of the JDBC driver and Redland Provider which makes use of the ODBC driver, would simply make use of a suitably configured JDBC or ODBC connect string to enable Failover or Round Robin connections to be made with them.

7.8.1. ODBC

Failover Connect String format

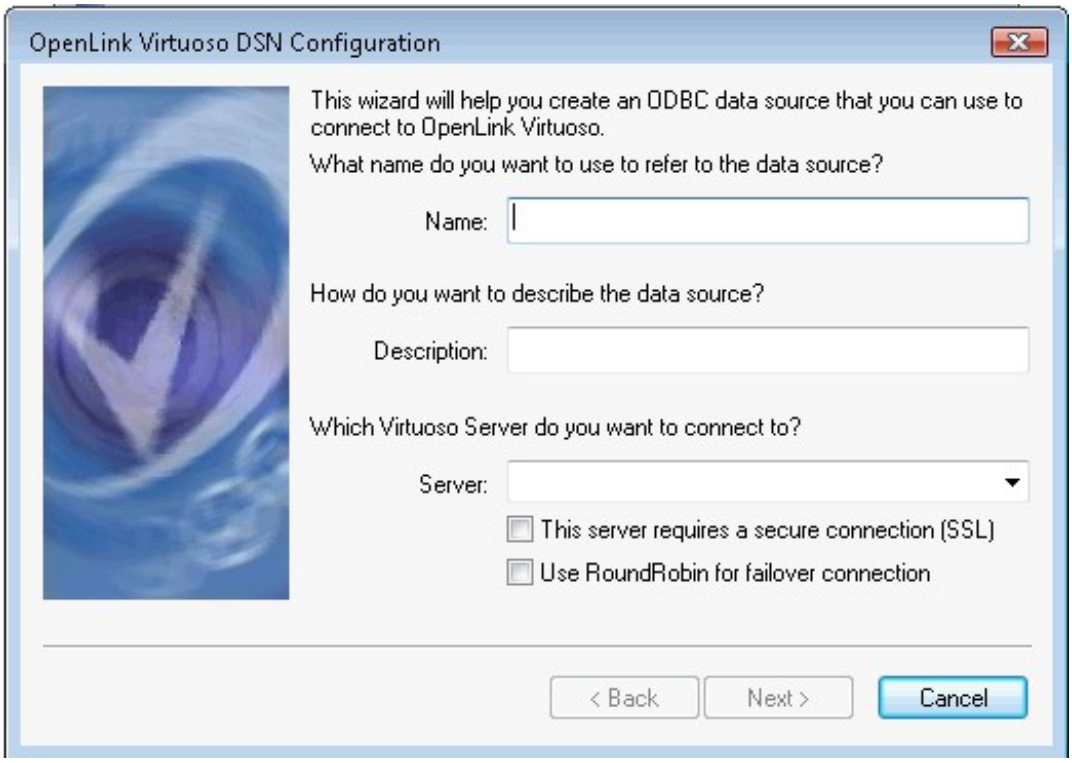
```
Driver={OpenLink Virtuoso};Host=server1:port1,server2:port2,server3:port3;UID=dba;PWD=dba;
```

Round Robin Connect String format

```
Driver={OpenLink Virtuoso};Host=server1:port1,server2:port2,server3:port3;UID=dba;PWD=dba;RoundRobin=Yes"
```

Or alternatively ensure the "use Round Robin for failover connection" check box in the setup dialog.

Figure 7.35. ODBC Round Robin Connect String format



7.8.2. ADO.Net

Failover Connect String format

```
Server=server1:port1,server2:port2,server3;UserId=dba;Password=dba;
```

Round Robin Connect String format

```
Server=server1:port1,server2:port2,server3;UserId=dba;Password=dba;Round Robin=true;Pooling=false;
```

7.8.3. JDBC

Failover Connect String format

```
jdbc:virtuoso://server1:port1,server2:port2,server3/UID=dba/PWD=dba/;
```

Round Robin Connect String format

```
jdbc:virtuoso://server1:port1,server2:port2,server3:1111/UID=dba/PWD=dba/ROUNDROBIN=yes;
```

7.8.4. OLE DB

Failover Connect String format

```
Provider=VIRTOLEDB;Data Source=server1:port1,server2:port2,server3;User Id=dba;Password=dba;Initial Catal
```

Round Robin Connect String format

```
  Provider=VIRTOLEDB;Data Source=server1:port1,server2:port2,server3;User Id=dba;
Password=dba;Initial Catalog=Demo;Prompt=NoPrompt;Round Robin=true
```

7.8.5. Sesame

Failover Connect String format

```
VirtuosoRepository("server1:port1,server2:port2,server3", "uid", "pwd");
```

Round Robin Connect String format

```
VirtuosoRepository("server1:port1,server2:port2,server3", "uid", "pwd");
((VirtuosoRepository)repository).setRoundrobin(true);
```

Sample program

```
/*
 *  $Id$
 *
 *  This file is part of the OpenLink Software Virtuoso Open-Source (VOS)
 *  project.
 *
 *  Copyright (C) 1998-2024 OpenLink Software
 *
 *  This project is free software; you can redistribute it and/or modify it
 *  under the terms of the GNU General Public License as published by the
 *  Free Software Foundation; only version 2 of the License, dated June 1991.
 *
 *  This program is distributed in the hope that it will be useful, but
 *  WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 *  General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License along
 *  with this program; if not, write to the Free Software Foundation, Inc.,
 *  51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
 */

import java.sql.*;
import java.util.*;
import java.lang.*;

import org.openrdf.model.*;
import org.openrdf.query.*;
import org.openrdf.repository.*;
import org.openrdf.rio.*;
import virtuoso.jdbc4.*;

import virtuoso.sesame2.driver.*;

public class TestSesame2 {

  public static void main(String[] args) {

      Repository repository = new
        VirtuosoRepository("localhost:1111,localhost:1311,localhost:1312,localhost:1313", "dba", "dba");
      ((VirtuosoRepository)repository).setRoundrobin(true);

      RepositoryConnection con = null;
      Random rnd = new Random();
      for (int i = 0; i < 1000000; ) {
          try {
              if (null == con) {
                  System.out.println("New connection");
                  con = repository.getConnection();
              }

              TupleQuery query = con.prepareTupleQuery(
                      QueryLanguage.SPARQL, "INSERT INTO <test_g> { <sub" + i + "> <pred> <obj" + i+ "> .
```

```
                    " <r" + Math.abs (rnd.nextInt()) + "> <rndpred> <r" + Math.abs (rnd.nextInt ()) + "
                TupleQueryResult queryResult = query.evaluate();
                /*long count = 0;
                while (queryResult.hasNext())
                {
                    queryResult.next();
                    count++;
                    if (count % 1000 == 0)
                    {
                        System.out.println("Passed " + count + " results...");
                    }
                }
                */
                queryResult.close();
                i++;
                try { Thread.sleep(100); } catch (InterruptedException ie) { }

            } catch (Exception e) {
                String state = "";
                if (e.getCause() instanceof SQLException) {
                    state = ((SQLException)e.getCause()).getSQLState();
                }
                System.out.println("ERROR:" + state + " " + e.getCause());
                try { Thread.sleep(2000); } catch (InterruptedException ie) { }
                System.out.println("ERROR:" + e.toString ());
                if (state == "")
                  e.printStackTrace();
                try {
                    if (con != null /*&& (state == "08U01" || state == "S2801")*/) {
                        System.out.println("Closing Connection.");
                        con.close();
                        con = null;
                    }
                } catch (RepositoryException re) {
                    System.out.println("Test Failed.");
                    re.printStackTrace();
                    System.exit (1);
                }
            } finally {
            }
        }
    }
}
```

Chapter 8. Virtual Database Engine

Abstract

This chapter covers Remote Table Attachment, Usage of Remote Tables from ODBC, JDBC, ADO.NET, SQLX, XPath/XQuery, Linked Data Views.

8.1. Virtual Database (VDB) Engine

8.1.1. The Need for VDB Engines

Situation Analysis

As computer hardware, network protocols, database engines, applications, application servers, and desktop productivity tools, proliferate the enterprise, integration of disparate applications from disparate vendors is becoming an all too common problem.

Add the emergence of standards based Distributed Computing galvanized by the Internet infrastructure and associated Internet protocols to this picture, and the need for Integration is even higher.

Increasing the industry at large is looking to a new technology deliverable known as Universal Data Access Middleware to address these systems integration pains.

"With Universal Data Access (UDA), customers receive all of the benefits of a high-level and consistent Application Programming Interface (API) that abstracts all the database complexities while providing a capability that can be specified, controlled, and managed on its own to optimize the near universal need of programs for data access".

Source IDC, 1998 Middleware Markets & Trends

At OpenLink Software, it is our opinion that a new genre of UDA middleware called the "Virtual Database", is set to emerge as the dominant UDA middleware solution for addressing the integration challenges as they exist today, and tomorrow. This new UDA middleware format plays the role of a Universal Data Access manager, fusing traditional database functionality and traditional data access middleware functionality into a single independent packaged software solution.

Virtual Database Engines Defined

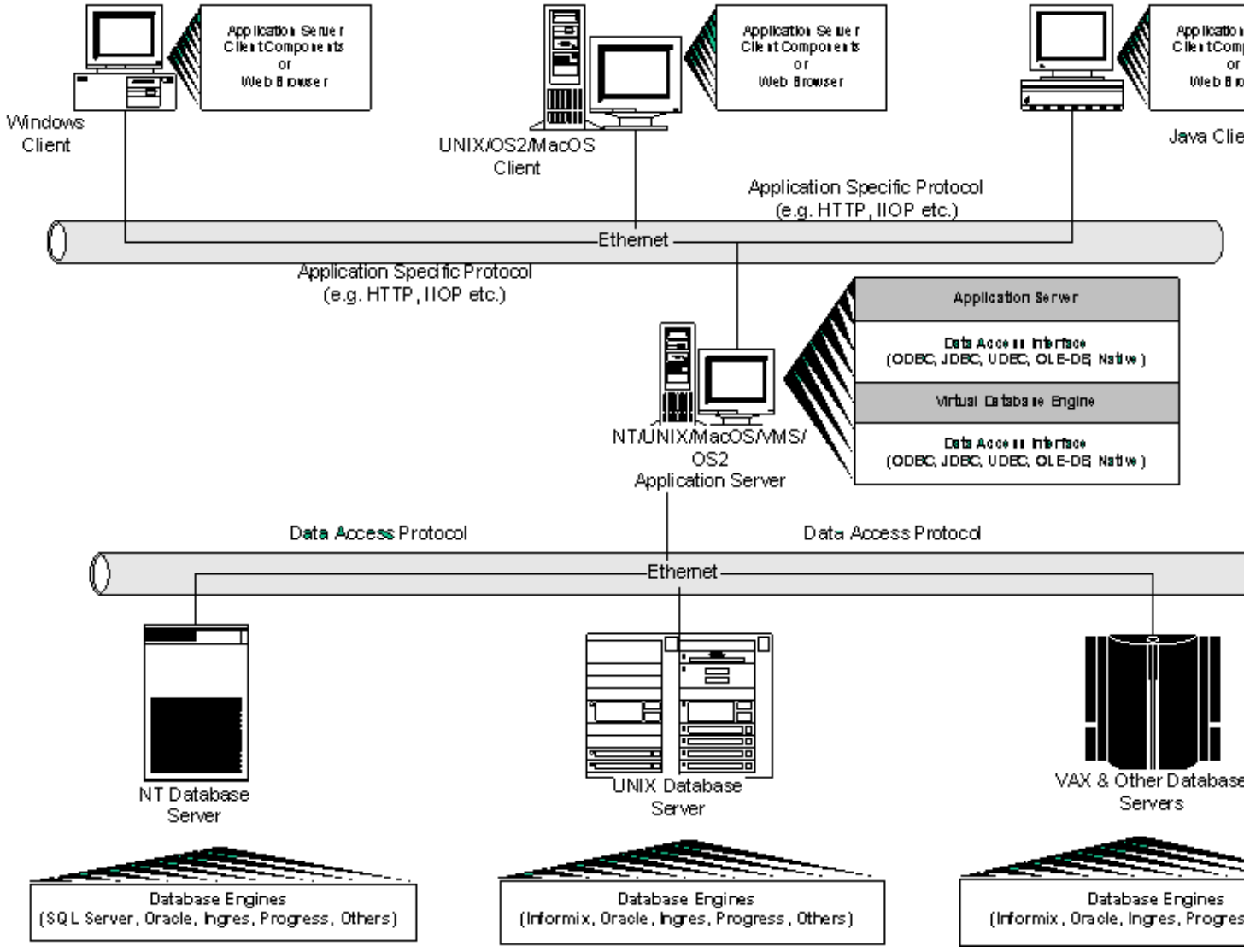
A Virtual Database (VDB) Engine is a UDA middleware format that transparently brings local and or remote heterogeneous databases together using logical database references called Data Source Names (DSN's). A VDB Engine exposes Metadata and Data held within these heterogeneous DSN's to clients applications and services homogeneously.

VDB Engines presume the existence of a number of Database Engines and Data Access Drivers provided by a variety of database vendors within an organization. VDB Engines provide transparent access to these heterogeneous databases via DSN's associated with the relevant data access drivers without exposing end-users or developers to the intricacies of heterogeneous data access.

Data Source Names (DSN's)

A Data Source Name is a logical reference that exposes database to standards compliant or native data access drivers. DSN's provide a flexible naming and binding service for database driven applications developers and end-users alike. Applications no longer need to be inextricably linked to specific database names or specific database engines.

Figure 8.1. Distributed Computing Infrastructure Incorporating A Virtual Database Engine



8.1.2. First Generation Virtual Database Products

Although the strict VDB definition may be new, there are a number of products that have been around for a while that attempt to address VDB issues. The list of such products includes The Microsoft JET Engine, Borland Database Engine (BDE), and IBM DataJoiner.

Microsoft JET

The Microsoft JET Engine lies at the heart of Microsoft Access, it is the piece of technology that allows you to link external and typically remote database tables into your local Access space via ODBC Data Sources. Once this link process has been completed, Access allows you to build Queries, Reports, Forms etc. using these external database tables as though they were Local Access tables. JET can also link to external tables hosted within desktop database engines via native interfaces.

The Microsoft JET Engine services are exposed via Microsoft provided data access interfaces such as: DAO, ADO, and OLE-DB. These interfaces are integral parts of most Microsoft applications, thereby exposing the benefits of the JET VDB transparently.

Borland Database Engine

The Borland Database Engine (BDE) from Inprise like the Microsoft JET Engine also facilitates external table linkage via ODBC Data Sources. The BDE also lets you link to external database tables via native database interfaces and there is no restriction to desktop database engines when you adopt this approach.

Although the BDE has a published set of APIs, it is predominantly used by Inprise applications in very much the same way JET is used by Microsoft applications.

IBM DataJoiner

DataJoiner from IBM provides the ability access heterogeneous data sources via IBM DB/2 Client Application Enablers. It does support ODBC and JDBC as client interfaces and makes use of Native or ODBC based data access for external Data I/O.

8.1.3. VDB Implementation Issues

The essential components that affect the implementation of VDB Engines are, High-Level Data Access Interfaces, Low-Level Data Access Interfaces and Traditional Database Functionality.

High-Level Data Access Interfaces

A VDB Engine's capabilities are exposed via High Level Data Access interfaces. For the purpose of this document, a high level data access interface is an interface utilized predominantly by applications, as opposed to middleware developers for achieving application database independence. A high level data access interface sits atop Low-Level data access interfaces, providing an abstraction layer that serves to simplifying the process of database independent application development.

A number of High Level Data Access standards exist today, the more prevalent being:

Data Access Objects (DAO)
Remote Data Objects (RDO)
ActiveX Data Objects (ADO)
OLE-DB
JavaBlend
InfoBus

It is important to note that low-level Data access interfaces such as ODBC, UDBC, JDBC and OLE-DB transparently serve the high-level interfaces mentioned in the section above. Thus, in most cases VDB vendors will treat ODBC, UDBC, JDBC, and OLE-DB as high-level interfaces by providing VDB data access drivers conforming to these standards as part of the VDB deliverable.

Low-Level Data Access Interfaces

A VDB Engine's data I/O occurs via low-level data access interfaces to underlying database engines or data sources. In recent times the Open Database Connectivity (ODBC) API and the X/Open SQL Call Level Interface (CLI) have emerged as the dominant industry wide Low-Level Data Access Standards. OLE-DB from Microsoft is also emerging as a new low-level data access standard for relational and non-relational data in the Microsoft Component Object Model (COM) world. While JDBC is emerging like wise as the low-level data access standard for the burgeoning Java world.

A VDB may also be a Native Database Interface Client, making use of database engine vendor provided data access interfaces. Native interfaces are based upon Embedded SQL, an older format Low-Level data access interface that preceded the X/Open SQL CLI. It is important to note that ODBC from Microsoft, JDBC from JavaSoft, and UDBC from OpenLink Software are all derived from the X/Open SQL CLI.

Traditional Database Functionality

The degree to which a VDB implements a traditional database engine's functionality has a direct bearing on the intrinsic value of a VDB engine. Traditional database functionality is extensive, but for the purposes of this document, a core set of functionality

common to all commercial database engines has been assembled. The functionality list includes:

Query Language Support. standard syntax for interrogating, manipulating, describing, and securing data contained within a database. Examples include the Structured Query Language (SQL) for relational databases and the Object Query Language (OQL) for Object and Object-Relational Databases.

Query Processor. the mechanism used by a database engine to convert Query Language Statements into actual data retrieval instructions. In addition, this database component is responsible for ensuring Query Language syntax conformance, Query Execution Plan Assembly and Query Fulfillment.

Standard Data Types Support. data contained within a database must be describable using standard data types e.g. Character, Number, Date, etc.

VIEW Support. pre constructed query statements stored within a database, for the purpose of query simplification, or content and structural security.

Stored Procedure Support. Stored Procedures facilitate the embedding of application programming logic within a database. Their pre-compiled nature enhances data access performance by reducing message hops between database servers and database clients.

Scrollable Cursor Support. the process by which the result of a database query (known as a result-set) is traversed. Traversal occurs in either direction, backwards or forwards, using result-set chunks (known as row-sets). Resultset scrolling occurs when database engines exchange data with database clients.

Concurrency Control. the process through which a database engine supports multiple sessions running concurrently, across multiple database users and database client applications without compromising underlying data integrity or introducing quantum increases in application response times.

Transaction Support. ensures that database instructions can be grouped into logical units of execution that are Atomic, Consistent, Isolated from the effect of other units of execution affecting the same underlying data, and Durable.

Transaction Isolation. describes the ability of a database engine to provide transaction process partitioning options called Isolation Levels, that offer different ways of managing the effects of multiple and concurrent transactions affecting the same underlying data.

Distributed Transaction Support. describes the ability to preserve transaction atomicity, consistency, integrity, and durability across database servers hosted on the same or different database server machines within a networked environment. This involves supporting transaction Commits and Rollbacks using a 2-phase commit protocol.

User Definable Type Support. this is how a database engine allows end-users extend its base functionality. This is achieved by providing interfaces that allow end-users create new ways in which a database engine's data is described and manipulated.

Federated Database Support. data access and manipulation across database servers resident on the same machine.

Distributed Database Support. data access, and manipulation across database servers resident on the different machines within a networked environment.

Security. the process by which data, and data transmission is protected using a combination of database and operating system privileges, roles and roles hierarchies. It also includes the ability of a database engine to protect data transmitted to its clients using data encryption.

8.1.4. VDB Engine Components

The prior section outlined the critical implementation issues that affect the development and implementation of VDB Engines. These issues form the basis around which a component based framework for depicting VDB architectures has been derived.

The components that comprise a VDB Engine framework are as follows:

Data Access Drivers

The VDB component that forms the entry point to the VDB Engine's services, these drivers may or may not conform to industry standards. Applications and Services that sit atop a VDB Engine must have their data access layers written to the same Application Programming Interfaces (APIs) implemented by the Data Access Drivers provided by a VDB engine.

Security Manager

The VDB component that is responsible for protecting data and data transmission (using encryption) within the VDB Engine's domain. It is also responsible for managing Application, User, Group, Role and Domain privileges as they relate to the creation, manipulation and destruction of VDB data and metadata.

Query Manager

The VDB component that handles queries presented to it by the VDB Engine's data access drivers. It provides query syntax checking, query execution plan compilation, and query fulfillment services. A query processor is built in conformance to one or more query language specifications, the most notable being the Structured Query Language (SQL) for relational database engines, and the Object Query Language (OQL) for Object-Relational and Object Database engines.

Meta Data Manager

The VDB component that provides the Query Processor with information about the data entities from which the Query Processor's execution plan is derived. Metadata managers are also the components responsible for linking external data sources into the VDB domain and directing the Query Processor to the appropriate Data I/O manager.

Transaction Manager

The Transaction Manager component ensures that transactions are Atomic (clearly distinguishable units), Consistent (thereby preserving integrity of data), Isolated from the effect of other transactions, and Durable (such that the effects of committed transactions survive failure). The Transaction Manager ensures VDB Engines are capable of supporting Online Transaction Processing (OLTP) and Distributed Transaction oriented applications and services. Transaction Managers may be standards based implementing X/Open's XA Resource Manager Specifications. Distributed transaction support is implemented by using a two-phase commit protocol.

Concurrency Manager

The VDB component that ensures client applications and services are capable of opening multiple concurrent sessions that execute data INSERTS, UPDATES and DELETIONS, without implicitly reducing application response times or compromising data integrity. Concurrency control is delivered in one of two formats, Optimistic or Pessimistic depending on the response times desired by VDB client applications or services.

Local Data I/O Manager

VDB Engine's that provide local data storage uses this component for reading and writing data to disk. This is how a VDB provides traditional database engine data storage services.

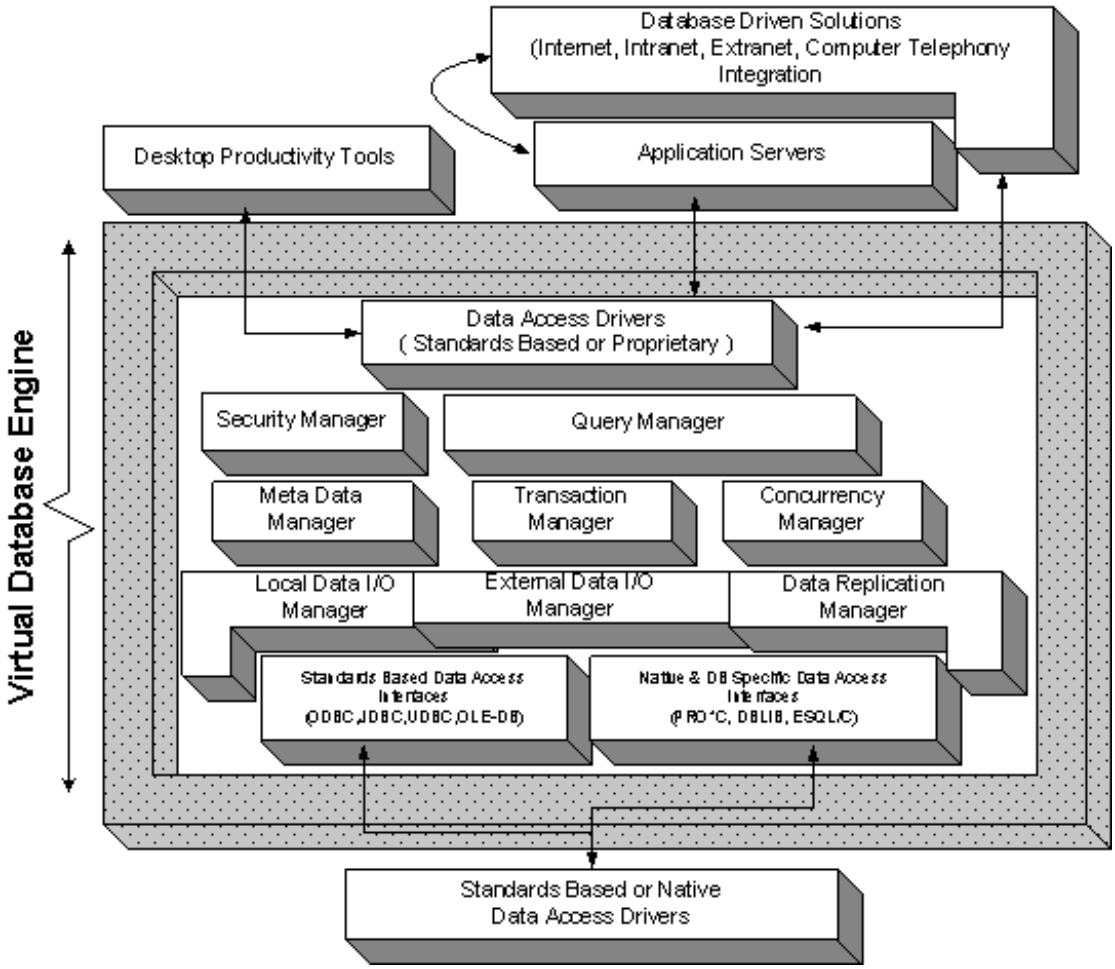
External Data I/O Manager

VDB component that handles data reads and writes to external data sources. The External Data I/O Manager be implemented using standard data access interfaces such as ODBC, JDBC, UDBC, OLE-DB or Native data source interfaces.

Replication Manager

Component that manages data migration and synchronization across two or more VDB servers within a distributed computing environment. This component acts as a data coordinator between the activities of Local Data I/O and External Data I/O Managers across VDB servers. The Replication Manager enables a VDB Engine offer automated bi-directional data, and metadata transformation services across heterogeneous data sources without end-user or developer intervention.

Figure 8.2. Virtual Database Engine Architecture & Components



VDB Implementation Approaches

There are no golden VDB implementation specifications, but the implementation of a VDB has a direct impact the degree to which you realize desired value from the VDB concept as a whole.

The VDB value proposition is simply stated as follows:

"To provide transparent access to heterogeneous data sources, independent of host operating system and underlying database engines ".

VDB implementations can be categorized as follows:

Table 8.1. VDB Implementation Categories

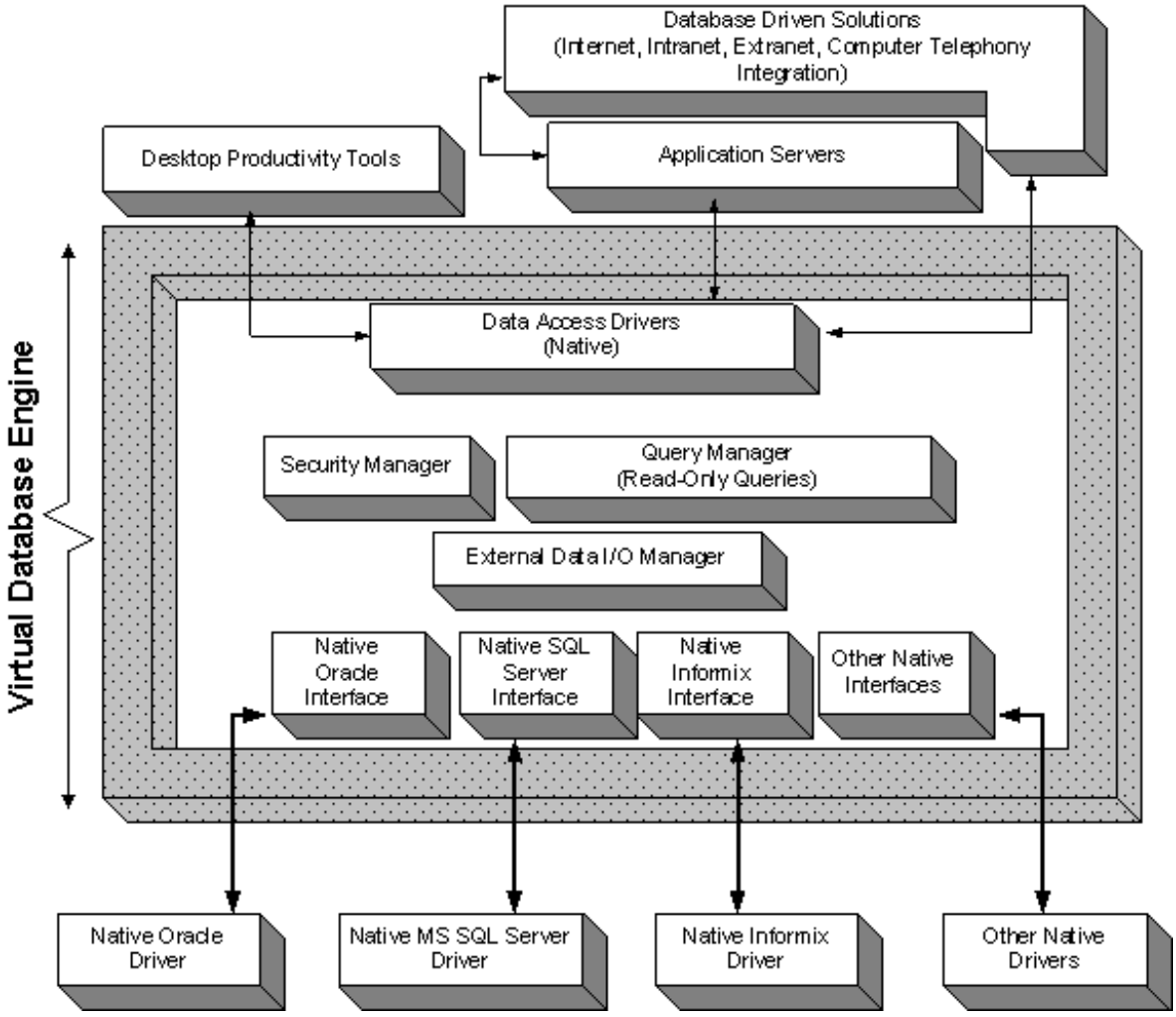
	VDB Data Access Interface	VDB External Data I/O	Traditional Database Functionality
Type 1	Native	Native	Partial
Type 2	Native	Native	Full
Type 3	Native	Standards Based	Partial
Type 4	Native	Standards Based	Full
Type 5	Standards Based	Native	Partial
Type 6	Standards Based	Native	Full
Type 7	Standards Based	Standards Based	Partial
Type 8	Standards Based	Standards Based	Full
Type 9	Standards Based	Standards Based or Native	Partial
Type 10	Standards Based	Standards Based or Native	Full

The sections that follow provide illustrations of the different VDB formats, depicting the components that provide the basis for the categorization used in the table above.

Type 1 VDB Engine

This category of VDB exposes its services to clients via Native and Proprietary high-level data access interfaces. Data I/O is achieved via native, proprietary, and data source specific low-level data access interfaces. This category of VDB does not possess a complete set of traditional database engine components.

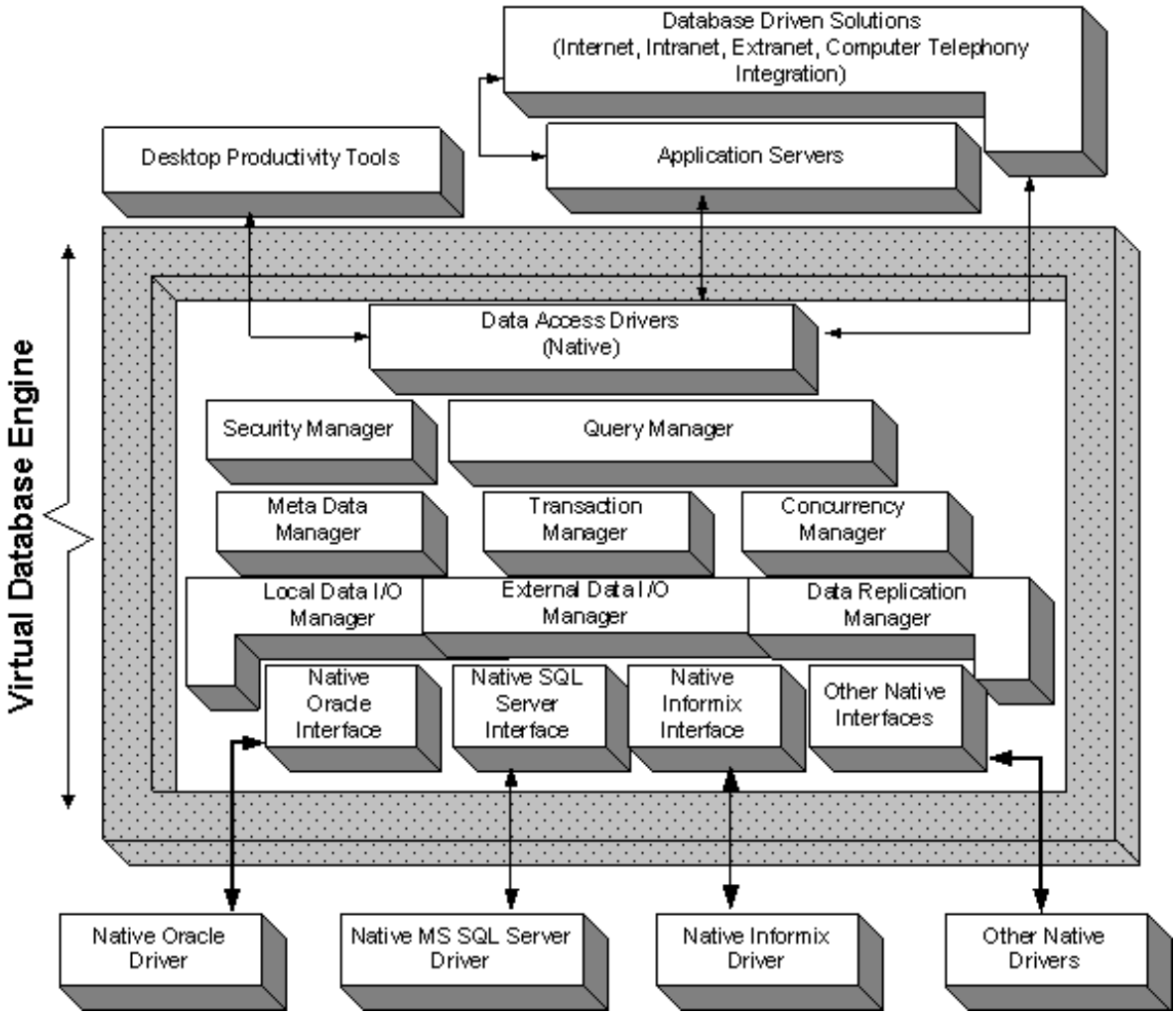
Figure 8.3. Type 1 VDB Engine Architecture



Type 2 VDB Engine

This category of VDB exposes its services to clients via Native and Proprietary high-level data access interfaces. External data I/O is achieved via native, proprietary, and data source specific low-level data access interfaces. This category of VDB possesses a complete set of traditional database engine components.

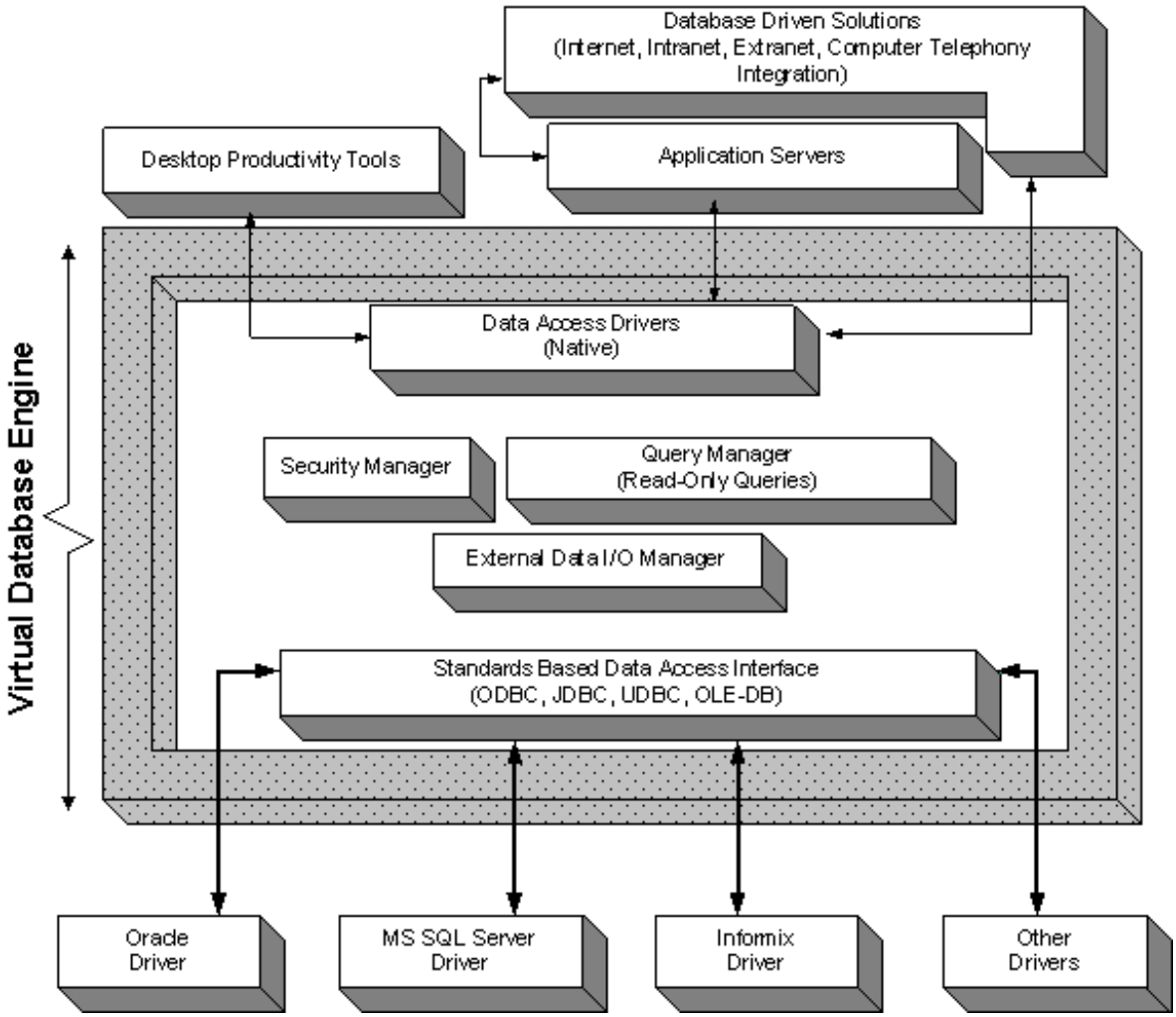
Figure 8.4. Type 2 VDB Engine Architecture



Type 3 VDB Engine

This category of VDB exposes its services to clients via Native and Proprietary high-level data access interfaces. Data I/O is achieved via Open, Standards based, and Database Independent low-level data access interfaces. This category of VDB does not possess a complete set of traditional database engine components.

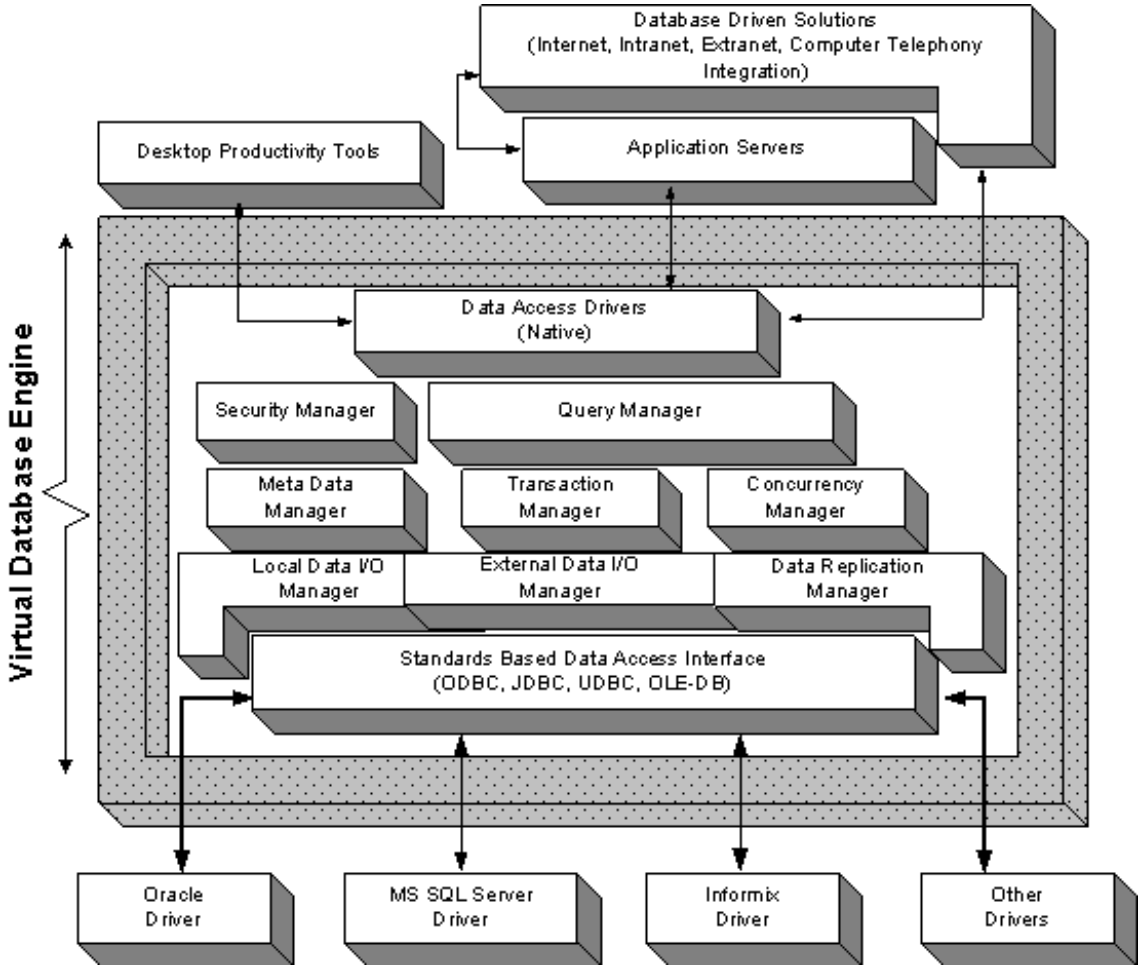
Figure 8.5. Type 3 VDB Engine Architecture



Type 4 VDB Engine

This category of VDB exposes its services to clients via Native and Proprietary high-level data access interfaces. External data I/O is achieved via Open, Standards based, and Database Independent low-level data access interfaces. This category of VDB possesses a complete set of traditional database engine components.

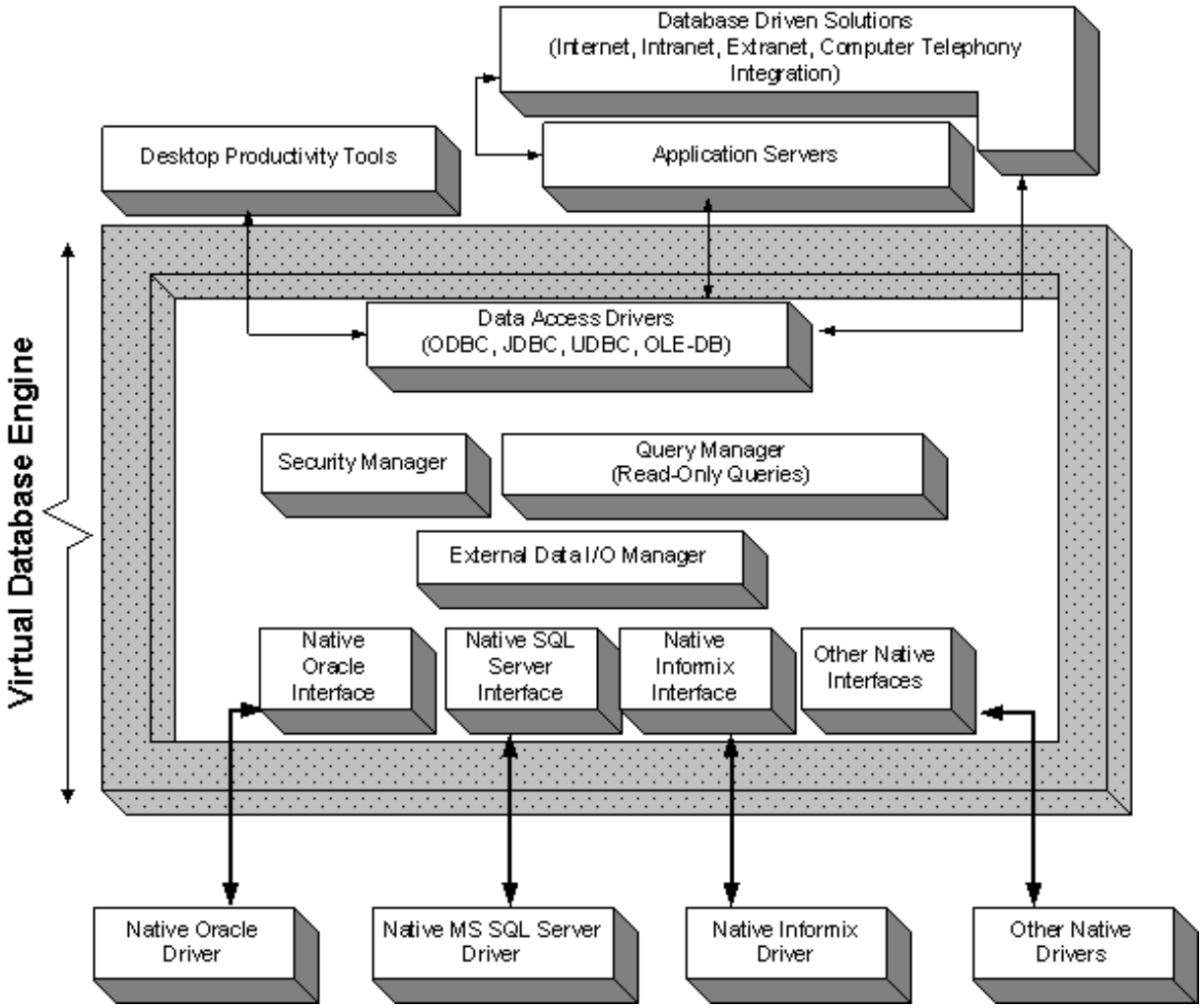
Figure 8.6. Type 4 VDB Architecture



Type 5 VDB Engine

This category of VDB exposes its services to clients via open and standards based high-level data access Interfaces. Data I/O is achieved via native, proprietary, and data source specific low-level interfaces. This category of VDB does not possess a complete set of traditional database engine components.

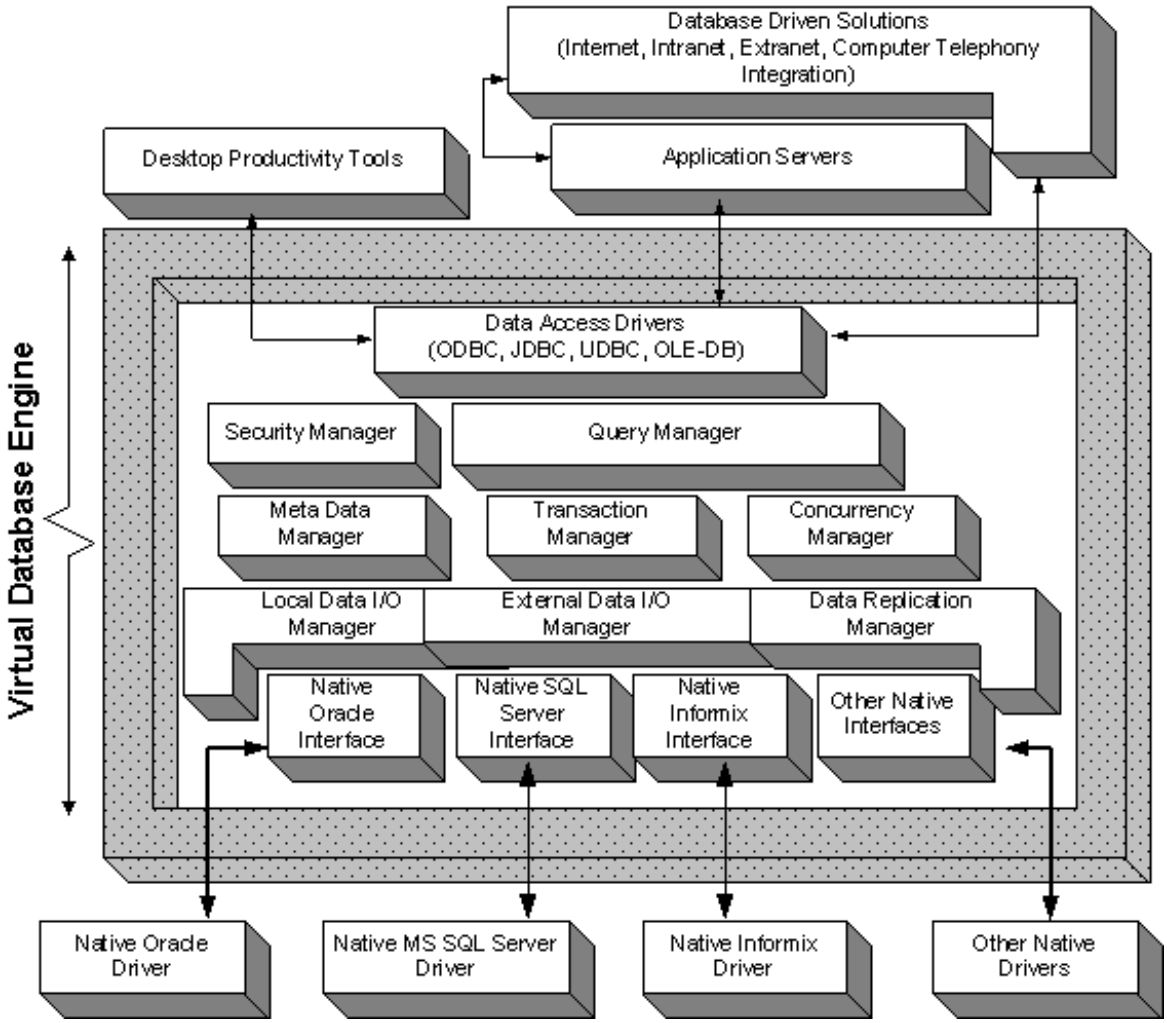
Figure 8.7. Type 5 VDB Engine Architecture



Type 6 VDB Engine

This category of VDB exposes its services to clients via open, standards based, high and low-level Interfaces. External data I/O is achieved via native, proprietary, and data source specific low-level interfaces. This category of VDB possesses a complete set of traditional database engine components.

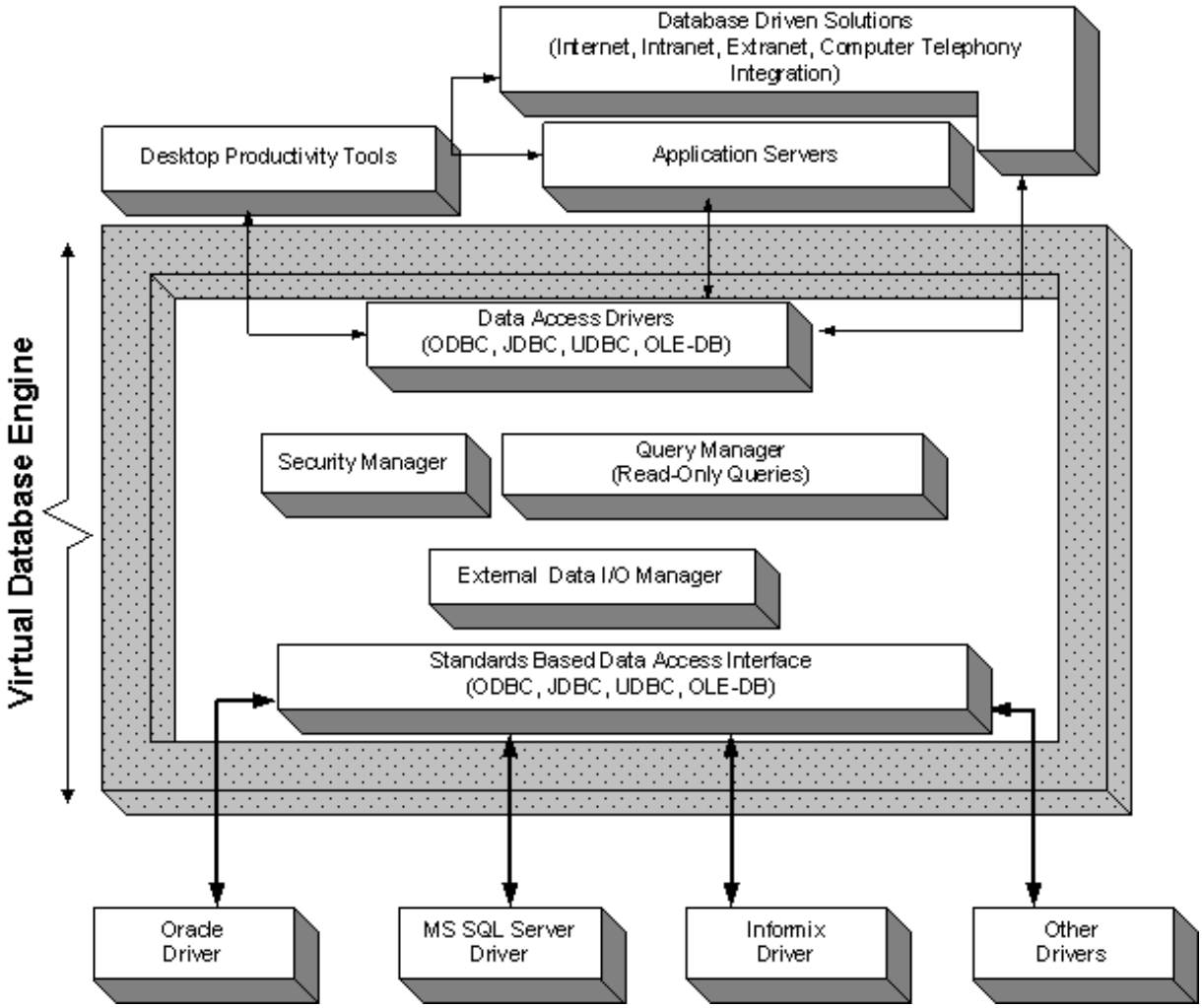
Figure 8.8. Type 6 - VDB Engine Architecture



Type 7 VDB Engine

This category of VDB exposes its services via open, standards based high-level data access interfaces. Data I/O is achieved via Open, Standards based, and Database Independent low-level data access interfaces. This category of VDB does not possess a complete set of traditional database engine components.

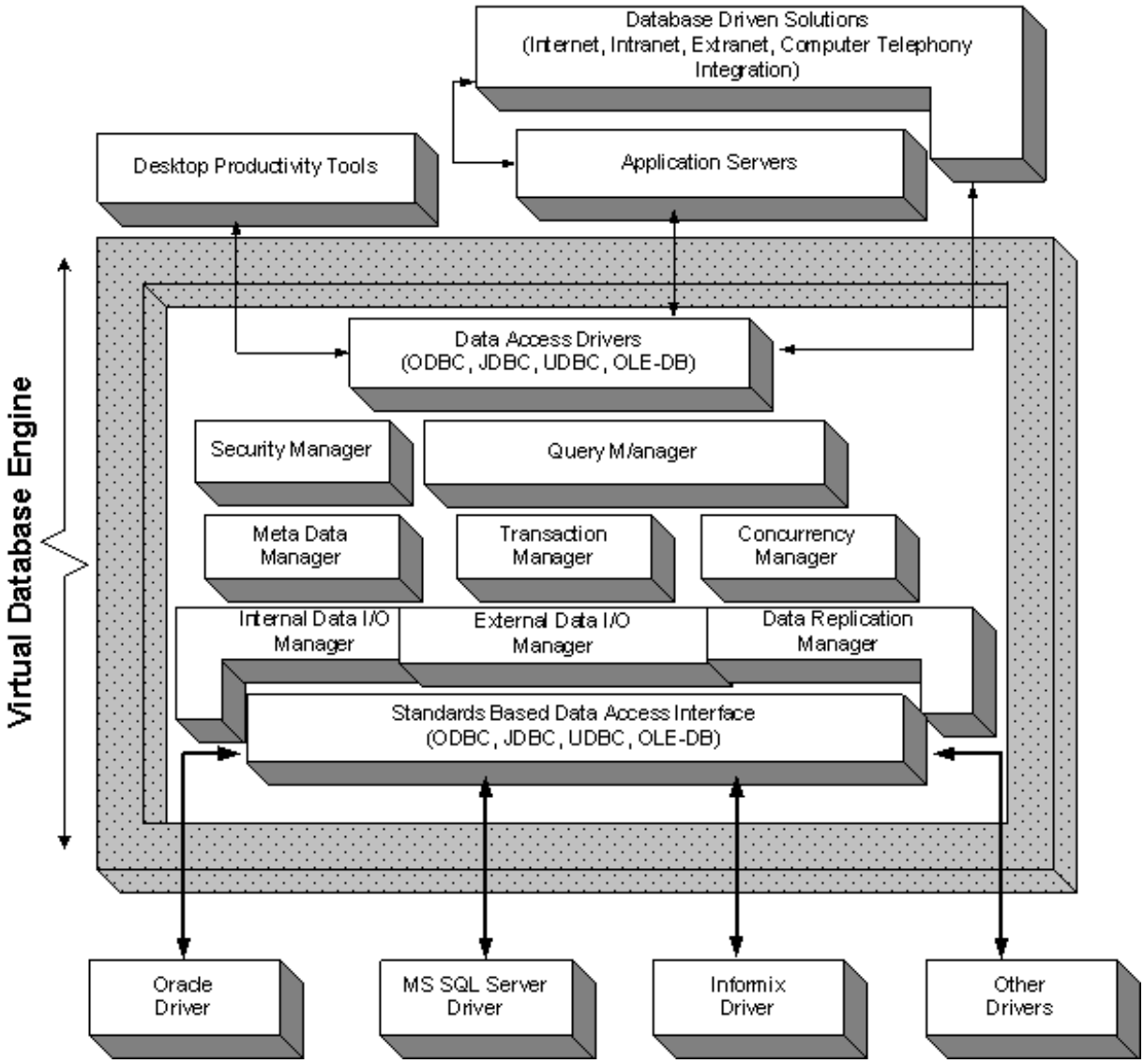
Figure 8.9. Type 7 VDB Engine Architecture



Type 8 VDB Engine

This category of VDB exposes its services via open, standards based, high and low-level interfaces. External data I/O is achieved via Open, Standards based, and Database Independent low-level data access interfaces. This category of VDB does possess a complete set of traditional database engine components.

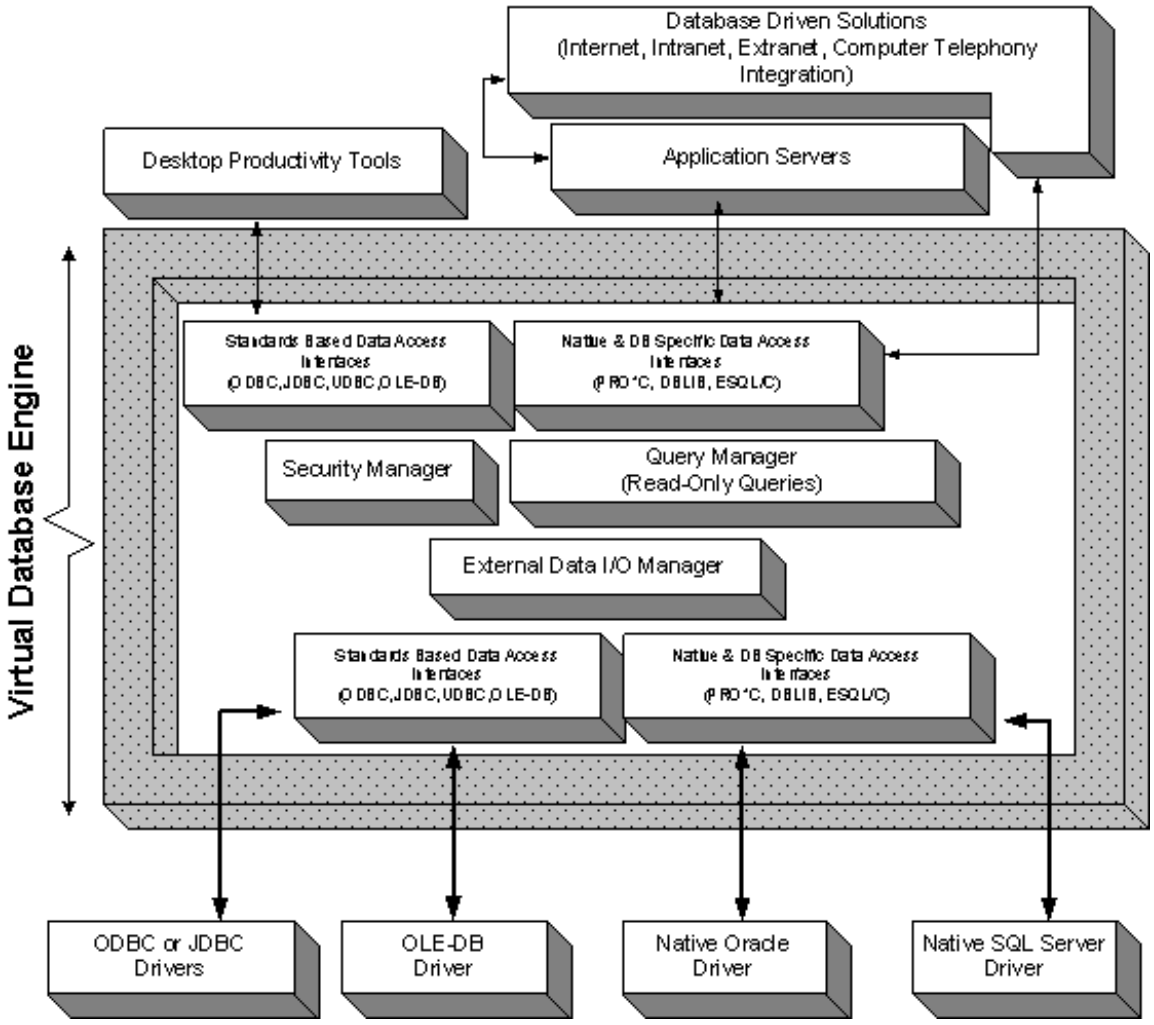
Figure 8.10. Type 8 VDB Engine Architecture



Type 9 VDB Engine

This category of VDB exposes its services via Open, Standards based, high-level data access interfaces. Data I/O is achieved by using either Open, Standards based, and Database Independent low-level data access interfaces or Native, Proprietary, and Database Specific low-level data access interfaces. This category of VDB does not possess a complete set of traditional database engine components.

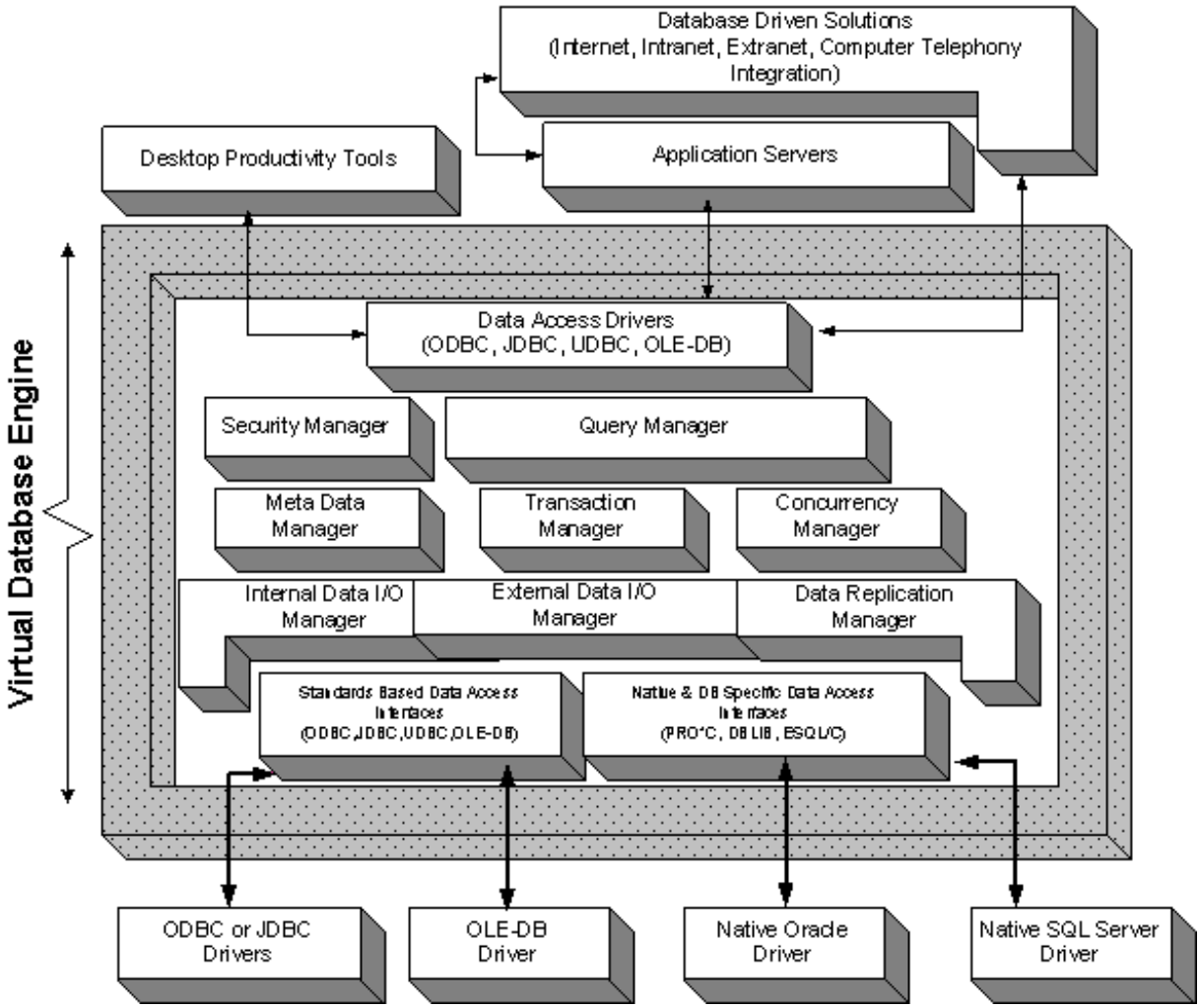
Figure 8.11. Type 9 VDB Architecture



Type 10 VDB Engine

This category of VDB exposes its services via Open, Standards based, high-level data access interfaces. External data I/O is achieved by using either Open, Standards based, and Database Independent low-level data access interfaces or Native, Proprietary, and Database Specific low-level data access interfaces. This category of VDB possesses a complete set of traditional database engine components.

Figure 8.12. Type 10 VDB Architecture



8.2. Using Microsoft Entity Frameworks to Access Oracle Schema Objects with Virtuoso

Abstract

This section details the steps required to provide Microsoft Entity Framework access to Oracle Schema Objects using the OpenLink Virtuoso Universal Server. This is achieved by linking the required Oracle Schema objects into Virtuoso using its built in Virtual Database engine, and then using the Virtuoso ADO.NET Entity Framework provider to query the remote Oracle Schema objects linked into the Virtuoso Server.

Prerequisites. The following prerequisites must be in place for this solution to be possible.

Oracle DBMS. An Oracle DBMS hosting the required Schema Objects needs to be available. In this section the *Oracle Human Resources* sample database will be used to demonstrate the process.

An Oracle DBMS hosting the required Schema Objects needs to be available. In this section the *Oracle Human Resources* sample database will be used to demonstrate the process.

ODBC Driver for Oracle. An Oracle ODBC Driver is required for Linking the Oracle Schema Objects into the Virtuoso Server. The OpenLink *ODBC Driver for Oracle* will be used in this section, for which a functional ODBC Data source name of "ora10ma" will be assumed to exist on the machine hosting the Virtuoso Server.

An Oracle ODBC Driver is required for Linking the Oracle Schema Objects into the Virtuoso Server. The OpenLink *ODBC Driver for Oracle* will be used in this section, for which a functional ODBC Data source name of "ora10ma" will be assumed to exist on the machine hosting the Virtuoso Server.

Virtuoso Universal Server. An Virtuoso installation including the Virtuoso Universal Server and ADO.NET Entity Framework

Provider is required. The Virtuoso components used must be Release 5.10.x or above, this being the minimum version containing support for Microsoft Entity Frameworks

An Virtuoso installation including the Virtuoso Universal Server and ADO.NET Entity Framework Provider is required. The Virtuoso components used must be Release 5.10.x or above, this being the minimum version containing support for Microsoft Entity Frameworks

Microsoft Visual Studio 2008 SP1. Microsoft Visual Studio 2008 Service Pack 1 is required, this being the only version containing the necessary Entity Framework support available at the time of writing.

Microsoft Visual Studio 2008 Service Pack 1 is required, this being the only version containing the necessary Entity Framework support available at the time of writing.

Tasks

Ensure Oracle Primary Keys (PKs) are not nullable. The Visual Studio 2008 Entity Data Model (EDM) requires that all primary keys are *NOT* Nullable, and will fail to generate an EDM if any are. Thus ensure any tables to be used are defined as not nullable in the Oracle database schema before attempting to generate an EDM. In the case of the Oracle Human Resources database all tables are not nullable, thus this should not be an issue in this case.

The Visual Studio 2008 Entity Data Model (EDM) requires that all primary keys are *NOT* Nullable, and will fail to generate an EDM if any are. Thus ensure any tables to be used are defined as not nullable in the Oracle database schema before attempting to generate an EDM. In the case of the Oracle Human Resources database all tables are not nullable, thus this should not be an issue in this case.

8.2.1. Install and configure OpenLink ODBC Driver for Oracle

The Virtuoso Virtual Database engine uses ODBC as the connectivity mechanism for linking remote database objects into its local schema. Thus an Oracle ODBC Driver must be available with a suitably configured DSN for connecting to the target database. The OpenLink ODBC Drivers for Oracle have been used in this section, although in theory any Oracle ODBC Driver can be used.

Installation instructions for the OpenLink ODBC Driver for Oracle are available from:

- Product Installation & Basic Configuration (ODBC)

8.2.2. Install and configure OpenLink Virtuoso Universal Server

- Install and configure OpenLink Virtuoso Universal Server

8.2.3. Linking Oracle tables into OpenLink Virtuoso

1. Start the Virtuoso Web User Interface

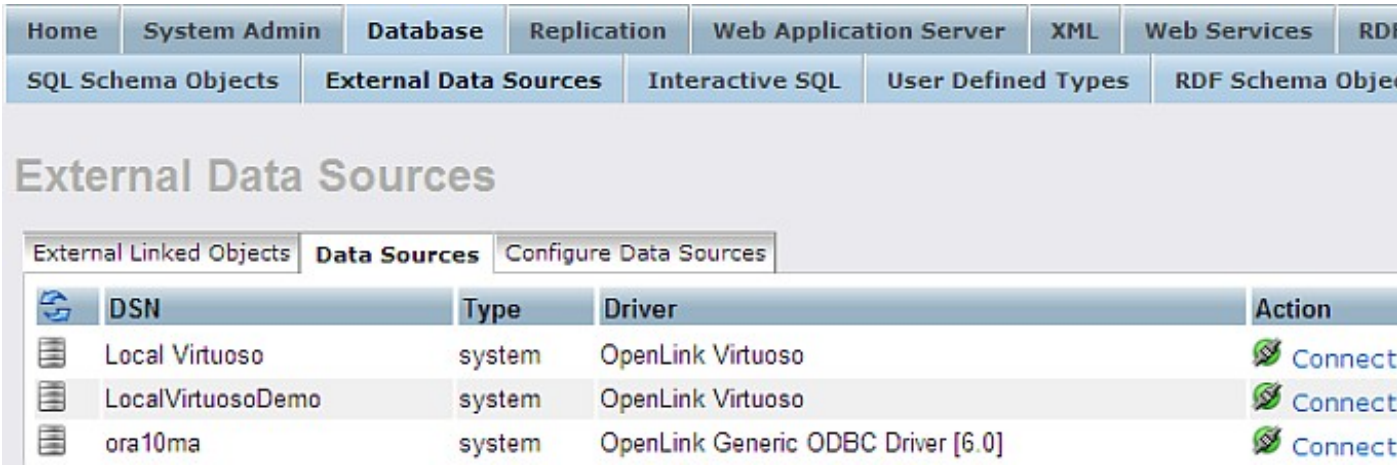
   Figure 8.13. Start

2. Select the "Conductor" link in the Left Frame and Login as the "dba" user.

Figure 8.14. Conductor

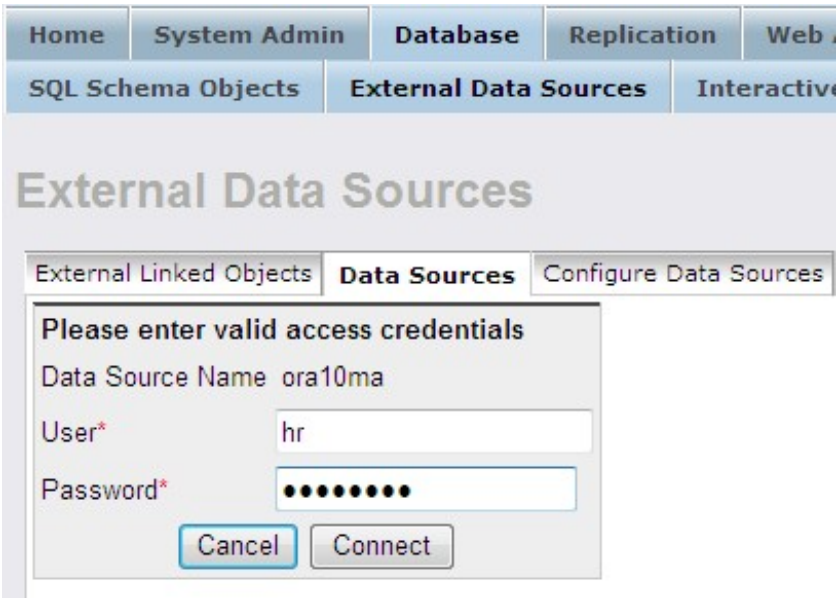
3. Select the "Databases" -> "External Data Source" -> "Data Sources" tab menu items

Figure 8.15. Databases



4. Select the "Connect" button for the "ora10ma" Oracle DSN.

Figure 8.16. Connect

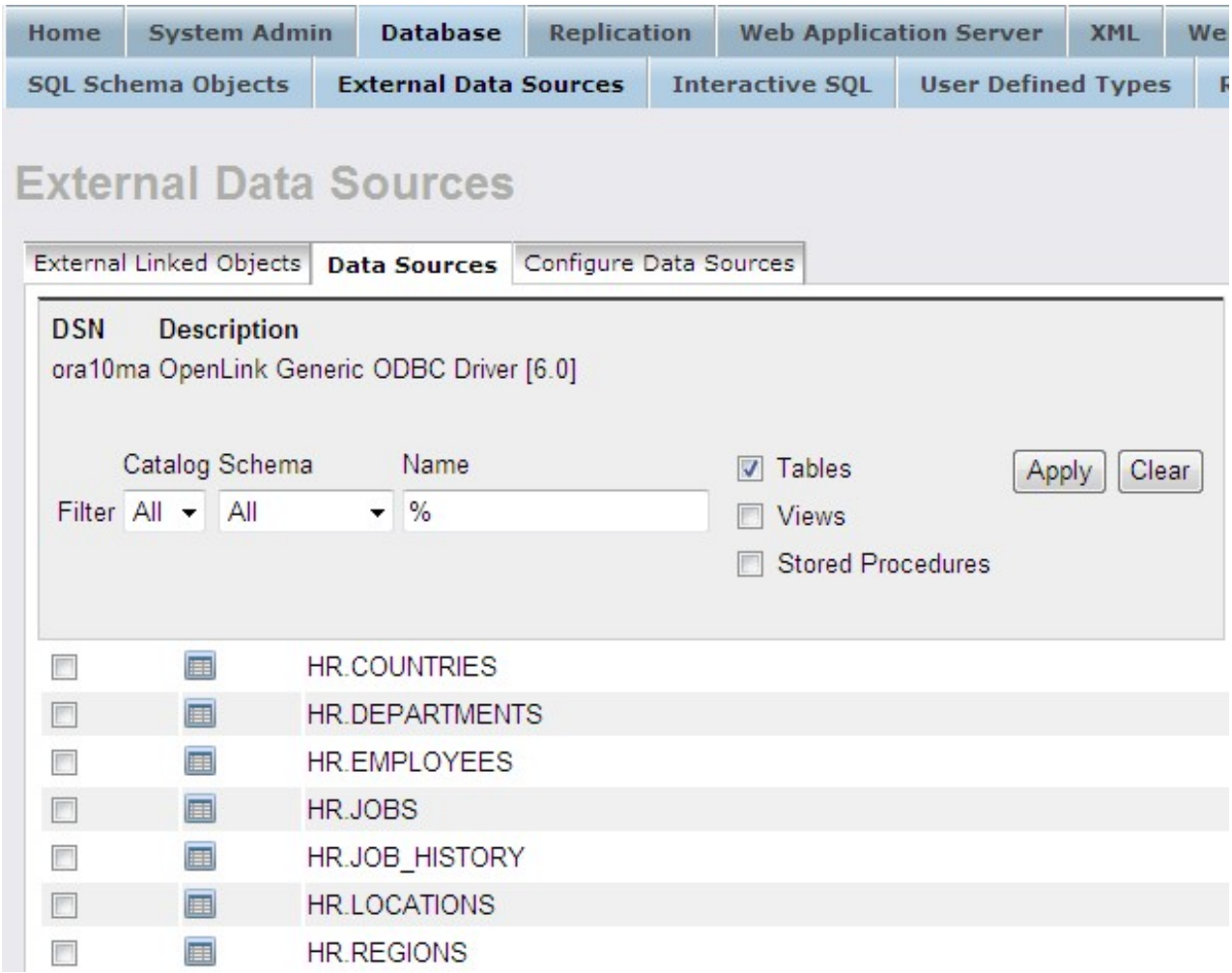


5. On successful connection Select the "Link Objects" button to obtain a list of available tables

Figure 8.17. Link Objects

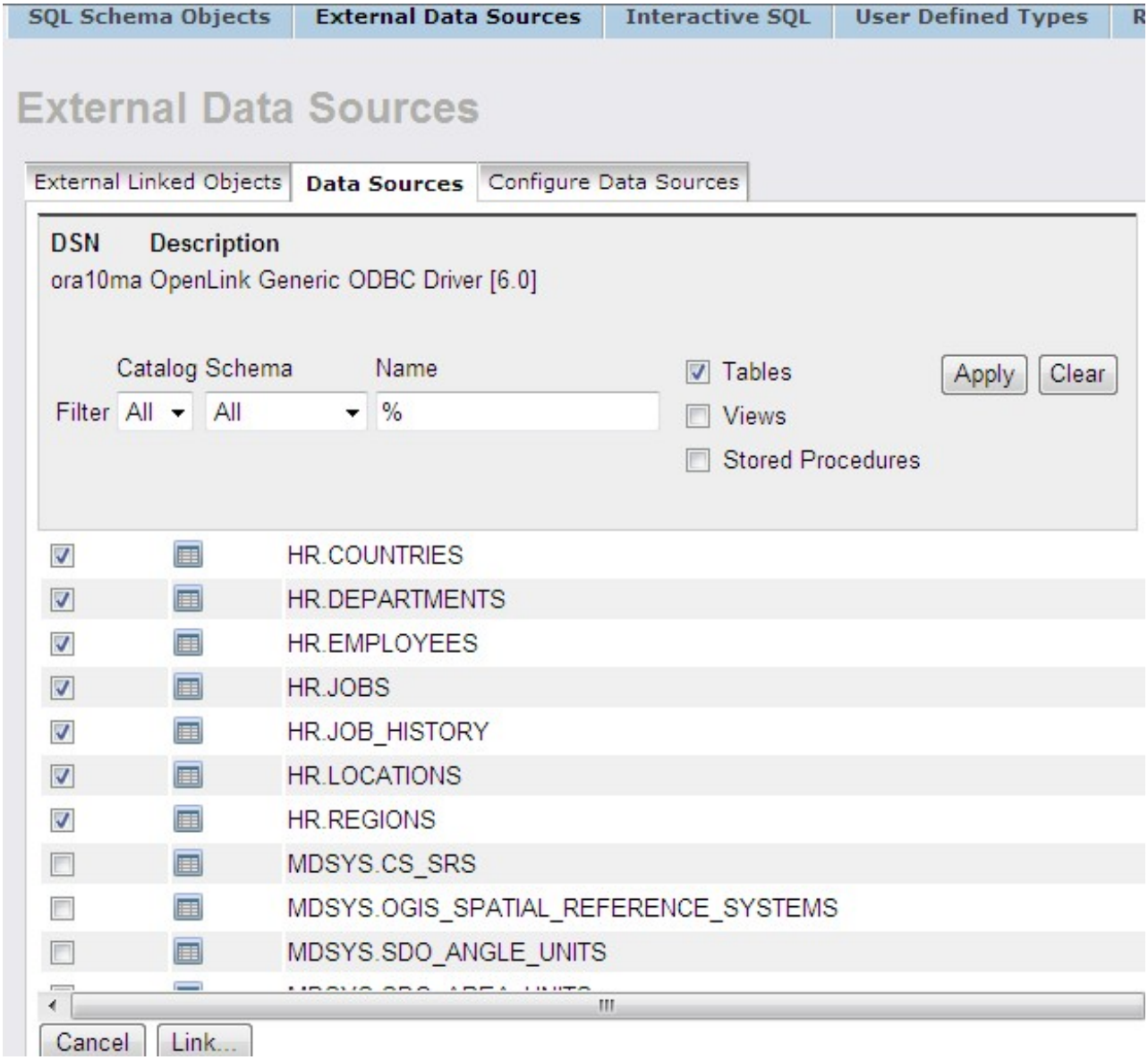
6. Select all the tables that are part of the "HR" catalog.

Figure 8.18. Select all tables



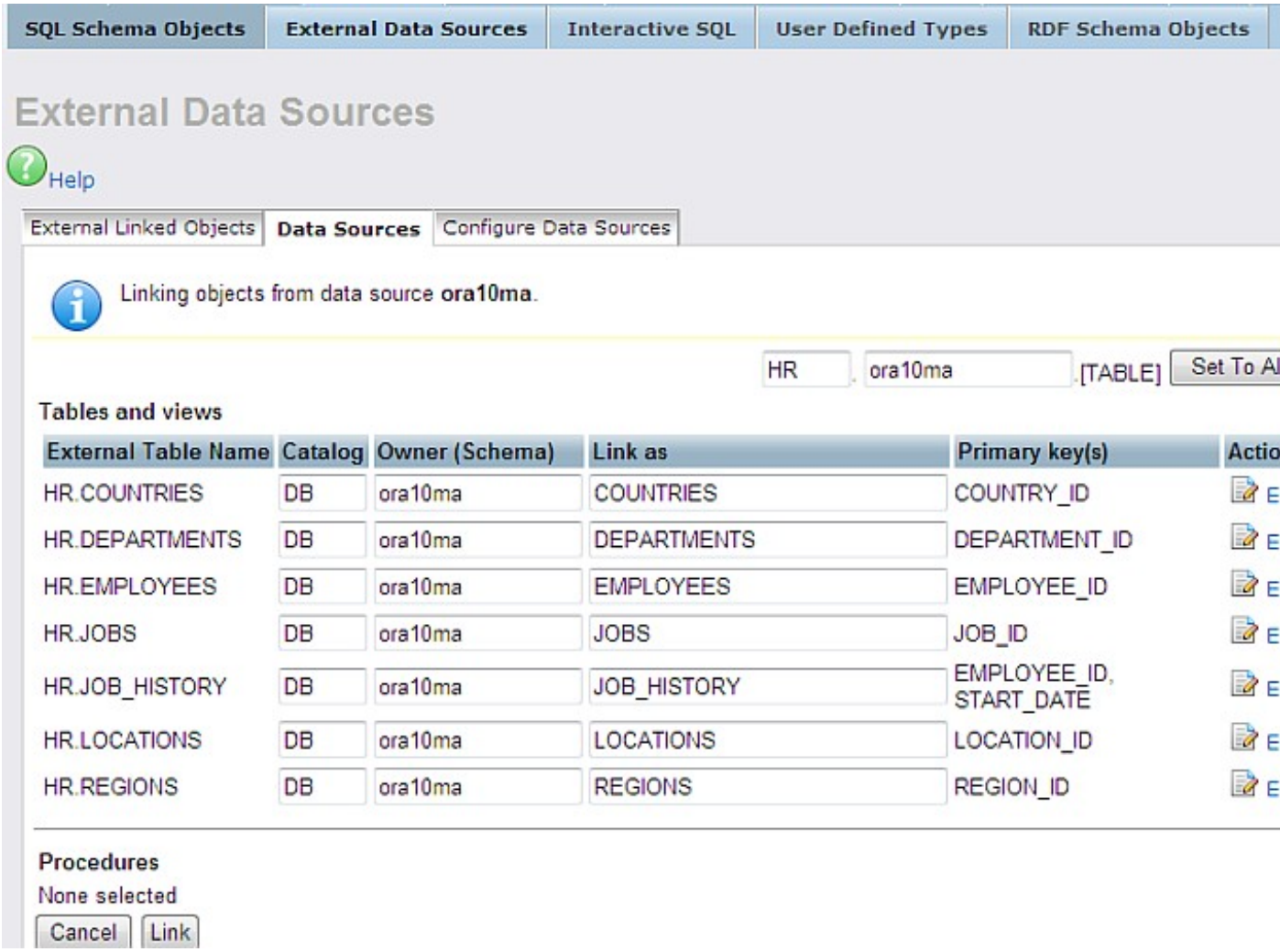
7. Change the Catalog for each table to be "HR" using the "Set All" button.

Figure 8.19. Catalog



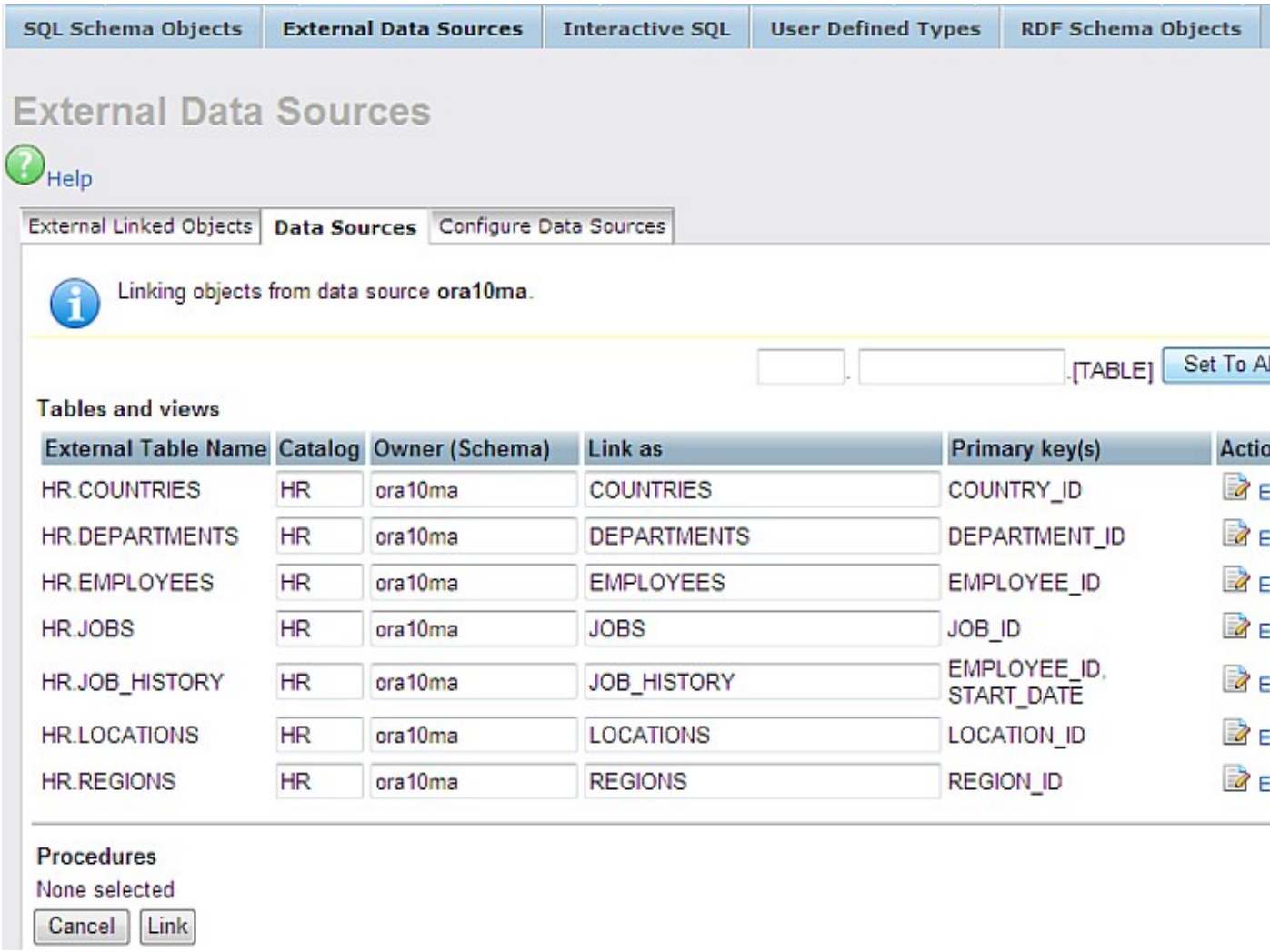
8. Ensure a primary key is assigned to all table that are to be used in the EDM generation phase.

Figure 8.20. Primary Key



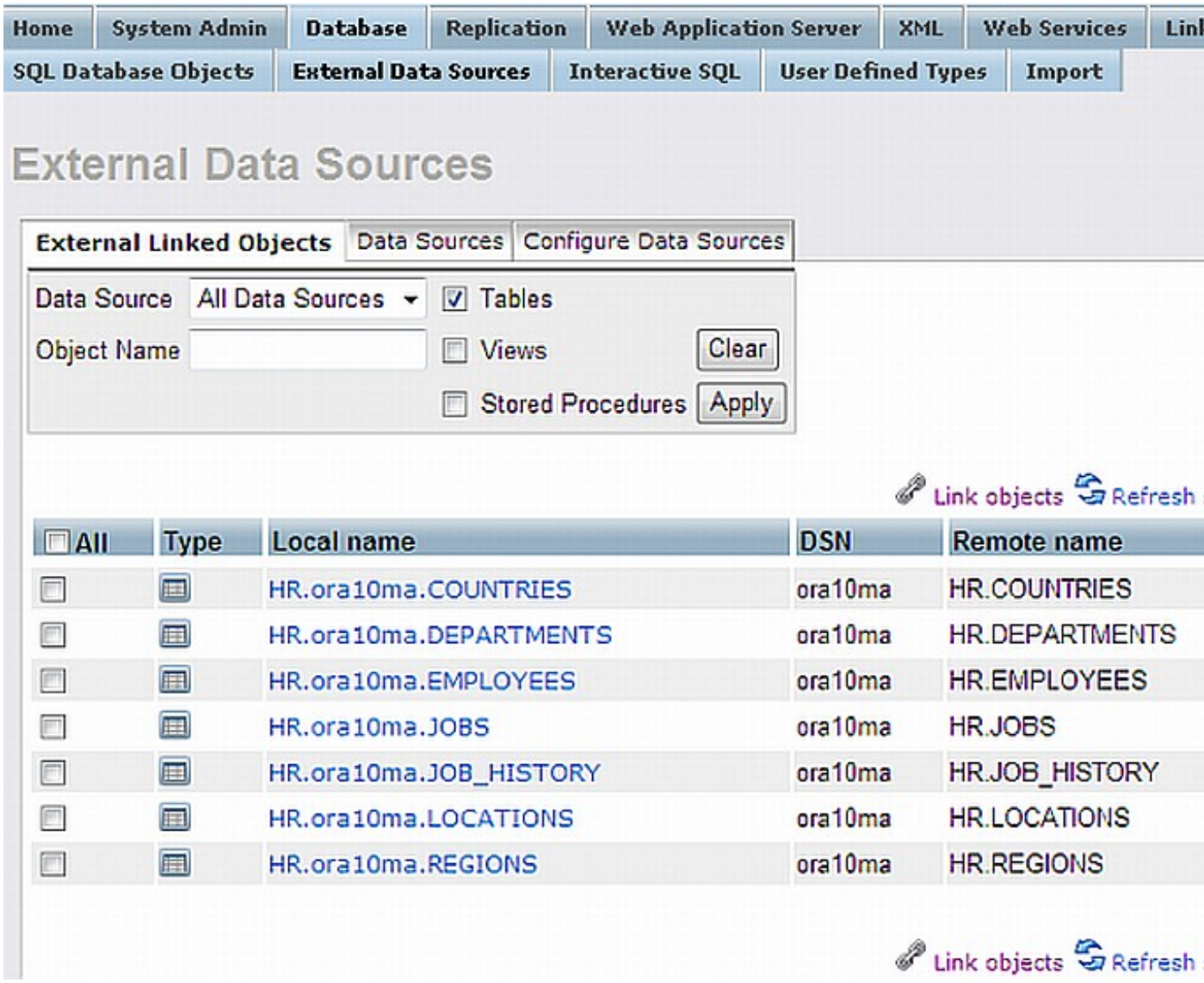
9. Select the "Link" button to link the selected tables into Virtuoso

Figure 8.21. "Link" button



10. On completion of the Link process the tables will be displayed in the "External Linked Objects" tab.

Figure 8.22. Completion

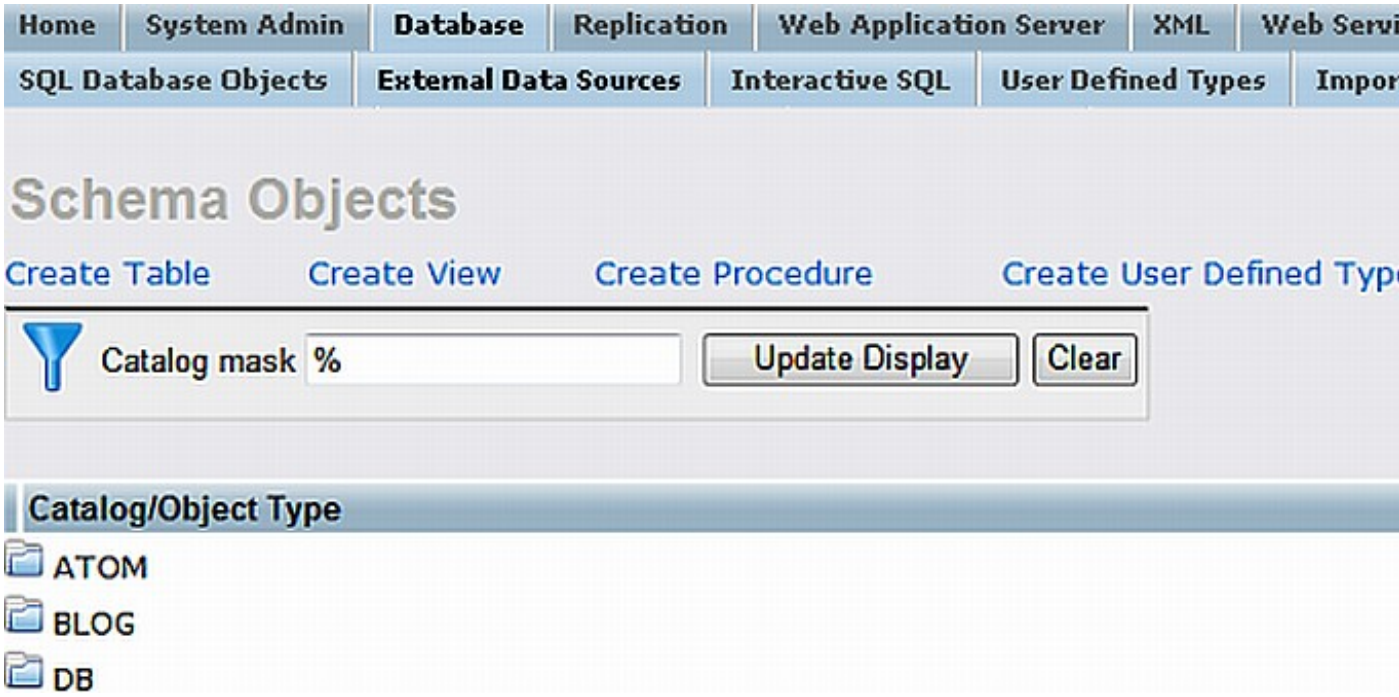


11. The linked tables can be queried by clicking on the hyperlink in the "Local Name" column of the "External Linked Objects" tab above, which loads the Virtuoso "Interactive SQL" interface with the required SQL "Select" for retrieving the remote table data . We shall use the "HR.ora10ma.COUNTRIES" table to demonstrate this.

Figure 8.23. Querying

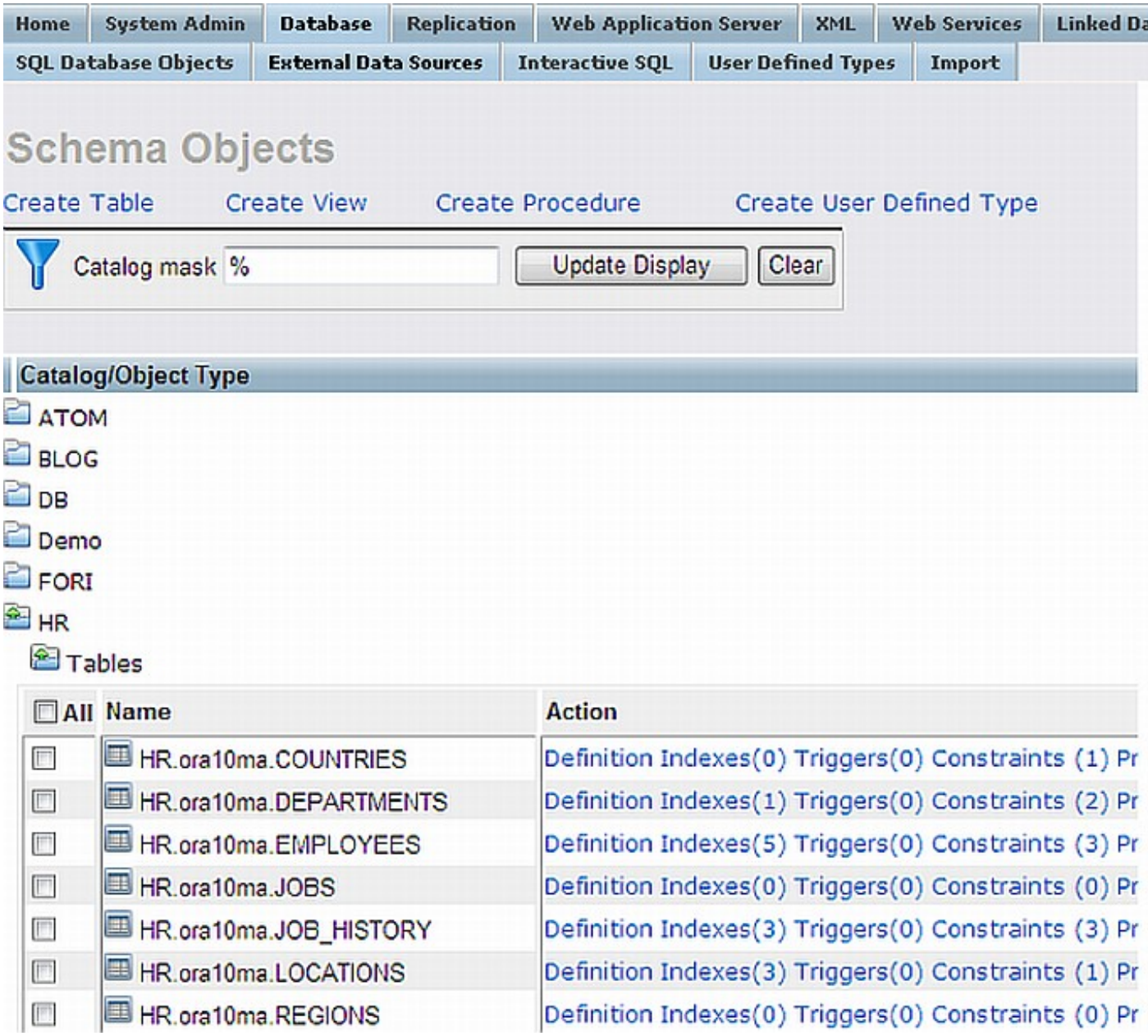
12. Then click the "Execute" button to run the query and retrieve the results from the remote table.

Figure 8.24. Execute



13. The tables can also be viewed as part of the Virtuoso "SQL Schema Objects" under the "HR" catalog name.

Figure 8.25. SQL Schema Objects



The Link process is now complete enabling the tables to be queried as if part of the Virtuoso Schema.

8.2.4. Creating EDM in Visual Studio 2008

The following steps can be used to create an Entity Data Model (EDM) for the Oracle Human Resources database:

1. Launch the Visual Studio 2008 SP1 IDE.

Figure 8.26. Visual Studio 2008 SP1 IDE

2. Create a

   Web Application

   project by going to the

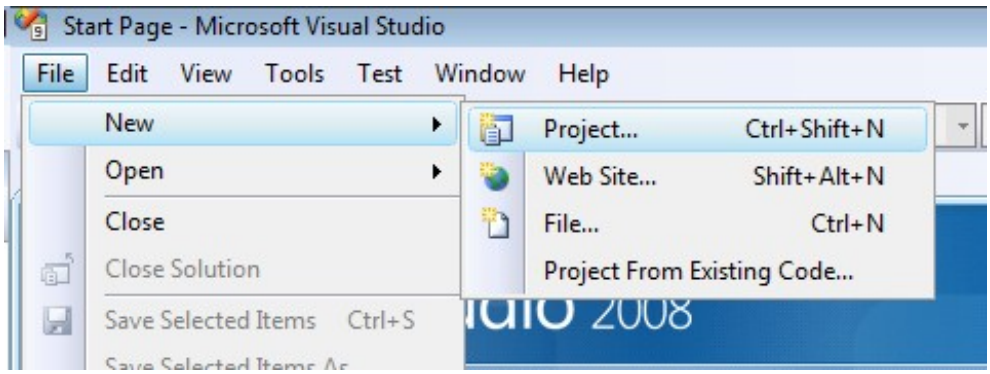
   File

   menu in Visual Studio and choosing

   New Project

   .

   Figure 8.27. Web Application



3. When the New Project window appears, choose either

   Visual Basic

   or

   Visual C#

   as your programming language.
4. Within the language category, click on

   Web

   and select

   ASP.NET Web Application

from the right-hand panel.

5. Choose a name for the project, for example

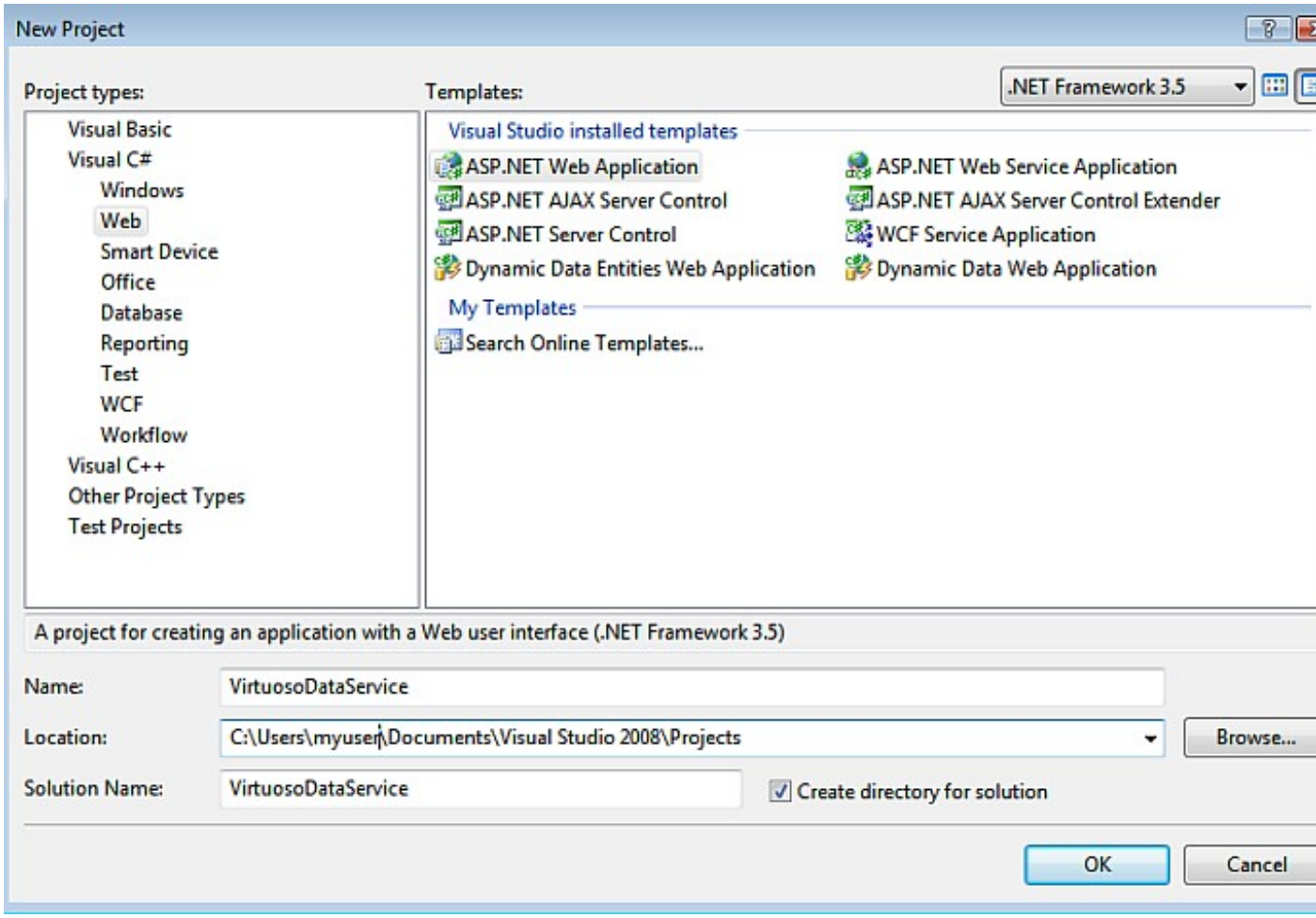
   VirtuosoDataService

   , and click

   OK

   .

Figure 8.28. name for the project

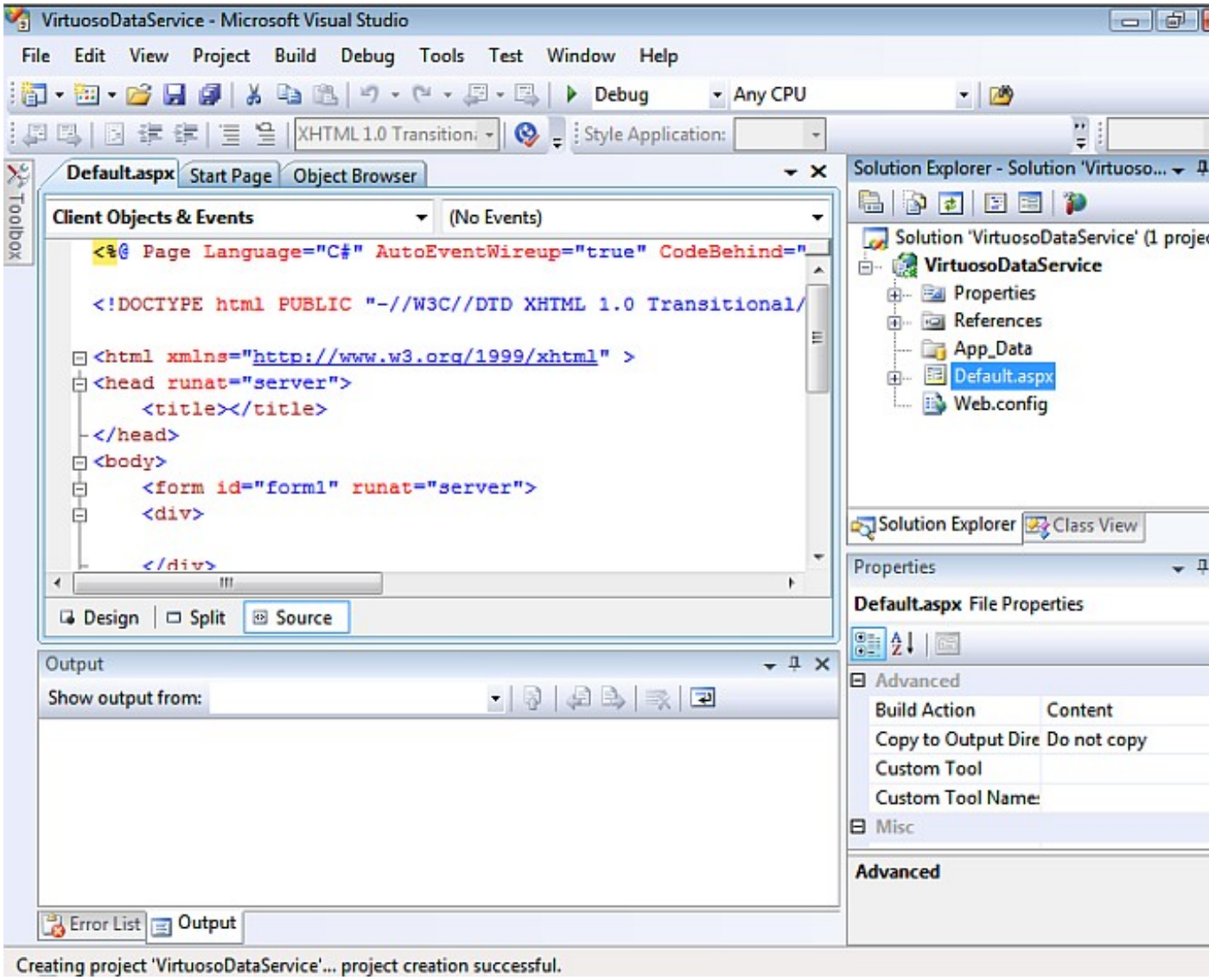


6. This will create a new project called

   VirtuosoDataService

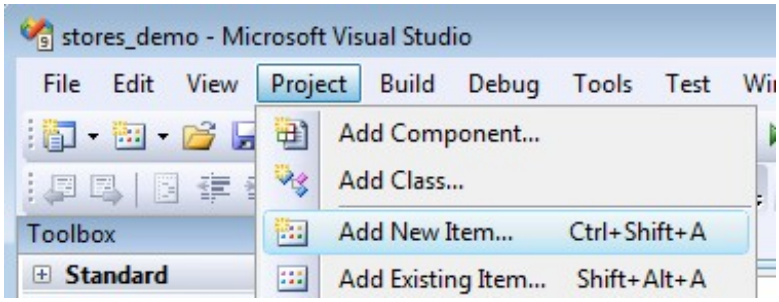
   .

Figure 8.29. create a new project



7. Select the Project -> Add New Item menu option.

Figure 8.30. VirtuosoDataService



8. The

Add New Item

dialog will appear. Choose the

ADO.NET Entity Data Model

template. Give it the name

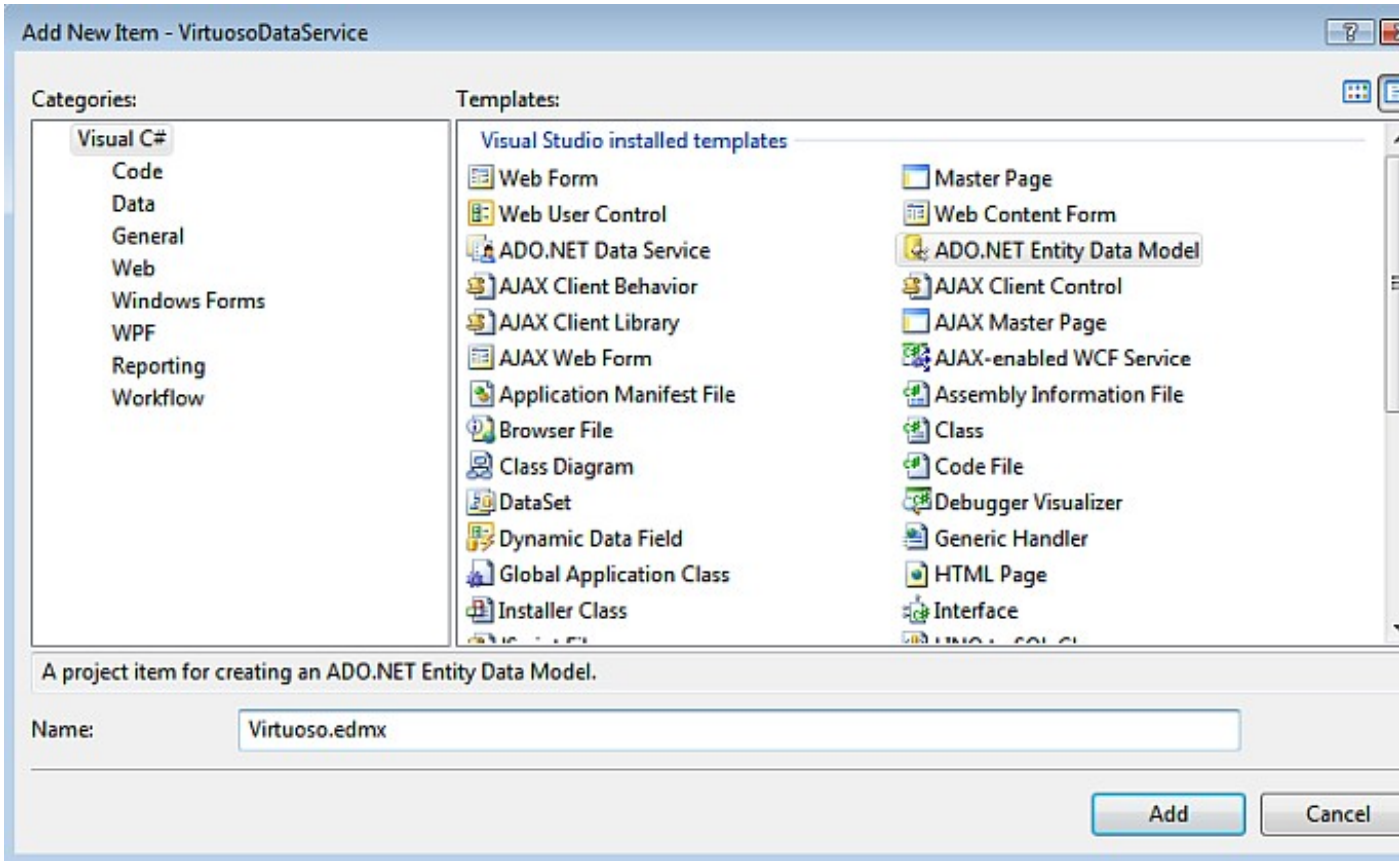
Virtuoso.edmx

and click

Add

to start the creation of the ADO.Net Entity Data Model.

Figure 8.31. Add New Item



9. In the

Entity Data Model Wizard

dialog

Choose Model Contents

page select the

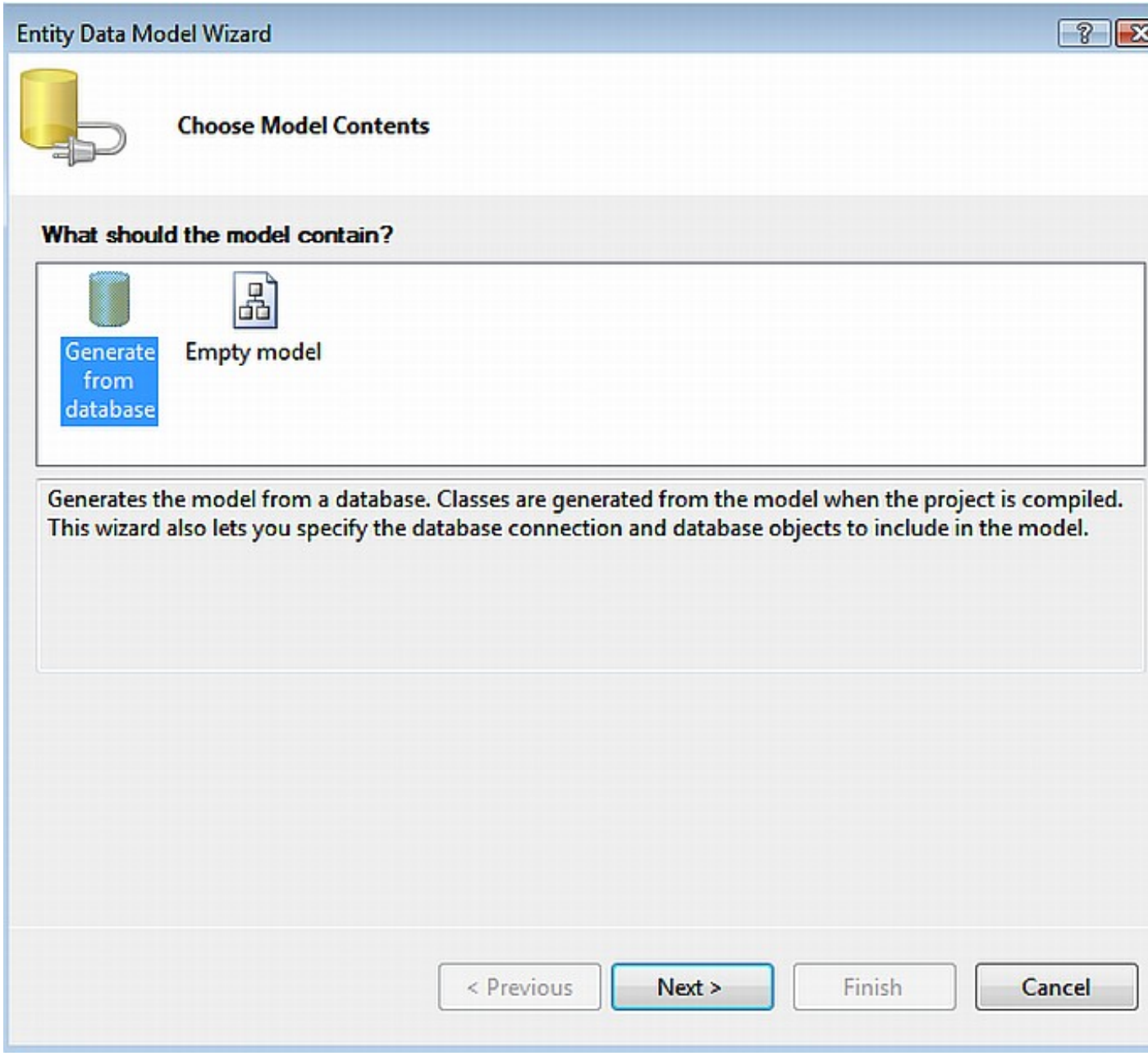
Generate from Database

model type and click

Next

.

Figure 8.32. Choose Model Contents



10. In the

    Entity Data Model Wizard

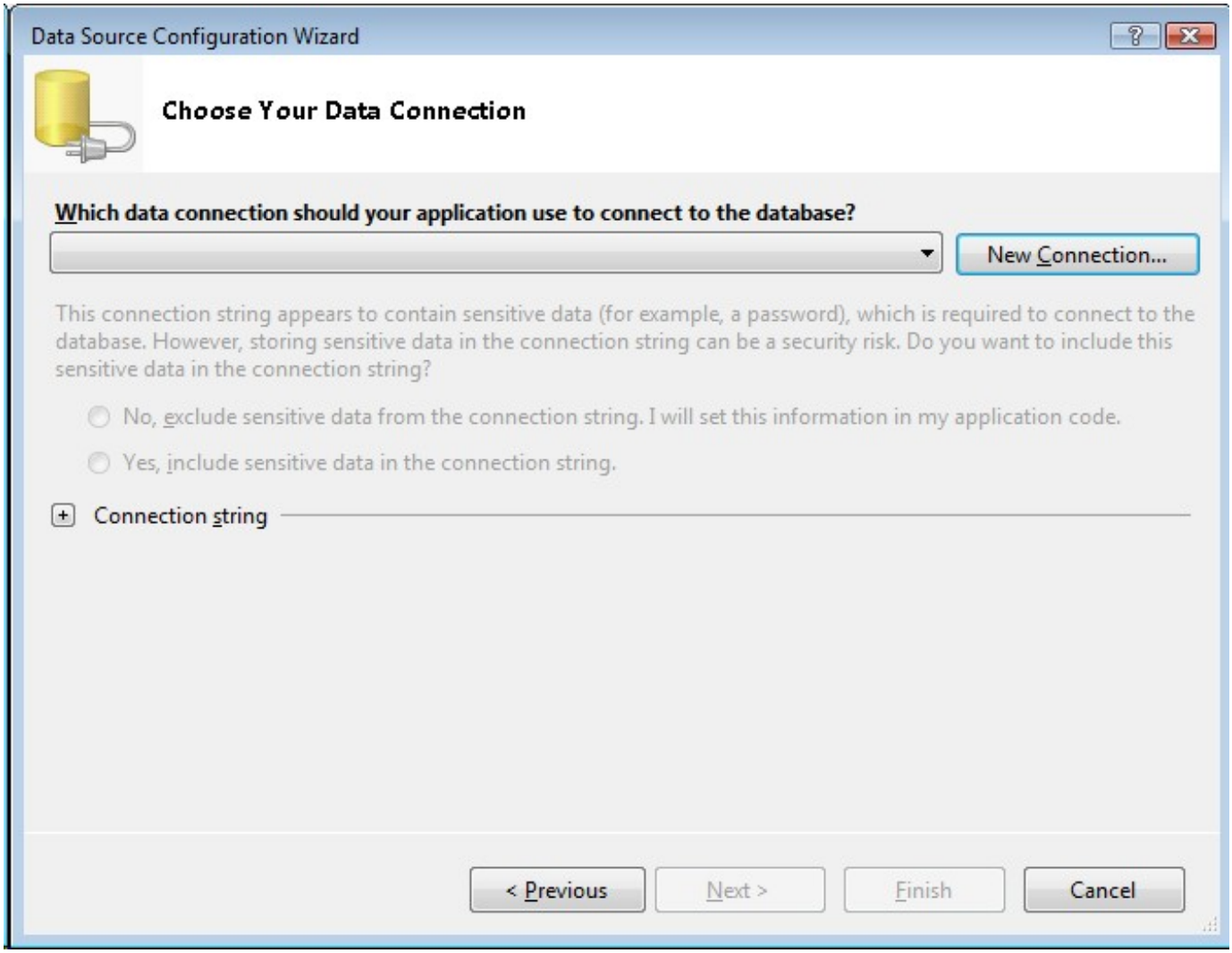
    dialog

    Choose your Data Connection

    page, select the

    New Connection button

Figure 8.33. Entity Data Model Wizard



11. In the

    Choose Data Source

    dialog, select the OpenLink

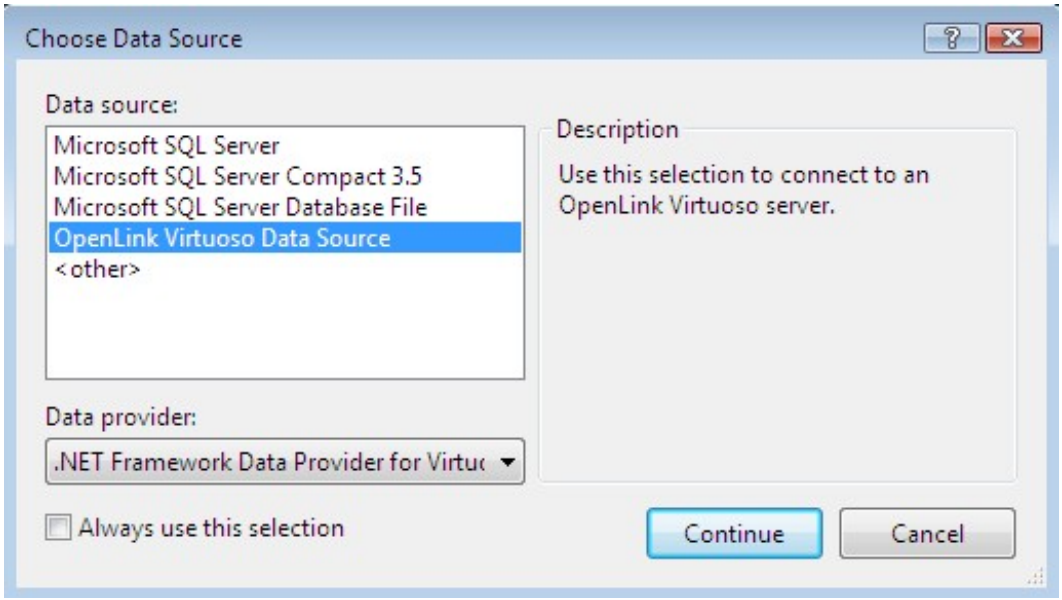
    Virtuoso Data Source

    from the list and click

    Continue

    .

Figure 8.34. Choose Data Source



12. In the

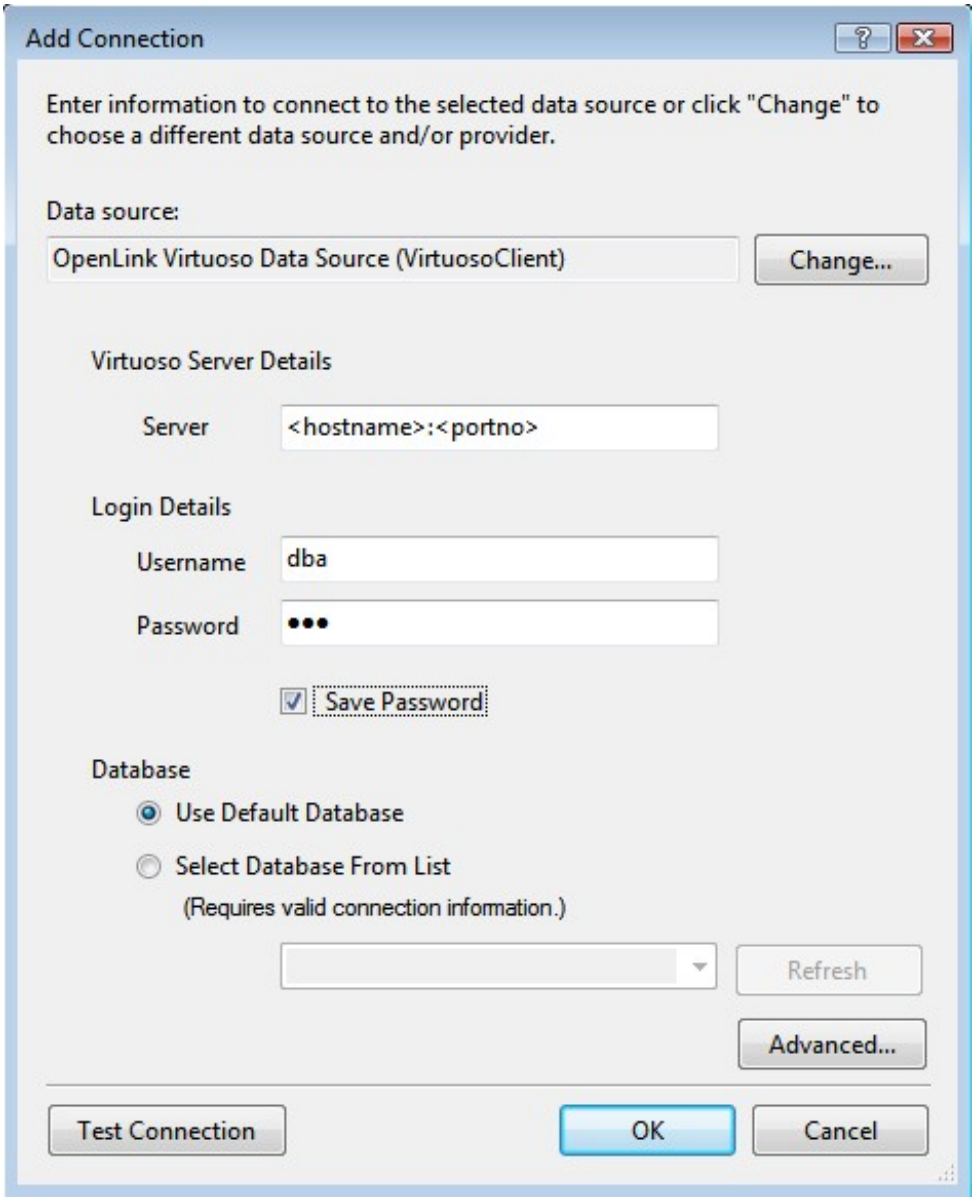
    Add Connection

    dialog, specify the

    hostname, portno, username and password

    for the target Virtuoso Server and check the Save Password check box.

    Figure 8.35. Connection Properties



13. Select the

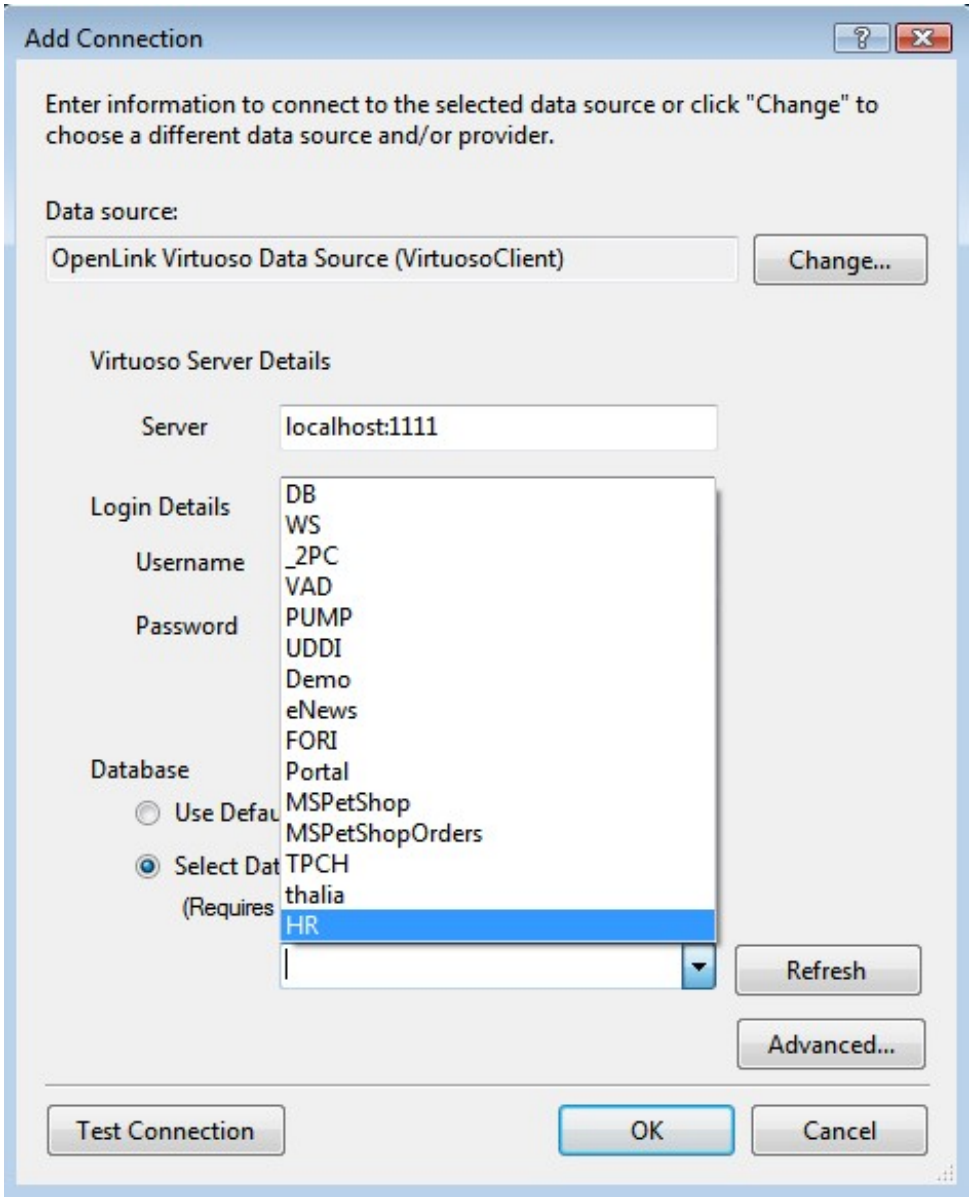
    Select Database From List

    radio button and choose the

    HR

    database from the drop down list.

Figure 8.36. Add connection

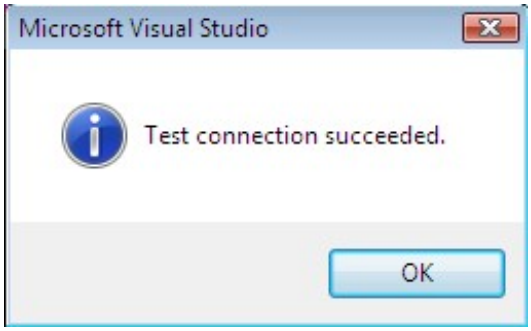


14. Press the

    Test Connection

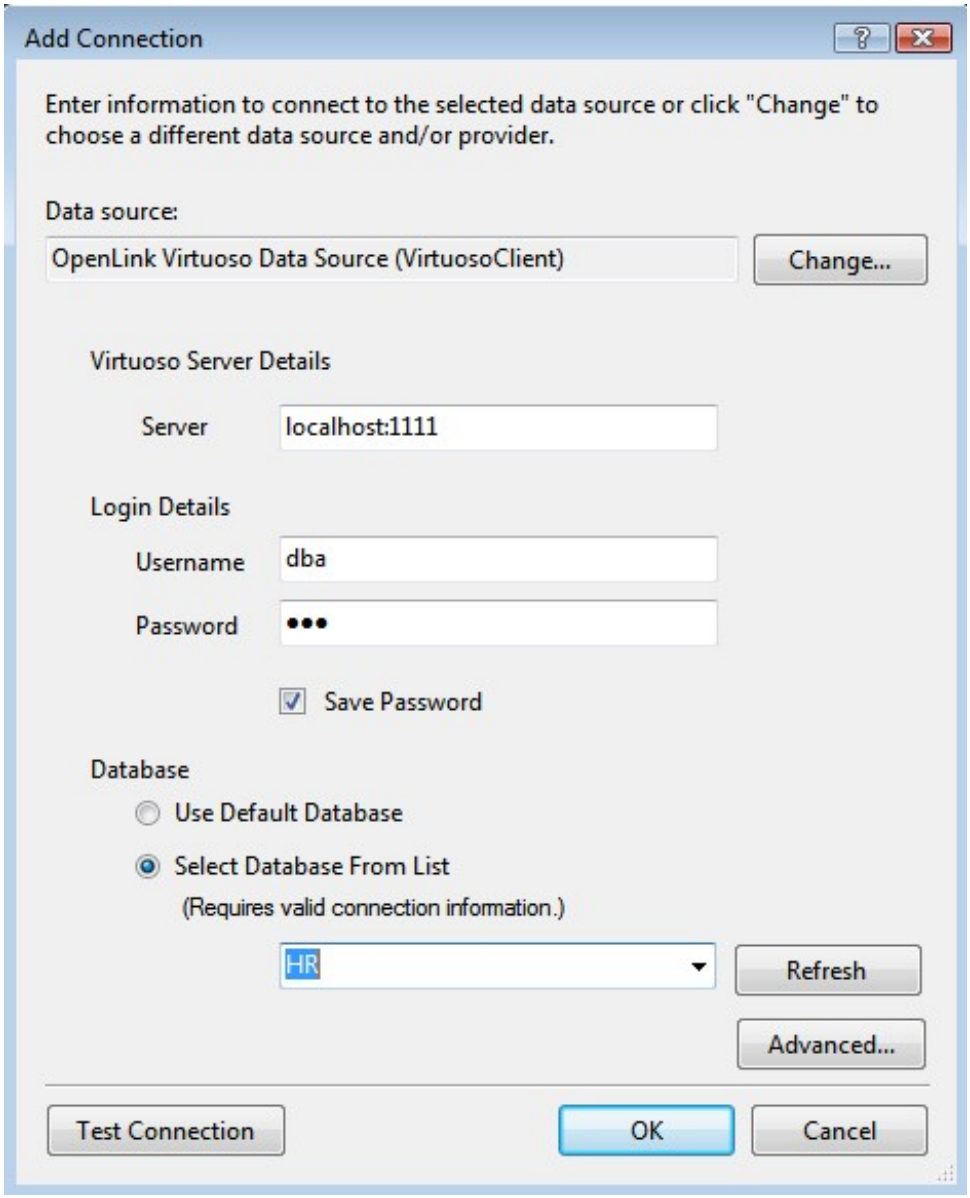
    dialog to verify that the database is accessible.

    Figure 8.37. Test Connection



15. Click OK to add the connection.

    Figure 8.38. Test Connection



16. Set the

    entity connect string

    name to

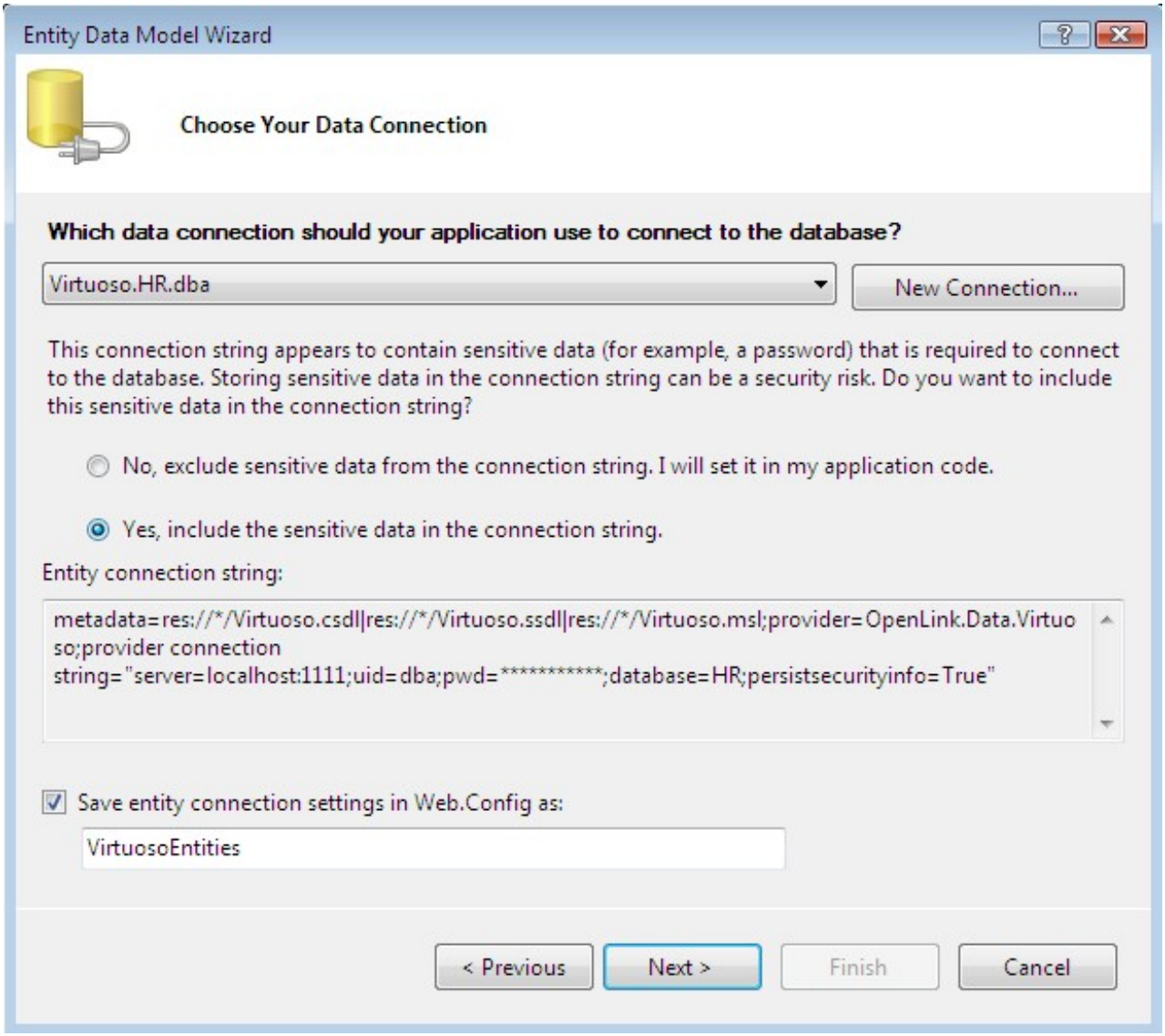
    VirtuosoEntities

    (note this name as it is required in the section on creating and ADO.Net Data Service) and click

    Next

    .

    Figure 8.39. entity connect string



17. In the

    Choose your Database Objects

    page select the

    Tables

    check box to select all tables in the HR catalog for addition to the Entity Data Model. Set the

    Model Namespace

    to

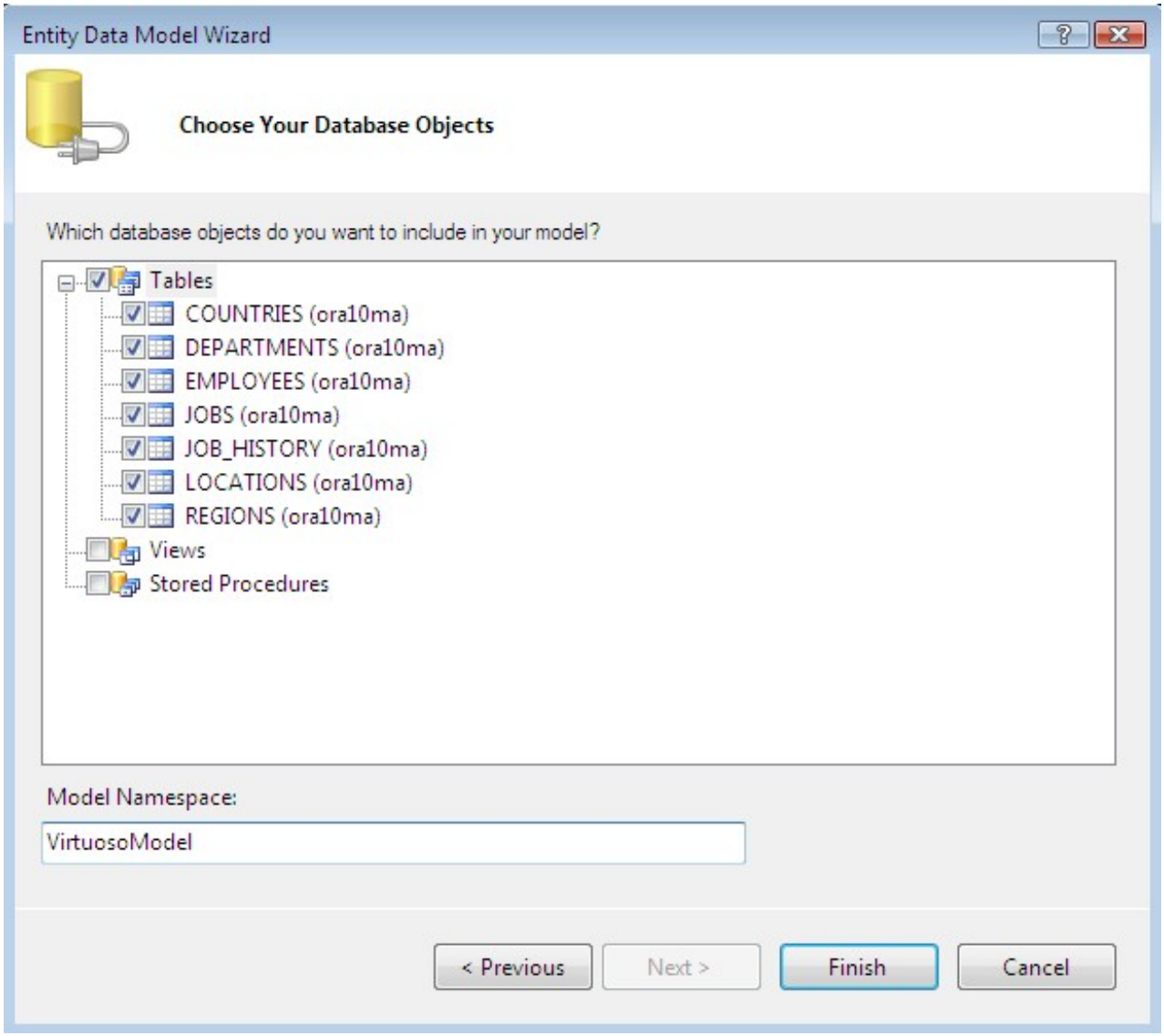
    VirtuosoModel

    and click

    Finish

    .

Figure 8.40. Database Objects

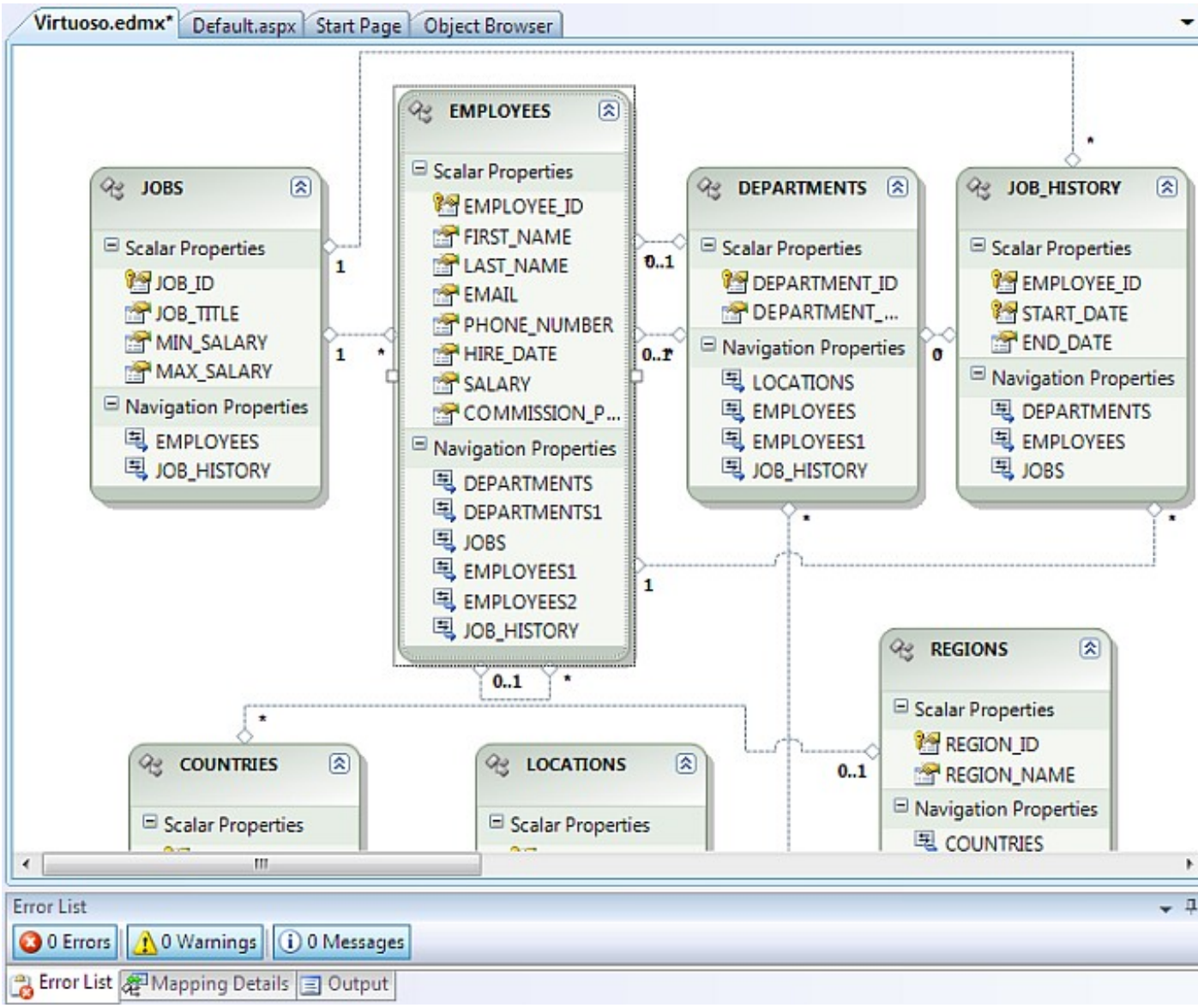


18. The

    Virtuoso.edmx

    EDM will be created with the tables and relationships displayed in the Visual Studio IDE

    Figure 8.41. Virtuoso.edmx



Creation for the Entity Data Model for the Oracle Human Resources database is now complete.

8.2.5. Using EDM to create Entity Framework based applications

Now that a Microsoft Entity Data Model has been created for the Oracle Human Resources database, Entity Framework applications can be created to make use of it.

Entity Frameworks based ADO.NET Data Service

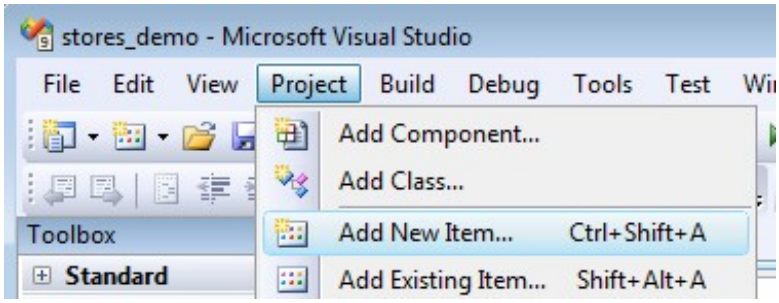
An ADO.Net Data Service for the Oracle tables can be created using the Entity Data Model created in the Creating EDM in Visual Studio 2008 section .

1. Open the

   VirtuosoDataService

   project created in the Creating EDM in Visual Studio 2008 section .
2. Select the Project -> Add New Item menu option.

Figure 8.42. VirtuosoDataService



3. The

   Add New Item

   dialog will appear. Choose the

   ADO.NET Data Service template

   . Give it the name

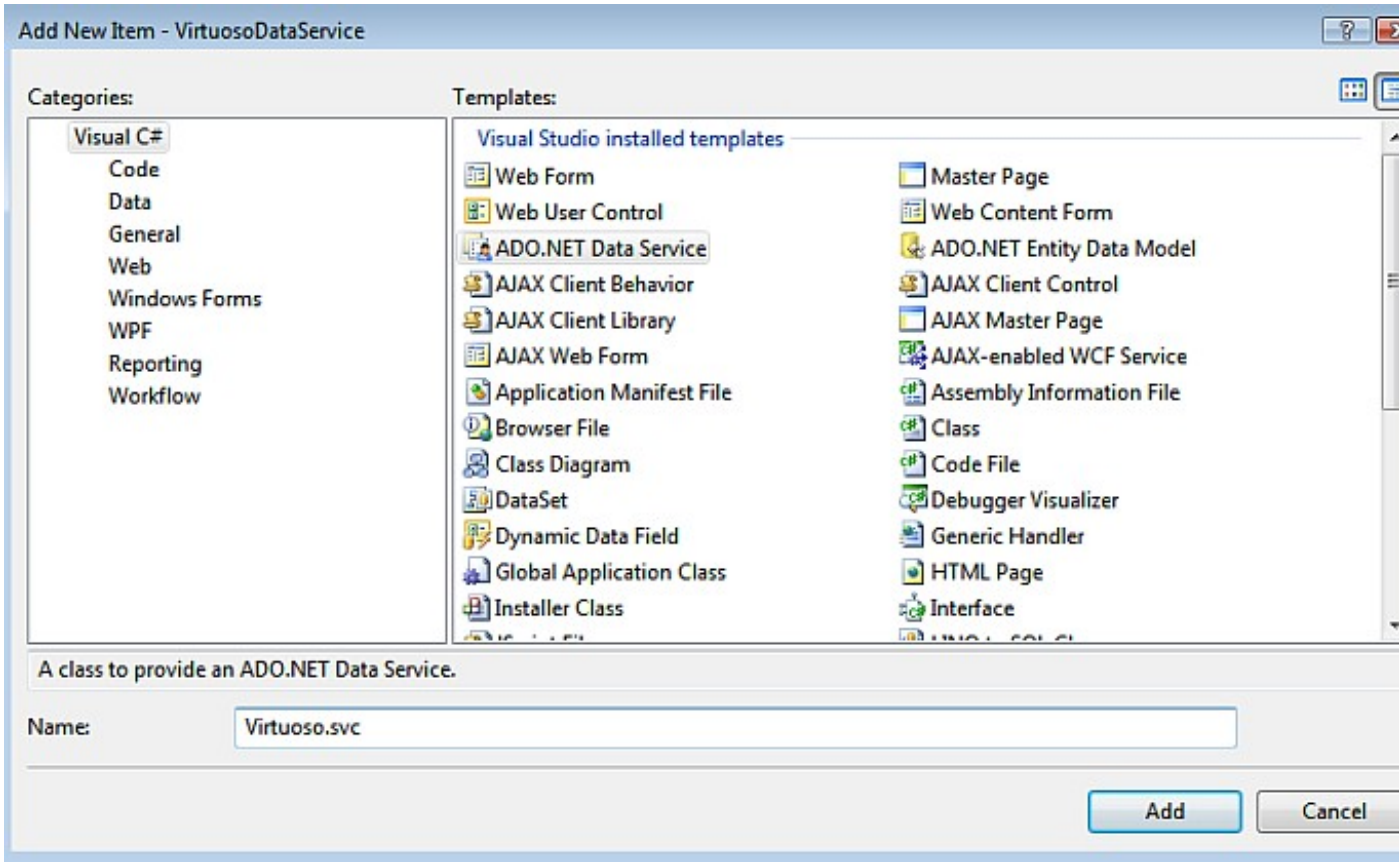
   Virtuoso.svc

   , and click

   Add

   to create the ADO.Net Data Service.

Figure 8.43. Add New Item



4. In the newly created

   Virtuoso.svc.cs

Data Service file, add the data source class name of

VirtuosoEntities

(note this is the name set in the Creating EDM in Visual Studio 2008 section) as the

DataService

name. Enable the access to the Data Service by adding the entry

config.SetEntitySetAccessRule("", EntitySetRights.All);*

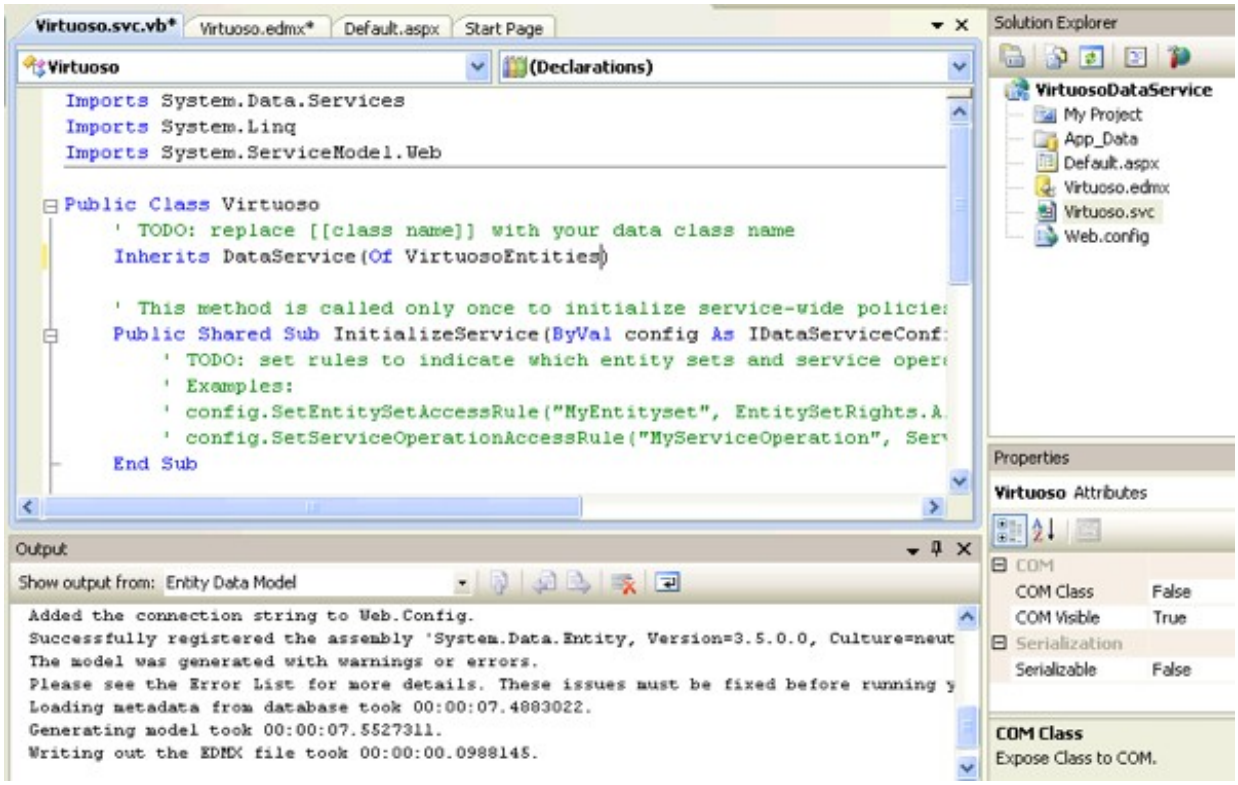
in the

InitializeService

method.

```
// C#

using System;
using System.Web;
using System.Collections.Generic;
using System.ServiceModel.Web;
using System.Linq;
using System.Data.Services;

namespace SimpleDataService
{
    public class Northwind : DataService<VirtuosoDemoEntities>
    {
        public static void InitializeService(IDataServiceConfiguration  config)
        {
            config.SetEntitySetAccessRule("*", EntitySetRights.All);
        }
    }
}
```

Figure 8.44. Virtuoso.svc.cs

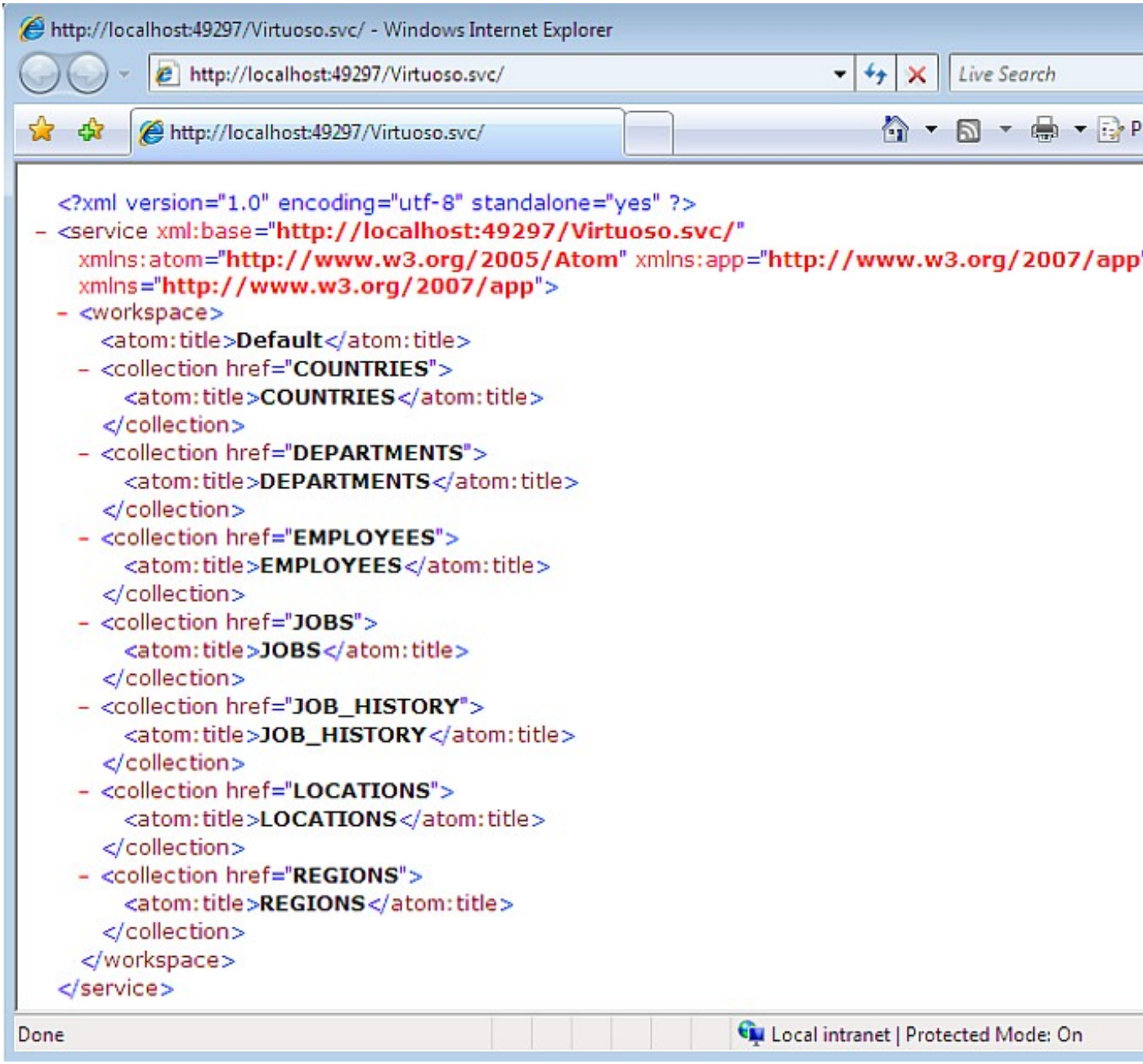


5. To test the Data Service, simply hit

   Ctrl+F5

   within Visual Studio. This will start the development web server, run the Data Services server inside and load a Web browser page displaying the list of available tables/entities for the HR database catalog.

Figure 8.45. Data Service test



6. To access a specific entity instance like the

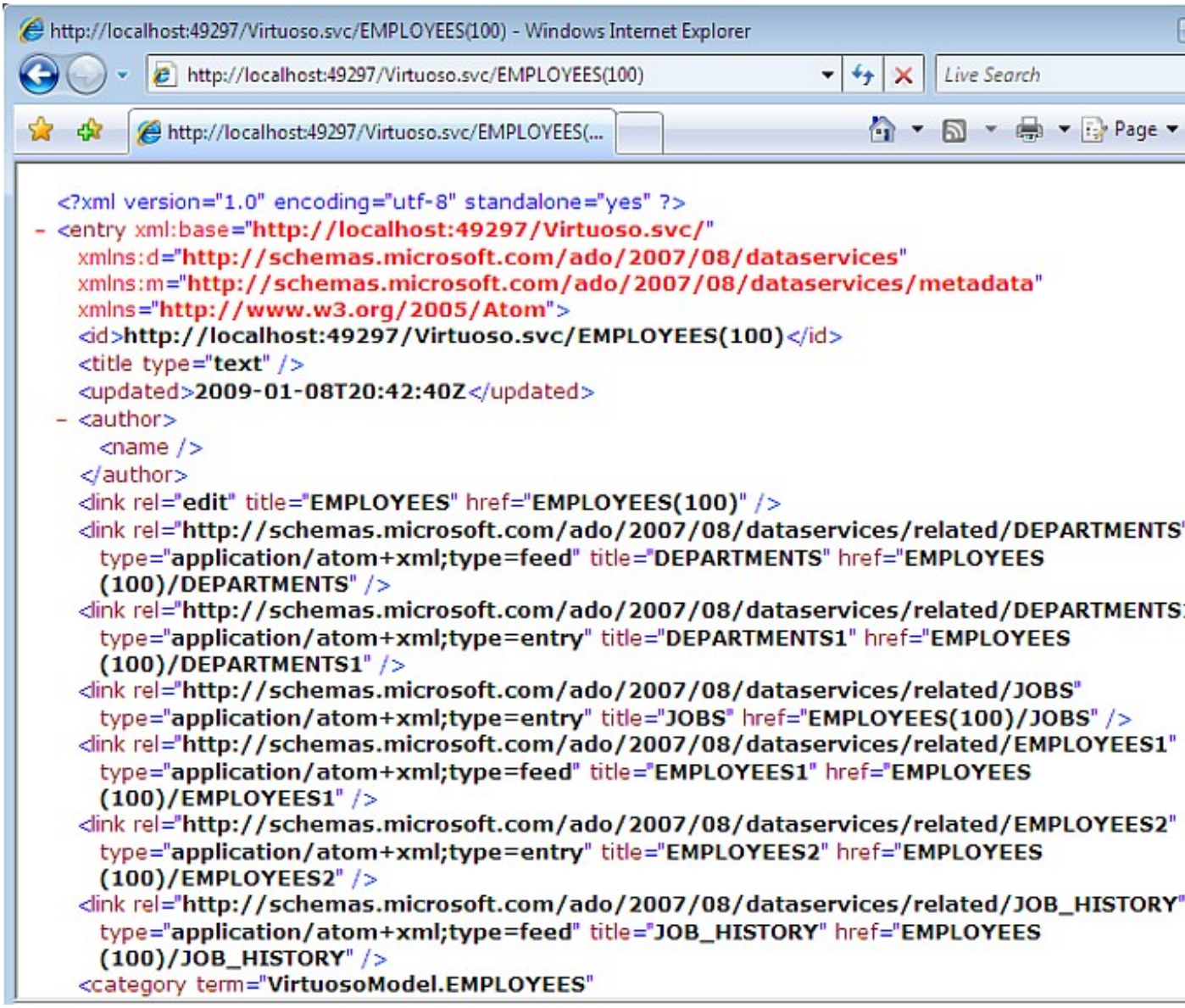
   EMPLOYEES

   table employee number

   100

   record, use this convention http://host/vdir/Virtuoso.svc/EMPLOYEES(100) .

   Figure 8.46. EMPLOYEES



Notes:

1. *Important*

   - To view

   Atom

   (the default format returned by an ADO.NET Data Service) in Internet Explorer, you must first ensure that

   Feed Reading View

   is turned

   off

   . This can be done on the

   Content tab

   of

Tools in Internet Options

.

2. If a Data Services entity instance URI page fails to load you can turn

   Verbose

   errors on by adding

   config.UseVerboseErrors = true;

   in the

   virtuoso.svc.cs InitializeService

   method to obtain more detailed information from the server as to why the page failed to load:

```
public static void InitializeService(IDataServiceConfiguration config)
{
config.UseVerboseErrors = true;
config.SetEntitySetAccessRule("*", EntitySetRights.All);
}
```

Visual Studio Windows DataGrid Form Application

This section details the steps required to create a simple Visual Studio 2008 Windows Form application, with associated DataGridView control for displaying data in selected tables from the target database.

1. Launch the Visual Studio 2008 SP1 IDE.

   Figure 8.47. Visual Studio 2008 SP1 IDE

2. Create a

   Web Application

   project by going to the

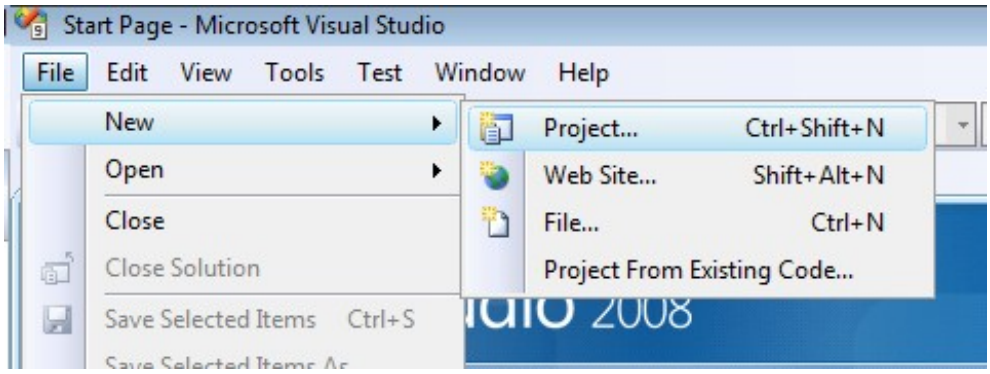
   File

   menu in Visual Studio and choosing

New Project

.

Figure 8.48. Web Application



3. When the New Project window appears, choose either

   Visual Basic

   or

   Visual C#

   as your programming language.
4. Within the language category, click on

   Windows

   and select

   Windows Form Application

   from the right-hand panel.
5. Choose a name for the project, for example

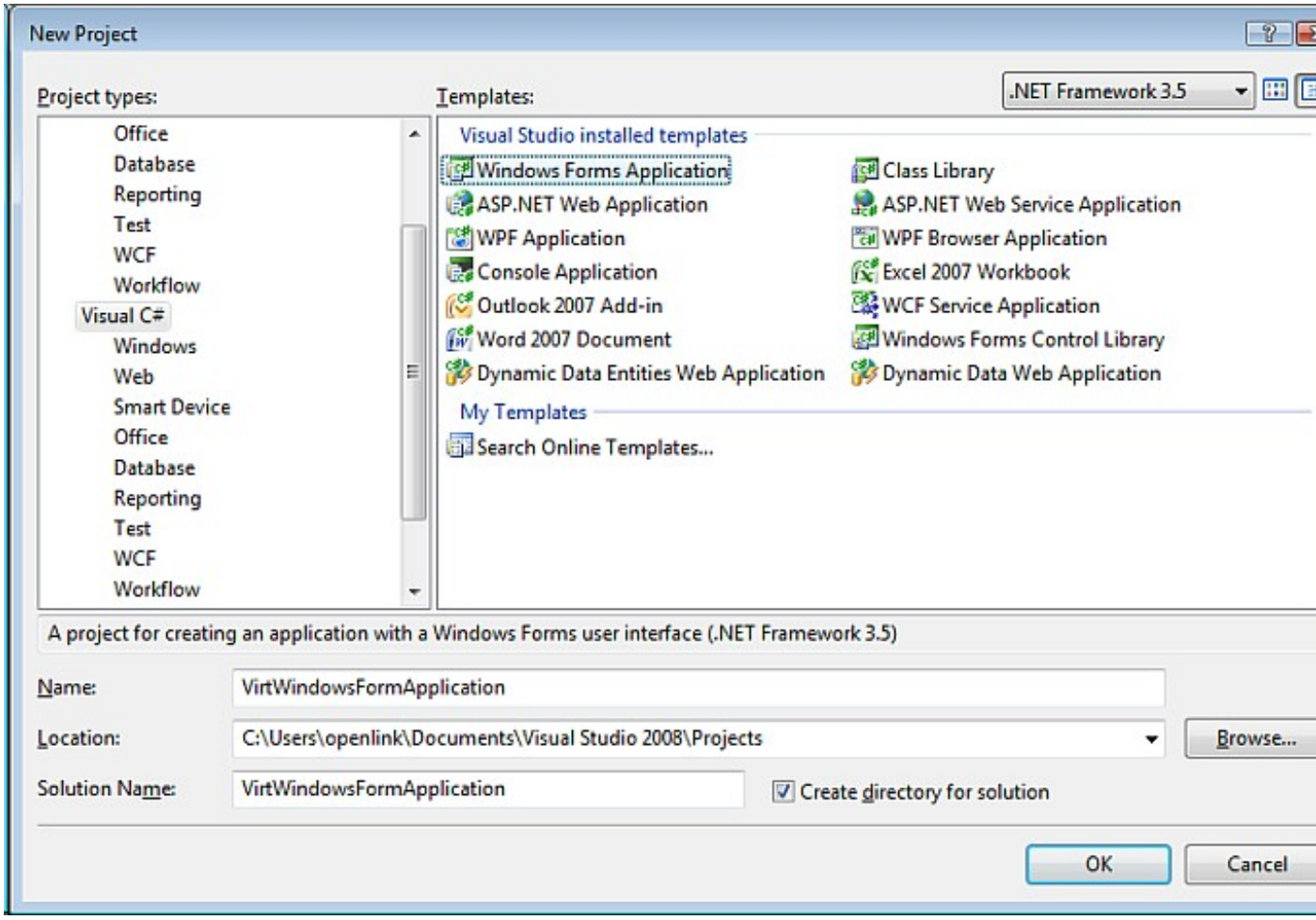
   VirtWindowsFormApplication

   , and click

   OK

   .

Figure 8.49. Web Application



6. In the

   Toolbox

   , expand

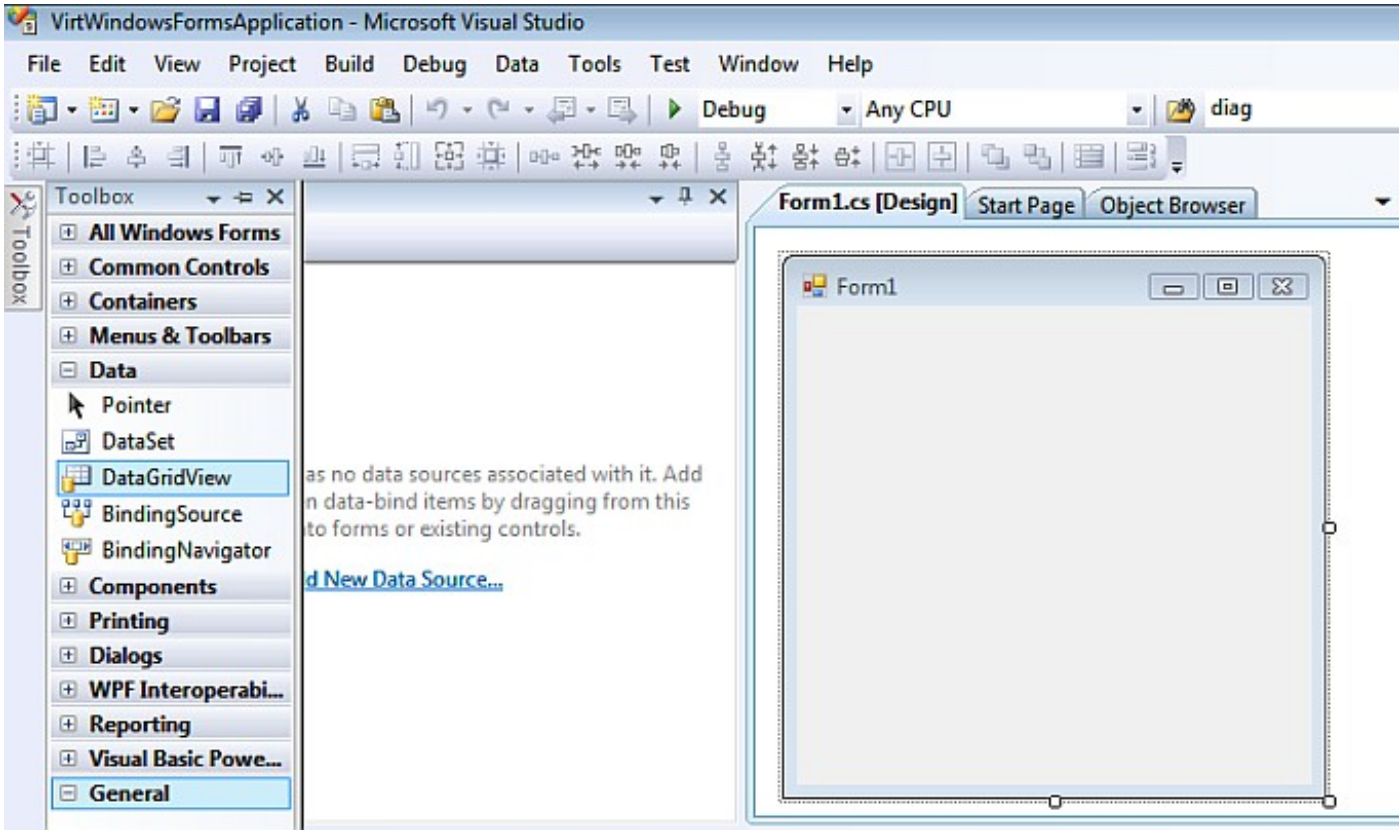
   Data Controls

   , and drag the

   DataGridView

   control onto the form.

   Figure 8.50. Toolbox



7. Click on the little

   arrow

   in the top right of the

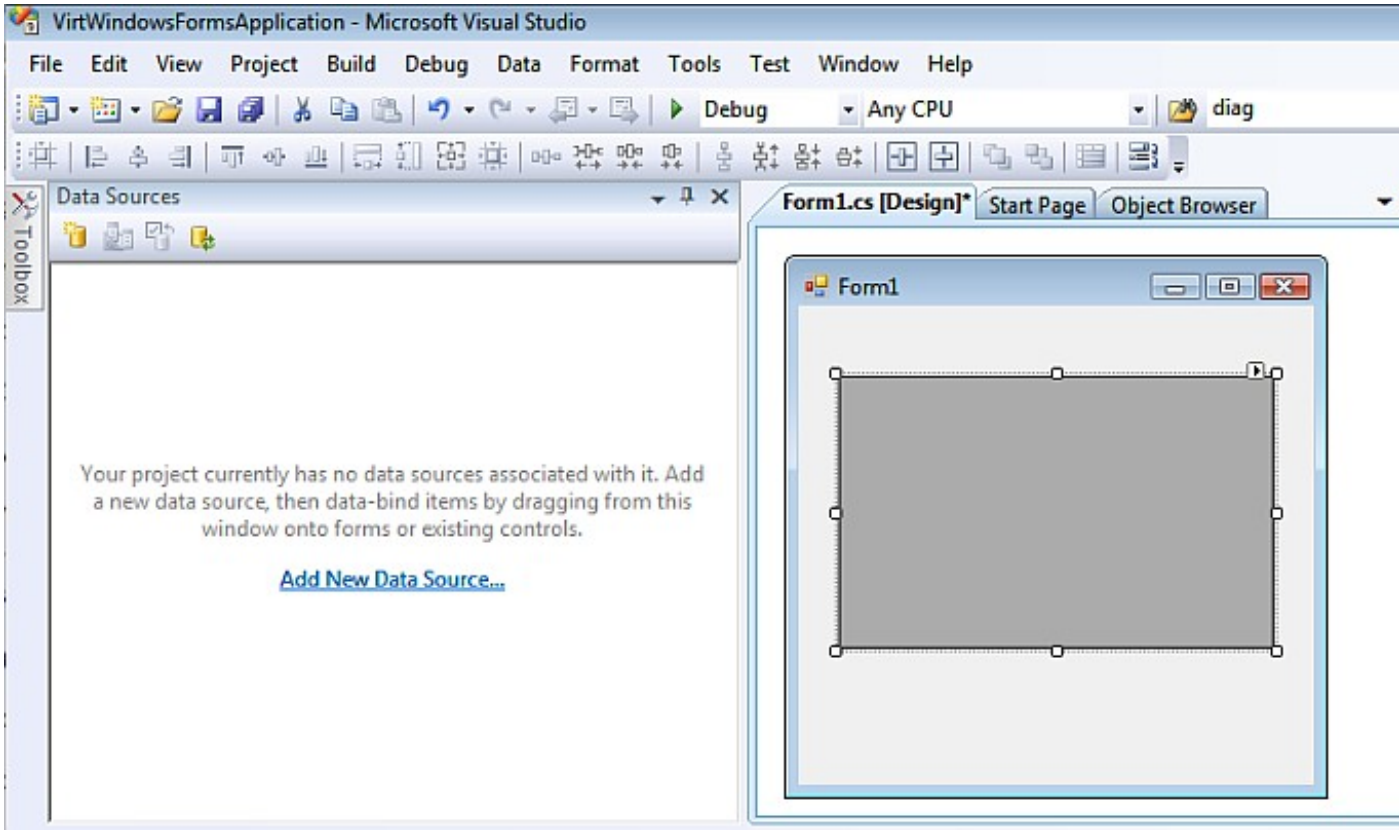
   DataGridView

   control. This loads the

   DataGridView Task

   menu.

Figure 8.51. DataGridView Task

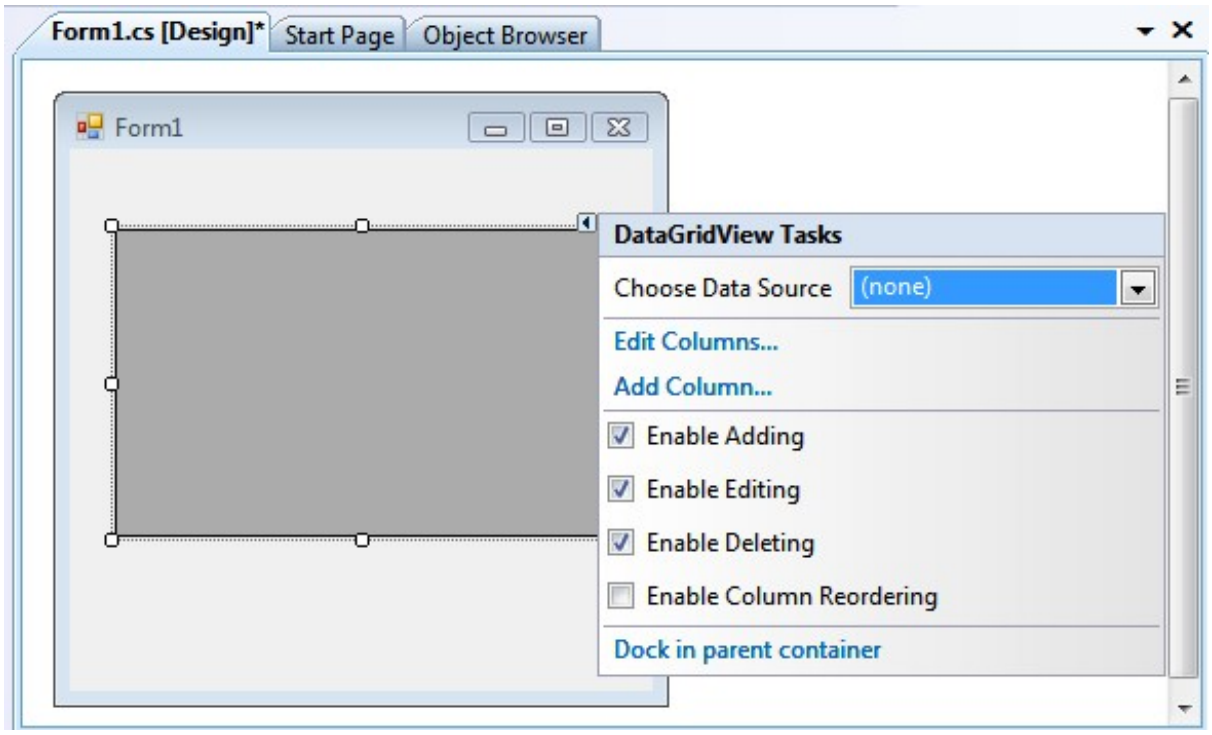


8. Click on the

   Choose Data Source

   list box.

Figure 8.52. Choose Data Source

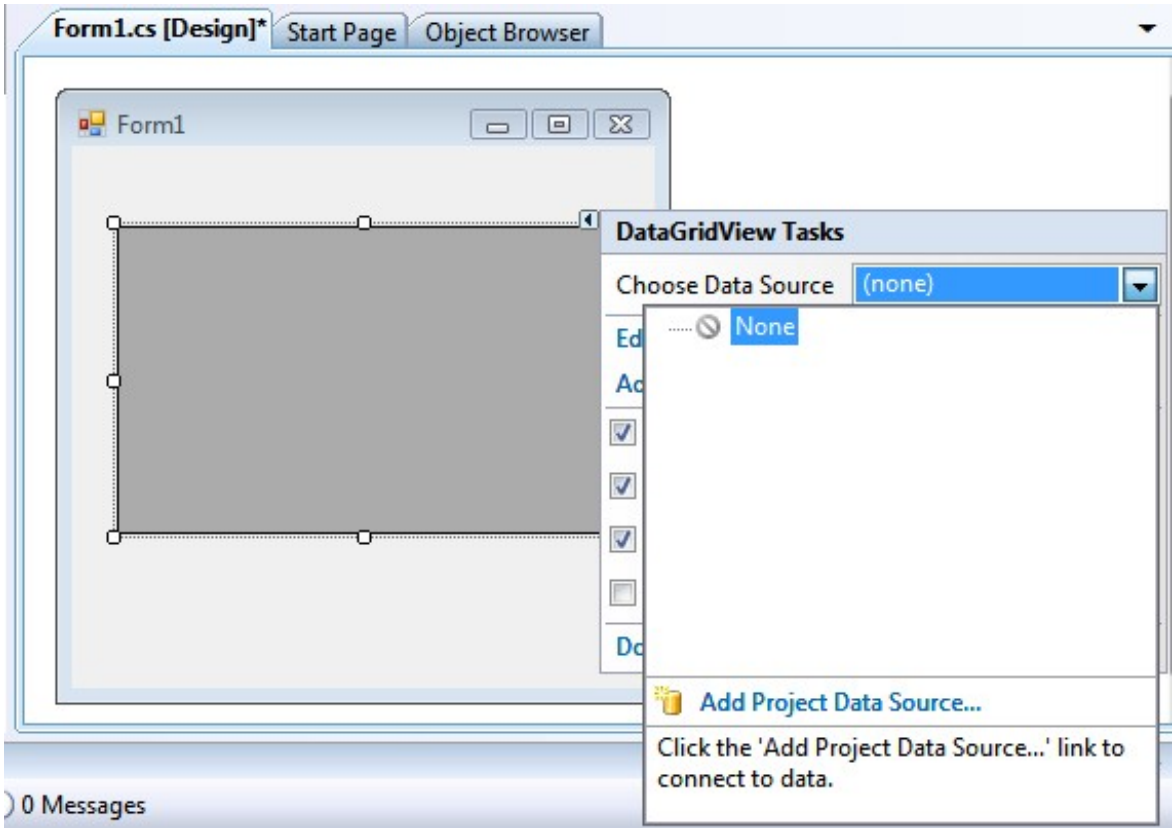


9. Click on the

   Add Project Data Source

link to connect to a data source.

Figure 8.53. Add Project Data Source



10. In the

    Data Source Configuration Wizard

    dialog

    Choose Data Source Type

    page select the

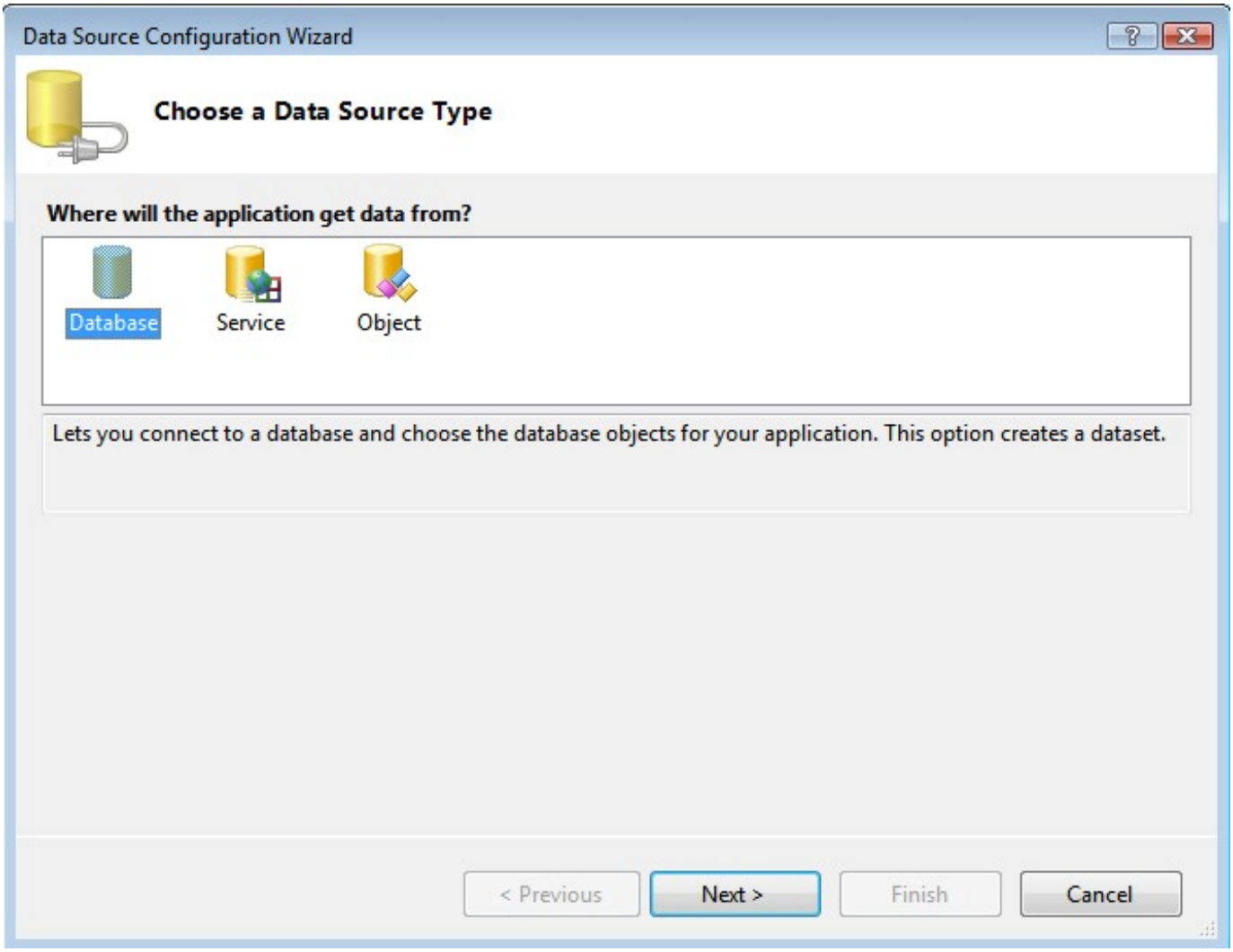
    Database

    data source type and click

    Next

    .

Figure 8.54. Data Source Type



11. In the

    Data Source Configuration Wizard

    dialog

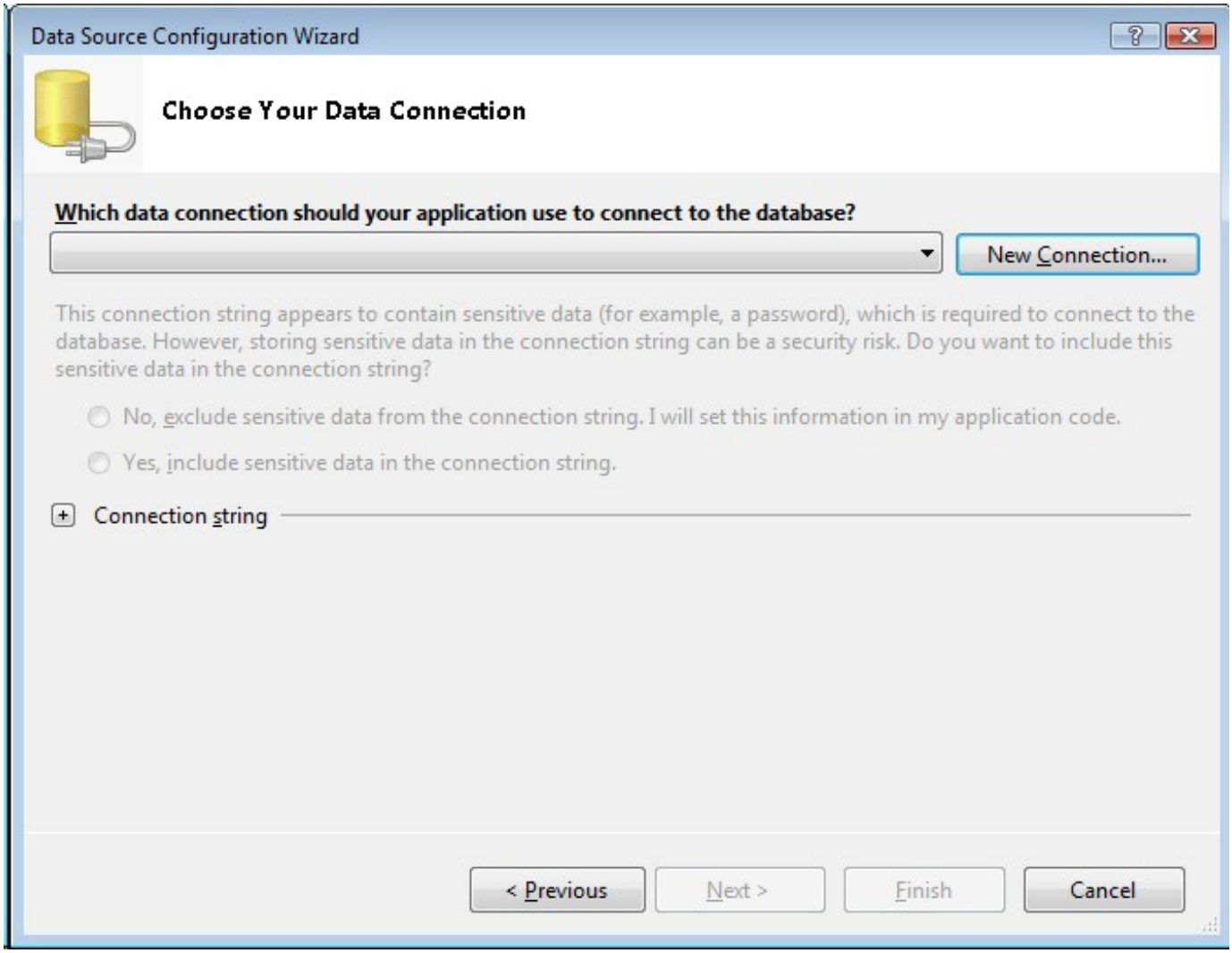
    Choose your Data Connection

    page, select the

    New Connection

    button

Figure 8.55. Data Source Configuration Wizard



12. In the

    Choose Data Source

    dialog, select the OpenLink

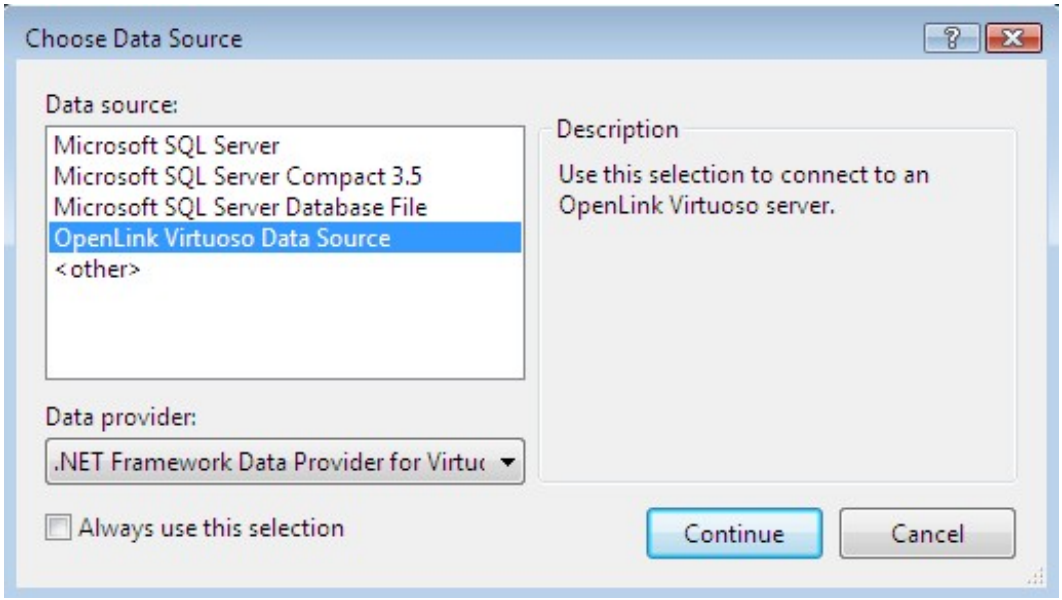
    Virtuoso Data Source

    from the list and click

    Continue

    .

Figure 8.56. Data Source



13. In the

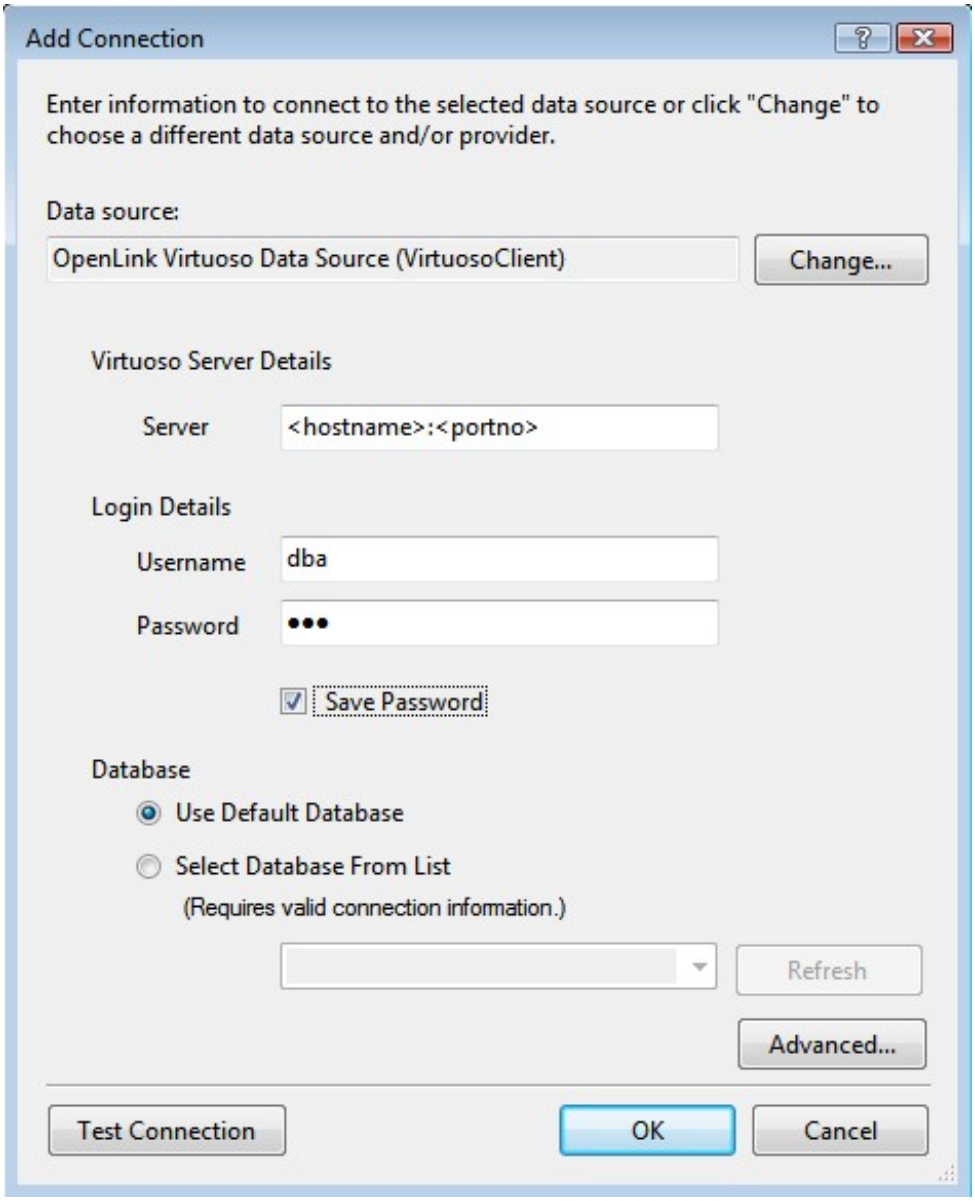
    Add Connection

    dialog, specify the

    hostname, portno, username and password

    for the target Virtuoso Server and check the Save Password check box.

    Figure 8.57. Connection Properties



14. Select the

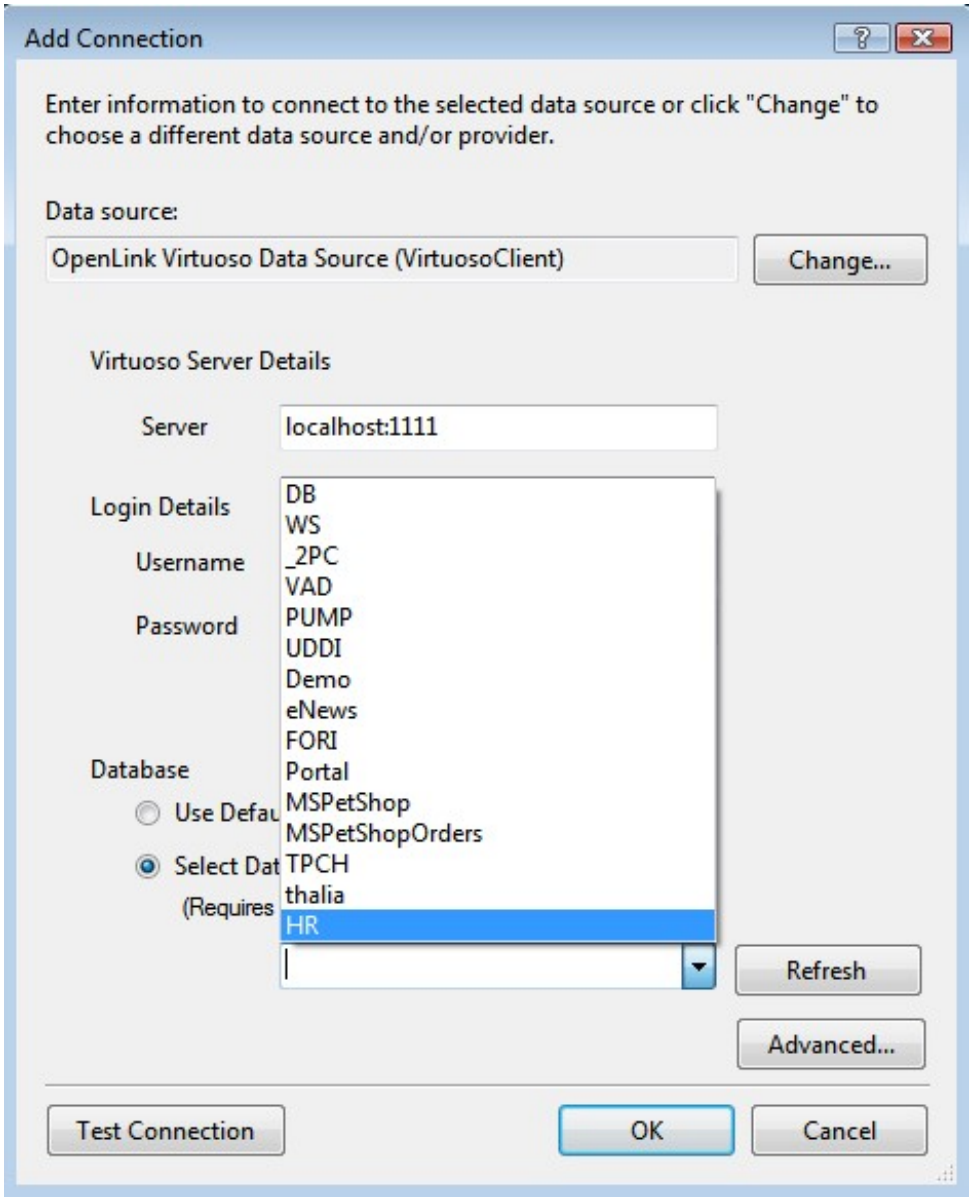
    Select Database From List

    radio button and choose the

    HR

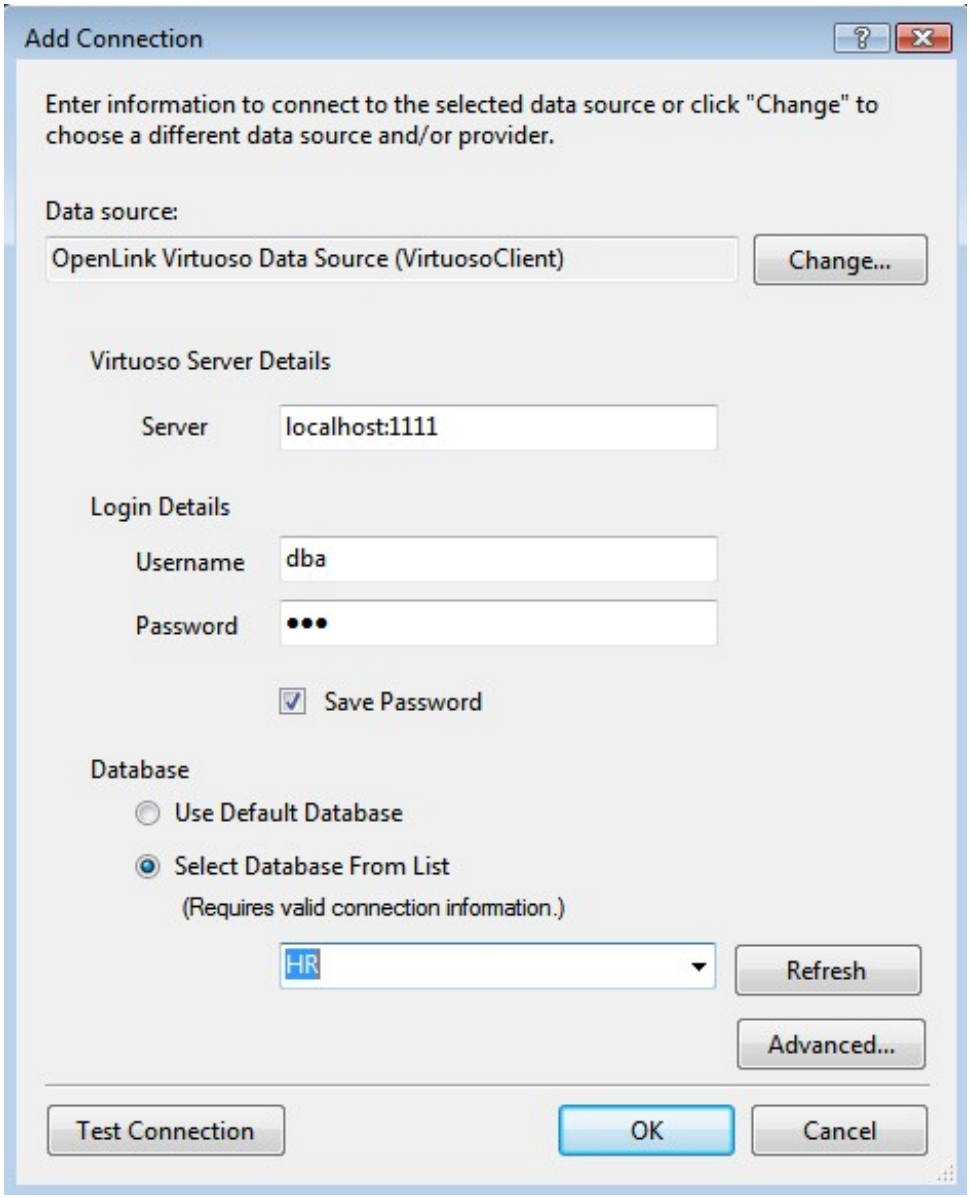
    database from the drop down list.

Figure 8.58. Add connection



15. Click OK to add the connection.

Figure 8.59. Add connection

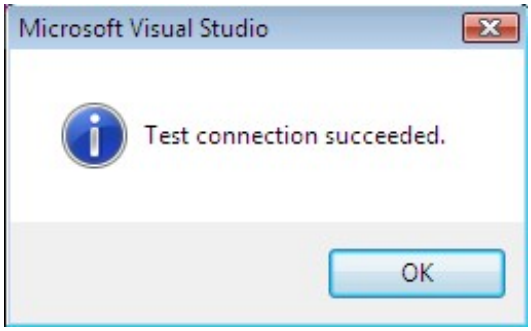


16. Press the

    Test Connection

    dialog to verify that the database is accessible.

    Figure 8.60. Test Connection



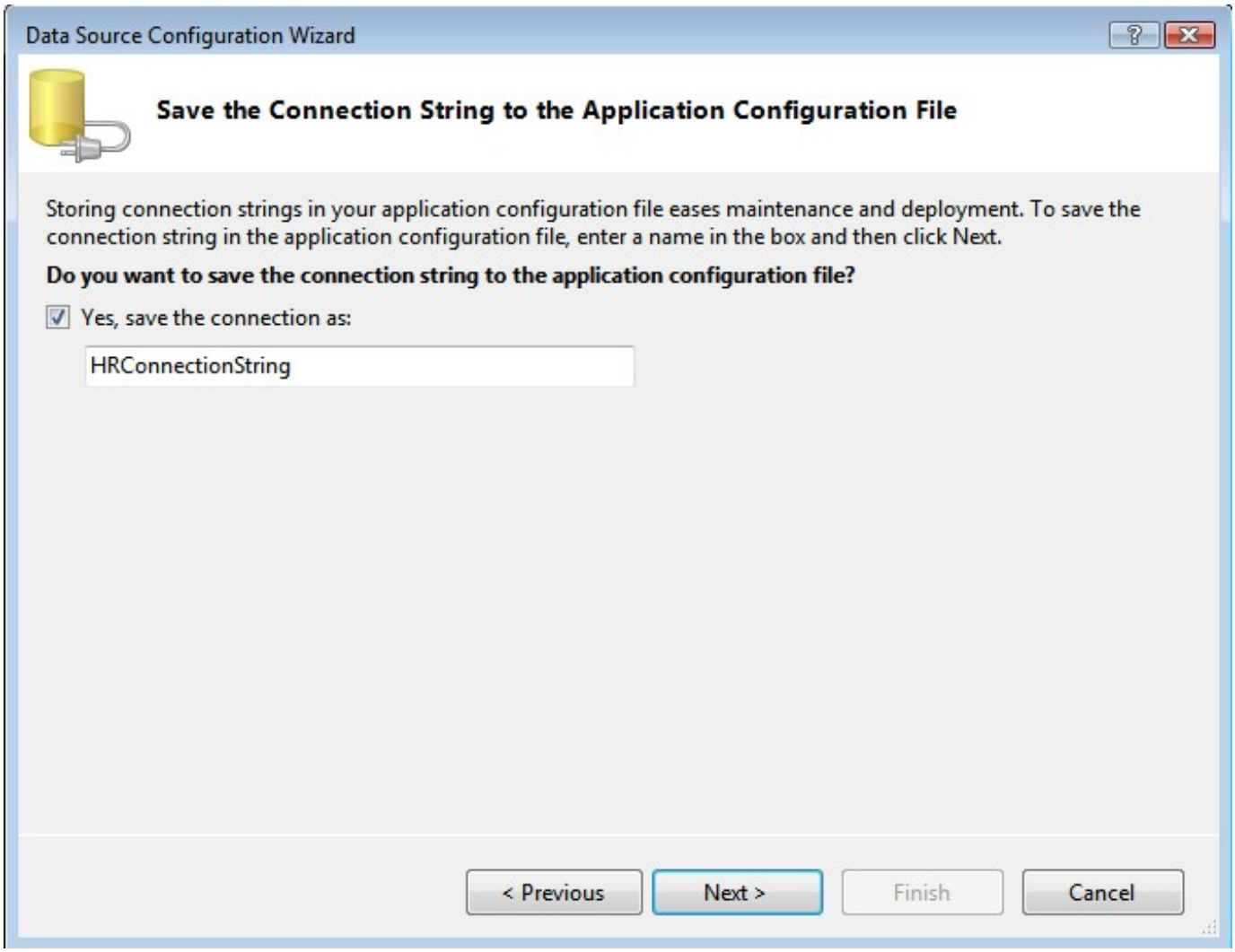
17. Leave the default connect string

    HRConnectionString

and click

Next

Figure 8.61. HRConnectionString



18. From the list of available tables returned for the HR database, select the

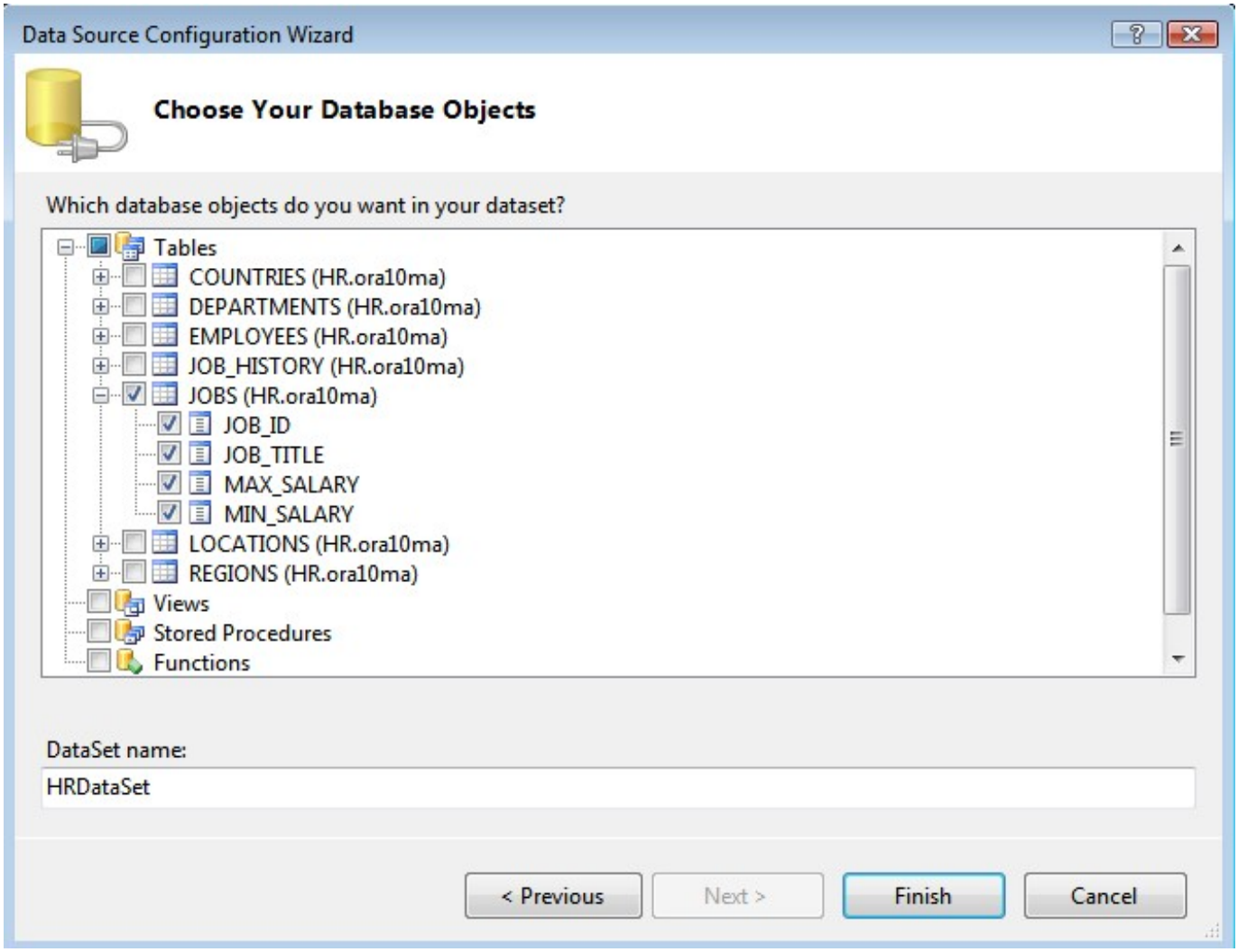
    JOBS

    table to be associated with the

    DataGridView

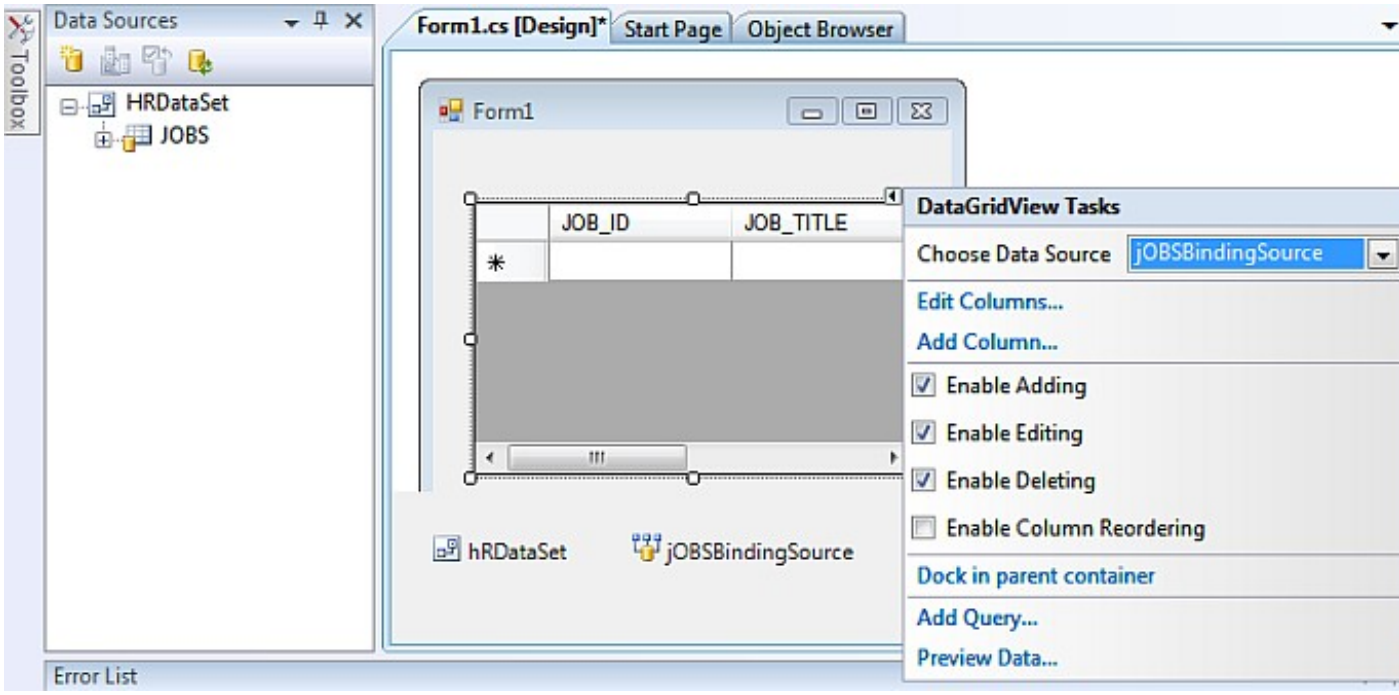
    control.

Figure 8.62. HR database



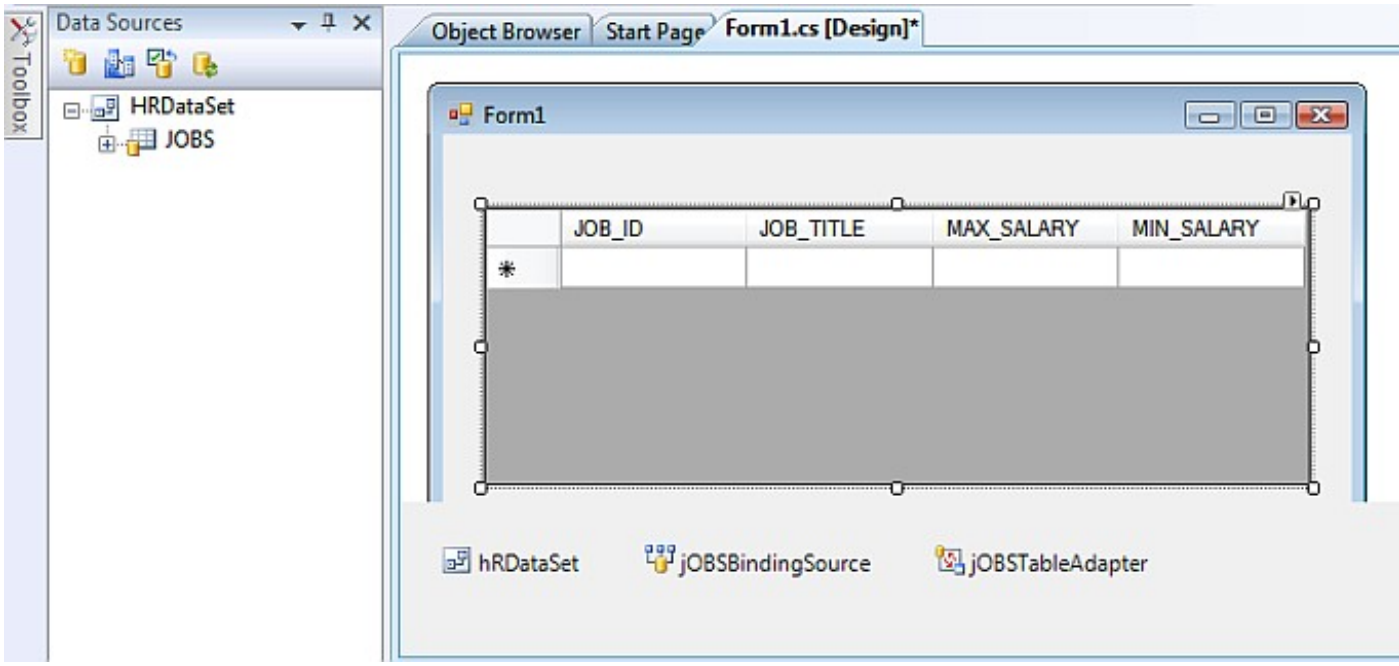
19. The columns names of the select table will be displayed in the DataGridView.

Figure 8.63. DataGridView



20. Resize the Form and DataGridView to allow all columns to be visible, if possible.

Figure 8.64. Resize the Form and DataGridView



21. To test the application, simply hit

    Ctrl+F5

    within Visual Studio or select

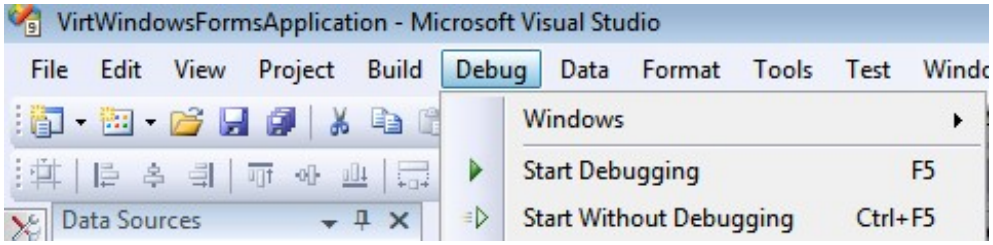
    Start Debugging

    from the

    Debug

    menu.

Figure 8.65. Start Debugging



22. The data from the

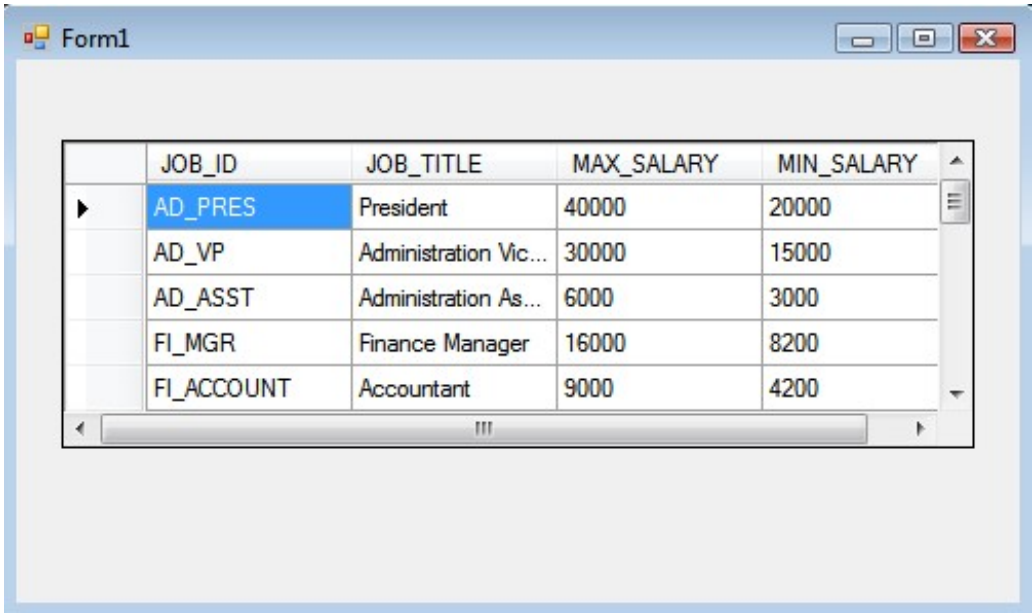
    JOBS

    table will be displayed in the

    DataGrid

    .

Figure 8.66. DataGrid



The task is now complete.

8.3. Using Microsoft Entity Frameworks to Access Progress Schema Objects with Virtuoso

Abstract

This section details the steps required to provide Microsoft Entity Framework access to Progress Schema Objects using the OpenLink Virtuoso Universal Server. This is achieved by Linking the required Progress Schema objects into Virtuoso using its built in Virtual Database engine, and then using the Virtuoso ADO.Net Entity Framework provider to query the remote Progress Schema objects linked into the Virtuoso Server.

Prerequisites

Progress OpenEdge 10. A Progress DBMS hosting the required Schema Objects needs to be available. In this section, the *isports* sample database will be used to demonstrate the process.

A Progress DBMS hosting the required Schema Objects needs to be available. In this section, the *isports* sample database will be used to demonstrate the process.

OpenLink ODBC Driver for Progress (SQL-92). A Progress ODBC Driver is required to link the Progress Schema Objects into the Virtuoso Server. The OpenLink *ODBC Driver for Progress (SQL-92)* will be used in this section, for which a functional ODBC Data source name of "*prs101c* " will be assumed to exist on the machine hosting the Virtuoso Server.

A Progress ODBC Driver is required to link the Progress Schema Objects into the Virtuoso Server. The OpenLink *ODBC Driver for Progress (SQL-92)* will be used in this section, for which a functional ODBC Data source name of "*prs101c* " will be assumed to exist on the machine hosting the Virtuoso Server.

OpenLink Virtuoso 5.10.x. An OpenLink Virtuoso Universal Server installation including the Virtuoso Universal Server and ADO.NET Entity Framework Provider is required. The Virtuoso components must be Release 5.10.x or above, this being the minimum version containing support for Microsoft Entity Frameworks.

An OpenLink Virtuoso Universal Server installation including the Virtuoso Universal Server and ADO.NET Entity Framework Provider is required. The Virtuoso components must be Release 5.10.x or above, this being the minimum version containing support for Microsoft Entity Frameworks.

Microsoft Visual Studio 2008 + SP1. Microsoft Visual Studio 2008 with Service Pack 1 is required, this being the only version containing the necessary Entity Framework support available at the time of writing.

Microsoft Visual Studio 2008 with Service Pack 1 is required, this being the only version containing the necessary Entity Framework support available at the time of writing.

Tasks

Ensure Progress Primary Keys (PKs) are not nullable. The Visual Studio 2008 Entity Data Model (EDM) requires that all primary keys are NOT Nullable, and will fail to generate an EDM if any are nullable. Thus ensure any tables to be used are defined as not nullable in the Progress database schema before attempting to generate an EDM.

The Visual Studio 2008 Entity Data Model (EDM) requires that all primary keys are NOT Nullable, and will fail to generate an EDM if any are nullable. Thus ensure any tables to be used are defined as not nullable in the Progress database schema before attempting to generate an EDM.

It seems that, by default, several Primary Keys (PKs) in the isports database allows <NULL> values. It seems somewhat nonsensical - that a unique key used, specifically, to identify a row in a table can be allowed to be <NULL>.

This issue is best addressed directly in the database schema, by redefining those PKs that allow <NULL> so as not to allow <NULL>. Progress does not seem to talk in terms of <NULL>, instead using the term "Mandatory"

The following section will provide more specific details about how to ensure Progress PKs are not nullable:

- Ensuring Progress PKs are not nullable

8.3.1. Install and configure OpenLink ODBC Driver for Progress (SQL-92)

The Virtuoso Virtual Database engine uses ODBC as the connectivity mechanism for linking remote database objects into its local schema. Thus a Progress ODBC Driver must be available with a suitably configured DSN for connecting to the target database. The OpenLink ODBC Drivers for Progress have been used in this section, although in theory any Progress ODBC Driver can be used.

Installation instructions for the OpenLink ODBC Driver for Oracle are available from:

- Product Installation & Basic Configuration (ODBC)

8.3.2. Install and configure OpenLink Virtuoso Universal Server

- Install and configure OpenLink Virtuoso Universal Server

8.3.3. Linking Progress tables into Virtuoso

1. Start the Virtuoso Web User Interface

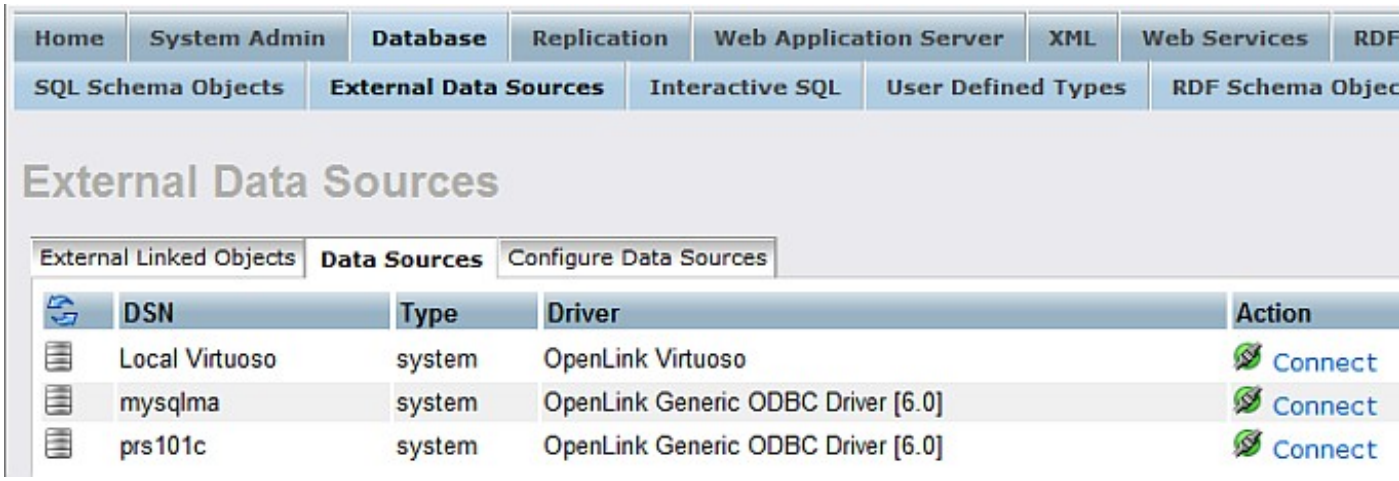
   Figure 8.67. Start

2. Select the "Conductor" link in the Left Frame and Login as the "dba" user.

Figure 8.68. Conductor

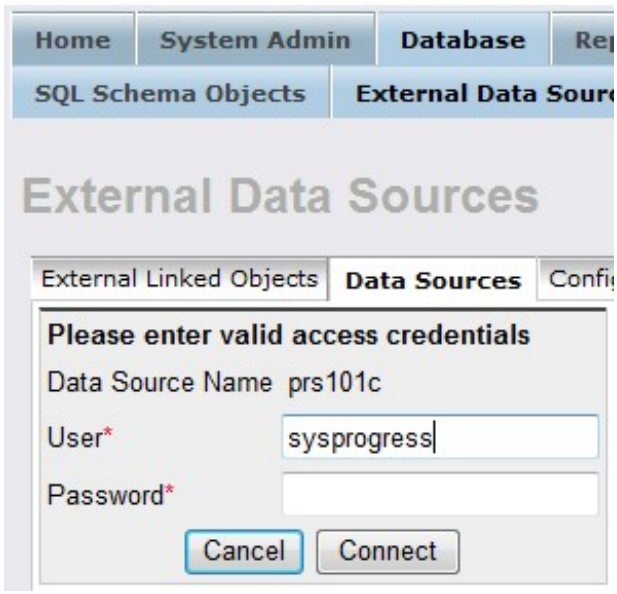
3. Select the "Databases" -> "External Data Source" -> "Data Sources" tab menu items

Figure 8.69. Databases



4. Select the "Connect" button for the "prs10ma" Oracle DSN.

Figure 8.70. Connect

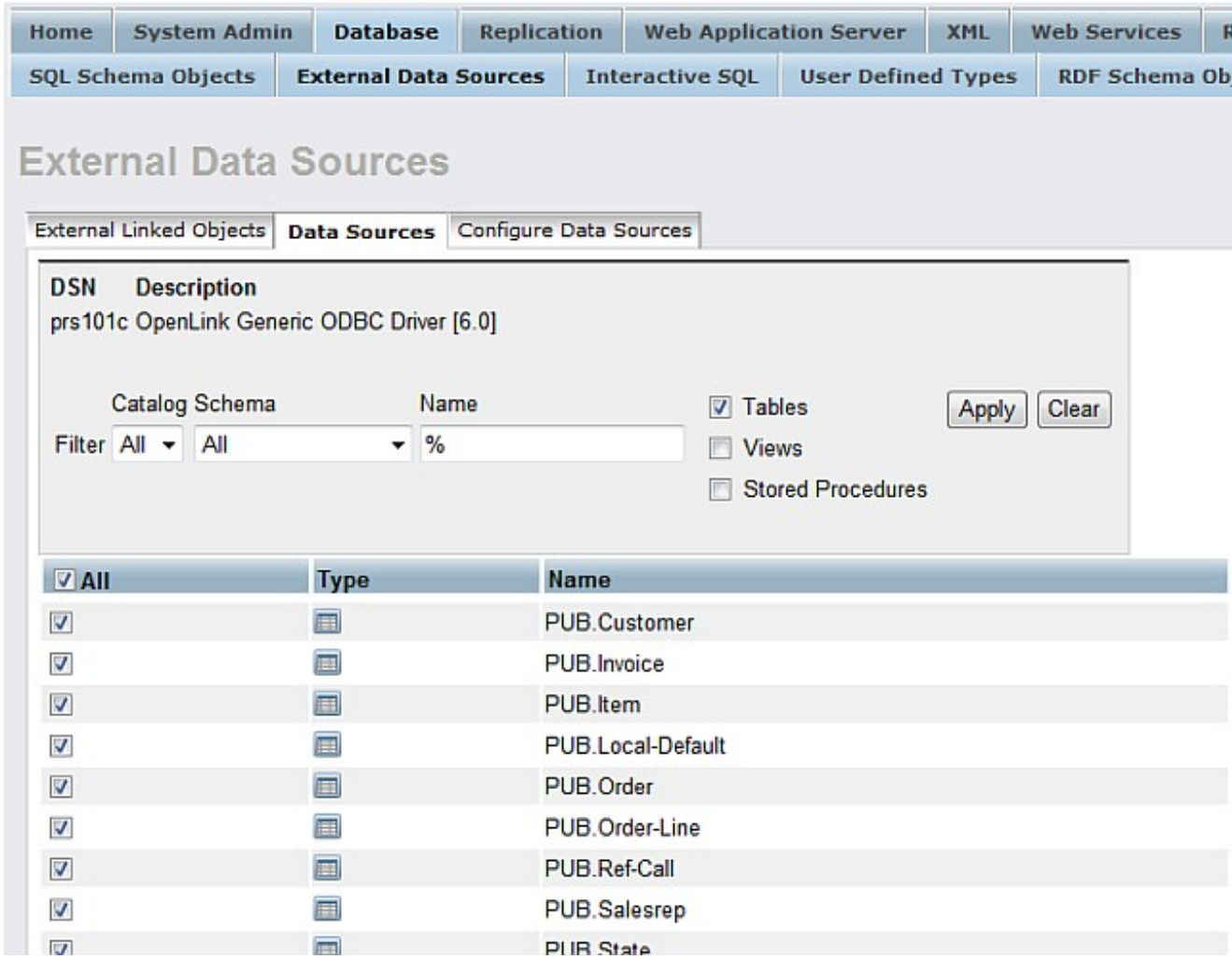


5. On successful connection Select the "Link Objects" button to obtain a list of available tables

Figure 8.71. Link Objects

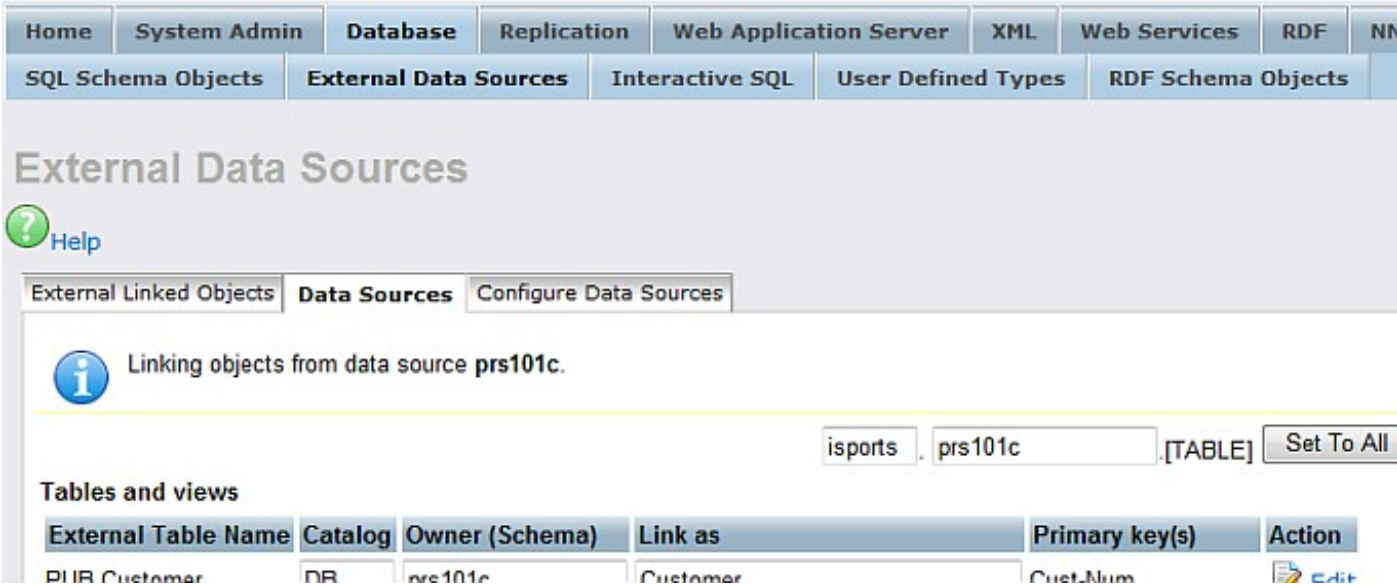
6. Select all the tables that are part of the "isports" catalog.

Figure 8.72. Select all tables



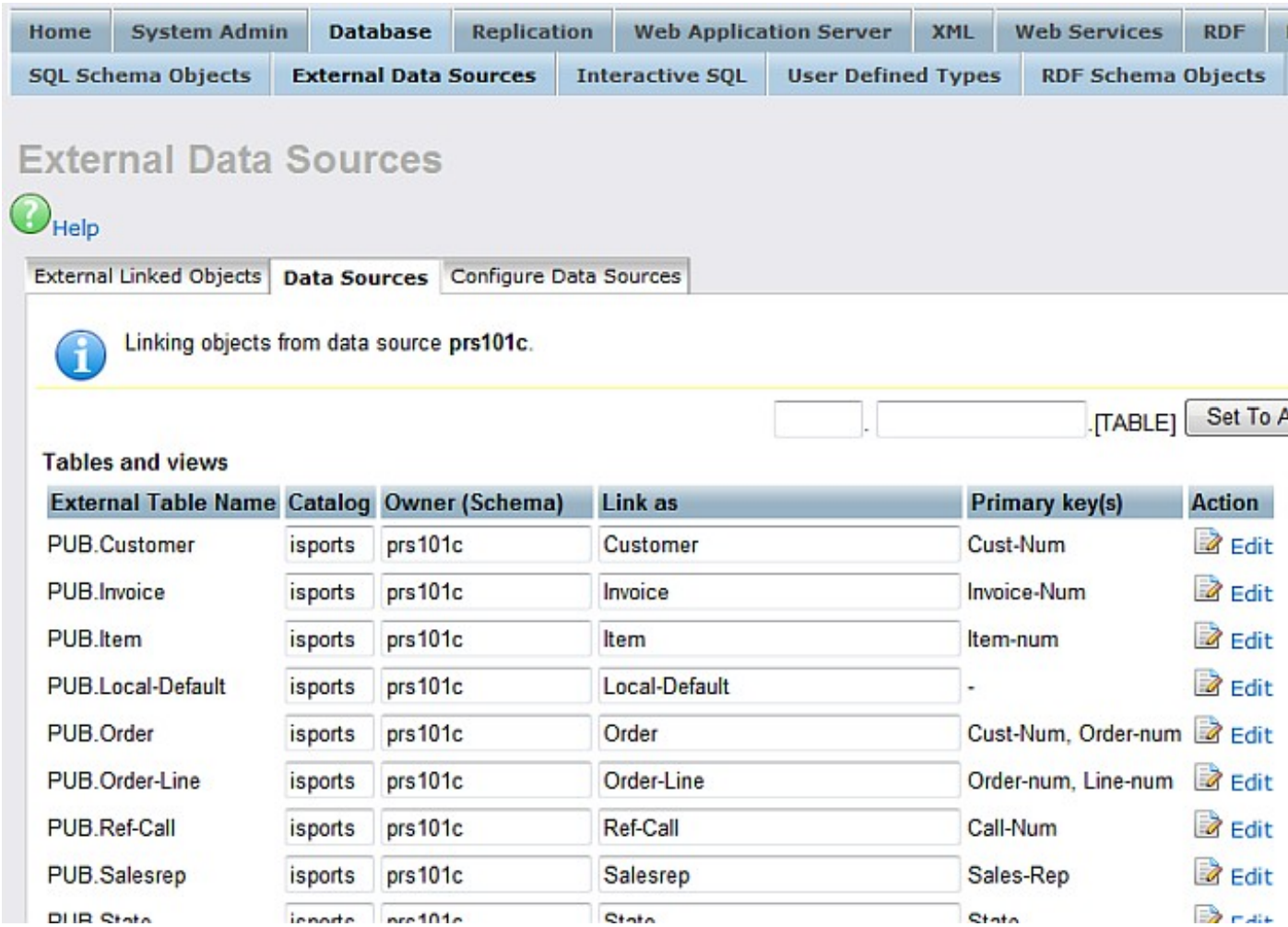
7. Change the Catalog for each table to be "isports" using the "Set All" button.

Figure 8.73. Catalog



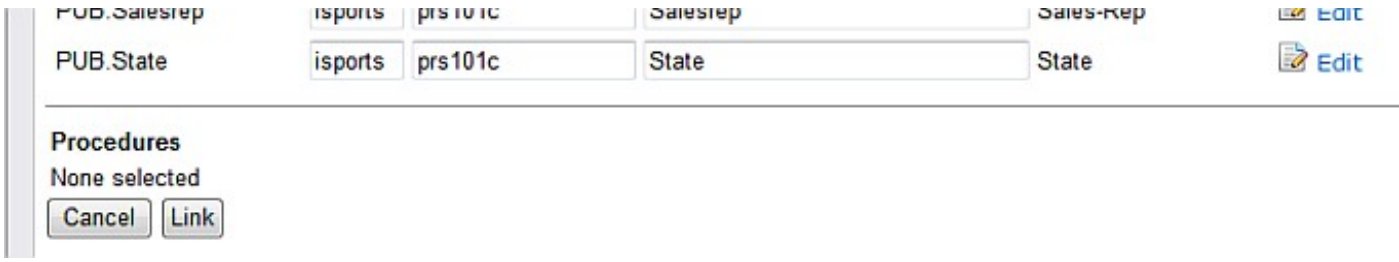
8. All the catalog names are changed to be "isports".

Figure 8.74. Catalog names



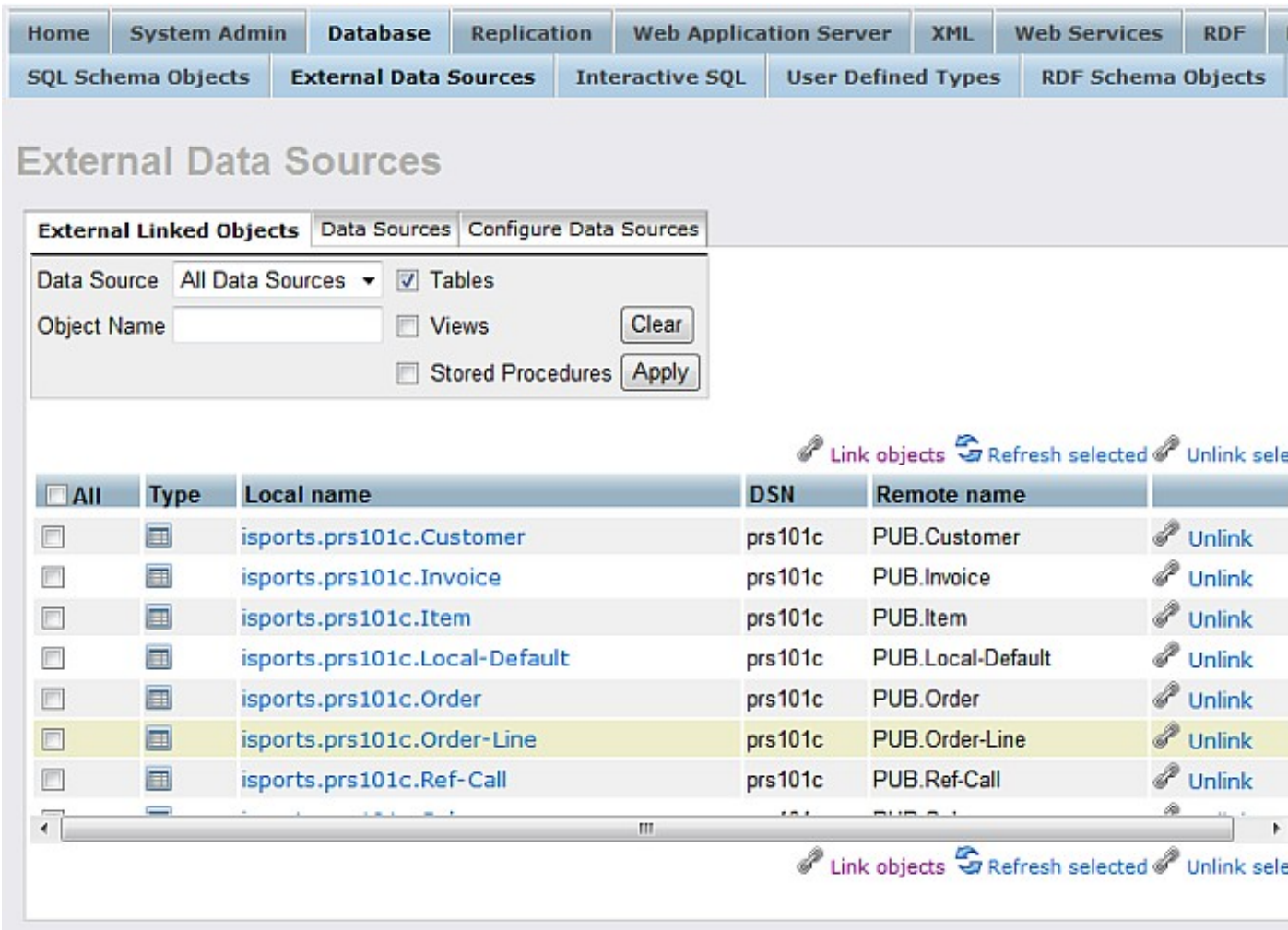
9. Select the "Link" button to link the selected tables into Virtuoso

Figure 8.75. "Link" button



10. On completion of the Link process the tables will be displayed in the "External Linked Objects" tab.

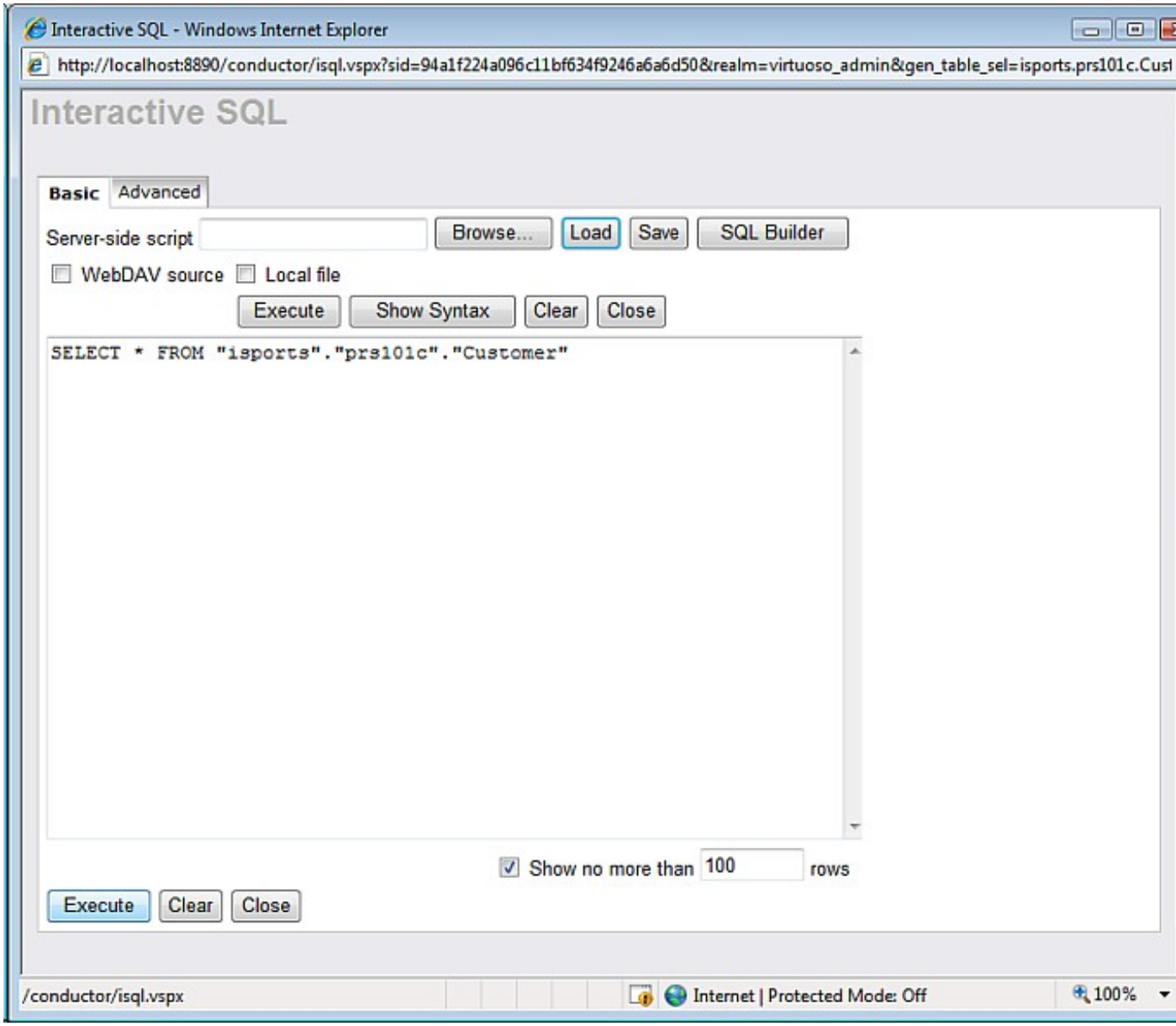
Figure 8.76. Completion



11. At this point you can test the remotely linked tables by clicking on the link that accompanies each table. e.g. isports.prs101c.Customer.

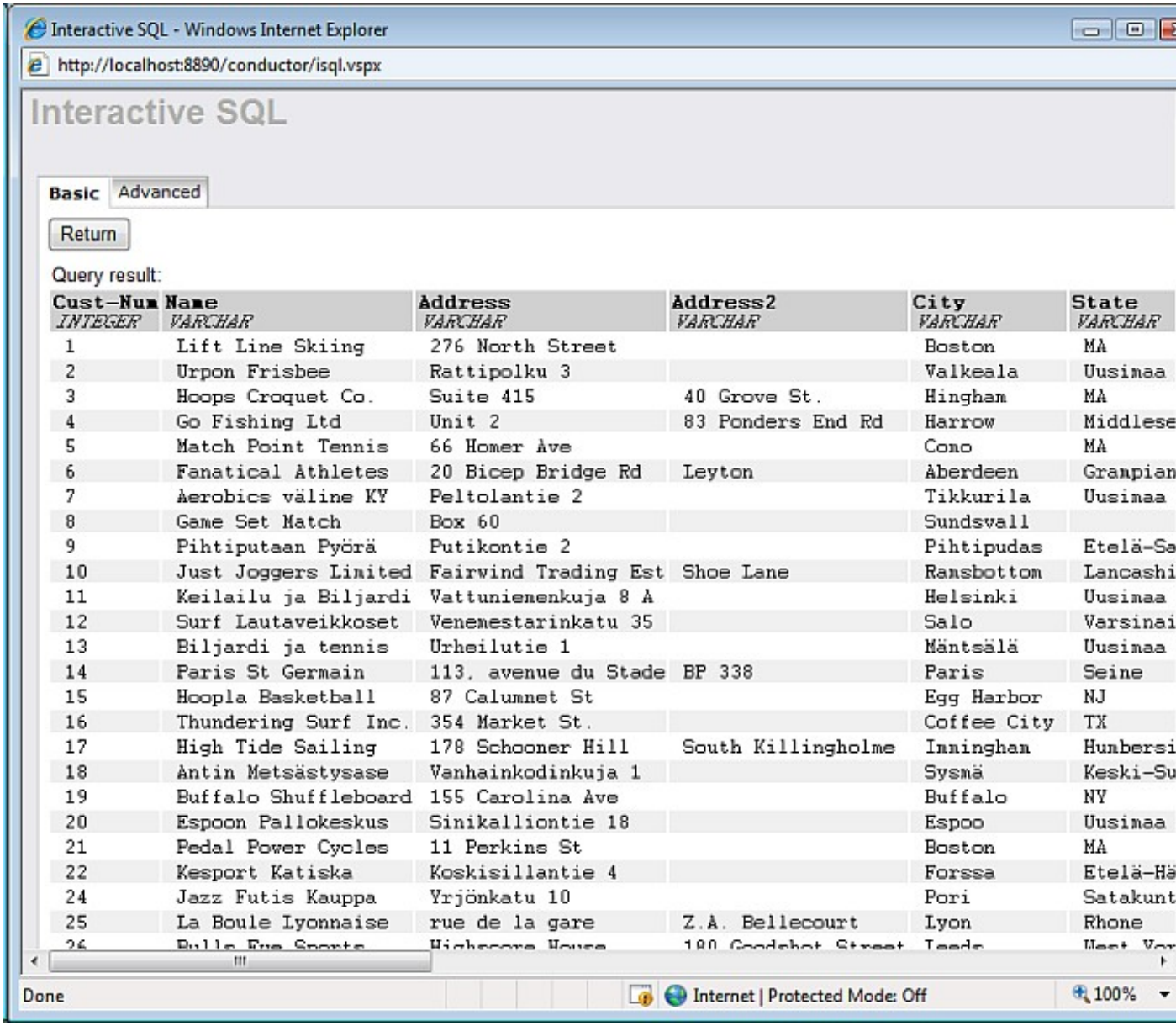
    This will display the Interactive ISQL interface which will have been already populated with a suitable SQL statement.

Figure 8.77. Querying



12. Select Execute to see data from the remotely linked table.

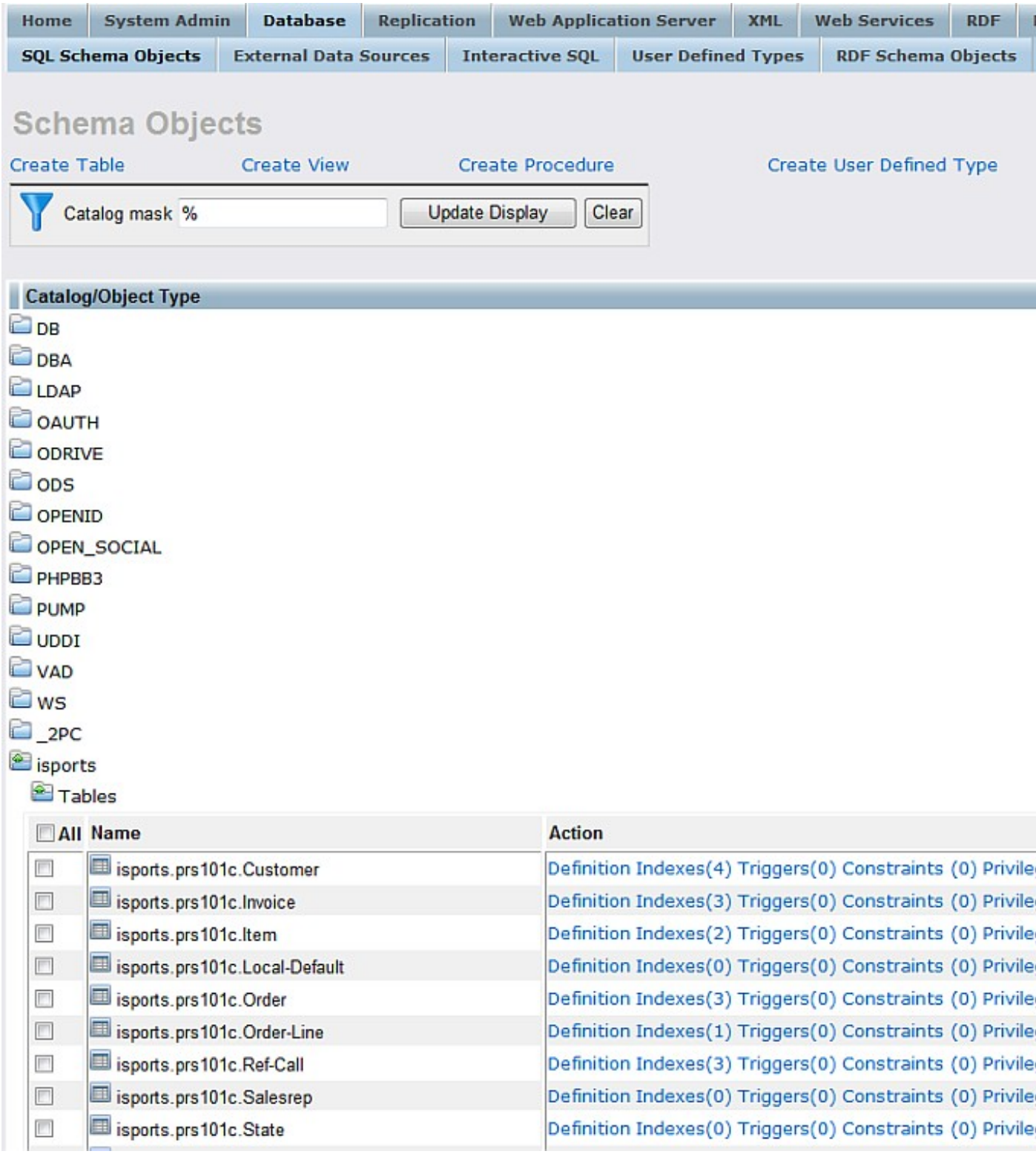
Figure 8.78. Execute



Cust-Num INTEGER	Name VARCHAR	Address VARCHAR	Address2 VARCHAR	City VARCHAR	State VARCHAR
1	Lift Line Skiing	276 North Street		Boston	MA
2	Urpon Frisbee	Rattipolku 3		Valkeala	Uusimaa
3	Hoops Croquet Co.	Suite 415	40 Grove St.	Hingham	MA
4	Go Fishing Ltd	Unit 2	83 Ponders End Rd	Harrow	Middlese
5	Match Point Tennis	66 Homer Ave		Cono	MA
6	Fanatical Athletes	20 Bicep Bridge Rd	Leyton	Aberdeen	Grampian
7	Aerobics väline KY	Peltolantie 2		Tikkurila	Uusimaa
8	Game Set Match	Box 60		Sundsvall	
9	Pihtiputaan Pyörä	Putikontie 2		Pihtipudas	Etelä-Sa
10	Just Joggers Limited	Fairvind Trading Est	Shoe Lane	Ramsbottom	Lancashi
11	Keilailu ja Biljardi	Vattuniemenkuja 8 A		Helsinki	Uusimaa
12	Surf Lautaveikkoset	Venemestarinkatu 35		Salo	Varsinai
13	Biljardi ja tennis	Urheilutie 1		Mäntsälä	Uusimaa
14	Paris St Germain	113, avenue du Stade	BP 338	Paris	Seine
15	Hoopla Basketball	87 Calumnet St		Egg Harbor	NJ
16	Thundering Surf Inc.	354 Market St.		Coffee City	TX
17	High Tide Sailing	178 Schooner Hill	South Killingholme	Inninghan	Humbersi
18	Antin Metsästysase	Vanhainkodinkuja 1		Sysmä	Keski-Su
19	Buffalo Shuffleboard	155 Carolina Ave		Buffalo	NY
20	Espoon Pallokeskus	Sinikalliontie 18		Espoo	Uusimaa
21	Pedal Power Cycles	11 Perkins St		Boston	MA
22	Kesport Katiska	Koskisillantie 4		Forssa	Etelä-Hä
24	Jazz Futis Kauppa	Yrjönkatu 10		Pori	Satakunt
25	La Boule Lyonnaise	rue de la gare	Z.A. Bellecourt	Lyon	Rhone

13. The tables can also be viewed as part of the Virtuoso "SQL Schema Objects" under the "isports" catalog name.

Figure 8.79. SQL Schema Objects



The Link process is now complete enabling the tables to be queried as if part of the Virtuoso Schema.

8.3.4. Creating EDM in Visual Studio 2008

The following steps can be used to create an Entity Data Model (EDM) for the Progress isports database:

1. Launch the Visual Studio 2008 SP1 IDE.

Figure 8.80. Visual Studio 2008 SP1 IDE

2. Create a

   Web Application

   project by going to the

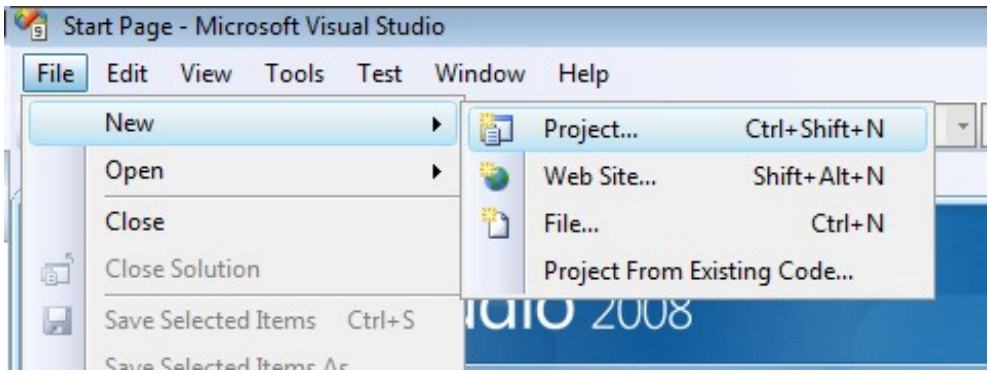
   File

   menu in Visual Studio and choosing

   New Project

   .

   Figure 8.81. Web Application



3. When the New Project window appears, choose either

   Visual Basic

   or

   Visual C#

   as your programming language.
4. Within the language category, click on

   Web

   and select

   ASP.NET Web Application

from the right-hand panel.

5. Choose a name for the project, for example

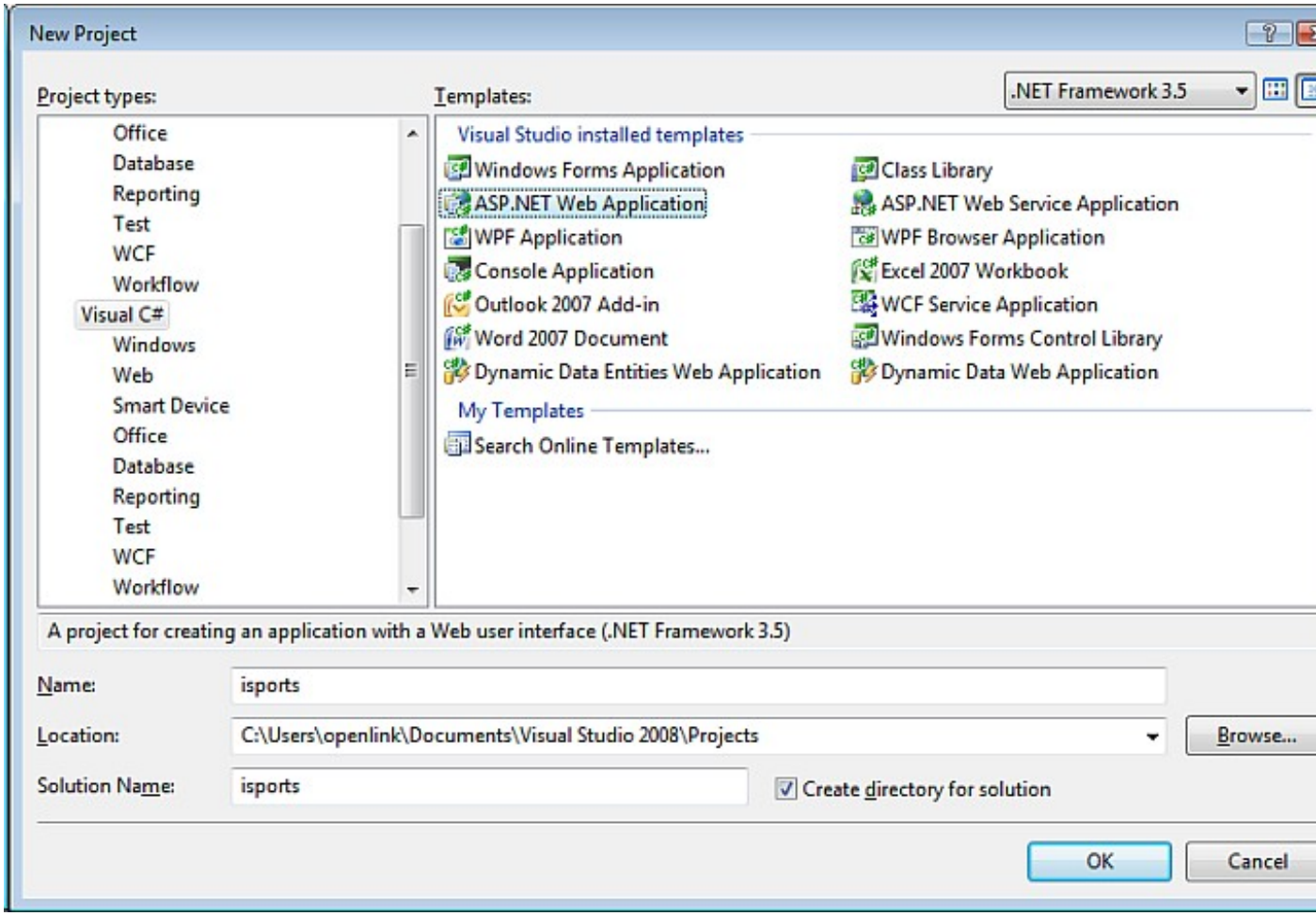
   isports

   , and click

   OK

   .

Figure 8.82. Name for the project

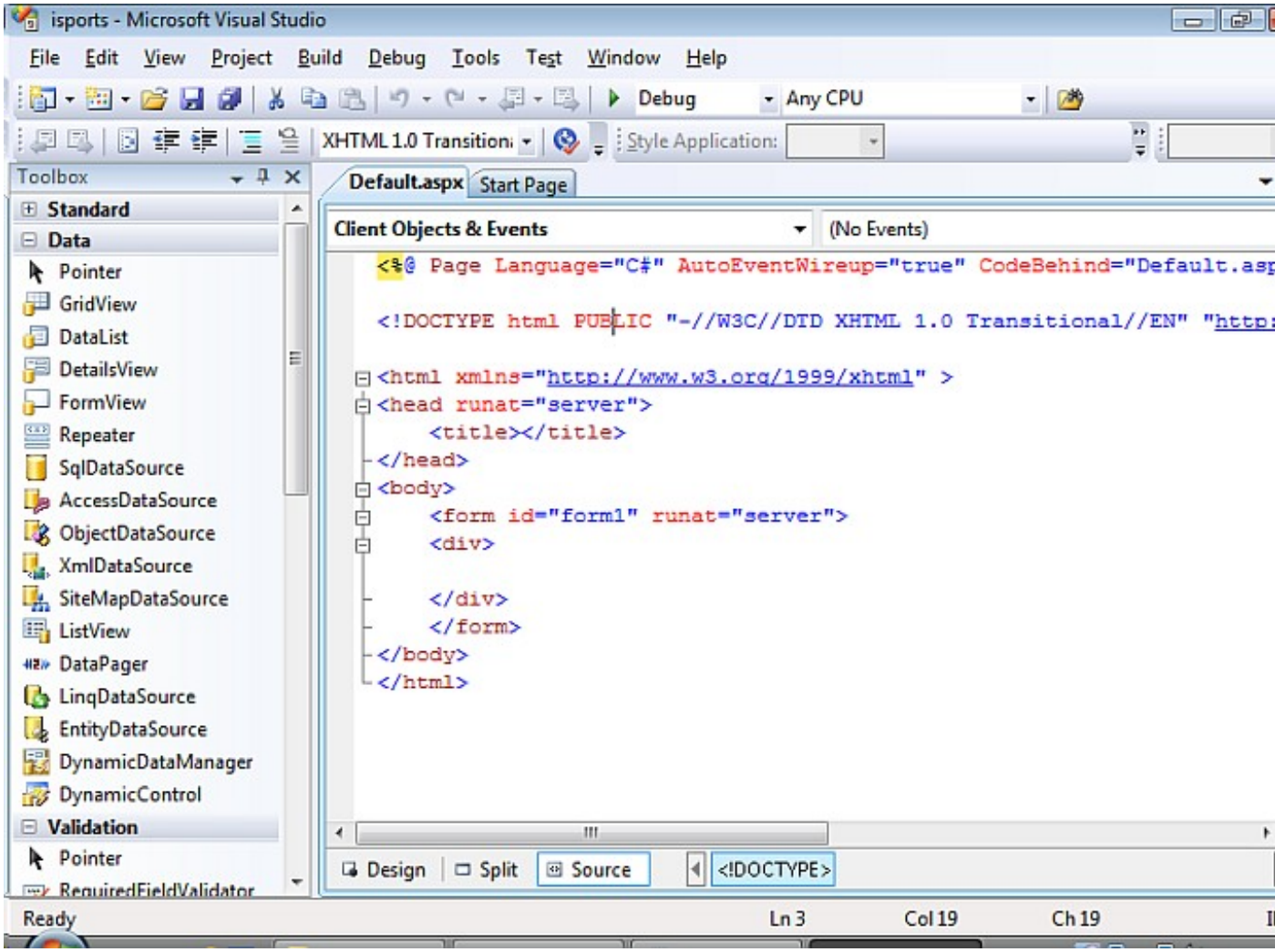


6. This will create a new project called

   isports

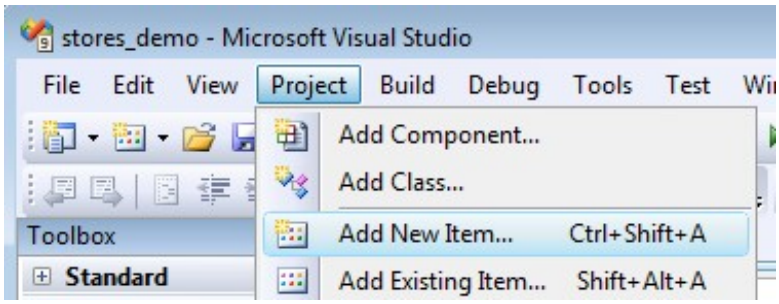
   .

Figure 8.83. create a new project



7. Select the Project -> Add New Item menu option.

Figure 8.84. New Item



8. The

Add New Item

dialog will appear. Choose the

ADO.NET Entity Data Model

template. Give it the name

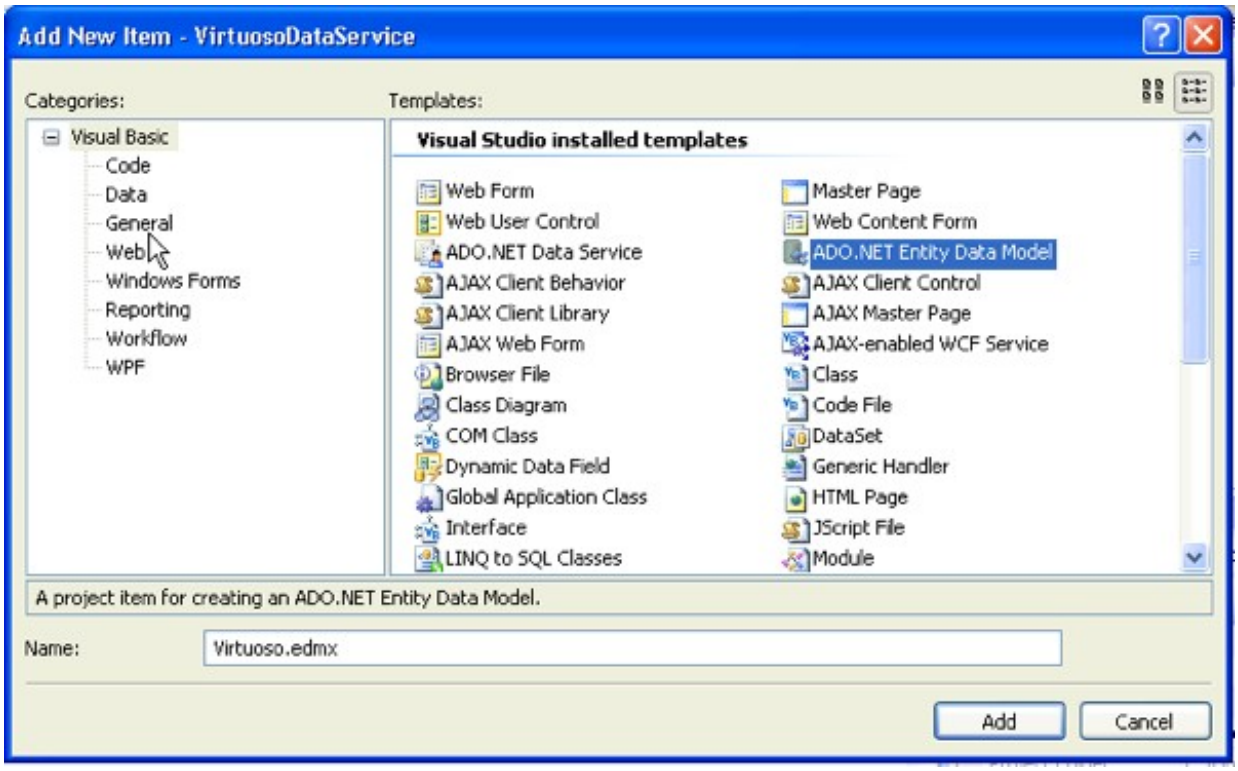
Model1.edmx

and click

Add

to start the creation of the ADO.Net Entity Data Model.

Figure 8.85. Add New Item



9. In the

Entity Data Model Wizard

dialog

Choose Model Contents

page select the

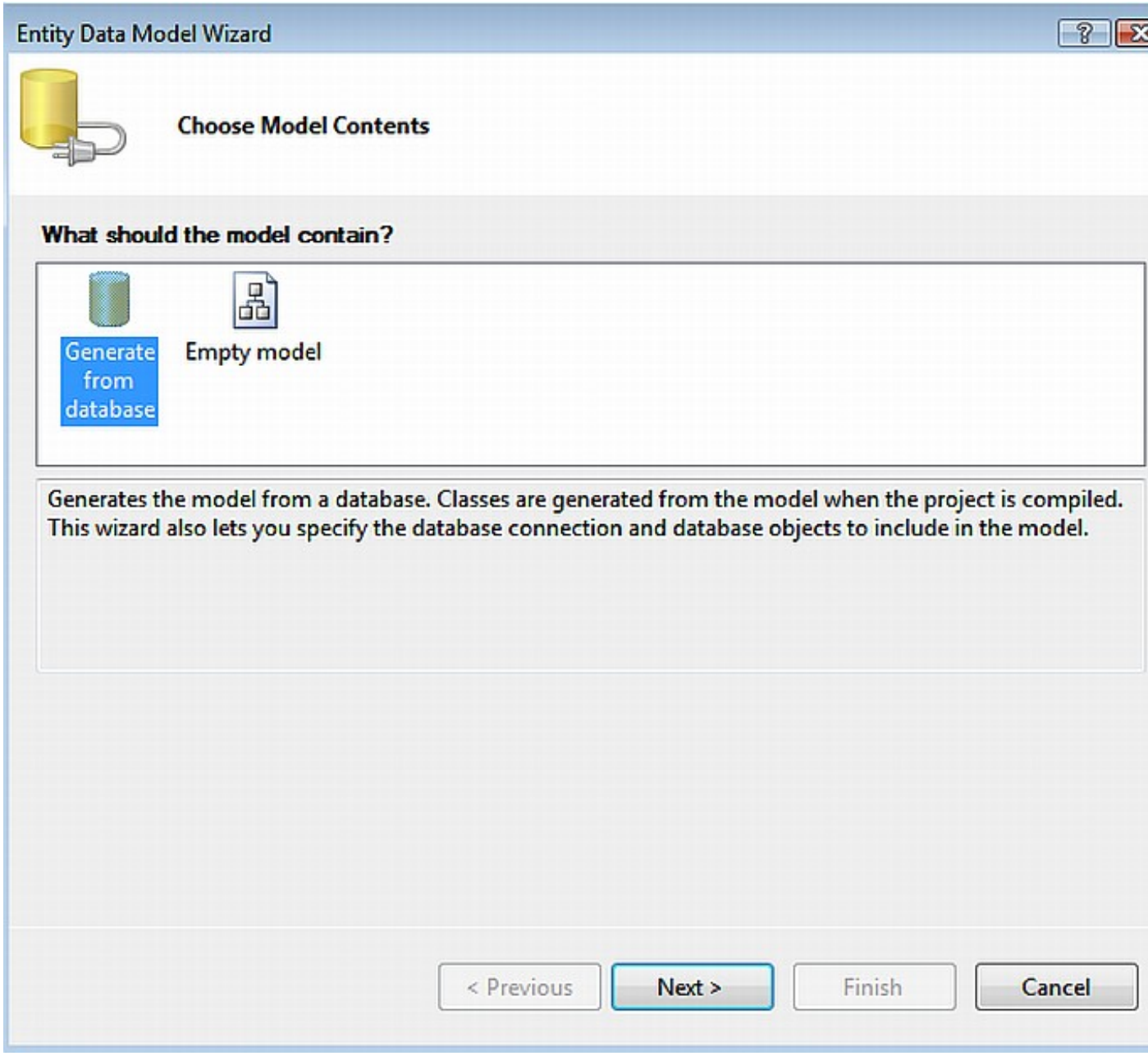
Generate from Database

model type and click

Next

.

Figure 8.86. Choose Model Contents



10. In the

    Entity Data Model Wizard

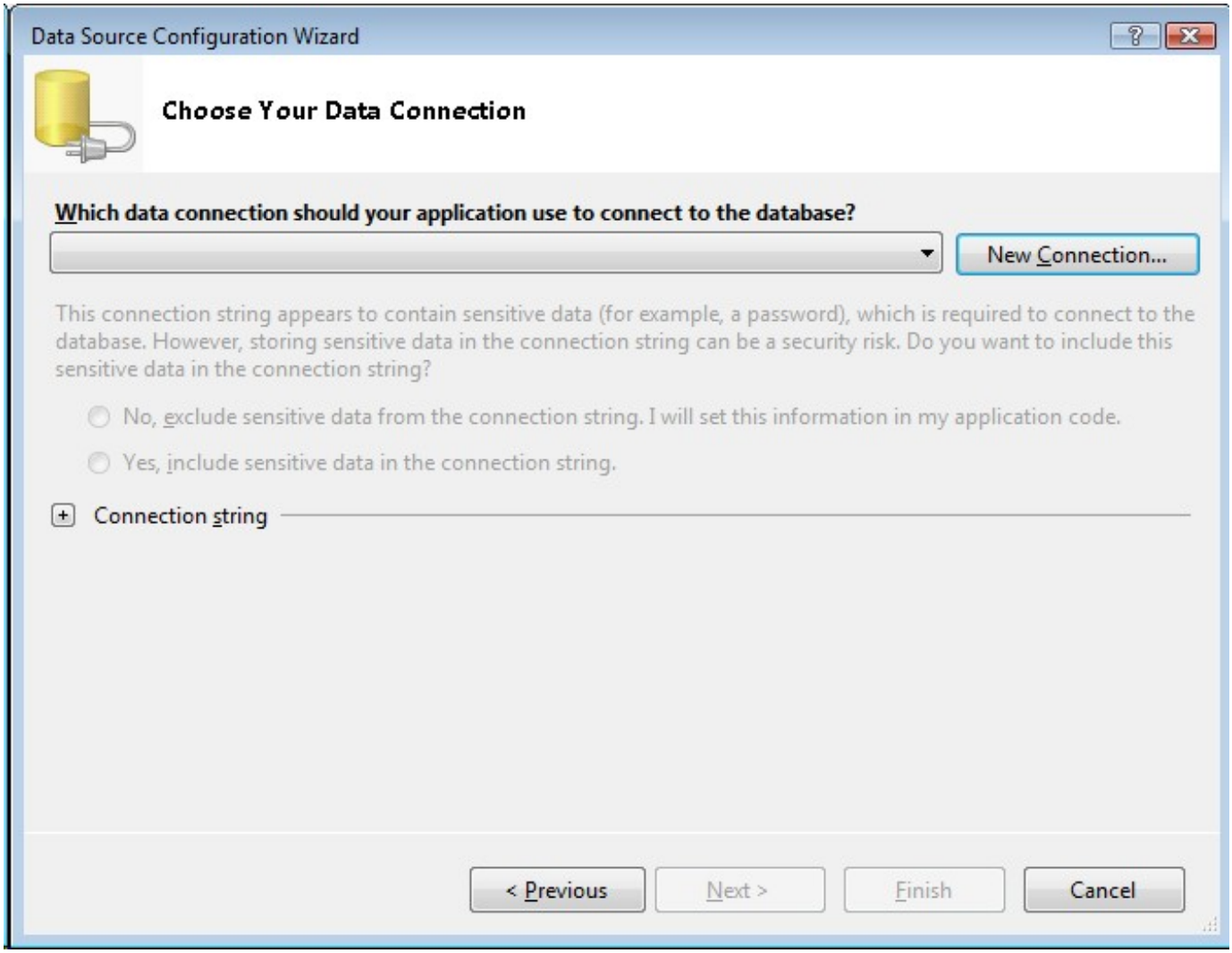
    dialog

    Choose your Data Connection

    page, select the

    New Connection button

Figure 8.87. Entity Data Model Wizard



11. In the

    Choose Data Source

    dialog, select the OpenLink

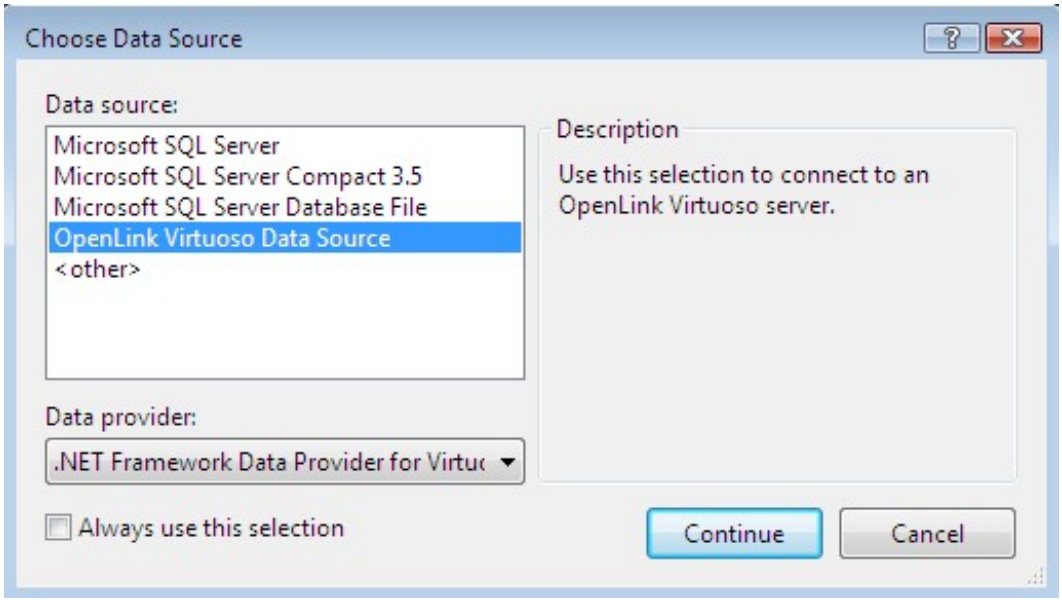
    Virtuoso Data Source

    from the list and click

    Continue

    .

Figure 8.88. Choose Data Source



12. In the

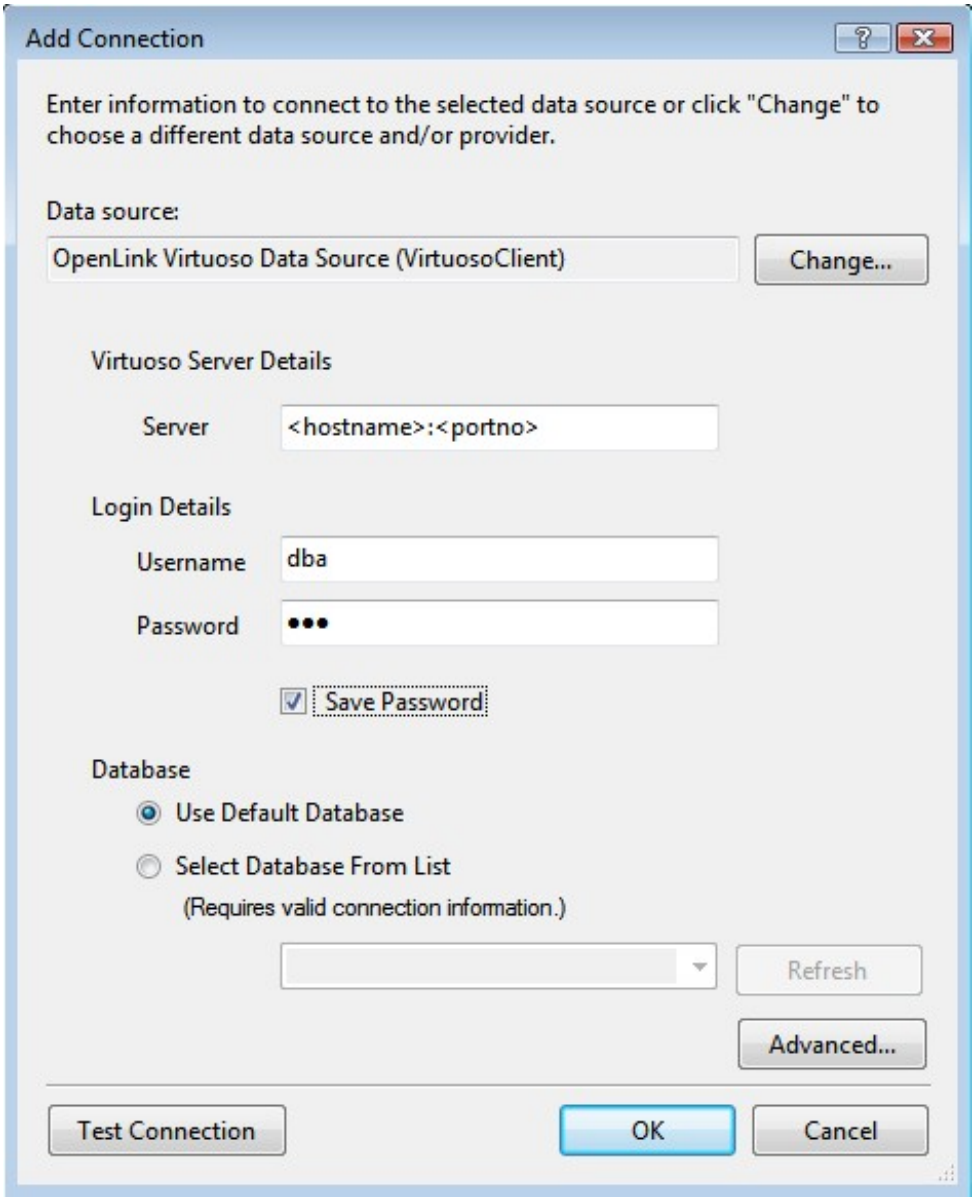
    Add Connection

    dialog, specify the

    hostname, portno, username and password

    for the target Virtuoso Server and check the Save Password check box.

    Figure 8.89. Connection Properties



13. Select the

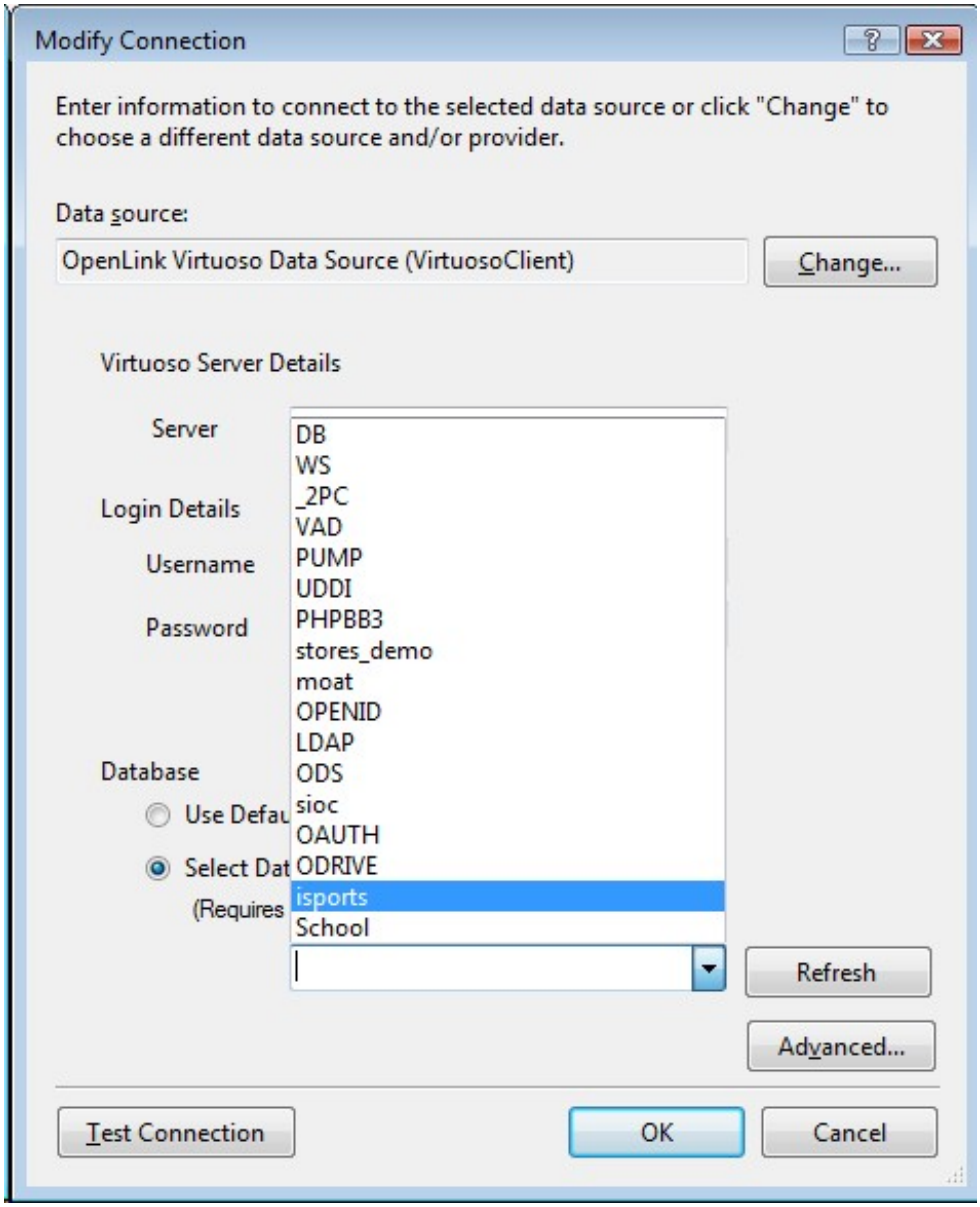
    Select Database From List

    radio button and choose the

    isports

    database from the drop down list.

Figure 8.90. Add connection

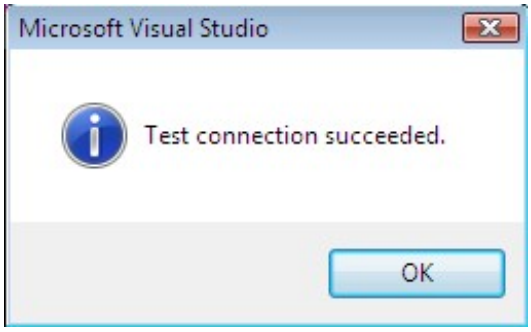


14. Press the

    Test Connection

    dialog to verify that the database is accessible.

    Figure 8.91. Test Connection



15. Set the

    entity connect string

name to

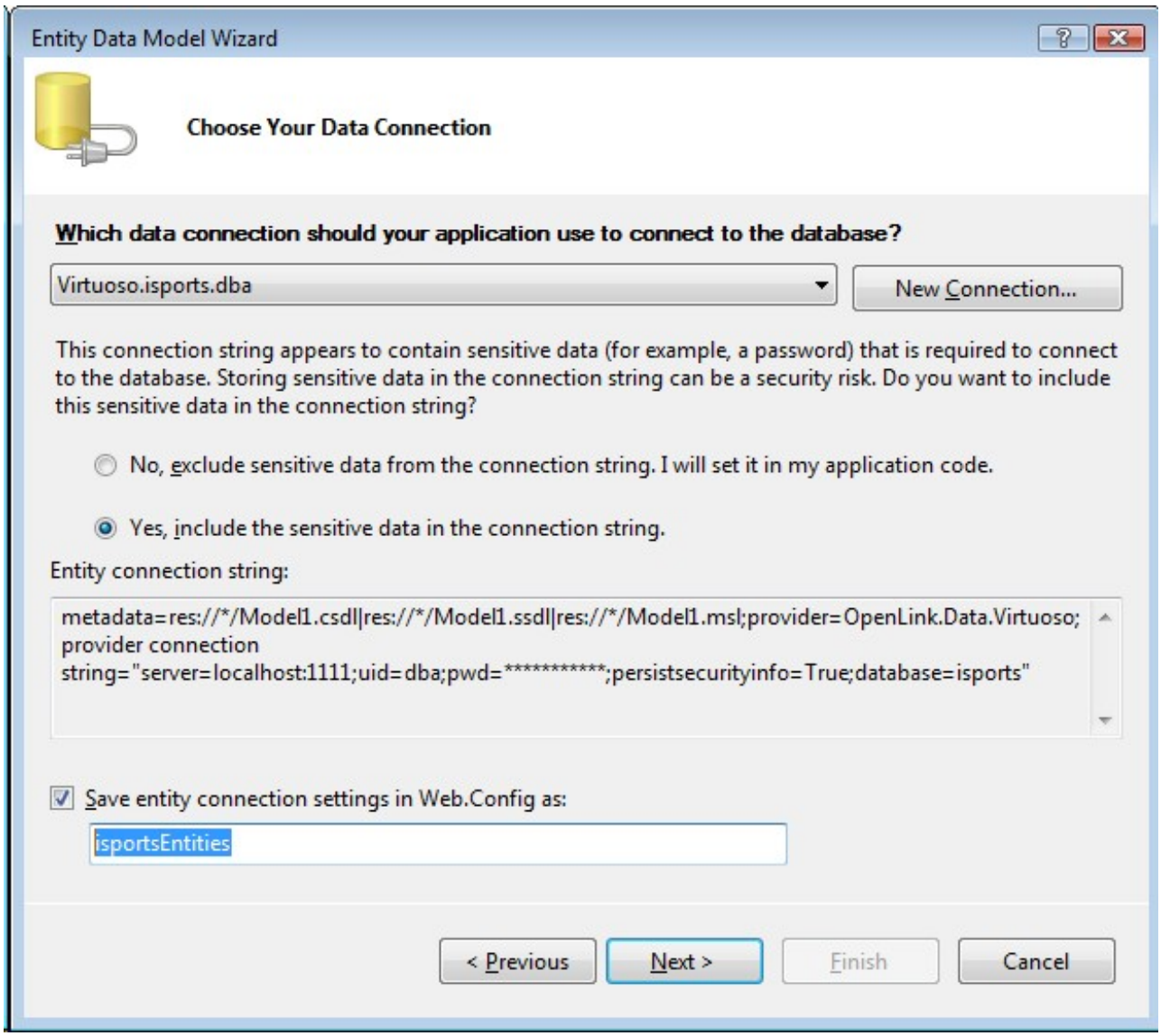
isportsEntities

(note this name as it is required in the section on creating and ADO.Net Data Service) and click

Next

.

Figure 8.92. entity connect string



16. In the

Choose your Database Objects

page select the

Tables

check box to select all tables in the isports catalog for addition to the Entity Data Model. Set the

Model Namespace

to

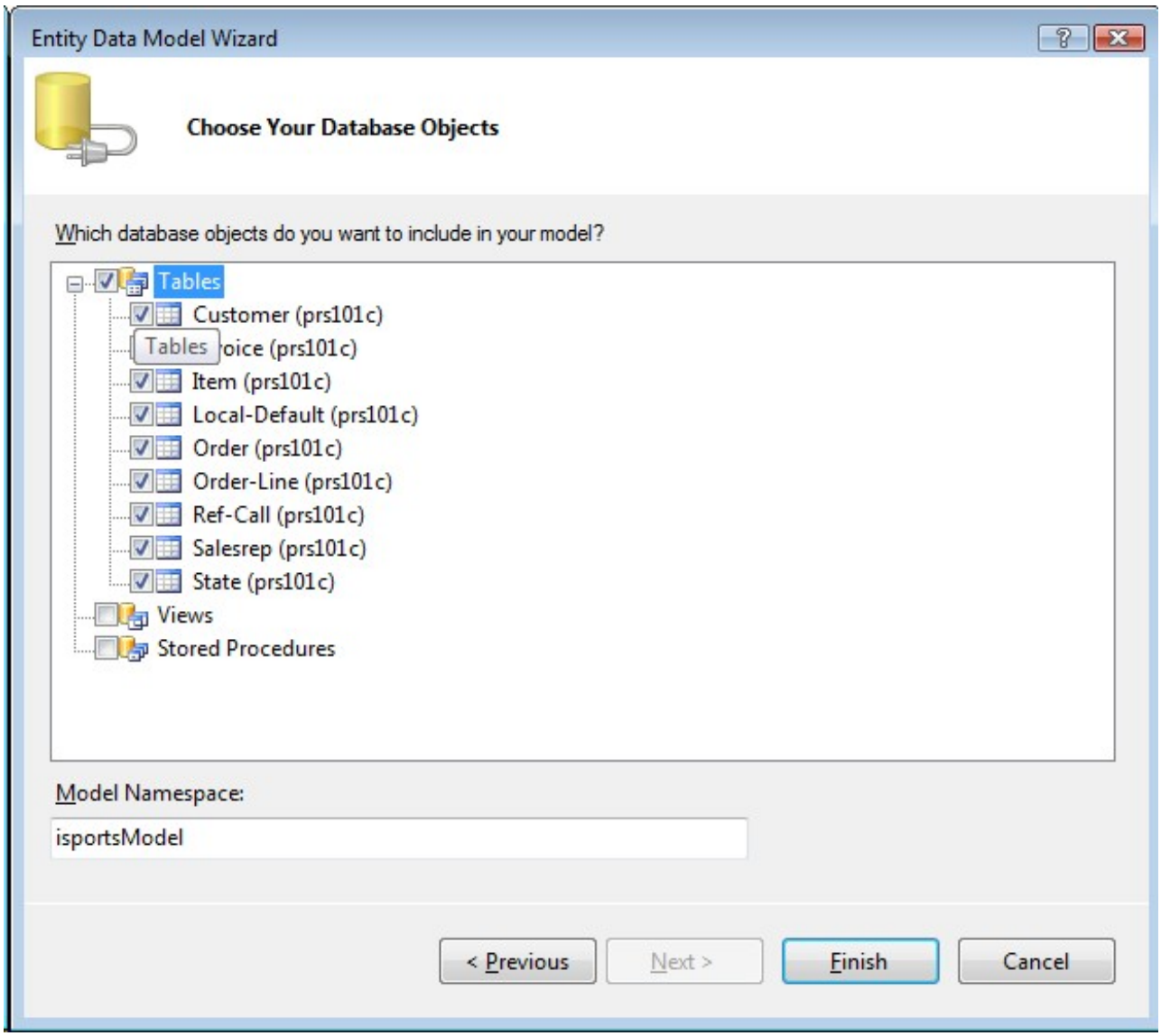
isportsModel

and click

Finish

.

Figure 8.93. Database Objects

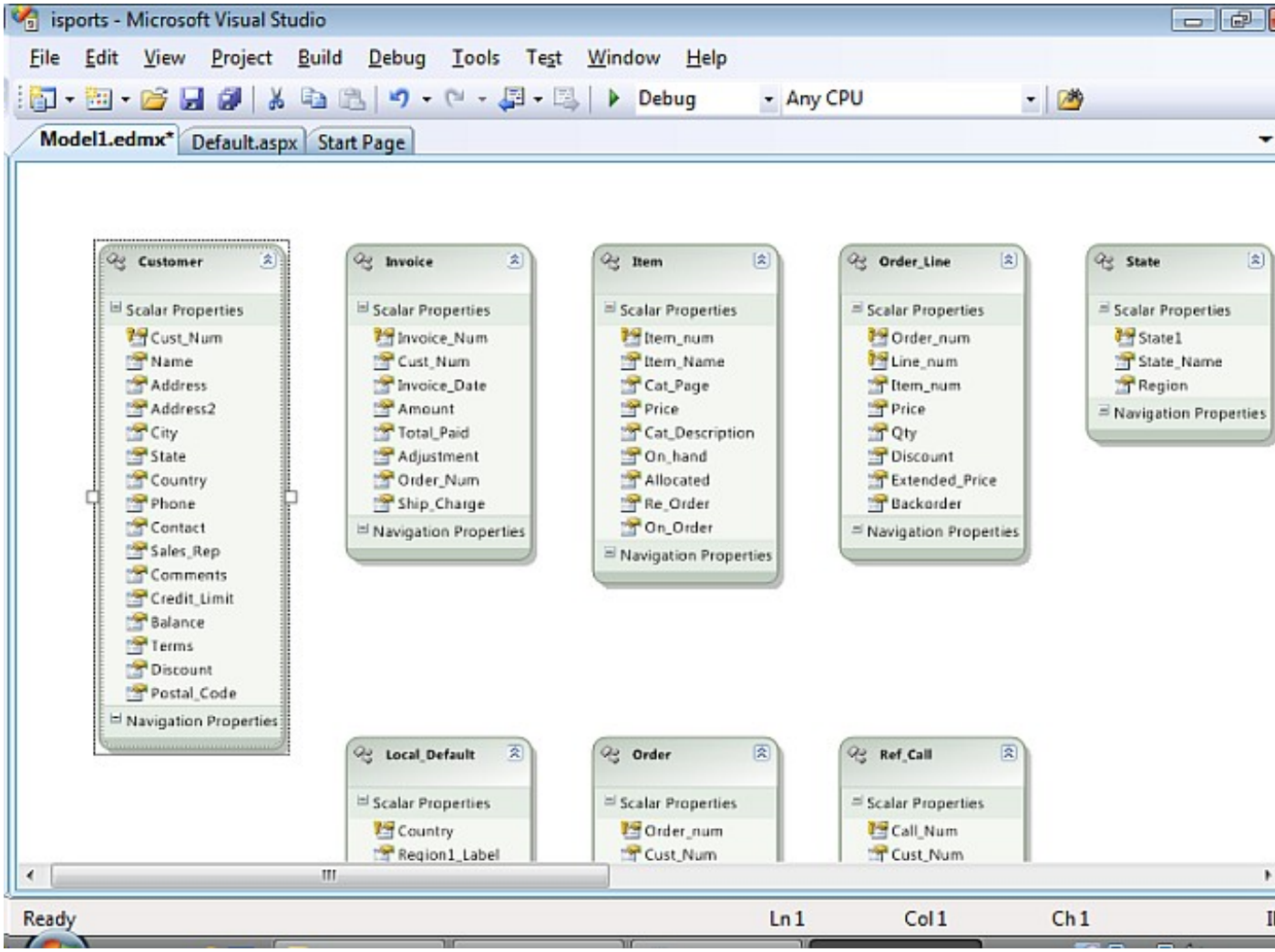


17. The

Model1.edmx

EDM will be created with the tables and relationships displayed in the Visual Studio IDE.

Figure 8.94. Virtuoso.edmx



Creation for the Entity Data Model for the Progress isports database is now complete.

8.3.5. Manually creating EDM Associations (FKs) for the Progress isports database

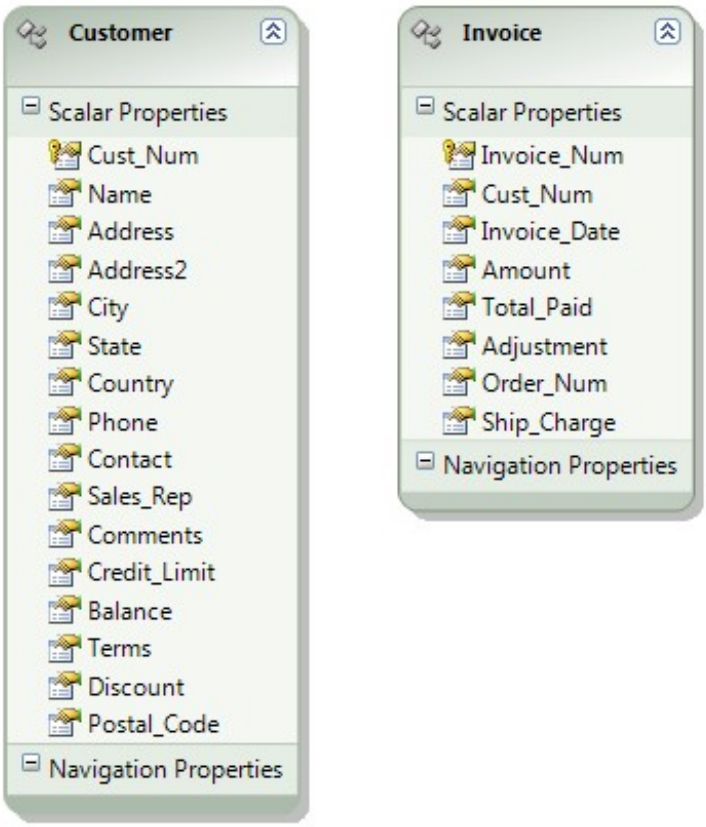
The following steps will detail what is required to manually create "Associations" in your Entity Data Model.

You will need to determine where these associations exist and their multiplicity (one-to-one, one-to-many, etc.) before commencing with the following steps.

Note: These steps will need to be repeated for each association.

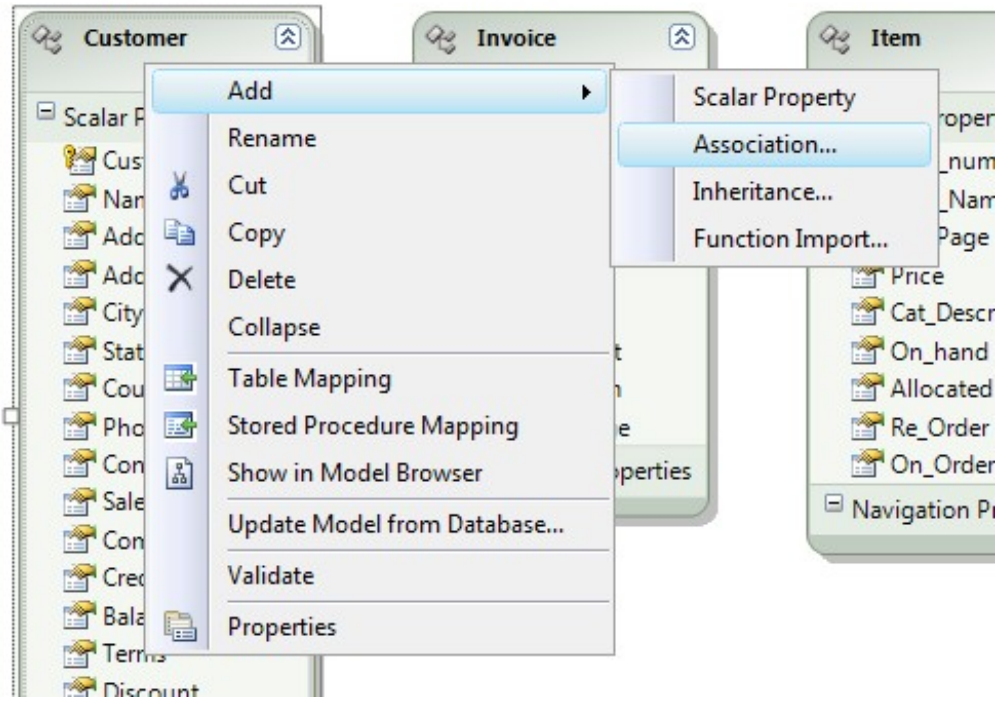
1. The first Association will deal with is the relationship between Customer and Invoice, identified by the presence of the scalar property Cust_Num in both entities. This is a one-to-many relationship, as a Customer may have any number of Invoices.

   Figure 8.95. Association



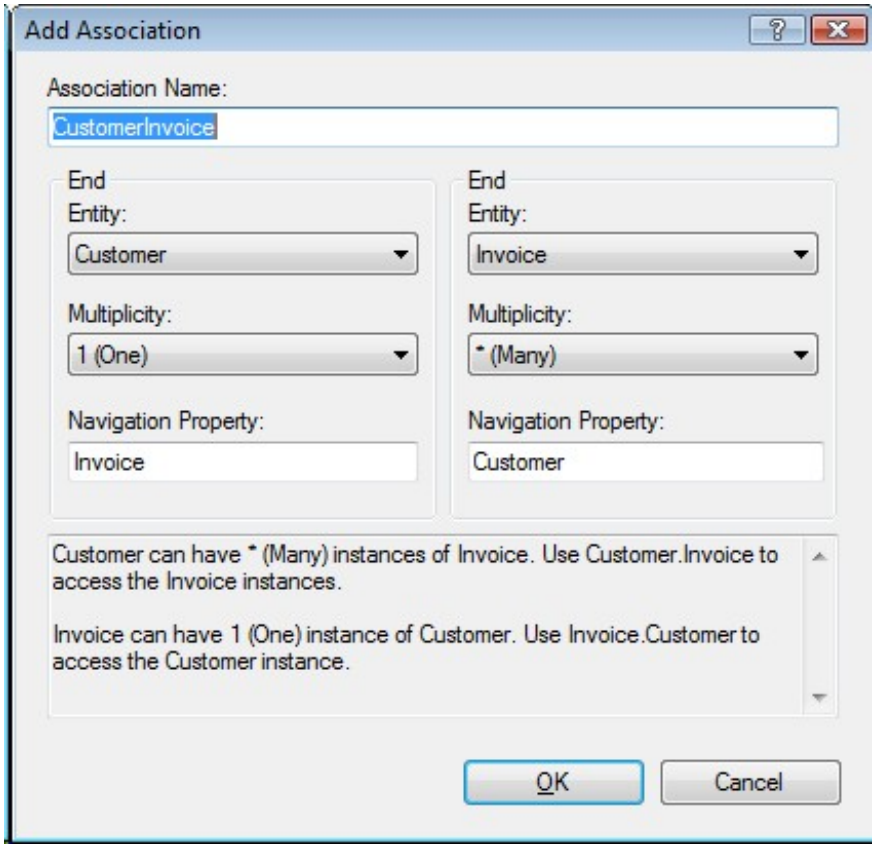
2. To add the Association, right click on the Customer entity then Add -> Association.

Figure 8.96. add the Association



3. You will now see the Add Association dialog.

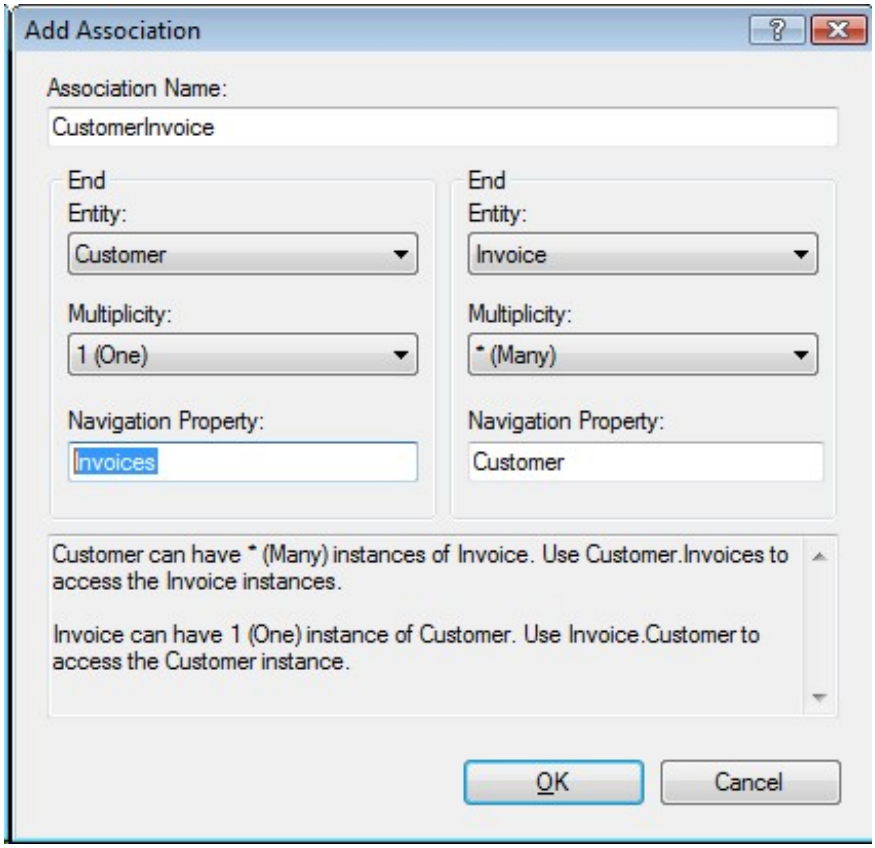
Figure 8.97. Add Association dialog



4. For this association the only thing that needs changing from the default, is the name of the Navigation Property from Invoice to invoices on the Customer end of the association.

   This better reflects the multiplicity of the association such that a Customer is associated with zero or many Invoices (plural).

Figure 8.98. Navigation Property name

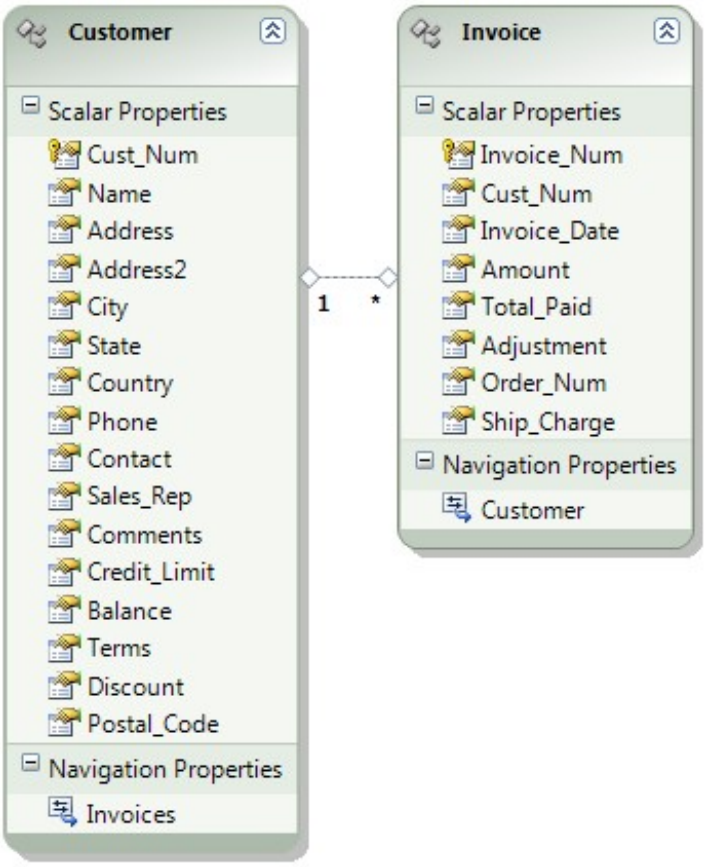


Typically, here is where you will:

- Select the entity at each end of the association
- Select the multiplicity of each end of the association, and
- Provide suitable names for the association and its navigation properties

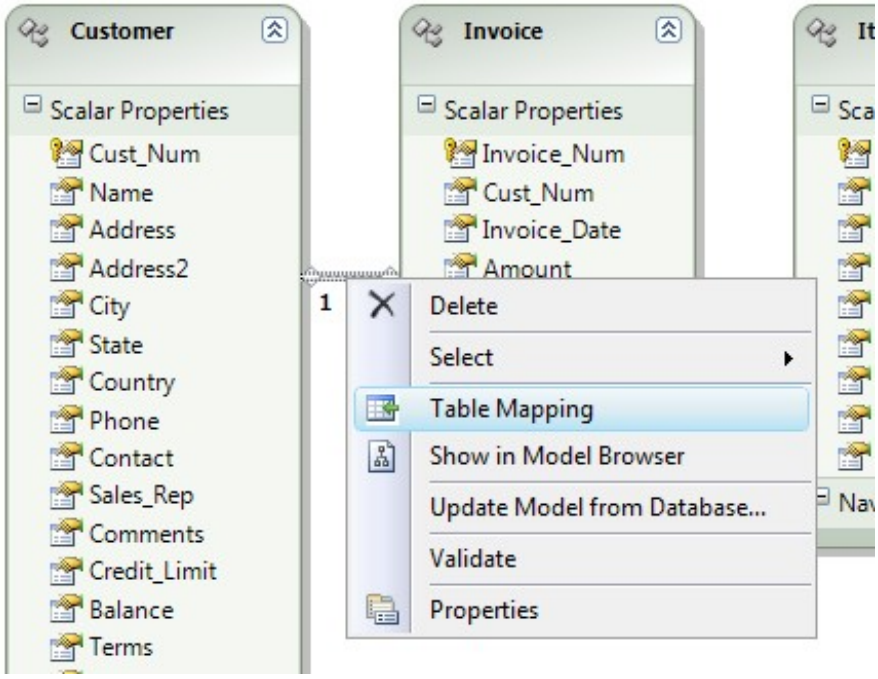
5. Once you then hit OK the diagram is refreshed to include the newly created association.

Figure 8.99. diagram



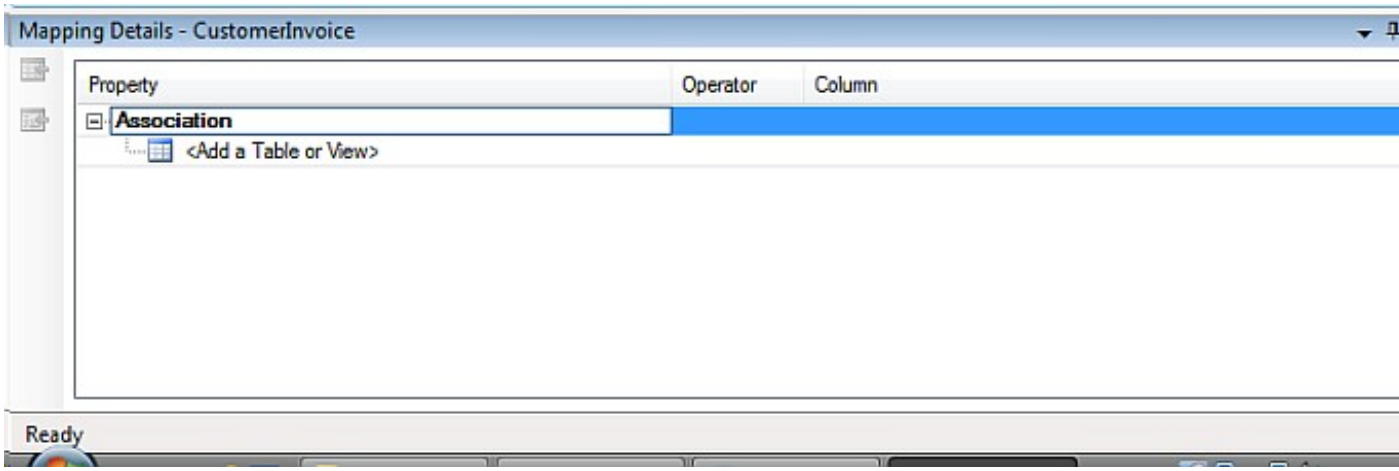
6. You now need to edit the mappings associated with the newly created association, so right-click the association on the diagram, and select Table Mapping to display the Mapping Details pane.

Figure 8.100. Table Mapping



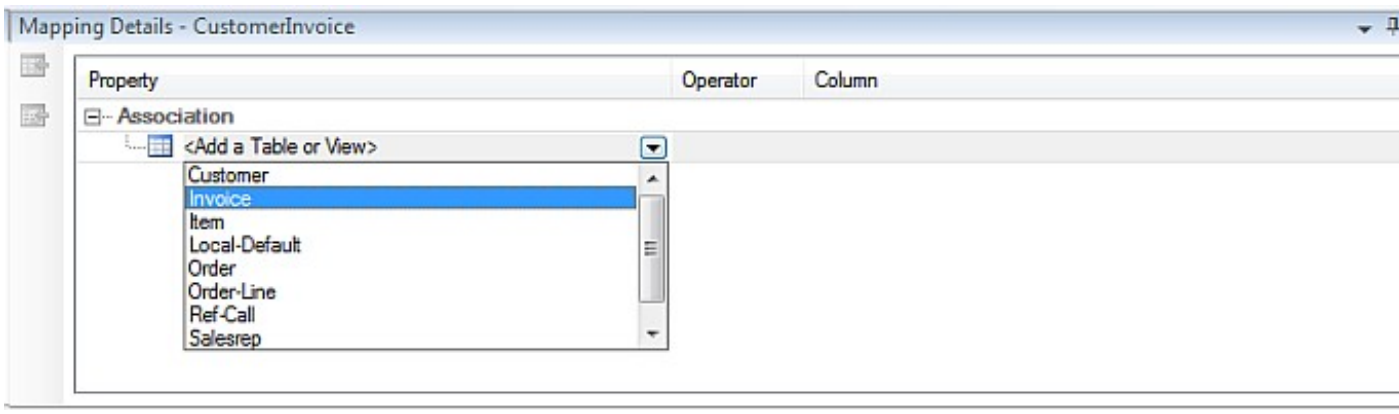
7. Click that line reading <Add a Table or View> to reveal a drop down list of all entities.

Figure 8.101. Add a Table or View



8. Here you need to select the entity on the right/far side of the association (the entity where the foreign key exists). In this example, it is the Invoice entity.

Figure 8.102. Entity



9. The Mapping Details pane now refreshes to display both ends of the association, requiring that you provide relevant target store data types in the Column column for the key fields, as depicted here.

Figure 8.103. Mapping Details

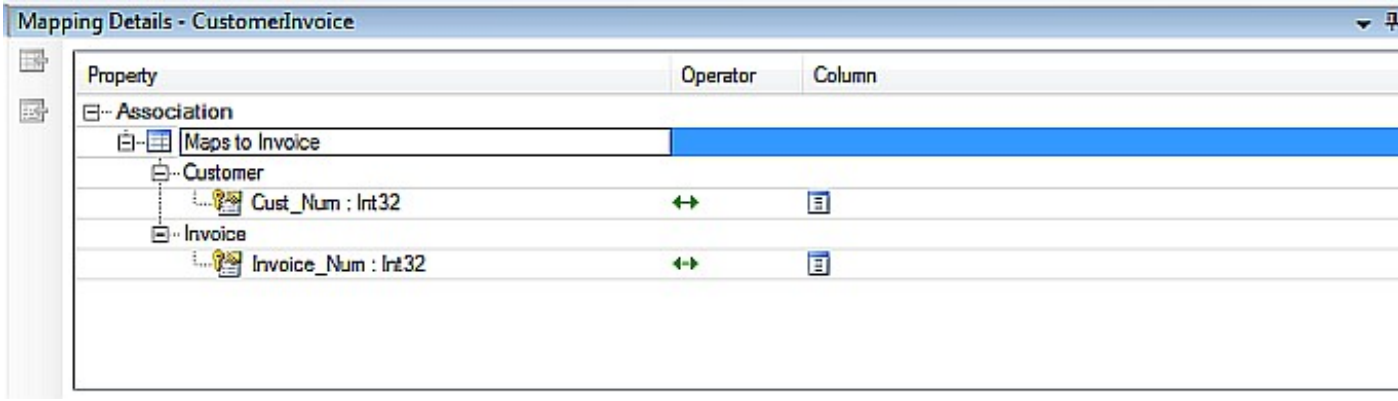


Figure 8.104. Mapping Details

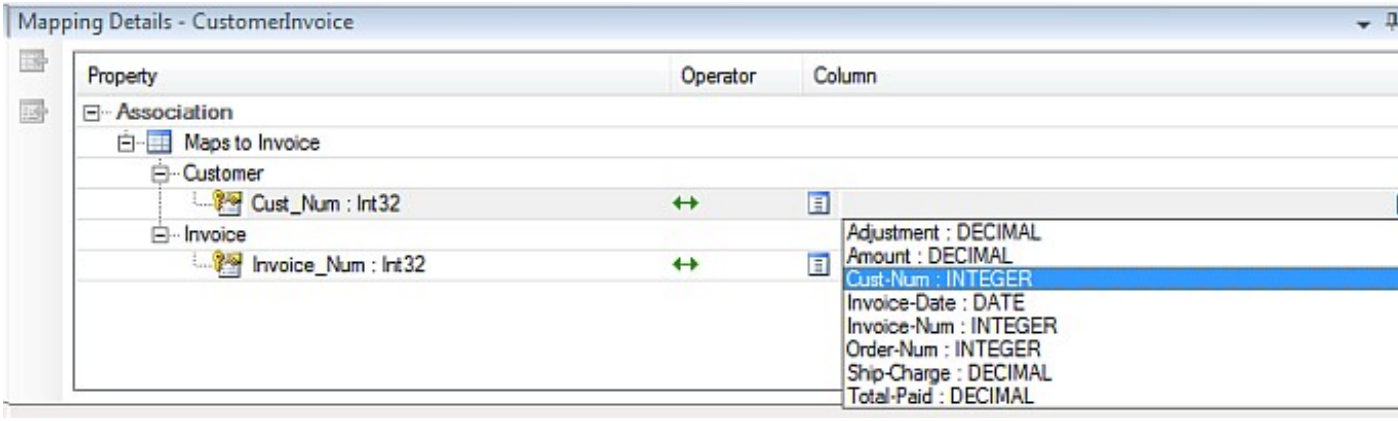
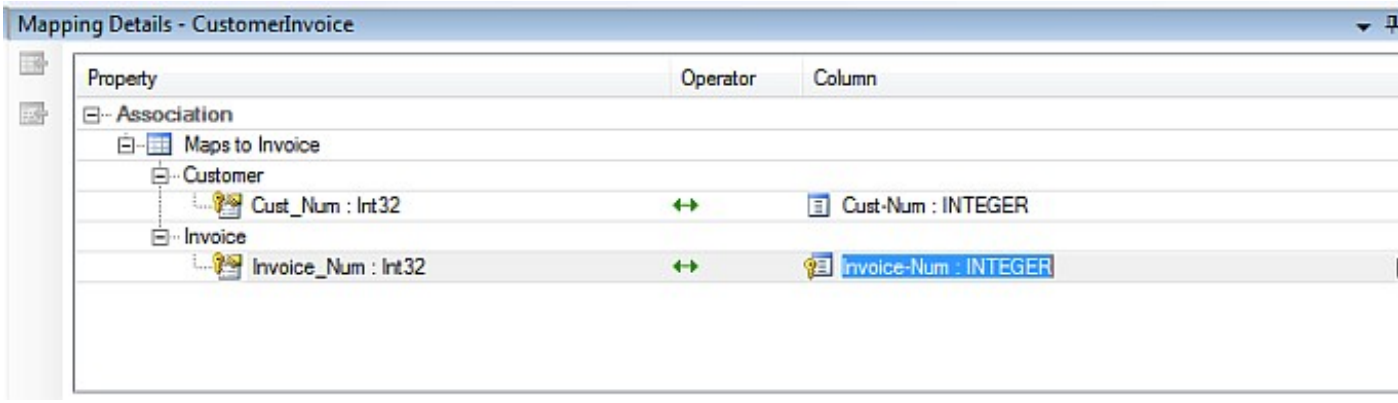
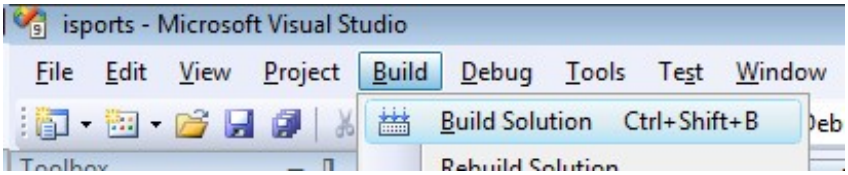


Figure 8.105. Mapping Details



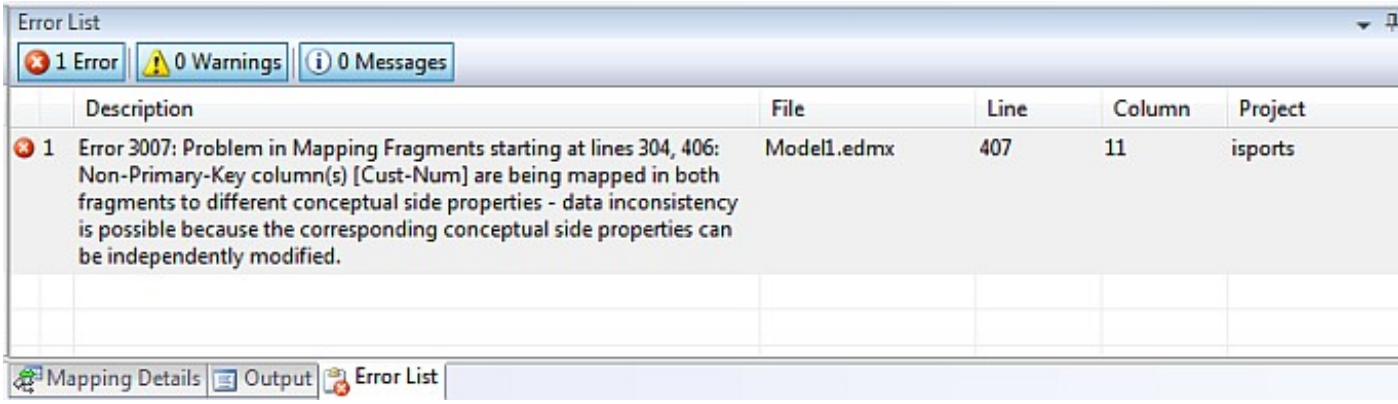
10. Once the mapping is complete, you can build the project using Build -> Build Solution. NOTE: It is worthwhile building as each association is made, since the error messages can be a little confusing.

Figure 8.106. Build the project



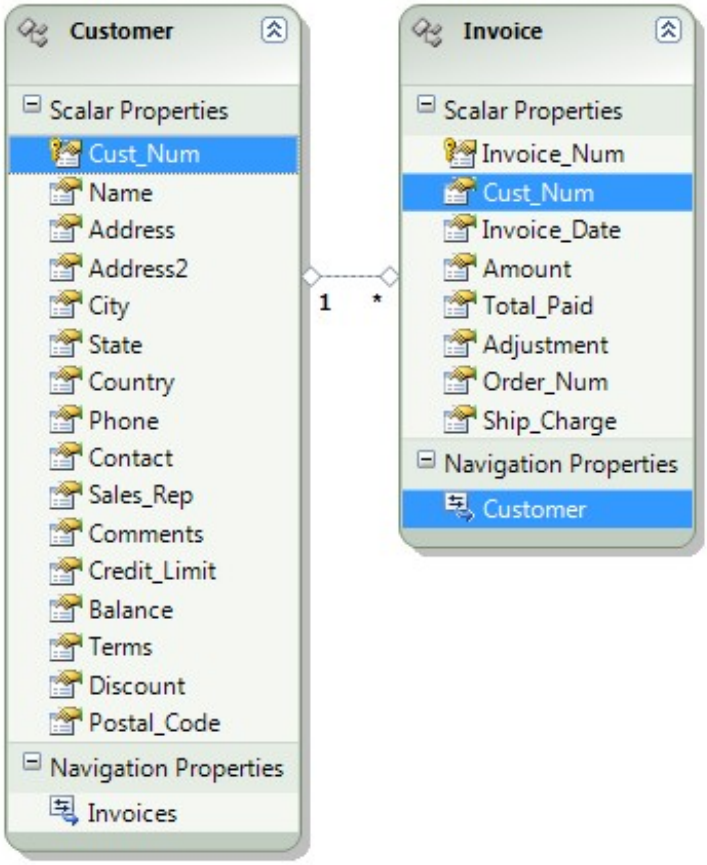
11. This should result in the following error:

Figure 8.107. Error



This error indicates that there are two source columns - in this case, the Invoice entity's Scalar Property Cust_Num and Navigation Property Customer, which are both mapped to the same target column - the Progress column Invoice.Cust_Num - and this is not supported.

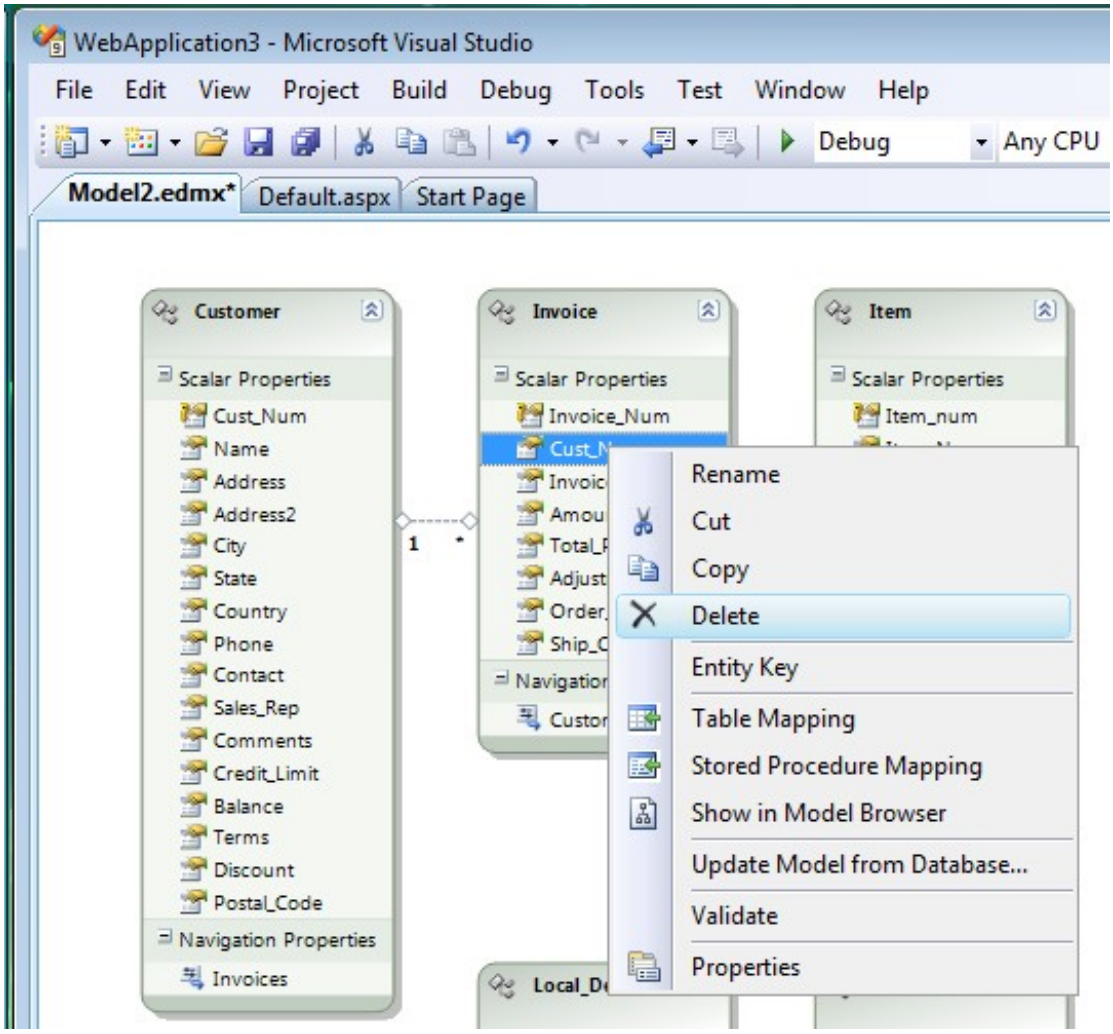
Figure 8.108. Error



12. The solution is simple! Simply delete the mapping of the Scalar Property Invoice.Cust_Num, since its only purpose is to hold data representing a relationship/association (it is a Foreign Key) which has already been represented by the newly created association and resulting Navigation Property Customer.

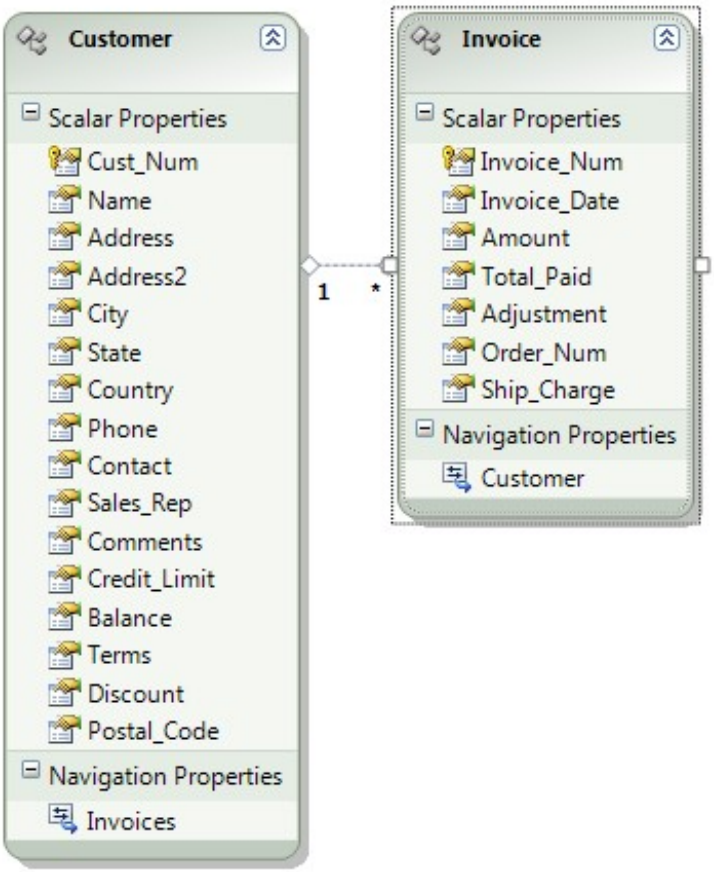
    Right click on Invoice.Cust_Num then Delete.

Figure 8.109. delete the mapping



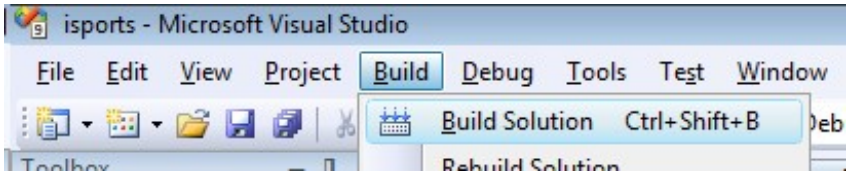
13. The model diagram will refresh to reflect this change.

Figure 8.110. Model Diagram



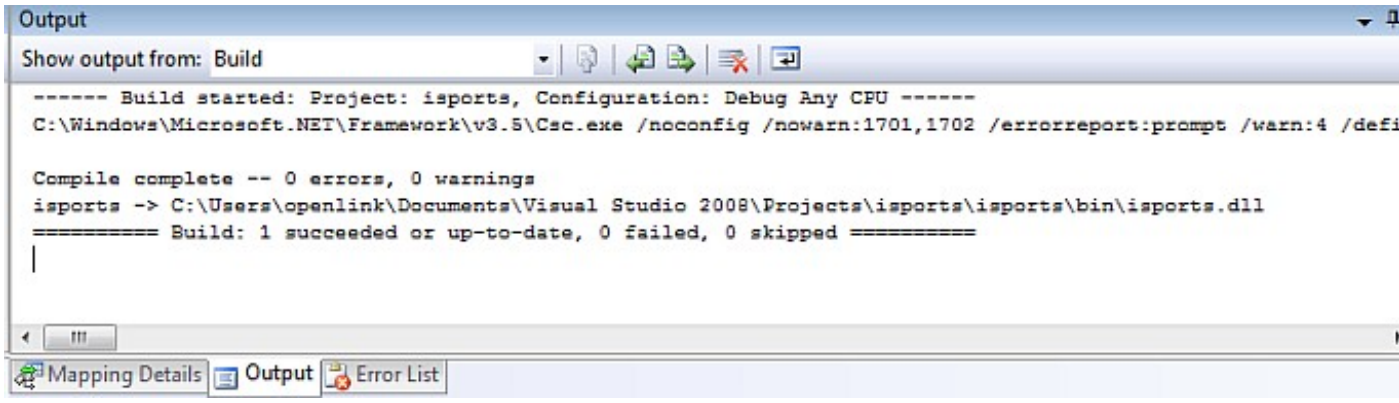
14. Build the project, again, using Build -> Build Solution.

Figure 8.111. Build project



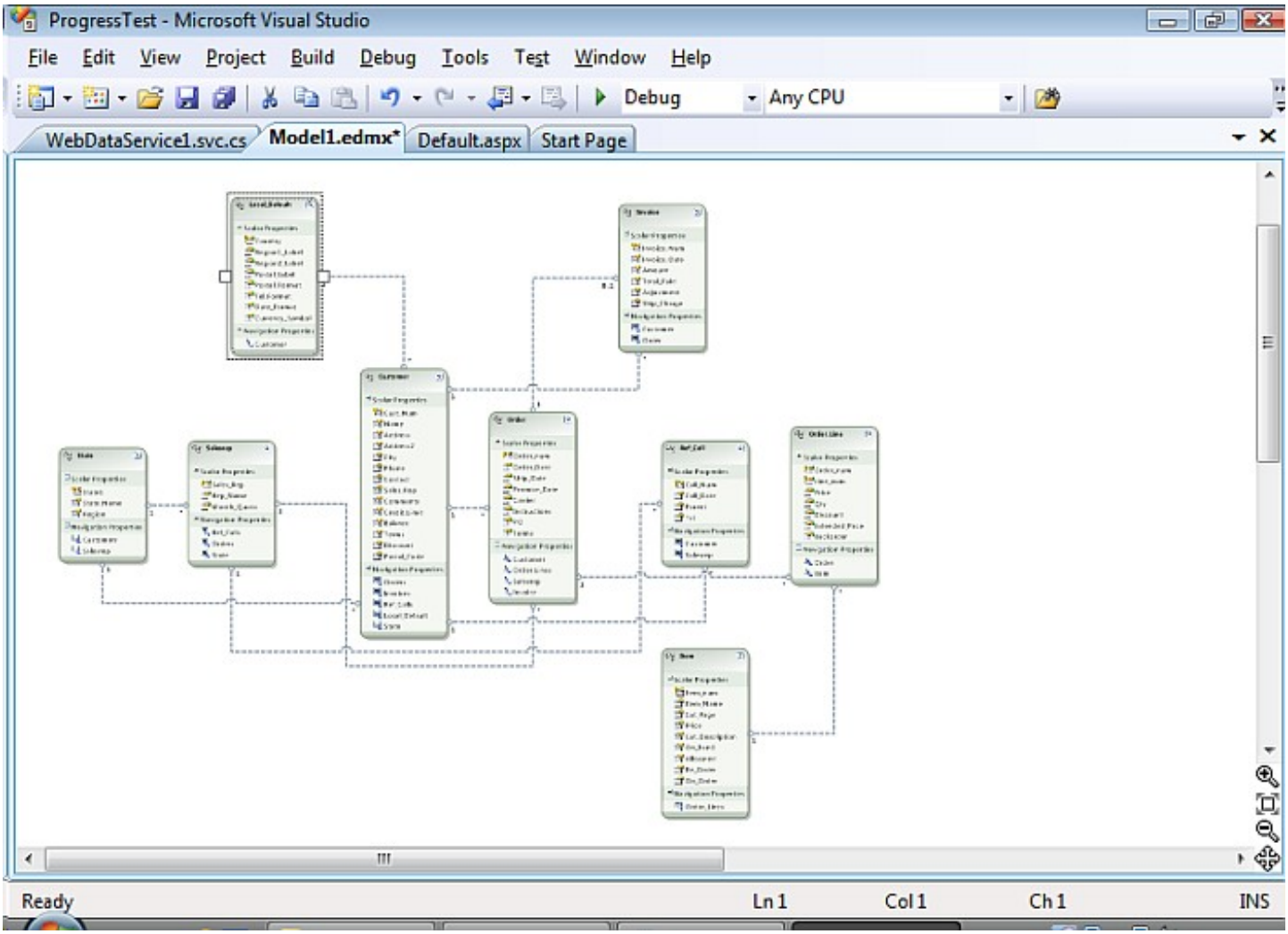
15. The project should now be fine.

Figure 8.112. Build Project



You will need to repeat these steps for each association until you have a completed Entity Data Model.

Figure 8.113. Entity Data Model



8.3.6. Using EDM to create Entity Framework based applications

Now that a Microsoft Entity Data Model has been created for the Progress isports database, Entity Framework applications can be created to make use of it.

Entity Framework Data Service

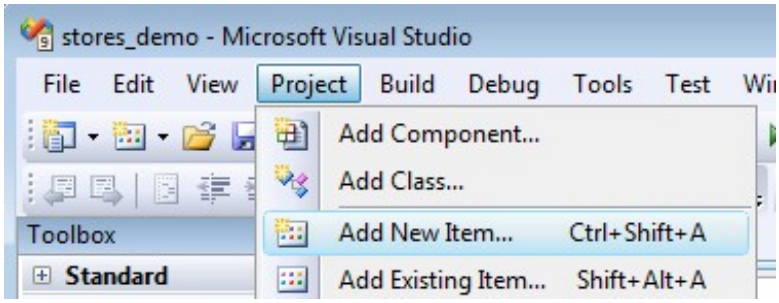
An ADO.Net Data Service for the Progress tables can be created using the Entity Data Model created in the Creating EDM in Visual Studio 2008 section .

1. Open the

   VirtuosoDataService

   project created in the Creating EDM in Visual Studio 2008 section .
2. Select the Project -> Add New Item menu option.

   Figure 8.114. VirtuosoDataService



3. The

   Add New Item

   dialog will appear. Choose the

   ADO.NET Data Service template

   . Give it the name

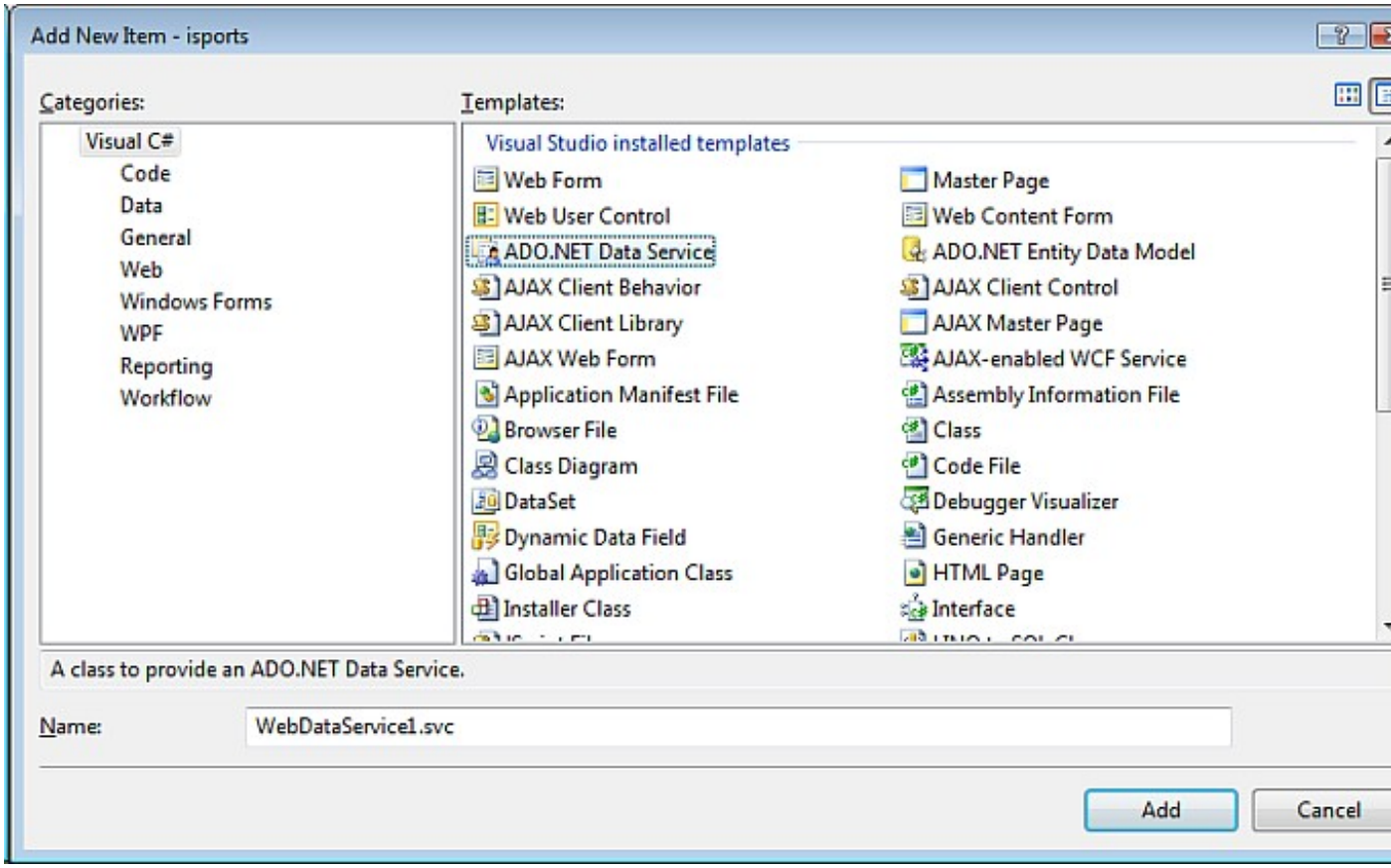
   WebDataService.svc

   , and click

   Add

   to create the ADO.Net Data Service.

   Figure 8.115. Add New Item



4. In the

   WebDataService1.svc.cs

Data Service file created add the data source class name of

isportsEntities

(note this is the name set in the Creating EDM in Visual Studio 2008 section) as the

DataService

name and enable the access to the Data Service by adding the entry

config.SetEntitySetAccessRule("", EntitySetRights.All);*

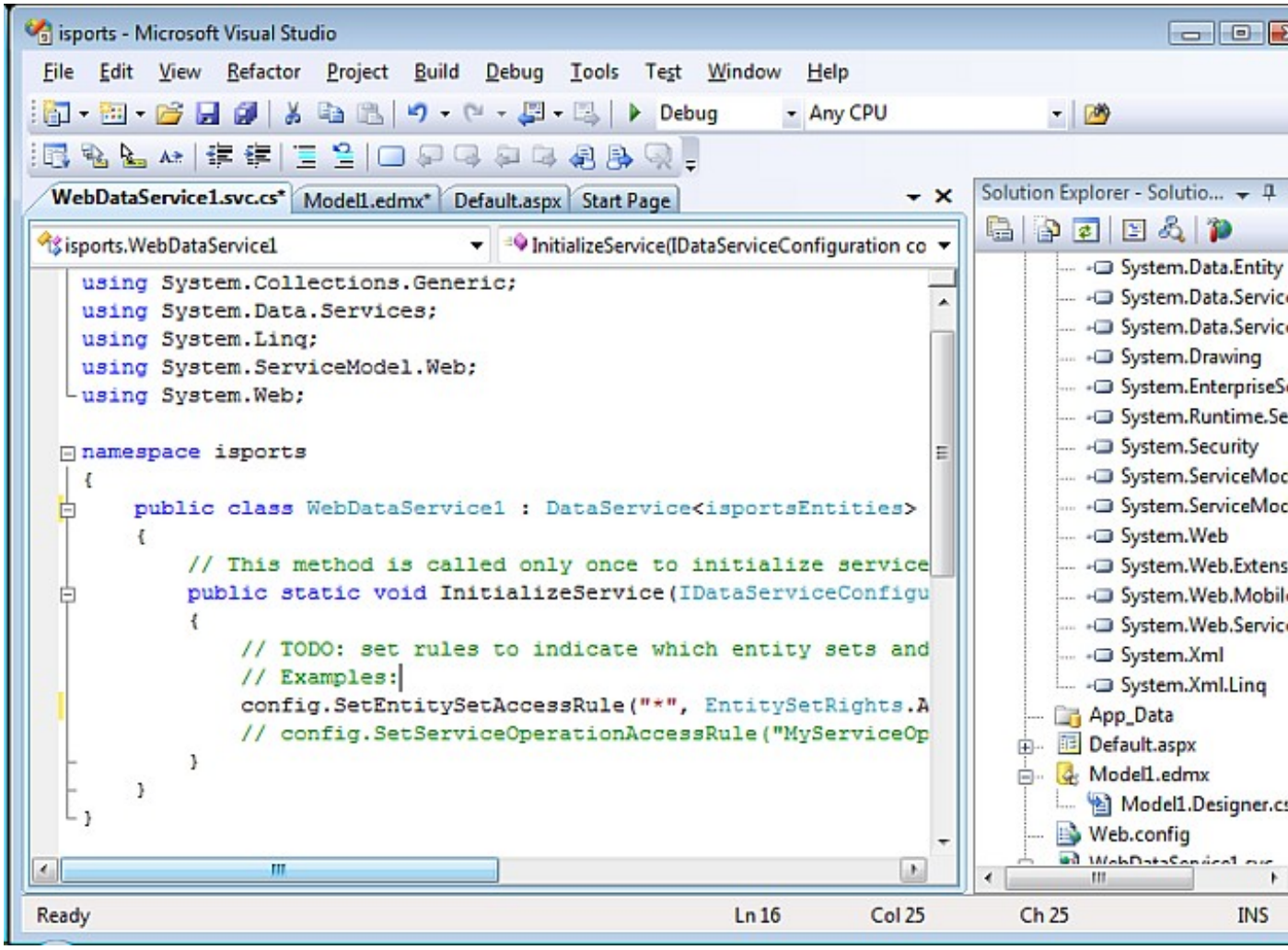
in the

InitializeService

method.

```
// C#

using System;
using System.Web;
using System.Collections.Generic;
using System.ServiceModel.Web;
using System.Linq;
using System.Data.Services;

namespace SimpleDataService
{
    public class Northwind : DataService<VirtuosoDemoEntities>
    {
        public static void InitializeService(IDataServiceConfiguration  config)
        {
            config.SetEntitySetAccessRule("*", EntitySetRights.All);
        }
    }
}
```

Figure 8.116. WebDataService1.svc.cs

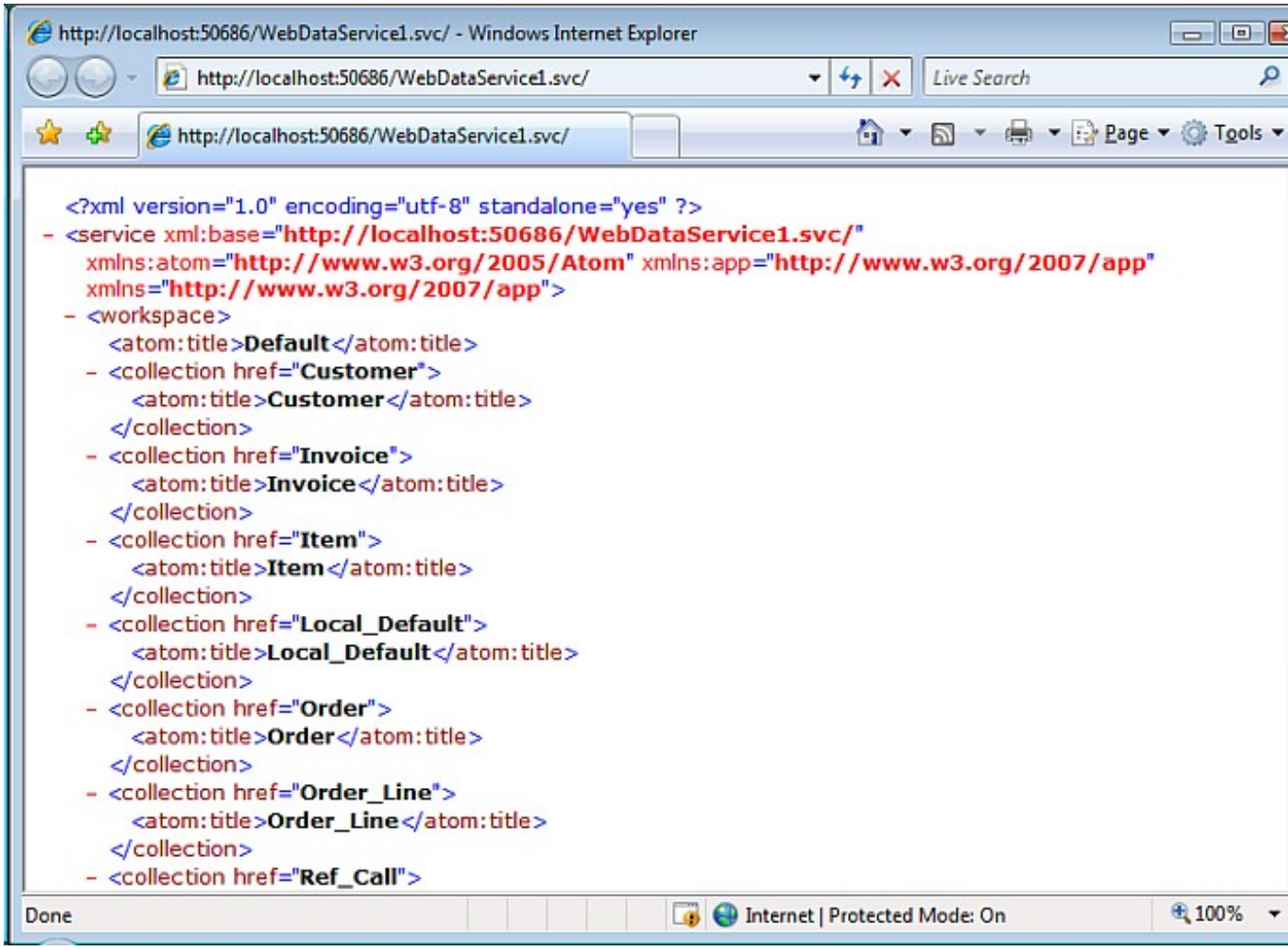


5. To test the Data Service, simply hit

   Ctrl+F5

   within Visual Studio. This will start the development web server, run the Data Services server inside and load a Web browser page displaying the list of available tables/entities for the isports database catalog.

Figure 8.117. Data Service test



6. To access a specific entity instance like the

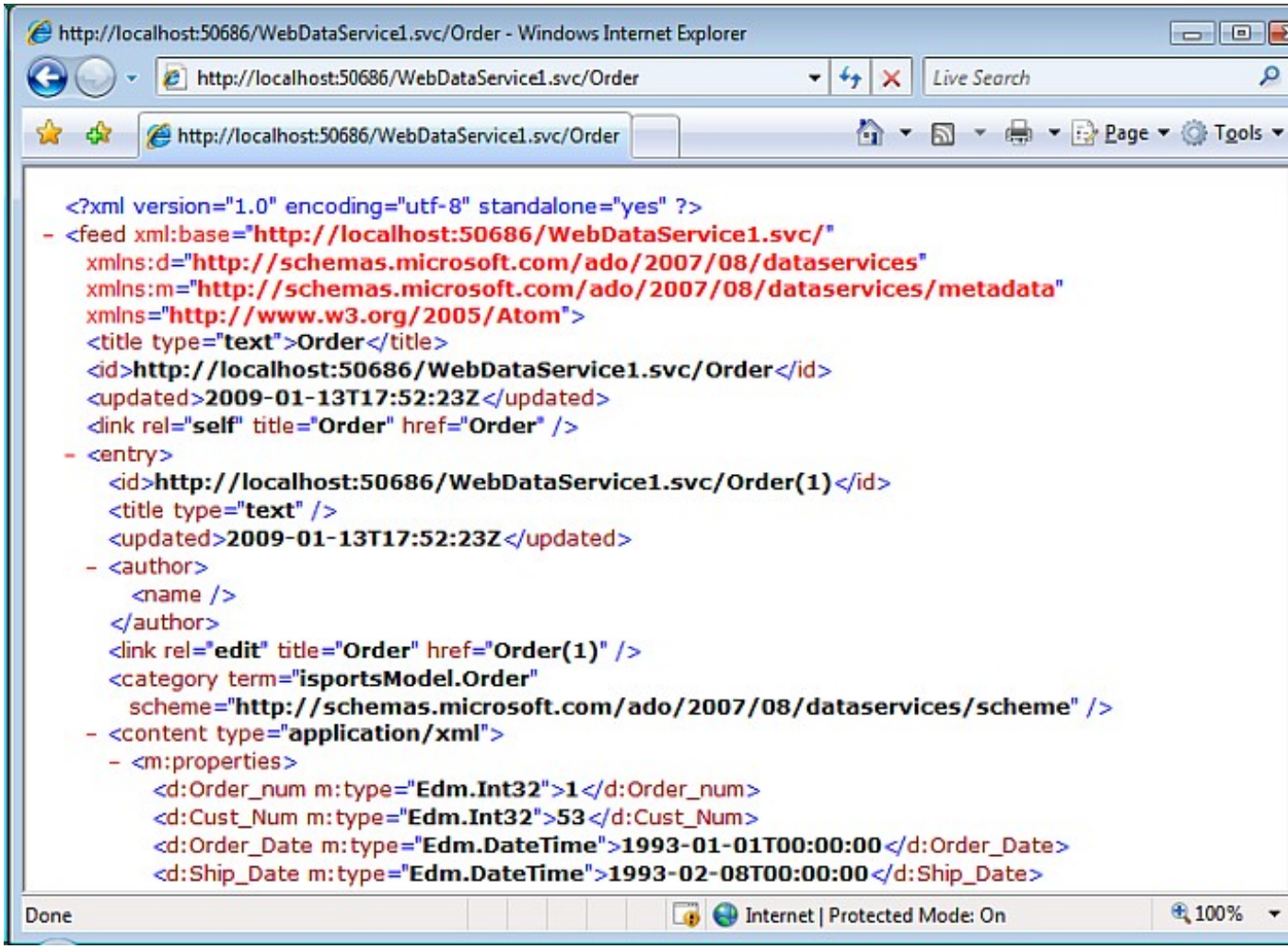
   EMPLOYEES

   table employee number

   100

   record, use this convention http://host/vdir/Virtuoso.svc/EMPLOYEES(100) .

   Figure 8.118. EMPLOYEES



Notes:

1. *Important*

   - To view

   Atom

   (the default format returned by an ADO.NET Data Service) in Internet Explorer, you must first ensure that

   Feed Reading View

   is turned

   off

   . This can be done on the

   Content tab

   of

   Tools in Internet Options

   .
2. If a Data Services entity instance URI page fails to load you can turn

Verbose

errors on by adding

config.UseVerboseErrors = true;

in the

virtuoso.svc.cs InitializeService

method to obtain more detailed information from the server as to why the page failed to load:

```
public static void InitializeService(IDataServiceConfiguration config)
{
config.UseVerboseErrors = true;
config.SetEntitySetAccessRule("*", EntitySetRights.All);
}
```

Visual Studio Windows DataGrid Form Application

This section details the steps required to create a simple Visual Studio 2008 Windows Form application, with associated DataGridView control for displaying data in selected tables from the target database.

1. Launch the Visual Studio 2008 SP1 IDE.

   Figure 8.119. Visual Studio 2008 SP1 IDE

2. Create a

   Web Application

   project by going to the

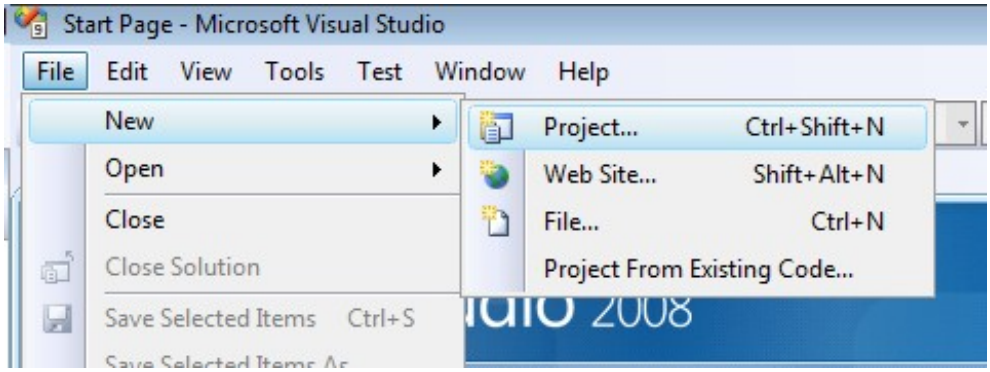
   File

   menu in Visual Studio and choosing

   New Project

   .

   Figure 8.120. Web Application



3. When the New Project window appears, choose either

   Visual Basic

   or

   Visual C#

   as your programming language.
4. Within the language category, click on

   Windows

   and select

   Windows Form Application

   from the right-hand panel.
5. Choose a name for the project, for example

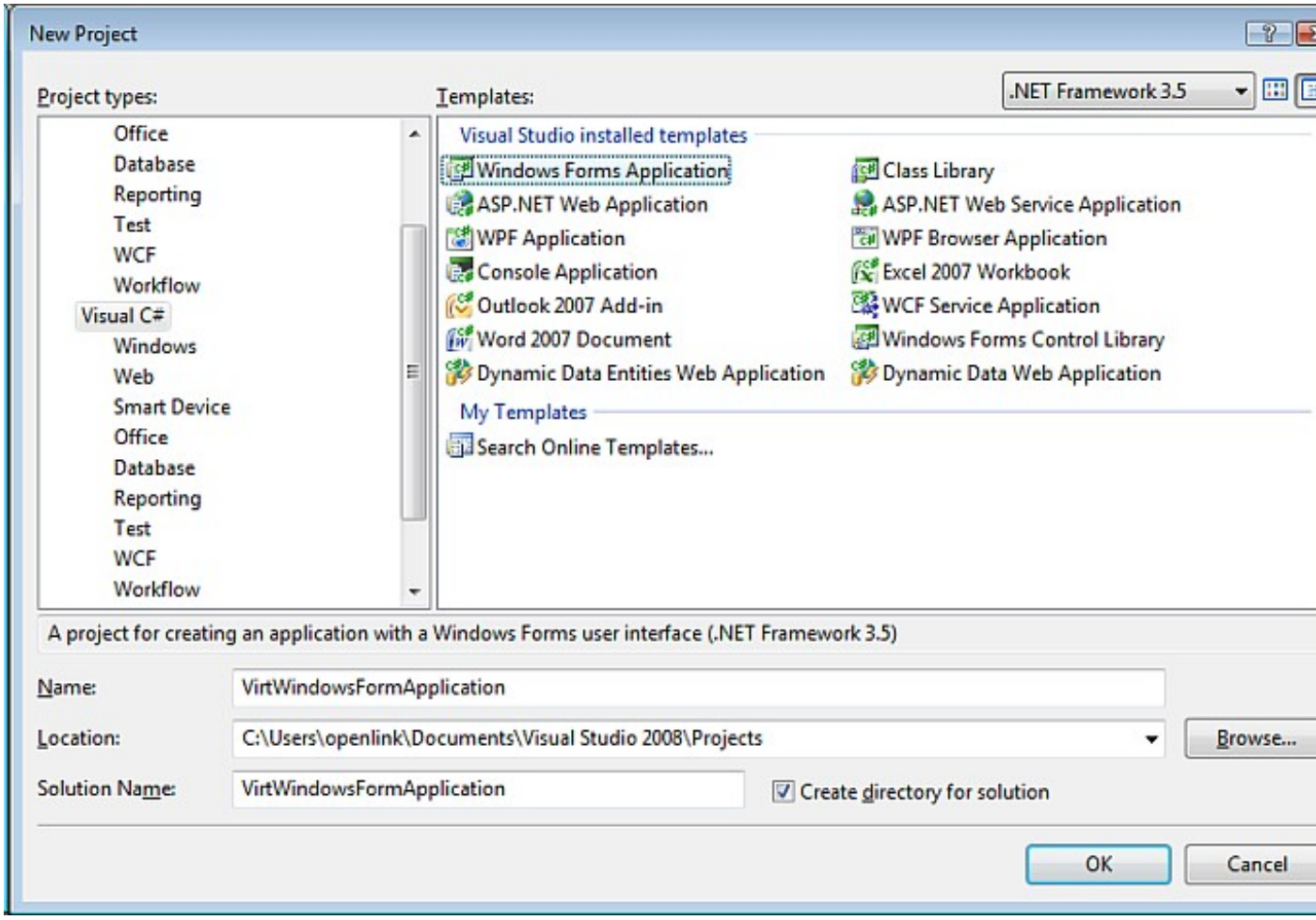
   VirtWindowsFormApplication

   , and click

   OK

   .

   Figure 8.121. Web Application



6. In the

   Toolbox

   , expand

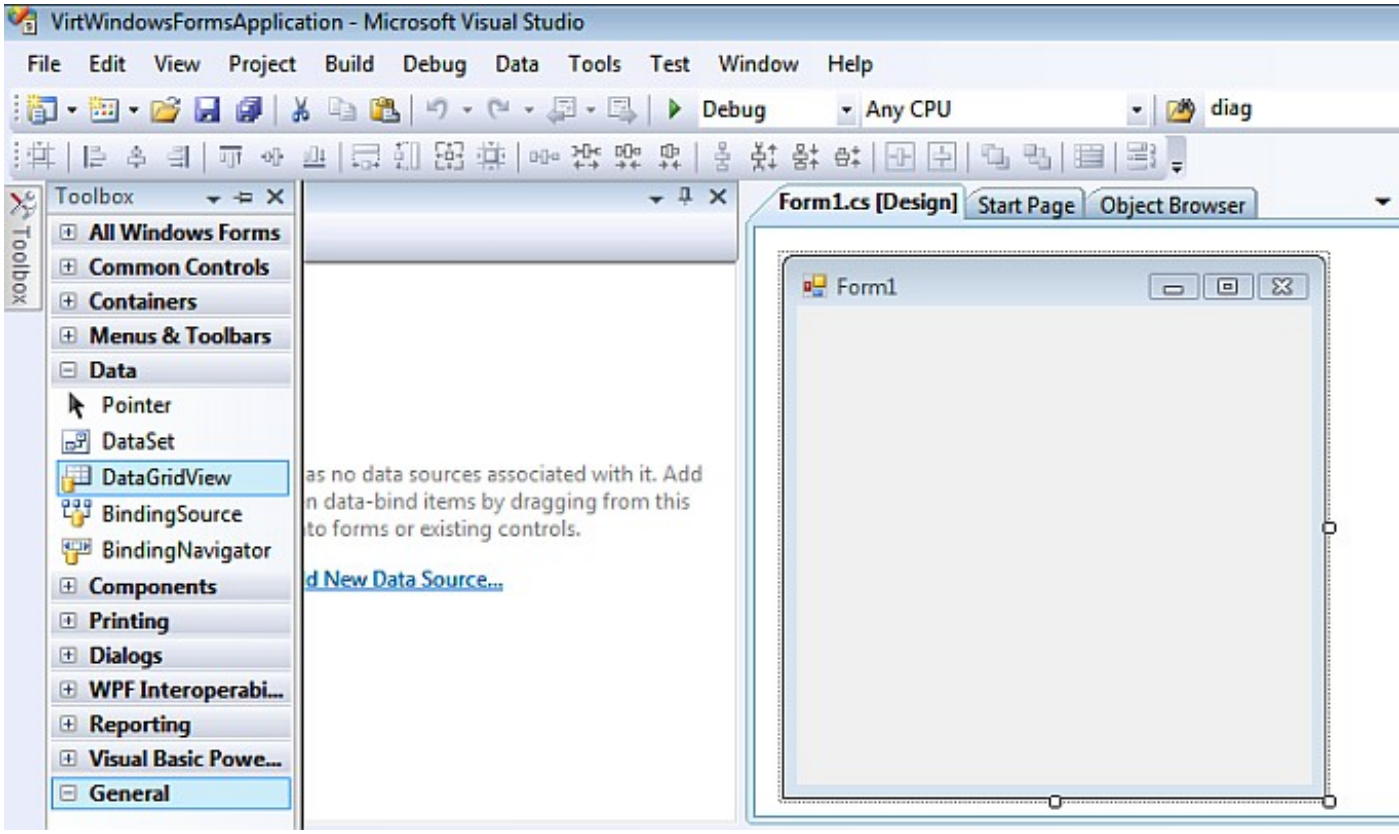
   Data Controls

   , and drag the

   DataGridView

   control onto the form.

   Figure 8.122. Toolbox



7. Click on the little

   arrow

   in the top right of the

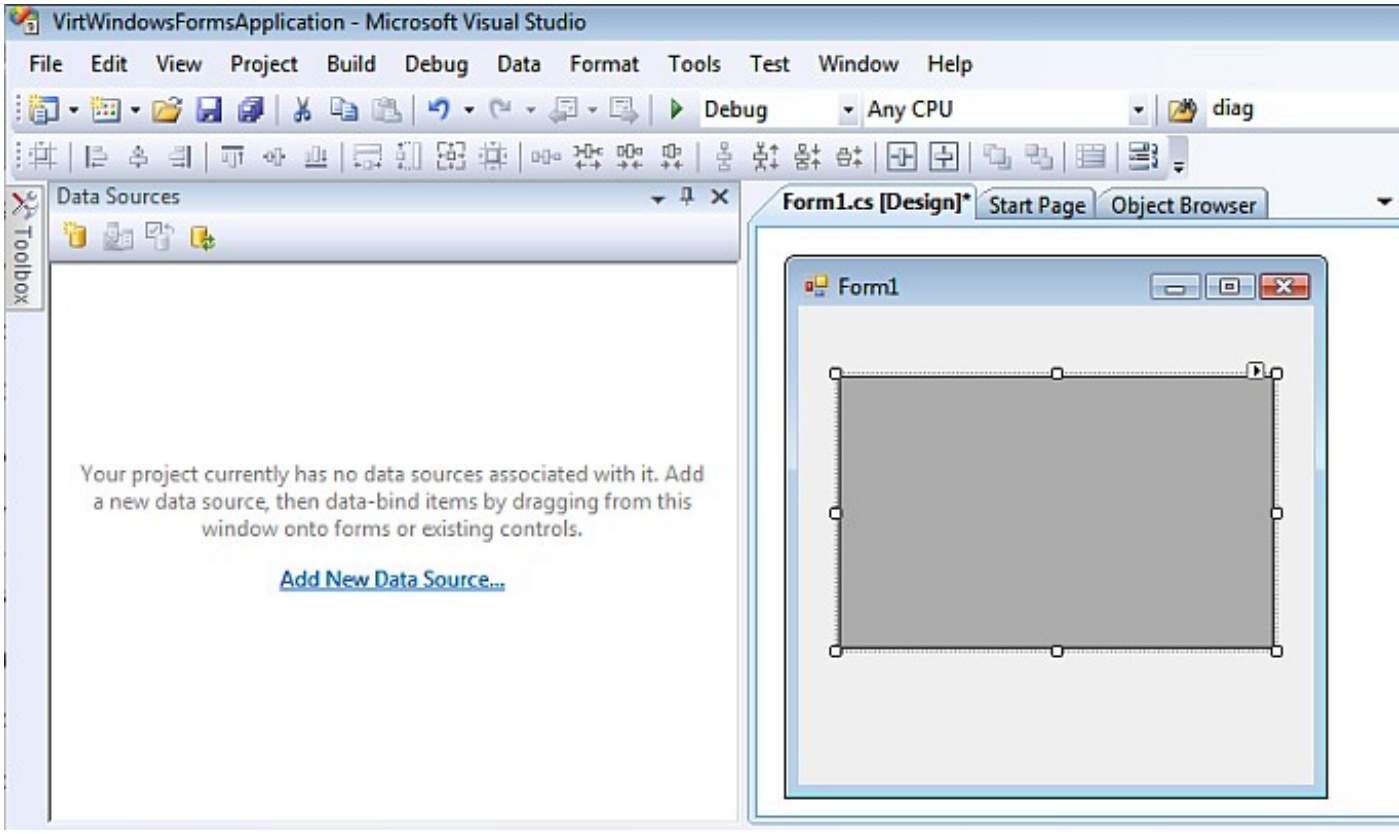
   DataGridView

   control. This loads the

   DataGridView Task

   menu.

Figure 8.123. DataGridView Task

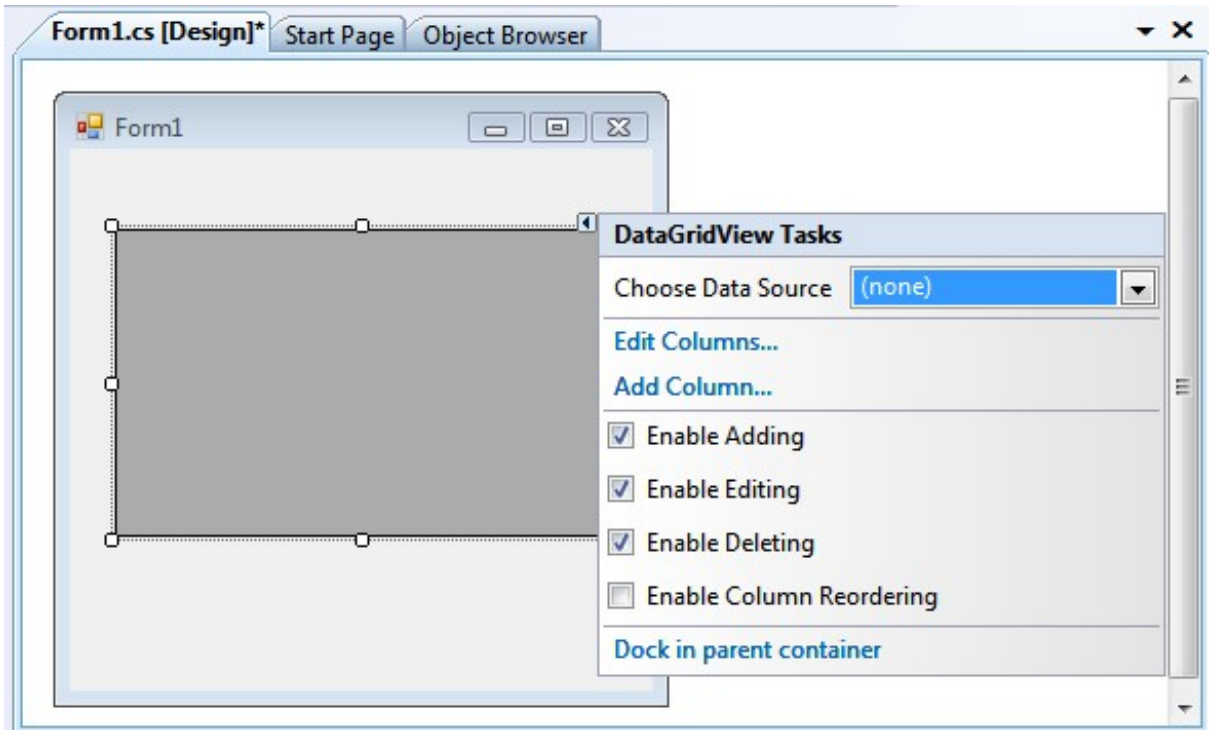


8. Click on the

   Choose Data Source

   list box.

Figure 8.124. Choose Data Source

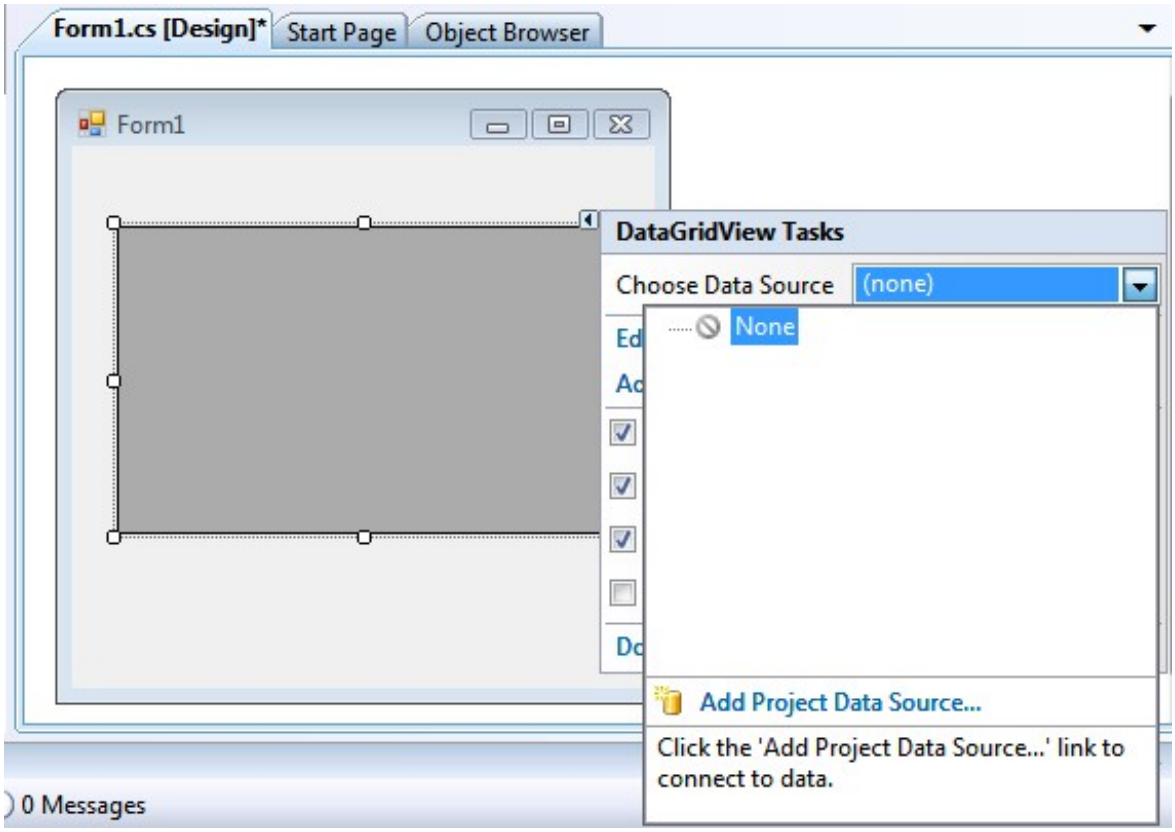


9. Click on the

   Add Project Data Source

link to connect to a data source.

Figure 8.125. Add Project Data Source



10. In the

    Data Source Configuration Wizard

    dialog

    Choose Data Source Type

    page select the

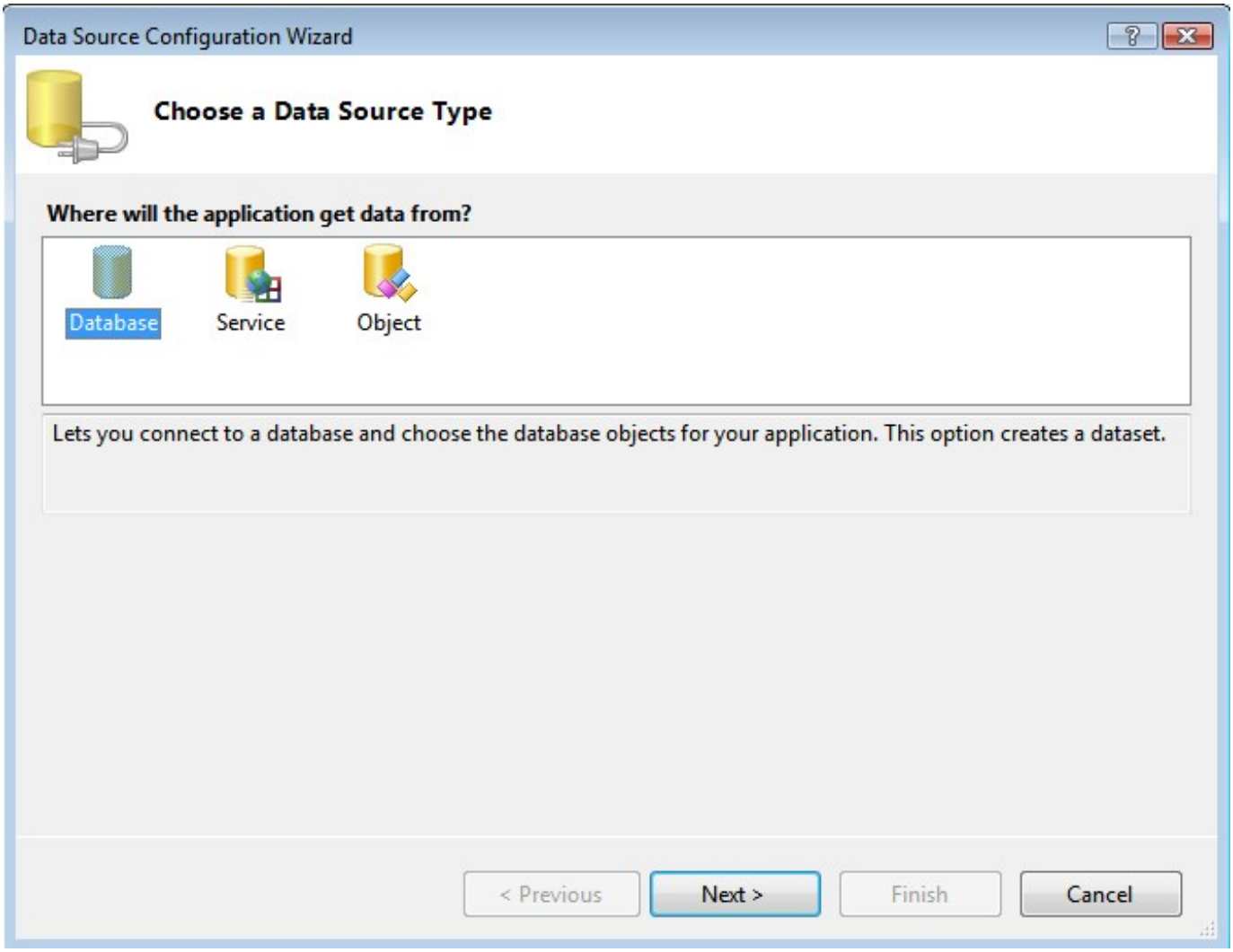
    Database

    data source type and click

    Next

    .

Figure 8.126. Data Source Type



11. In the

    Data Source Configuration Wizard

    dialog

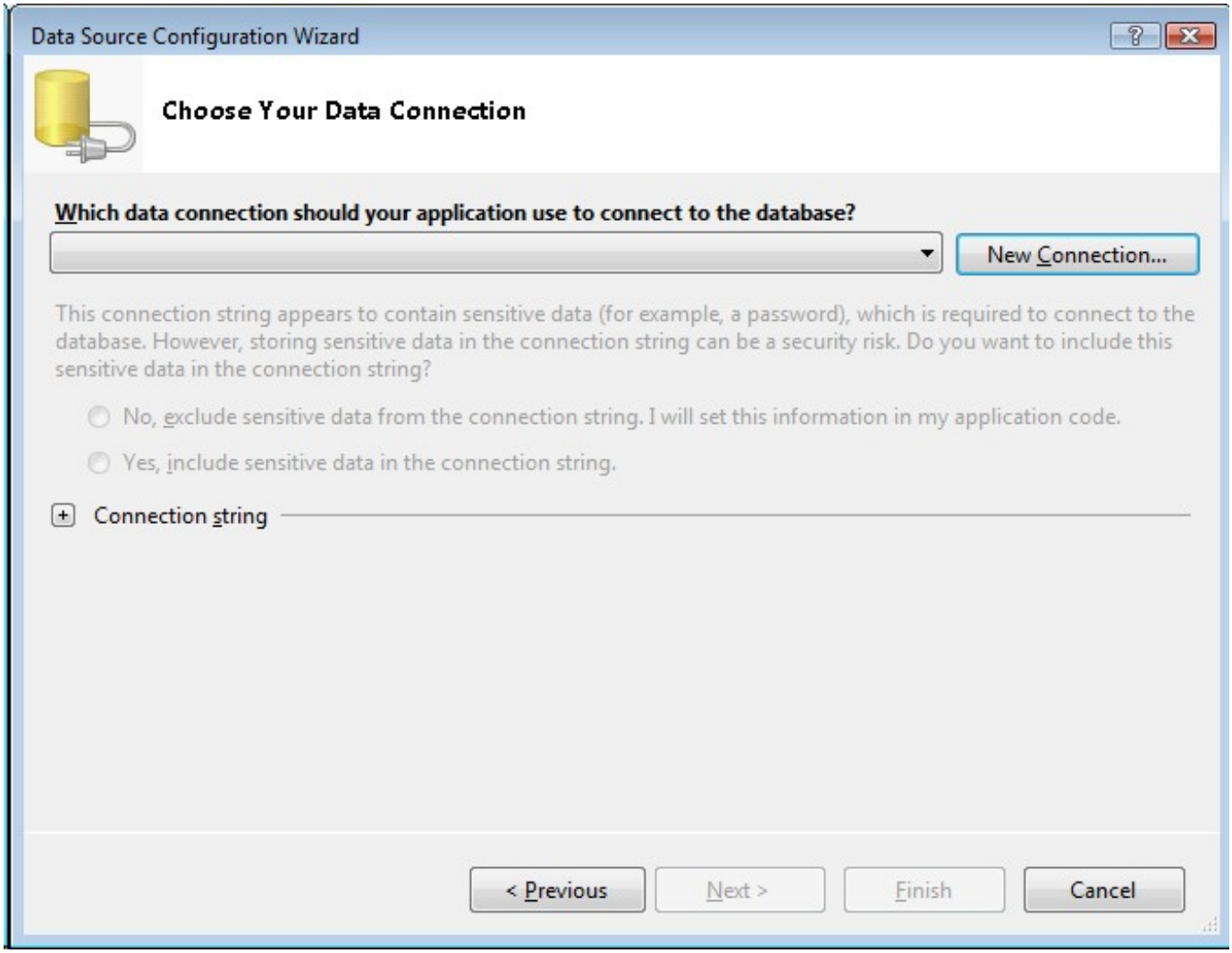
    Choose your Data Connection

    page, select the

    New Connection

    button

Figure 8.127. Data Source Configuration Wizard



12. In the

    Choose Data Source

    dialog, select the OpenLink

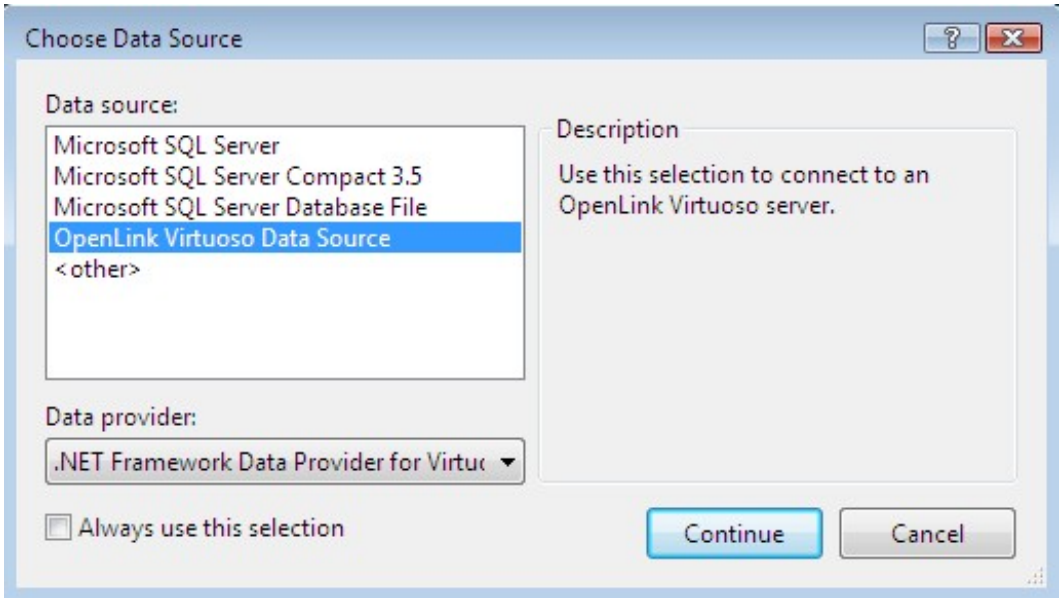
    Virtuoso Data Source

    from the list and click

    Continue

    .

Figure 8.128. Data Source



13. In the

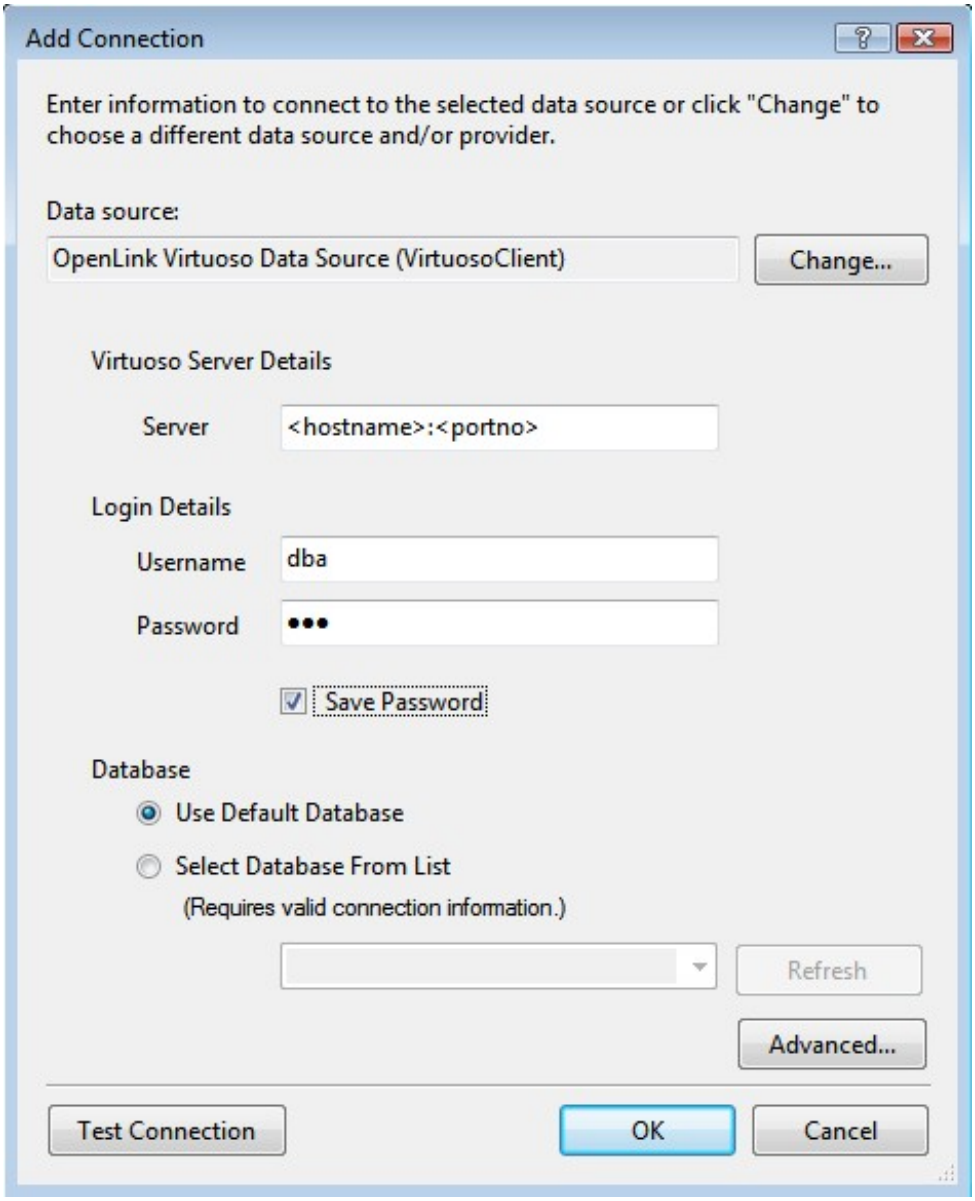
    Add Connection

    dialog, specify the

    hostname, portno, username and password

    for the target Virtuoso Server and check the Save Password check box.

    Figure 8.129. Connection Properties



14. Select the

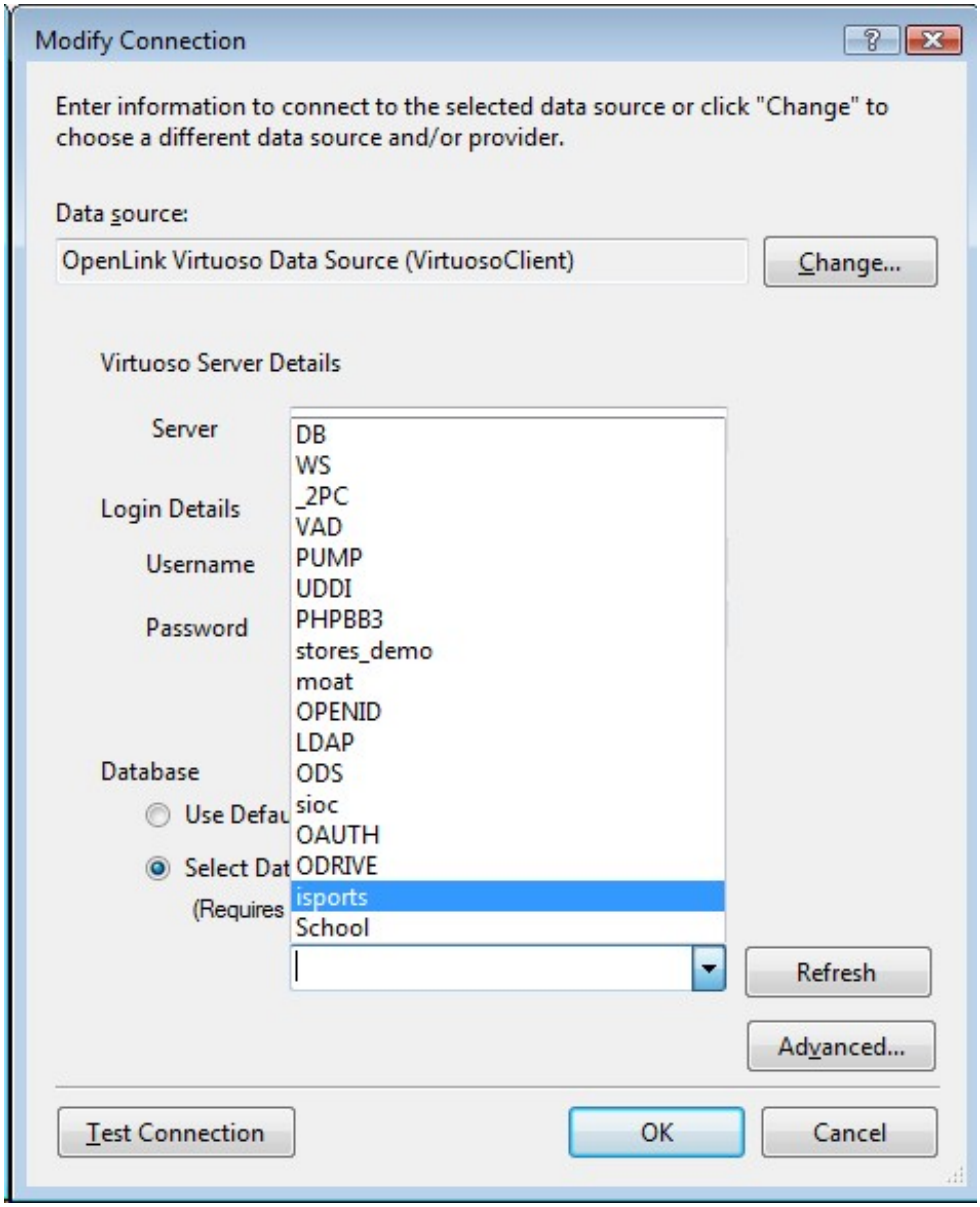
    Select Database From List

    radio button and choose

    isports

    from the drop down list.

    Figure 8.130. Add conneciton

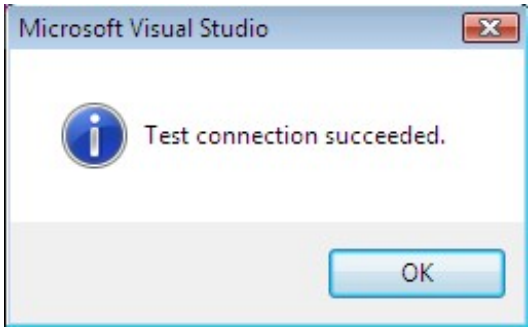


15. Press the

    Test Connection

    dialog to verify that the database is accessible.

    Figure 8.131. Test Connection



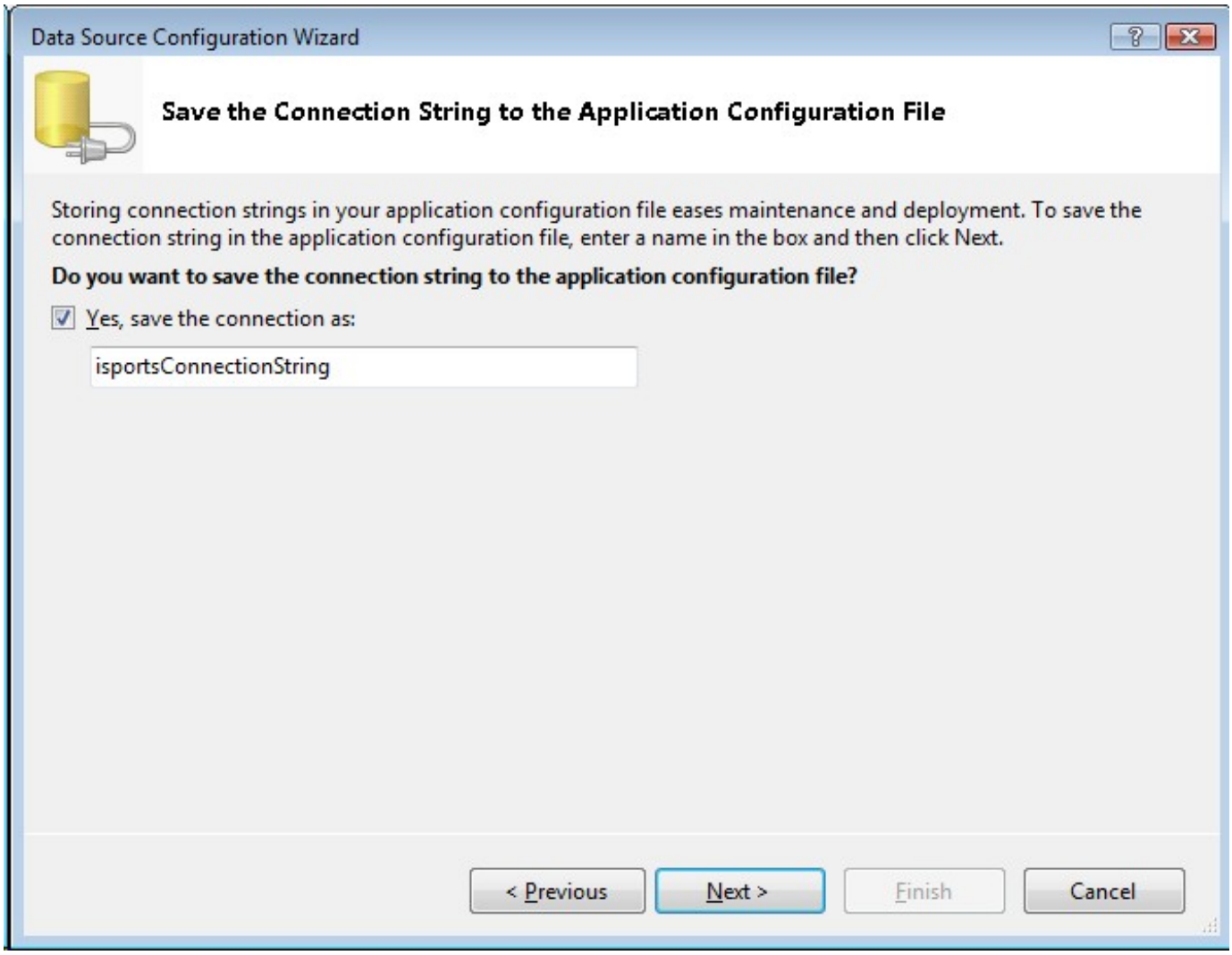
16. Leave the default connect string

    isportsConnectionString

and click

Next

Figure 8.132. isportsConnectionString



17. From the list of available tables returned for the isports database, select the

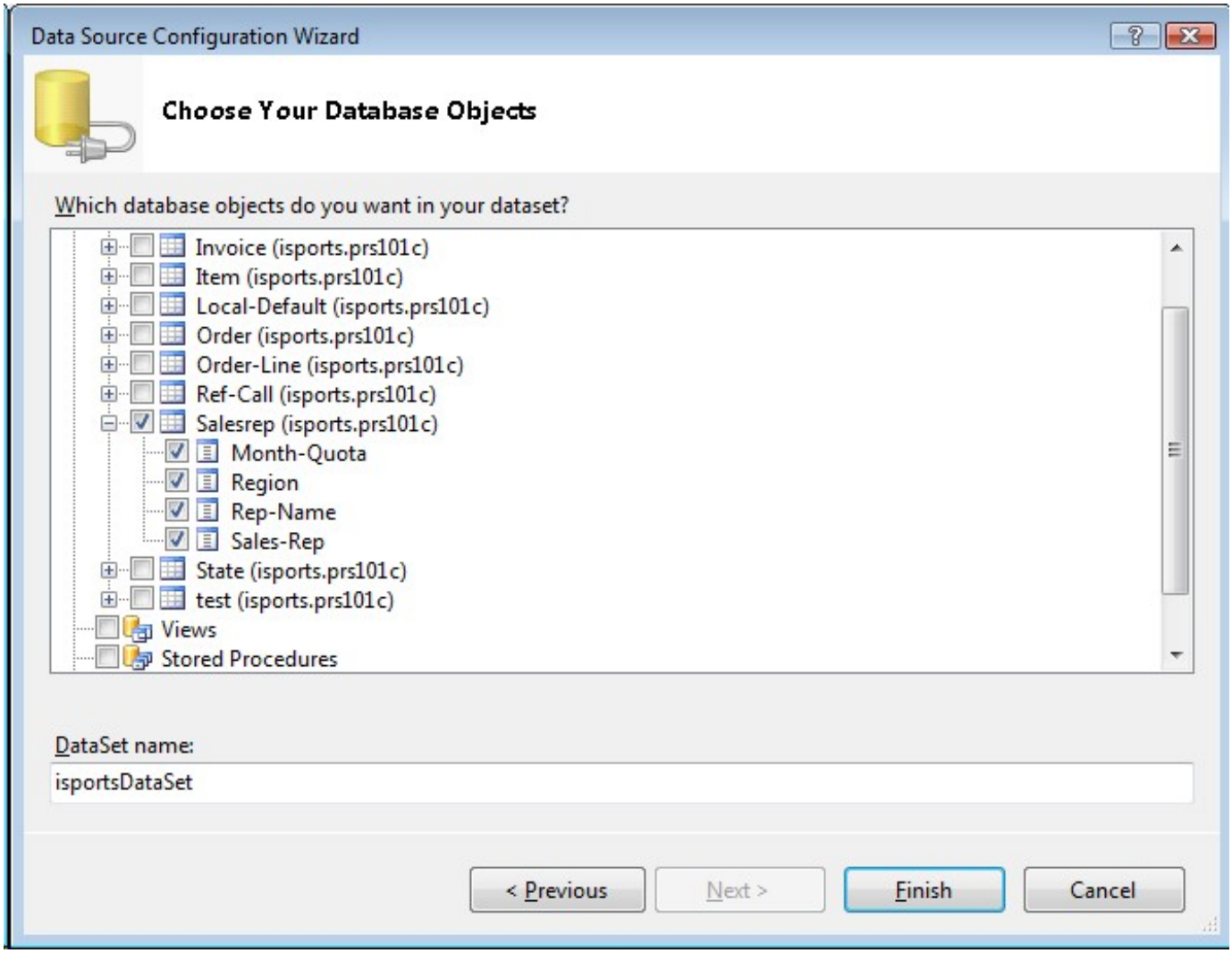
    JOBS

    table to be associated with the

    DataGridView

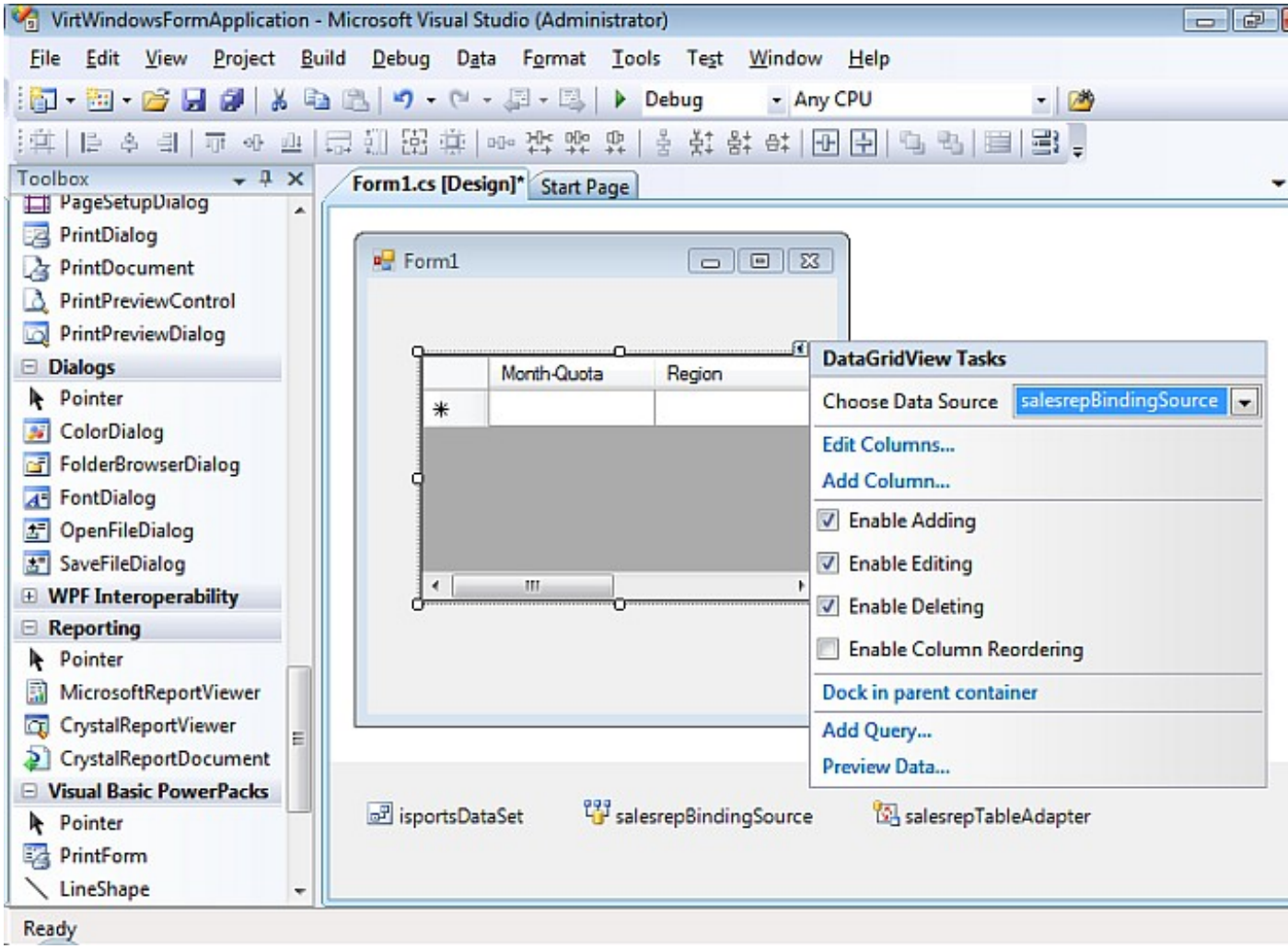
    control.

    Figure 8.133. isports database



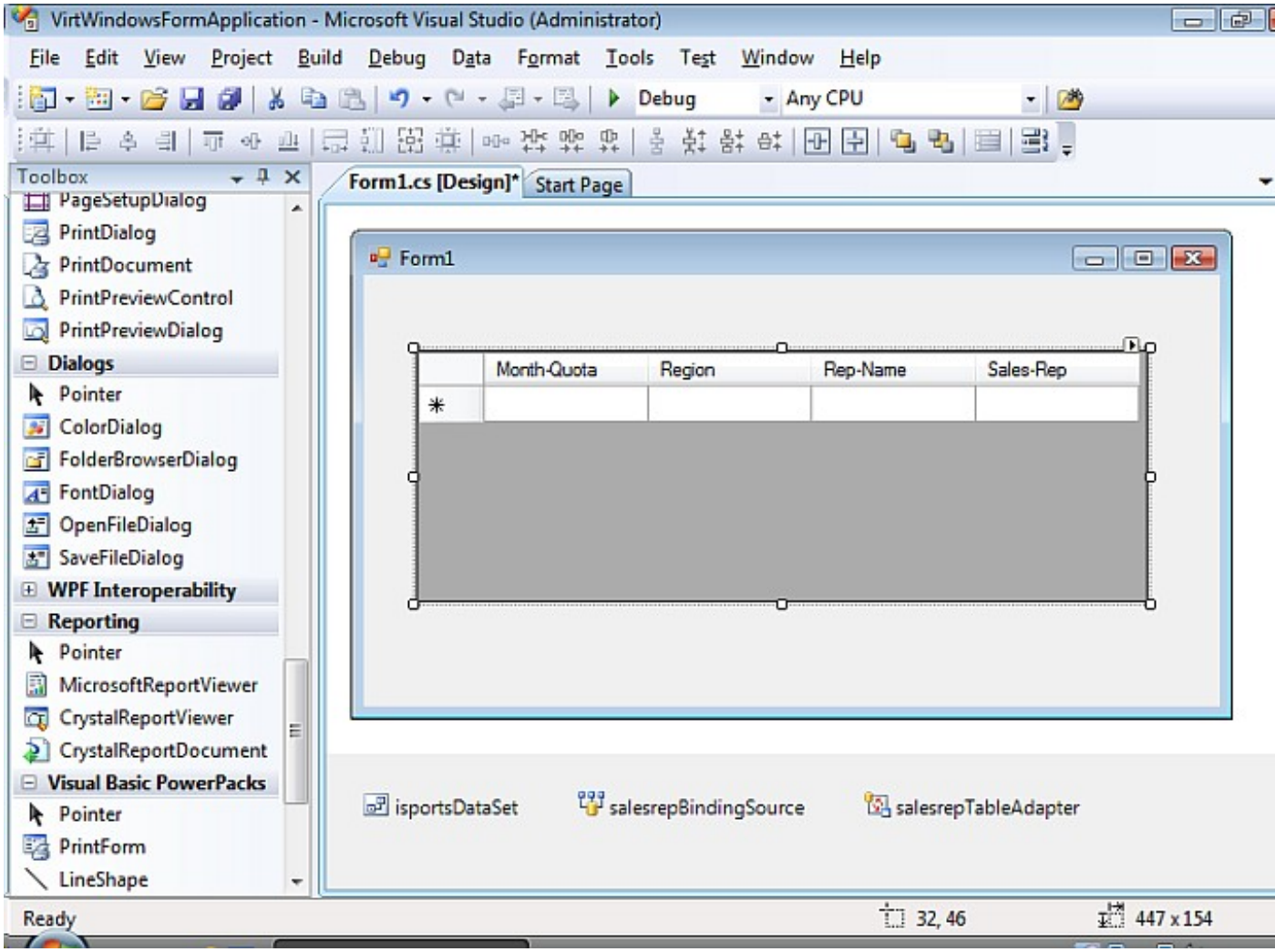
18. The columns names of the select table will be displayed in the DataGridView.

Figure 8.134. DataGridView



19. Resize the Form and DataGridView to allow all columns to be visible, if possible.

Figure 8.135. Resize the Form and DataGridView



20. To test the application, simply hit

    Ctrl+F5

    within Visual Studio or select

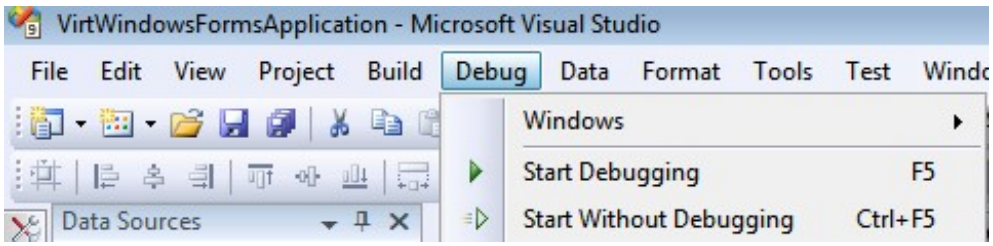
    Start Debugging

    from the

    Debug

    menu.

Figure 8.136. Start Debugging



21. The data from the

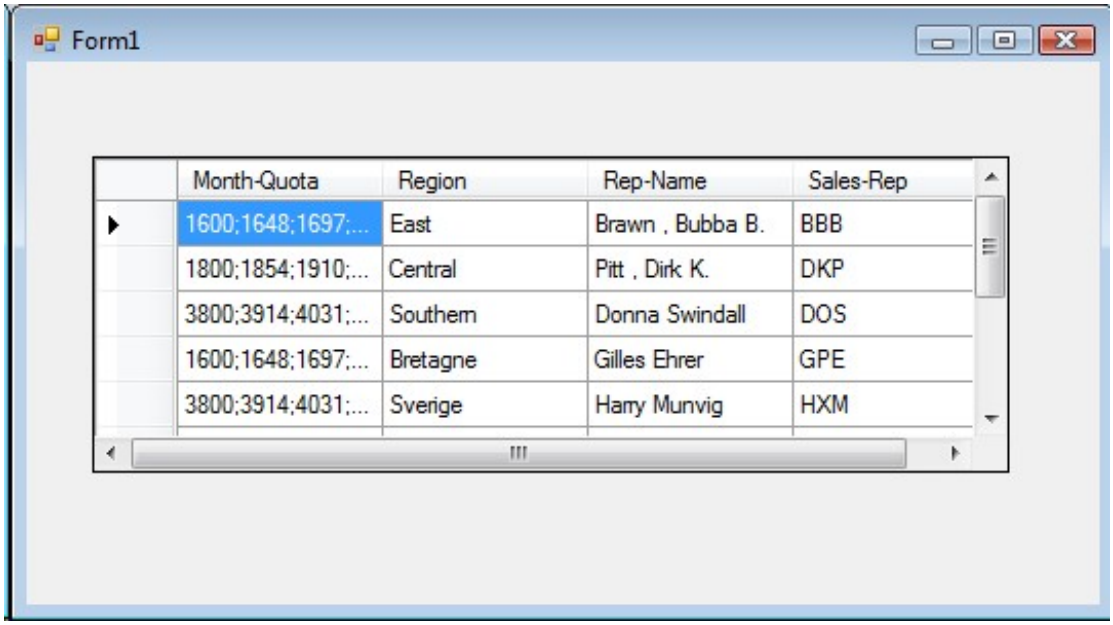
    JOBS

table will be displayed in the

DataGrid

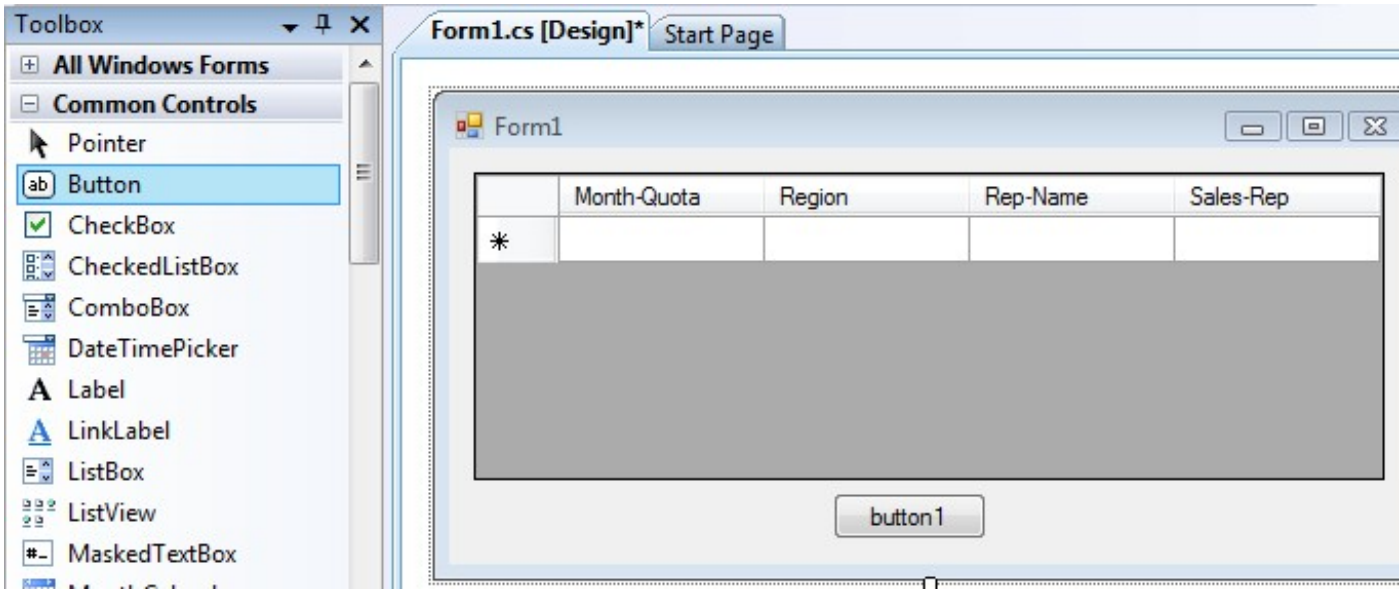
.

Figure 8.137. DataGrid



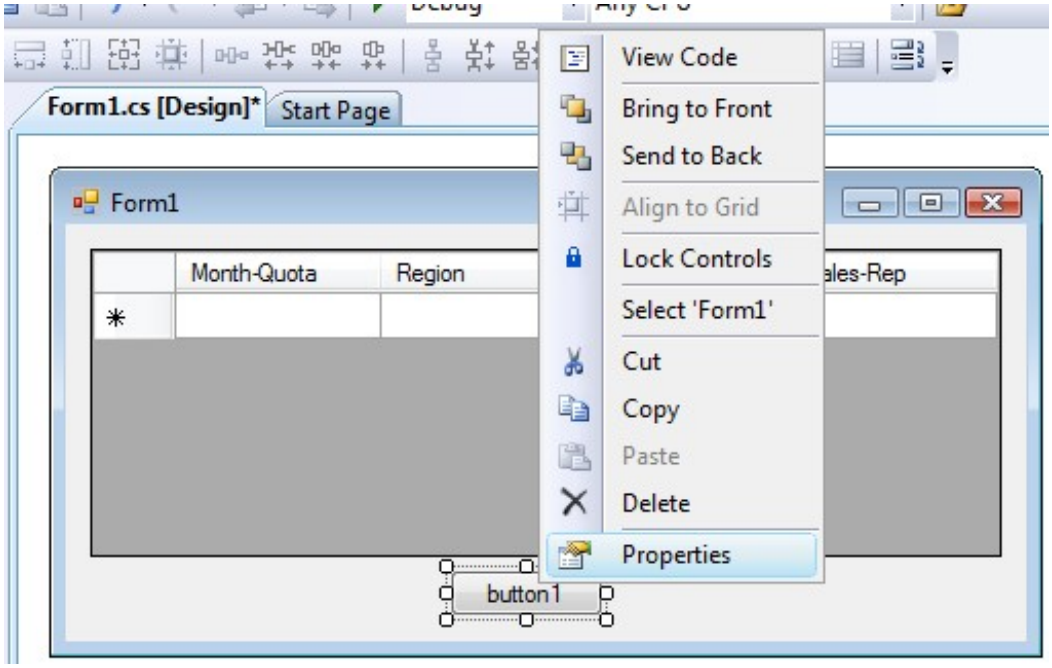
22. To make the DataGridView updateable, you will need to manually add some code to the project along with a suitable control to invoke the code. Drag a Button control onto the form.

Figure 8.138. DataGrid



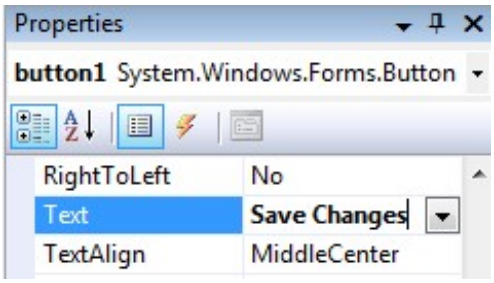
23. Right click on the Button and select Properties.

Figure 8.139. DataGrid



24. In the Properties view, edit the buttons Text property to read Save Changes and its (Name) property to read saveChanges.

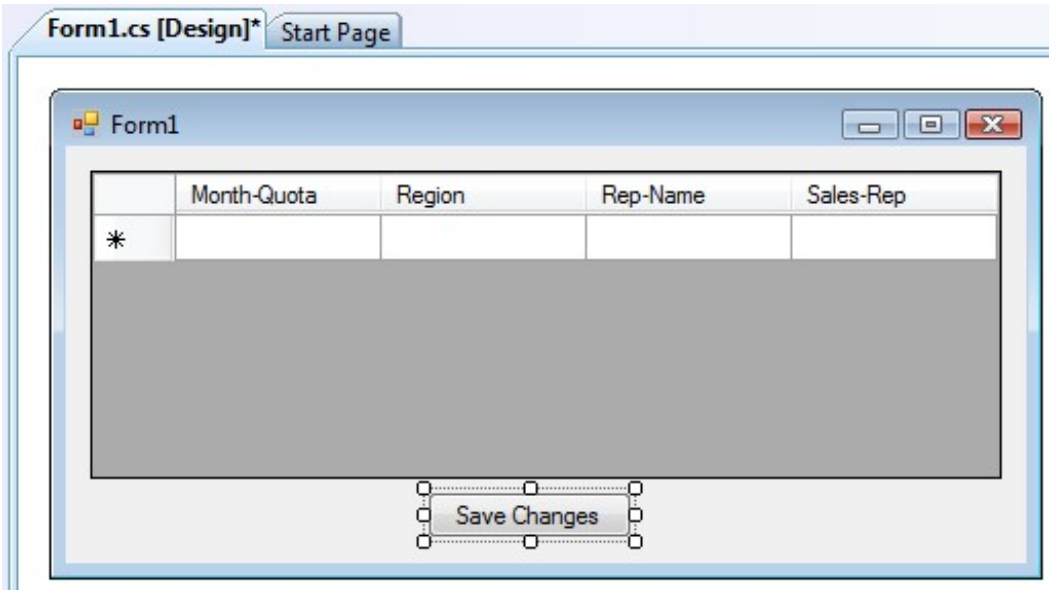
Figure 8.140. DataGrid



25. The button will now update to reflect these changes.

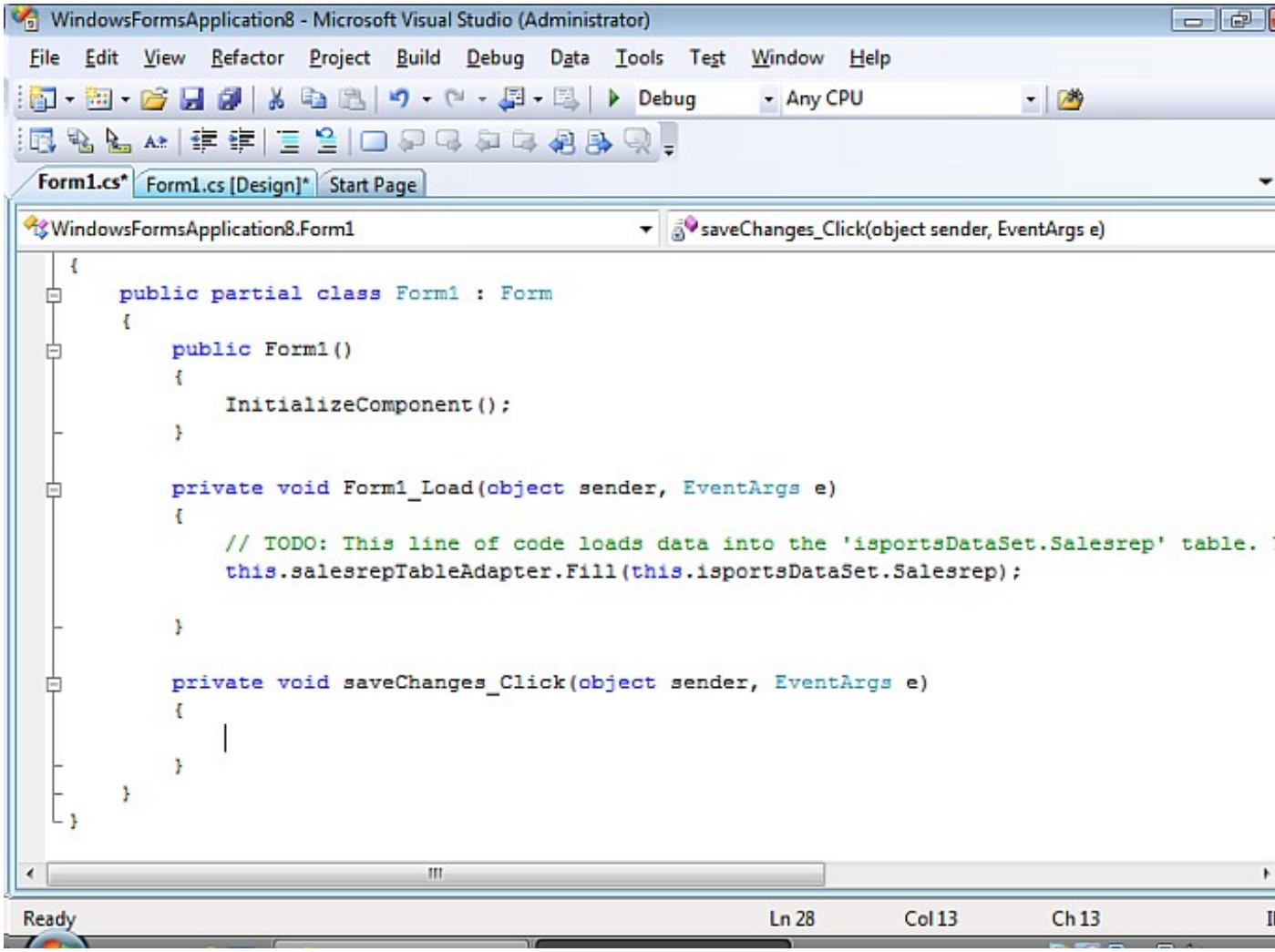
NOTE: You will need to resize the button to make the new text visible.

Figure 8.141. DataGrid



26. Double click the new button to generate the required event handler. It should take you directly to the area of code that will execute when the button is clicked.

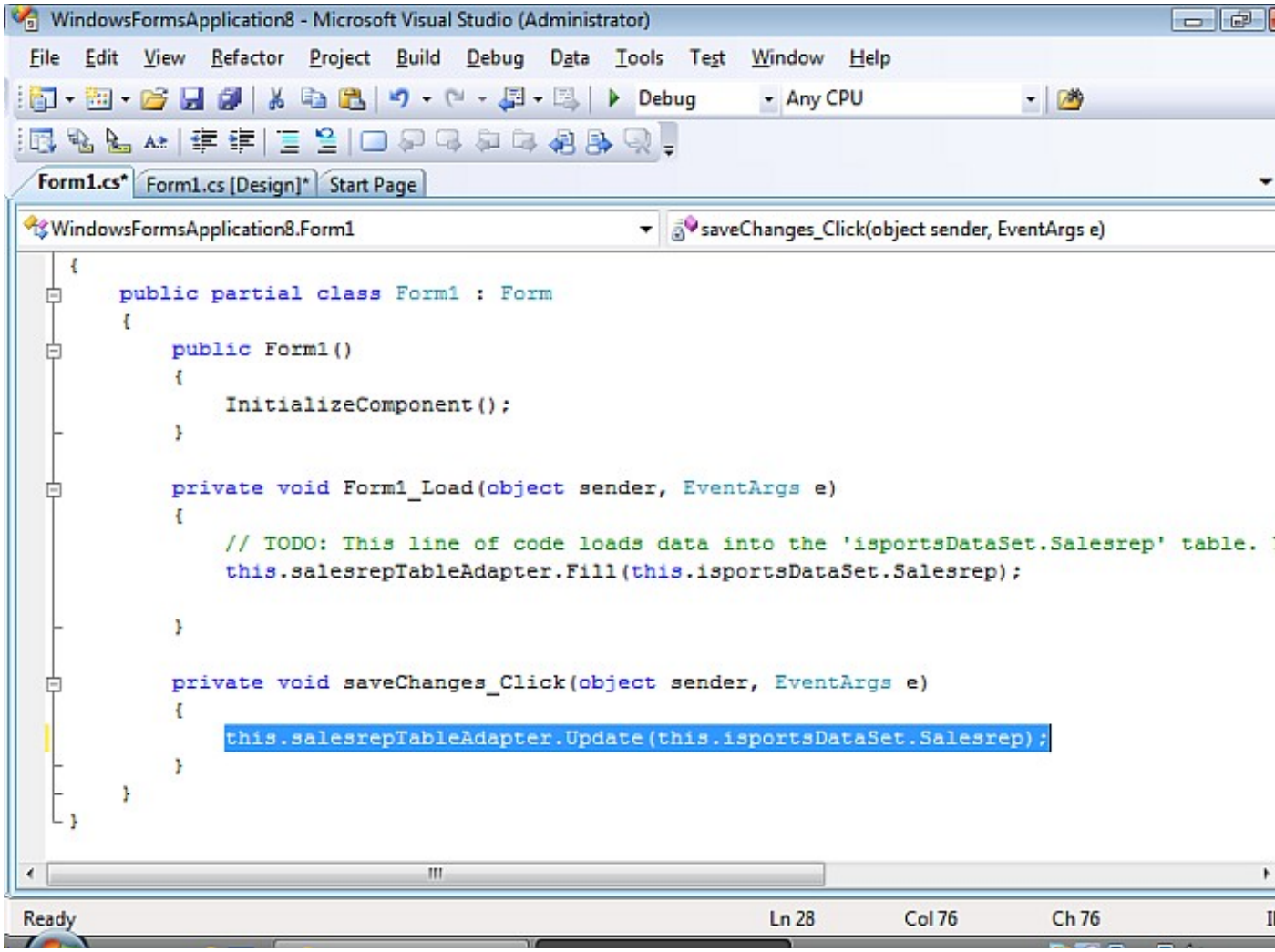
Figure 8.142. DataGrid



27. Edit the saveChanges_Click event handler code to include the following line.

```
private void saveChanges_Click(object sender, EventArgs e)
{
  this.msgsTableAdapter.Update(this.isportsDataSet.msgs);
}
```

Figure 8.143. DataGrid

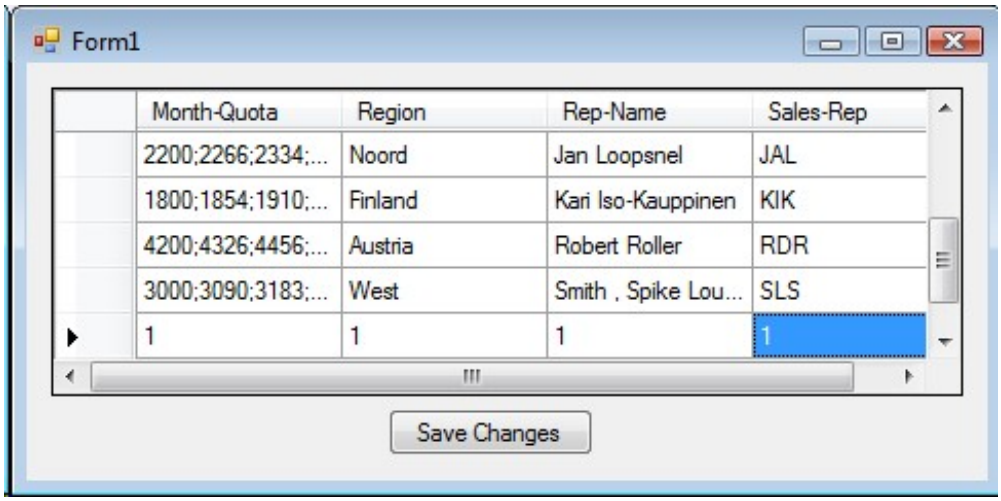


28. Now test the application again by hitting Ctrl+F5.

    Scroll to the empty row, at the bottom, and enter data for a new row then select Save Changes which will write the new row back to the database.

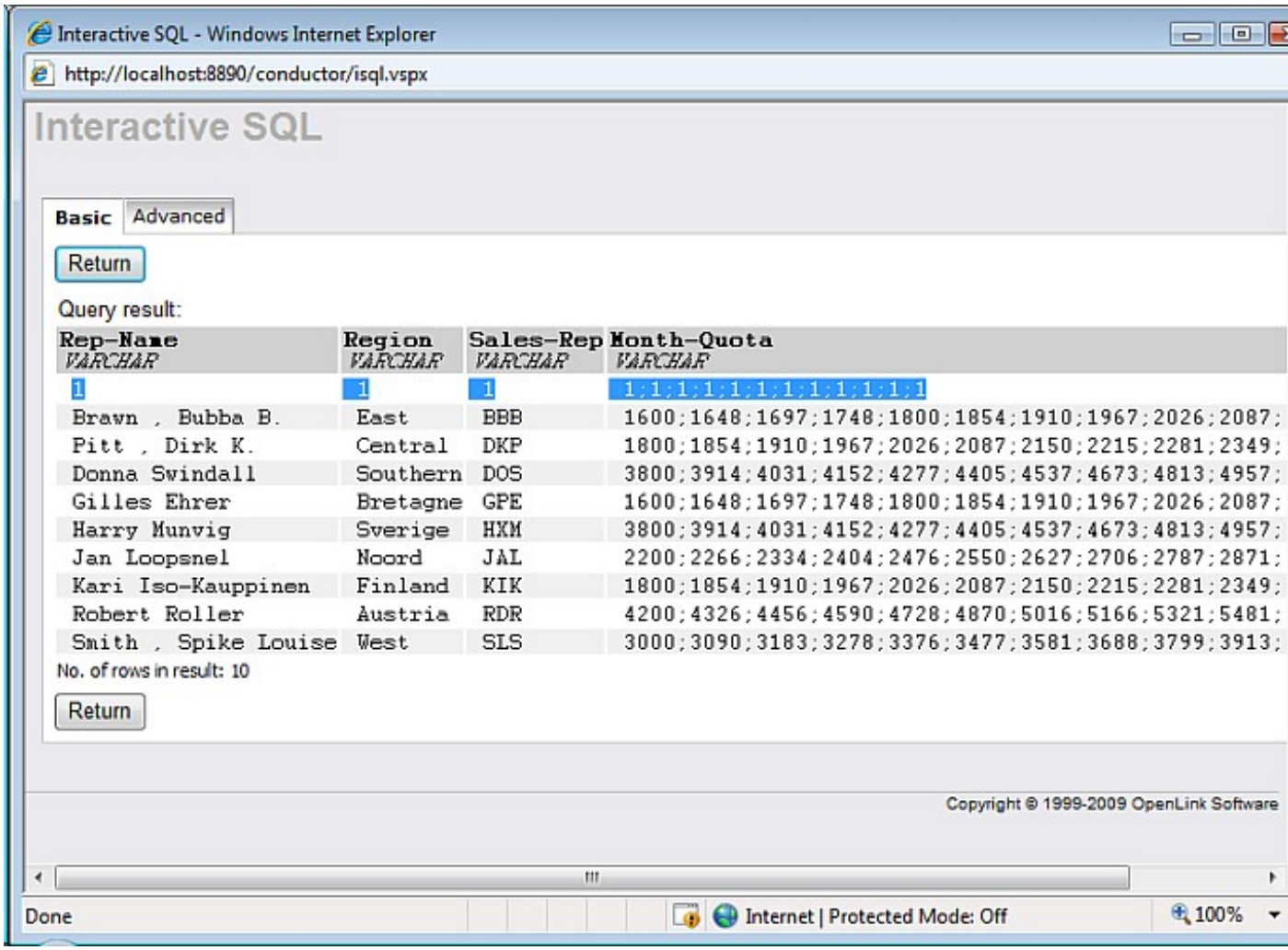
    Updates and deletes can be performed similarly.

Figure 8.144. DataGrid



29. You can use Interactive ISQL to test that the changes that have been written. Interactive ISQL Interface is detailed in the Linking Progress tables section.

Figure 8.145. DataGrid



The task is now complete.

8.4. Using Microsoft Entity Frameworks to Access Ingres Schema Objects with Virtuoso

Abstract

This section details the steps required to provide Microsoft Entity Framework access to Ingres Schema Objects using the OpenLink Virtuoso Universal Server. This is achieved by linking the required Ingres Schema objects into Virtuoso using its built in Virtual Database engine, and then using the Virtuoso ADO.NET Entity Framework provider to query the remote Ingres Schema objects linked into the Virtuoso Server.

Prerequisites. The following prerequisites must be in place for this solution to be possible.

Ingres DBMS. An Ingres DBMS hosting the required Schema Objects needs to be available. In this section the Ingres Tutorial sample database will be used to demonstrate the process.

An Ingres DBMS hosting the required Schema Objects needs to be available. In this section the Ingres Tutorial sample database will be used to demonstrate the process.

ODBC Driver for Ingres. An Ingres ODBC Driver is required for Linking the Ingres Schema Objects into the Virtuoso Server. The *OpenLink ODBC Driver for Ingres* will be used in this section, for which a functional ODBC Data source name of *ingiima* will be assumed to exist on the machine hosting the Virtuoso Server.

An Ingres ODBC Driver is required for Linking the Ingres Schema Objects into the Virtuoso Server. The *OpenLink ODBC Driver for Ingres* will be used in this section, for which a functional ODBC Data source name of *ingiima* will be assumed to exist on the machine hosting the Virtuoso Server.

Virtuoso Universal Server. An Virtuoso installation including the Virtuoso Universal Server and ADO.NET Entity Framework Provider is required. The Virtuoso components used must be Release 5.10.x or above, this being the minimum version containing support for Microsoft Entity Frameworks

An Virtuoso installation including the Virtuoso Universal Server and ADO.NET Entity Framework Provider is required. The Virtuoso components used must be Release 5.10.x or above, this being the minimum version containing support for Microsoft Entity Frameworks

Microsoft Visual Studio 2008 SP1. Microsoft Visual Studio 2008 Service Pack 1 is required, this being the only version containing the necessary Entity Framework support available at the time of writing.

Microsoft Visual Studio 2008 Service Pack 1 is required, this being the only version containing the necessary Entity Framework support available at the time of writing.

Tasks

Ensure Ingres Primary Keys (PKs) are not nullable. The Visual Studio 2008 Entity Data Model (EDM) requires that all primary keys are *NOT* Nullable, and will fail to generate an EDM if any are. Thus ensure any tables to be used are defined as not nullable in the Ingres database schema before attempting to generate an EDM. In the case of the Ingres database all tables are not nullable, thus this should not be an issue in this case.

The Visual Studio 2008 Entity Data Model (EDM) requires that all primary keys are *NOT* Nullable, and will fail to generate an EDM if any are. Thus ensure any tables to be used are defined as not nullable in the Ingres database schema before attempting to generate an EDM. In the case of the Ingres database all tables are not nullable, thus this should not be an issue in this case.

8.4.1. Install and configure OpenLink ODBC Driver for Ingres

The Virtuoso Virtual Database engine uses ODBC as the connectivity mechanism for linking remote database objects into its local schema. Thus an Ingres ODBC Driver must be available with a suitably configured DSN for connecting to the target database. The OpenLink ODBC Drivers for Ingres have been used in this section, although in theory any Ingres ODBC Driver can be used.

Installation instructions for the OpenLink ODBC Driver for Ingres are available from:

- Product Installation & Basic Configuration (ODBC)

8.4.2. Install and configure OpenLink Virtuoso Universal Server

- Install and configure OpenLink Virtuoso Universal Server

8.4.3. Linking Ingres tables into OpenLink Virtuoso

1. Start the Virtuoso Web User Interface

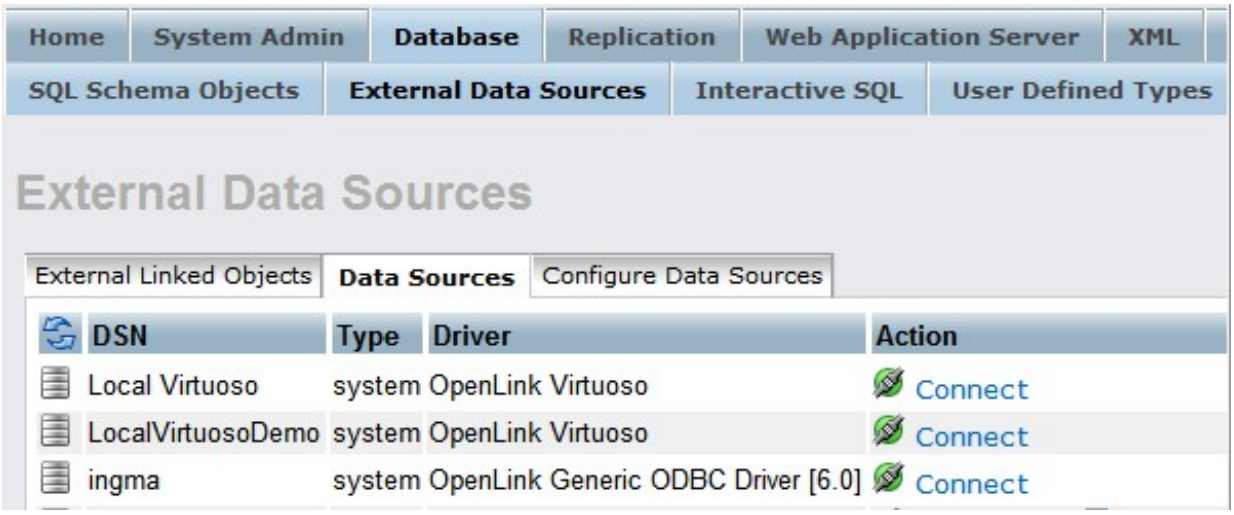
   Figure 8.146. Start

2. Select the "Conductor" link in the Left Frame and Login as the "dba" user.

Figure 8.147. Conductor

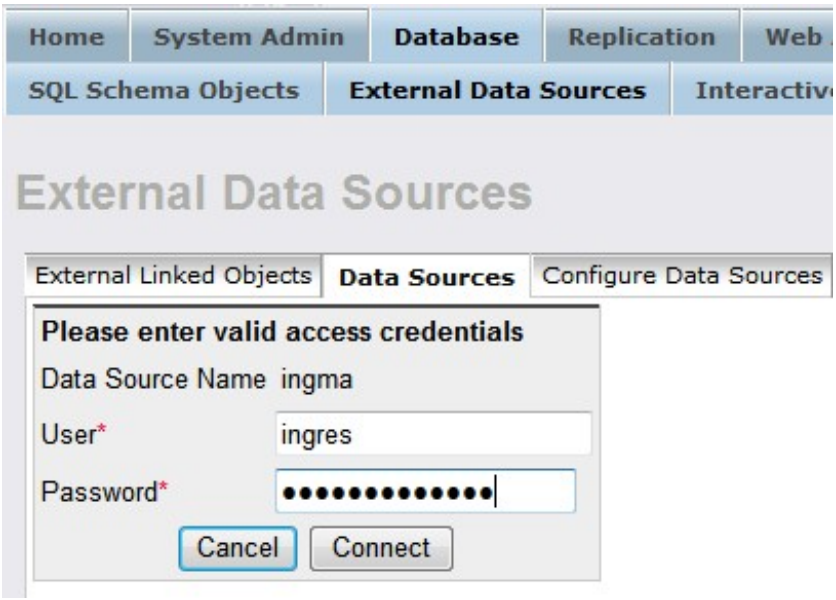
3. Select the "Databases" -> "External Data Source" -> "Data Sources" tab menu items

Figure 8.148. Databases



4. Select the "Connect" button for the "Ing10ma" Ingres DSN.

Figure 8.149. Connect

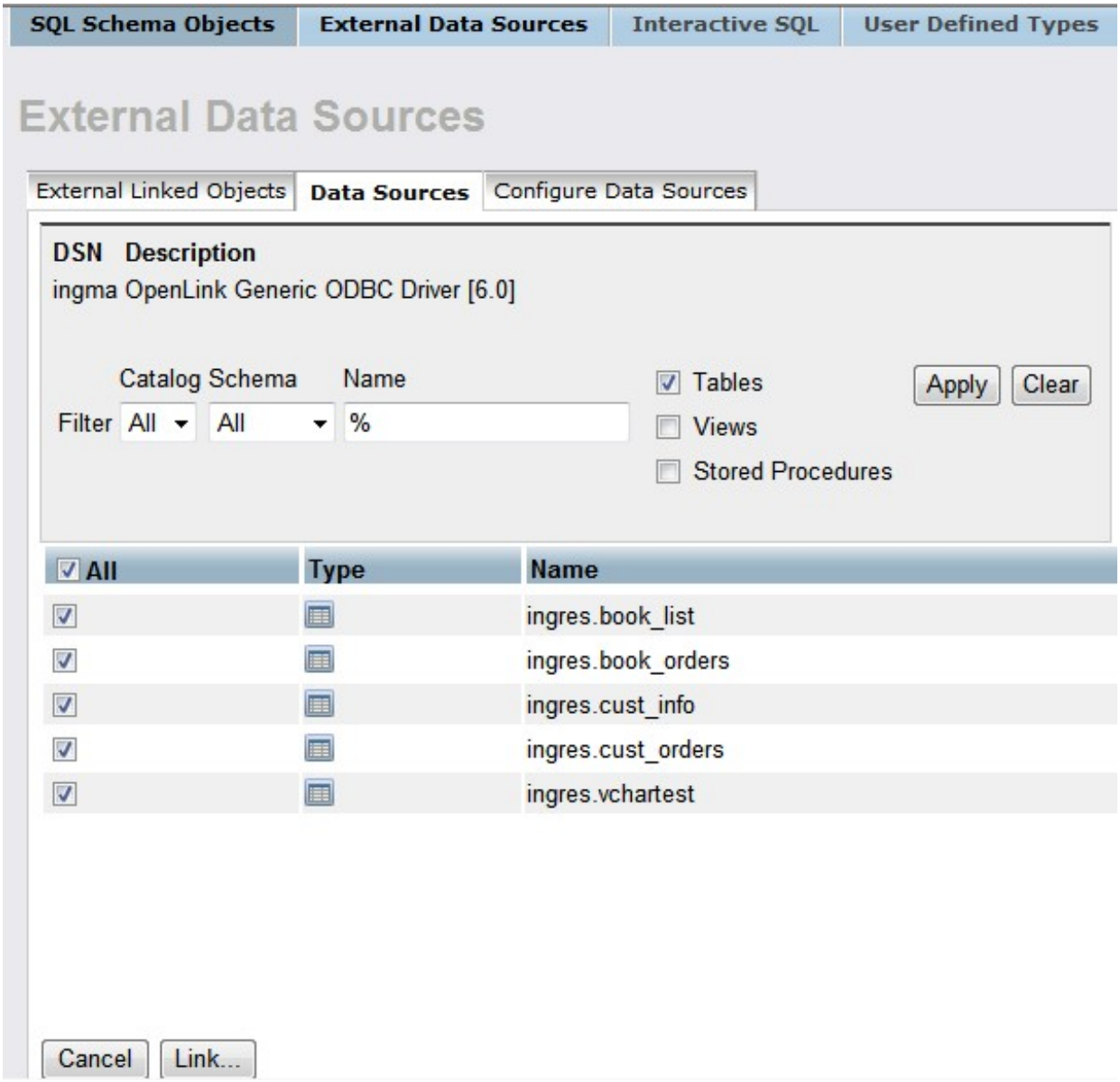


5. On successful connection Select the "Link Objects" button to obtain a list of available tables

Figure 8.150. Link Objects

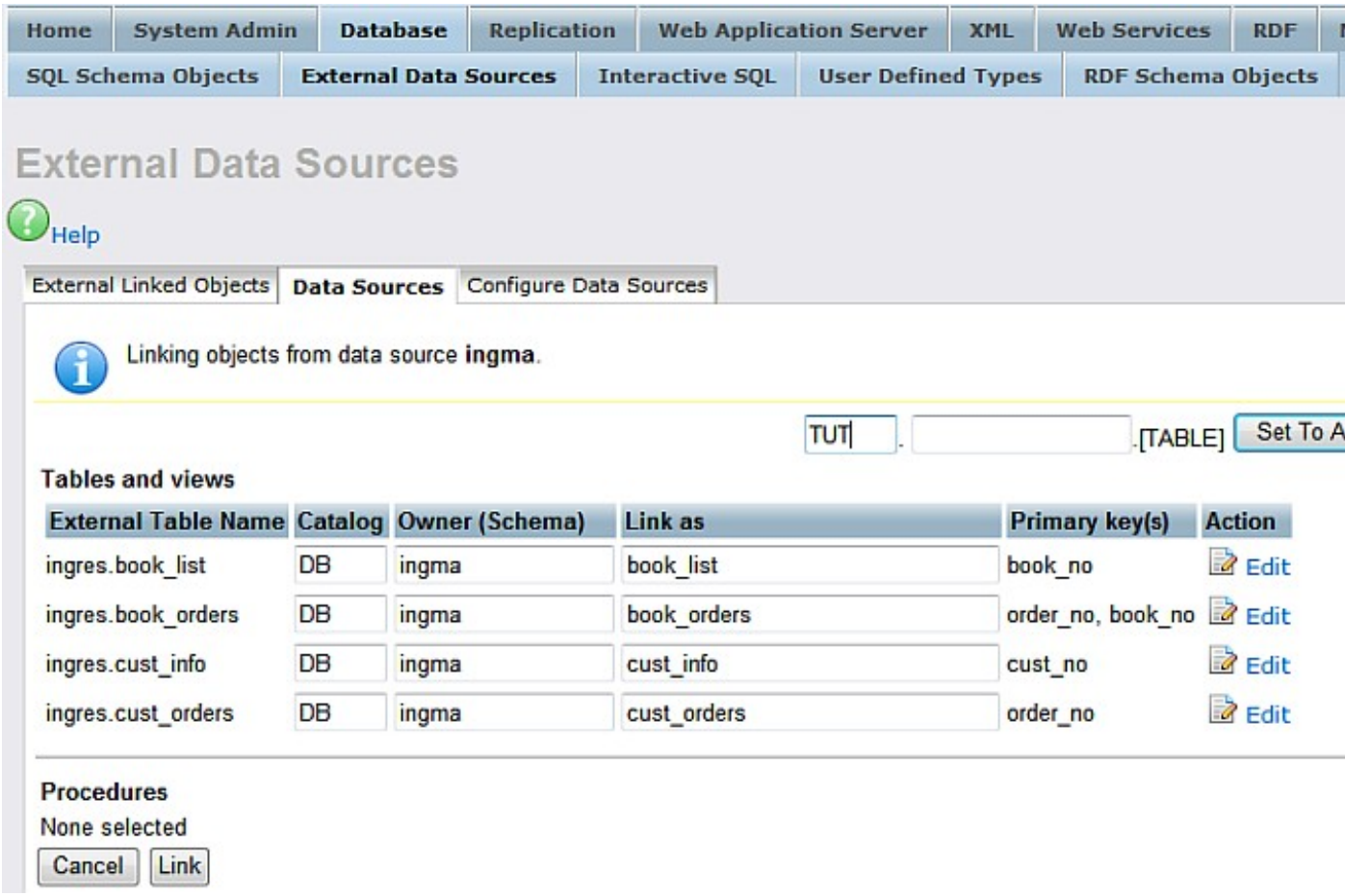
6. Select all the tables that are part of the "TUT" catalog and press the "Link" button to commence the linking process

Figure 8.151. Select all tables



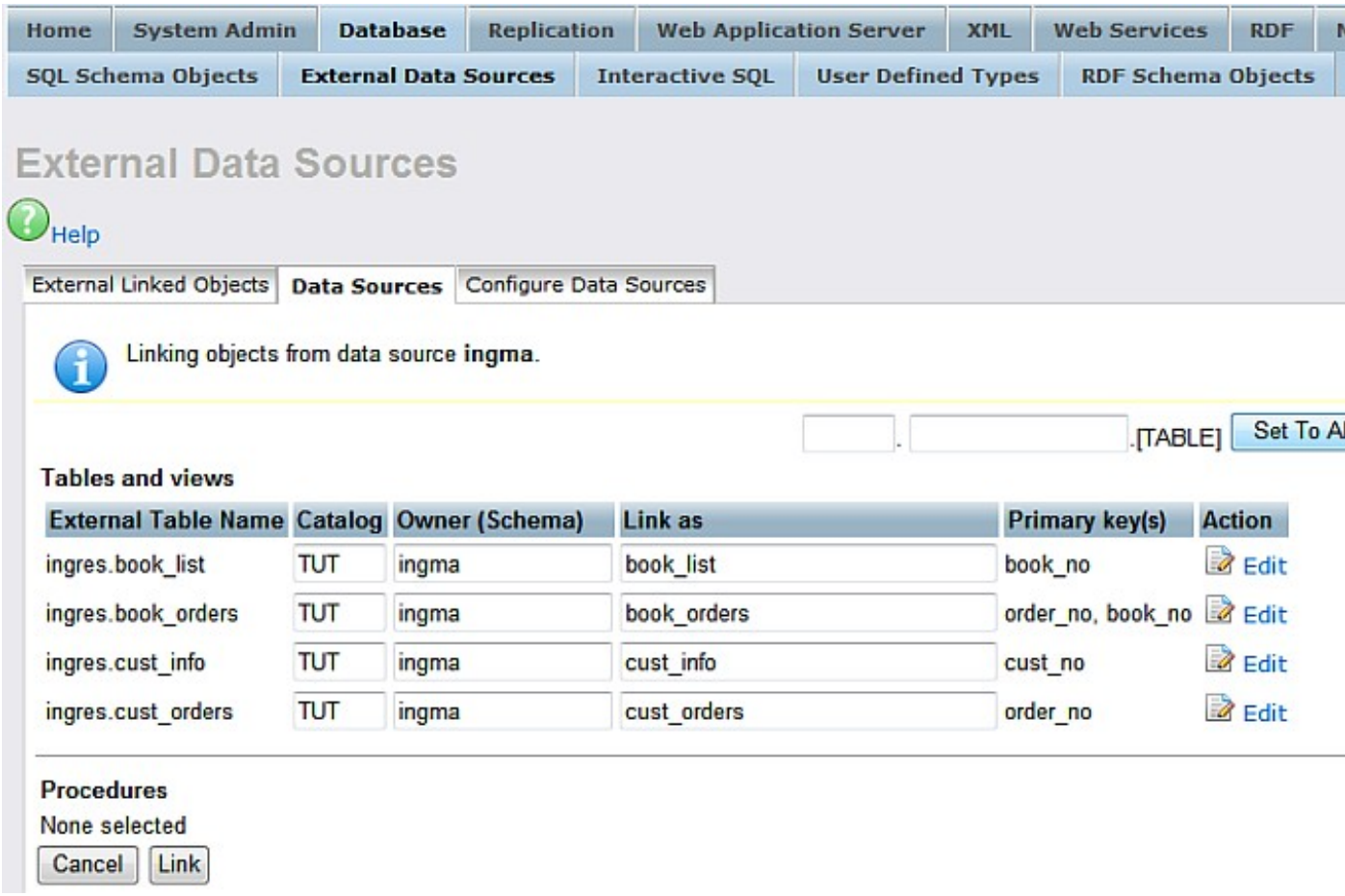
7. Change the Catalog for each table to be "TUT" using the "Set All" button.

Figure 8.152. Catalog



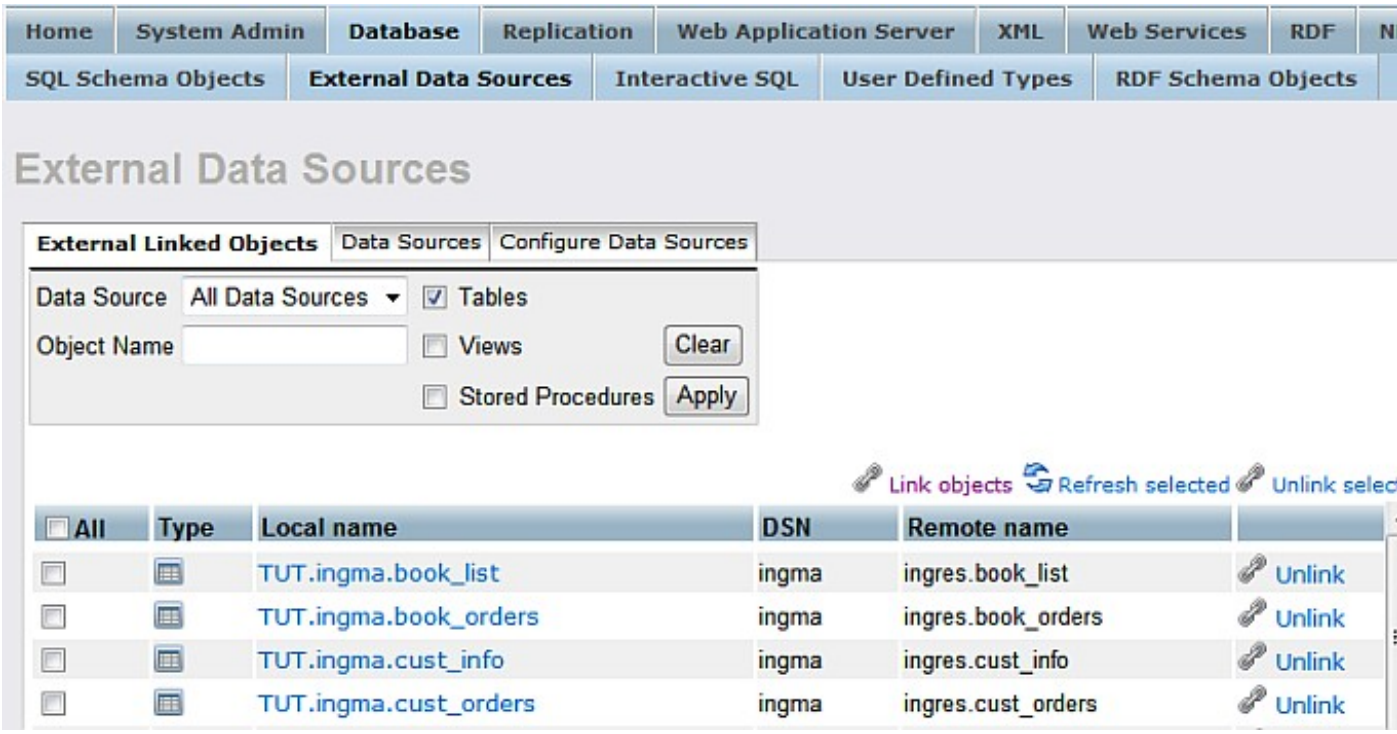
8. All the catalog names are changed to be "TUT".
9. Select the "Link" button to link the selected tables into Virtuoso

Figure 8.153. "Link" button



10. On completion of the Link process the tables will be displayed in the "External Linked Objects" tab.

Figure 8.154. Completion

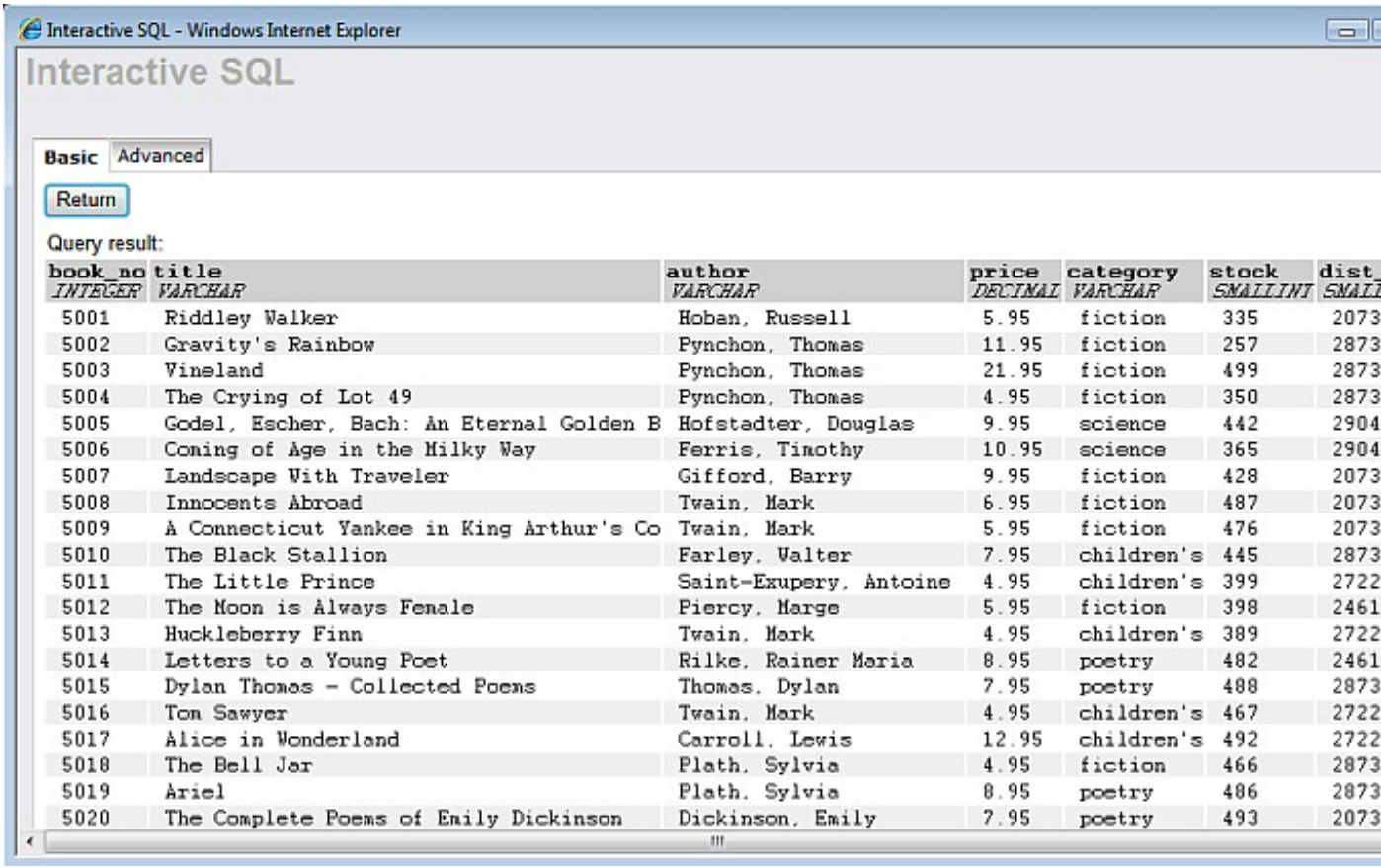


11. The linked tables can be queried by clicking on the hyperlink in the "Local Name" column of the "External Linked Objects" tab above, which loads the Virtuoso "Interactive SQL" interface with the required SQL "Select" for retrieving the remote table data. We shall use the "TUT.ingma.book_list" table to demonstrate this.

Figure 8.155. Querying

12. Then click the "Execute" button to run the query and retrieve the results from the remote table.

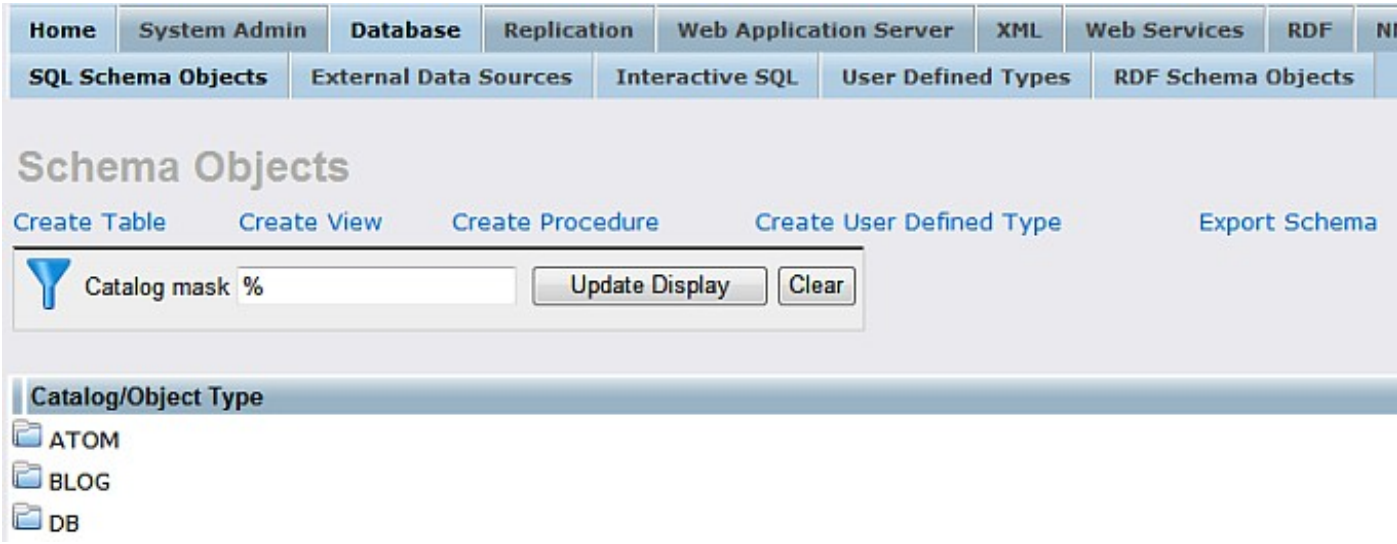
Figure 8.156. Execute



book_no INTEGER	title VARCHAR	author VARCHAR	price DECIMAL	category VARCHAR	stock SMALLINT	dist SMALL
5001	Riddley Walker	Hoban, Russell	5.95	fiction	335	2073
5002	Gravity's Rainbow	Pynchon, Thomas	11.95	fiction	257	2873
5003	Vineland	Pynchon, Thomas	21.95	fiction	499	2873
5004	The Crying of Lot 49	Pynchon, Thomas	4.95	fiction	350	2873
5005	Godel, Escher, Bach: An Eternal Golden B	Hofstadter, Douglas	9.95	science	442	2904
5006	Coming of Age in the Milky Way	Ferris, Timothy	10.95	science	365	2904
5007	Landscape With Traveler	Gifford, Barry	9.95	fiction	428	2073
5008	Innocents Abroad	Twain, Mark	6.95	fiction	487	2073
5009	A Connecticut Yankee in King Arthur's Co	Twain, Mark	5.95	fiction	476	2073
5010	The Black Stallion	Farley, Walter	7.95	children's	445	2873
5011	The Little Prince	Saint-Exupery, Antoine	4.95	children's	399	2722
5012	The Moon is Always Female	Piercy, Marge	5.95	fiction	398	2461
5013	Huckleberry Finn	Twain, Mark	4.95	children's	389	2722
5014	Letters to a Young Poet	Rilke, Rainer Maria	8.95	poetry	482	2461
5015	Dylan Thomas - Collected Poems	Thomas, Dylan	7.95	poetry	488	2873
5016	Tom Sawyer	Twain, Mark	4.95	children's	467	2722
5017	Alice in Wonderland	Carroll, Lewis	12.95	children's	492	2722
5018	The Bell Jar	Plath, Sylvia	4.95	fiction	466	2873
5019	Ariel	Plath, Sylvia	8.95	poetry	486	2873
5020	The Complete Poems of Emily Dickinson	Dickinson, Emily	7.95	poetry	493	2073

13. The tables can also be viewed as part of the Virtuoso "SQL Schema Objects" under the "TUT" catalog name.

Figure 8.157. SQL Schema Objects



The Link process is now complete enabling the tables to be queried as if part of the Virtuoso Schema.

8.4.4. Creating EDM in Visual Studio 2008

The following steps can be used to create an Entity Data Model (EDM) for the Ingres Tutorial database:

1. Launch the Visual Studio 2008 SP1 IDE.

Figure 8.158. Visual Studio 2008 SP1 IDE

2. Create a

   Web Application

   project by going to the

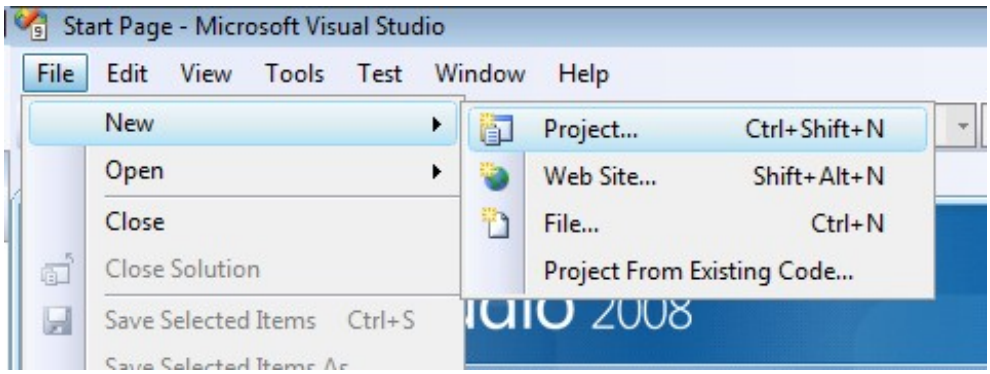
   File

   menu in Visual Studio and choosing

   New Project

   .

   Figure 8.159. Web Application



3. When the New Project window appears, choose either

   Visual Basic

   or

   Visual C#

   as your programming language.
4. Within the language category, click on

   Web

   and select

   ASP.NET Web Application

from the right-hand panel.

5. Choose a name for the project, for example

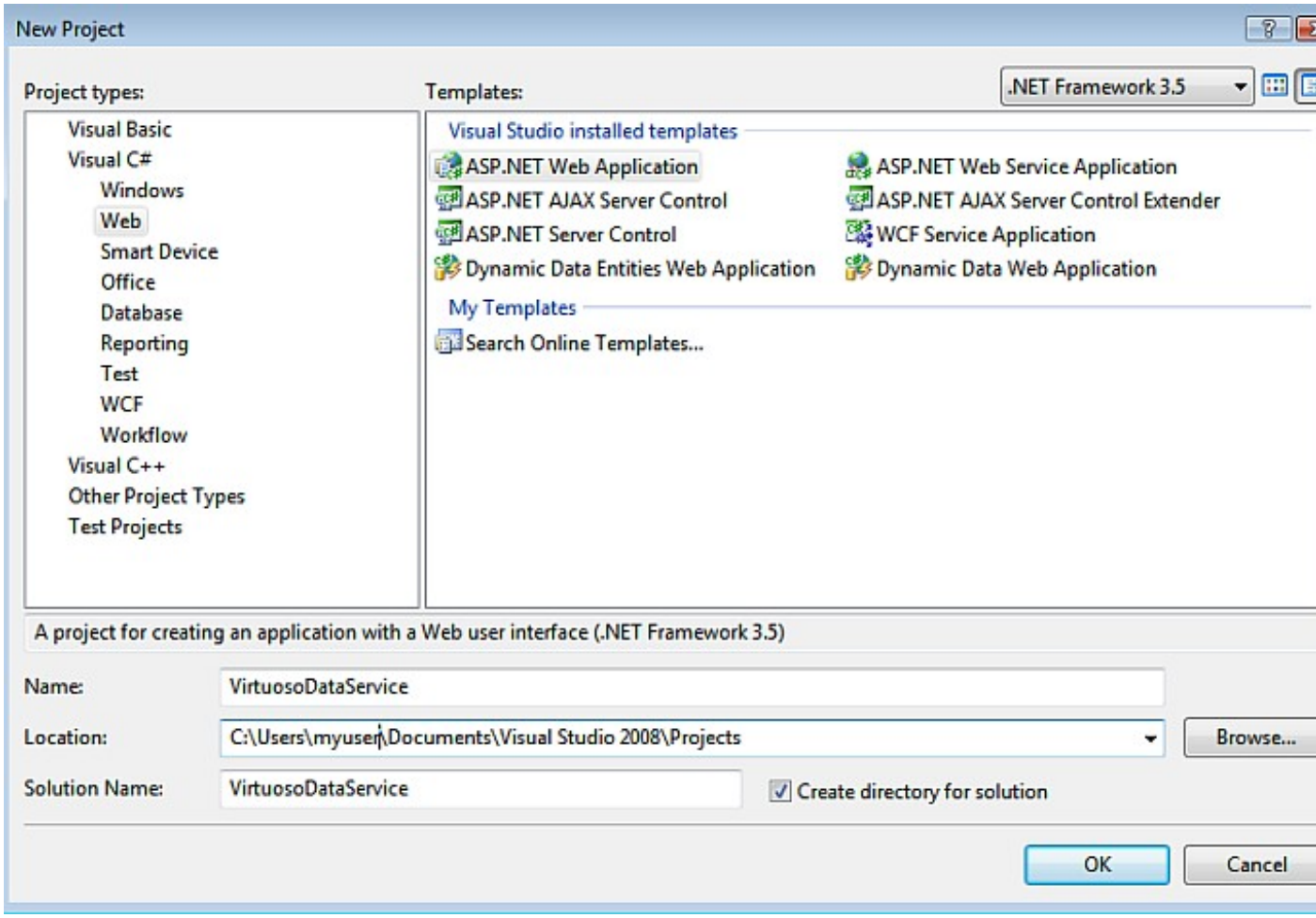
   VirtuosoDataService

   , and click

   OK

   .

   Figure 8.160. name for the project

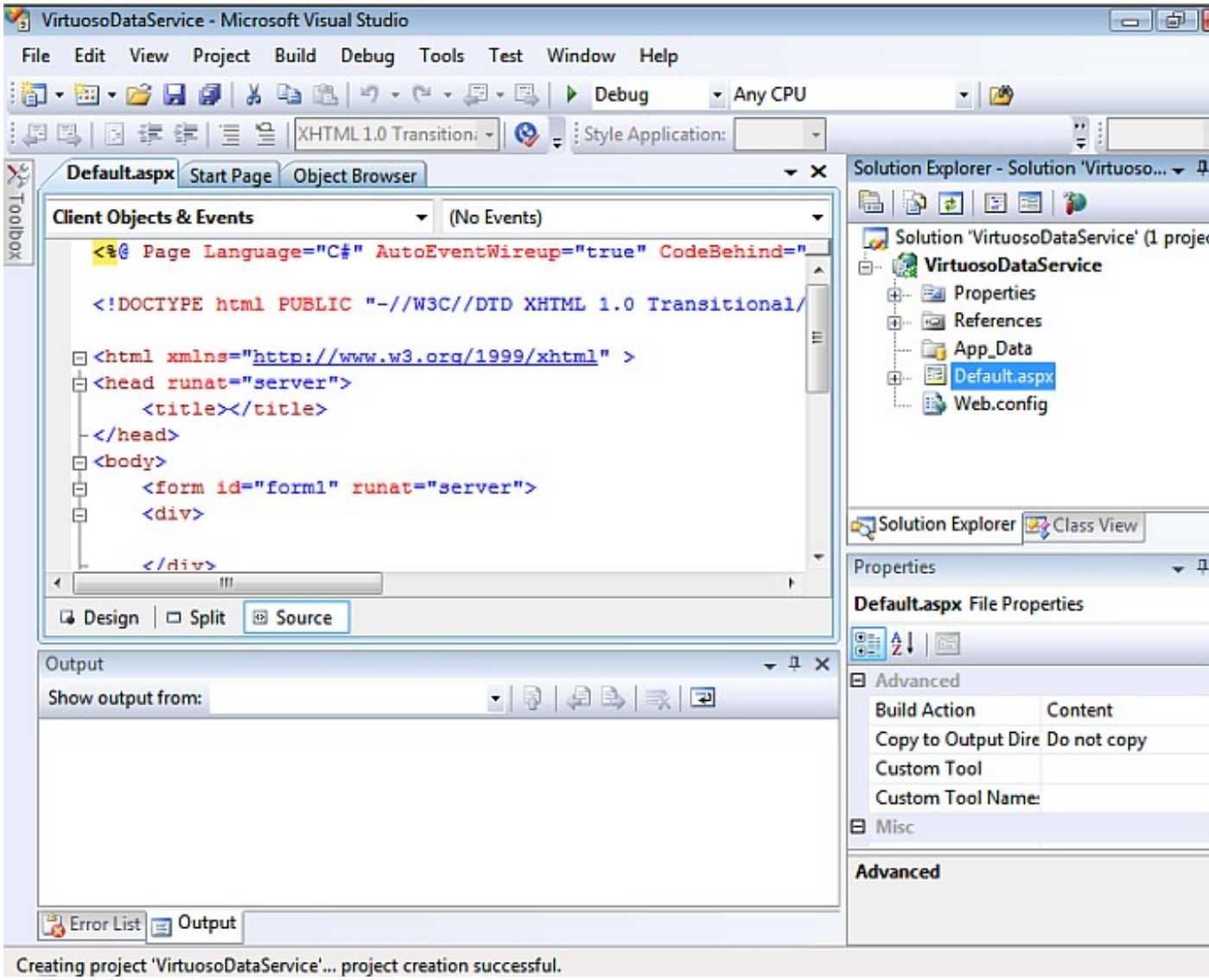


6. This will create a new project called

   VirtuosoDataService

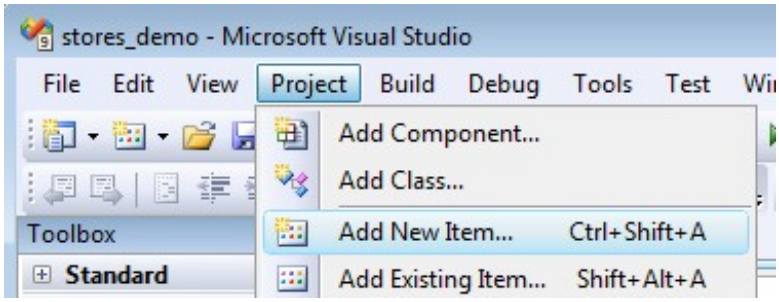
   .

   Figure 8.161. create a new project



7. Select the Project -> Add New Item menu option.

Figure 8.162. VirtuosoDataService



8. The

Add New Item

dialog will appear. Choose the

ADO.NET Entity Data Model

template. Give it the name

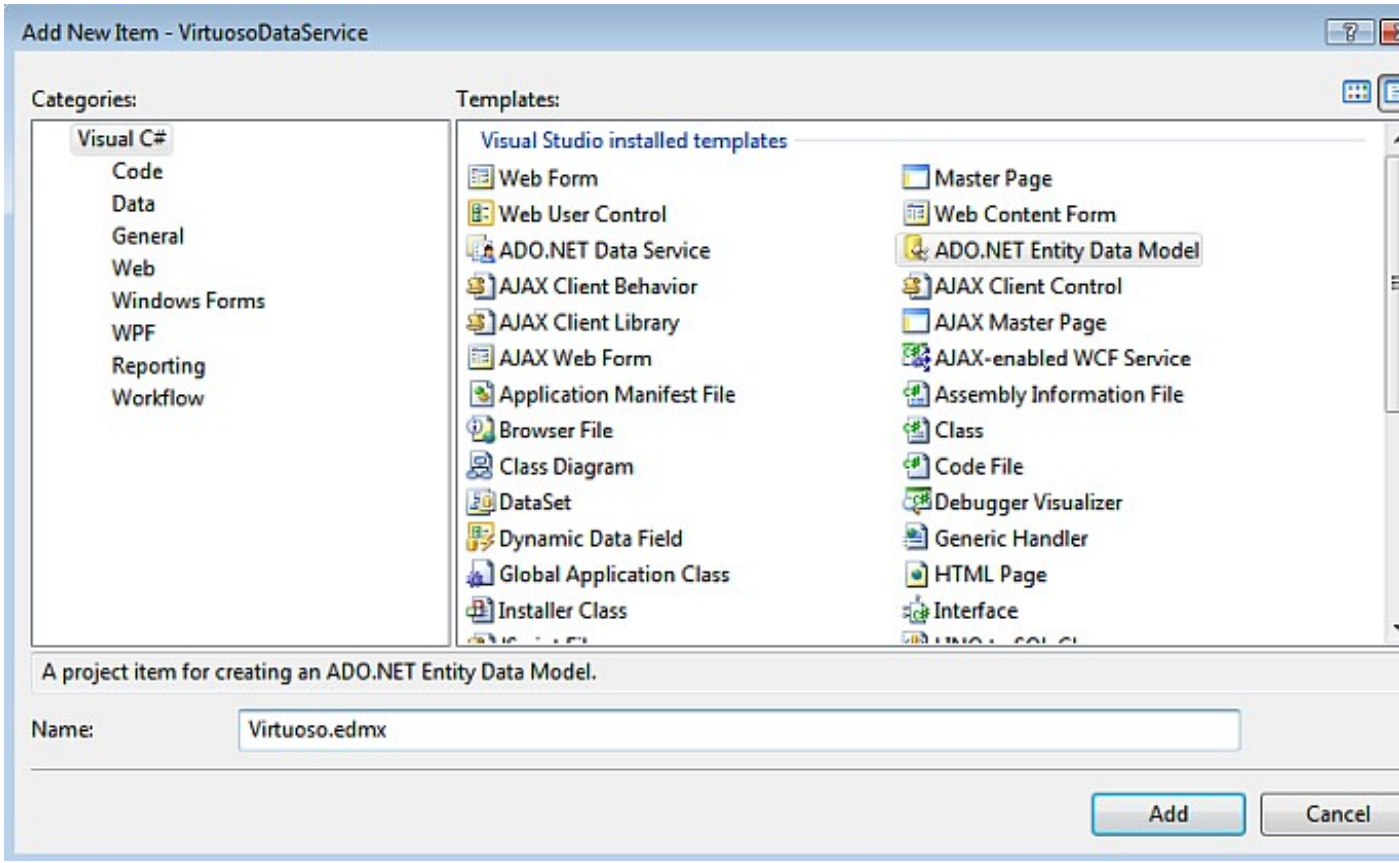
Virtuoso.edmx

and click

Add

to start the creation of the ADO.Net Entity Data Model.

Figure 8.163. Add New Item



9. In the

Entity Data Model Wizard

dialog

Choose Model Contents

page select the

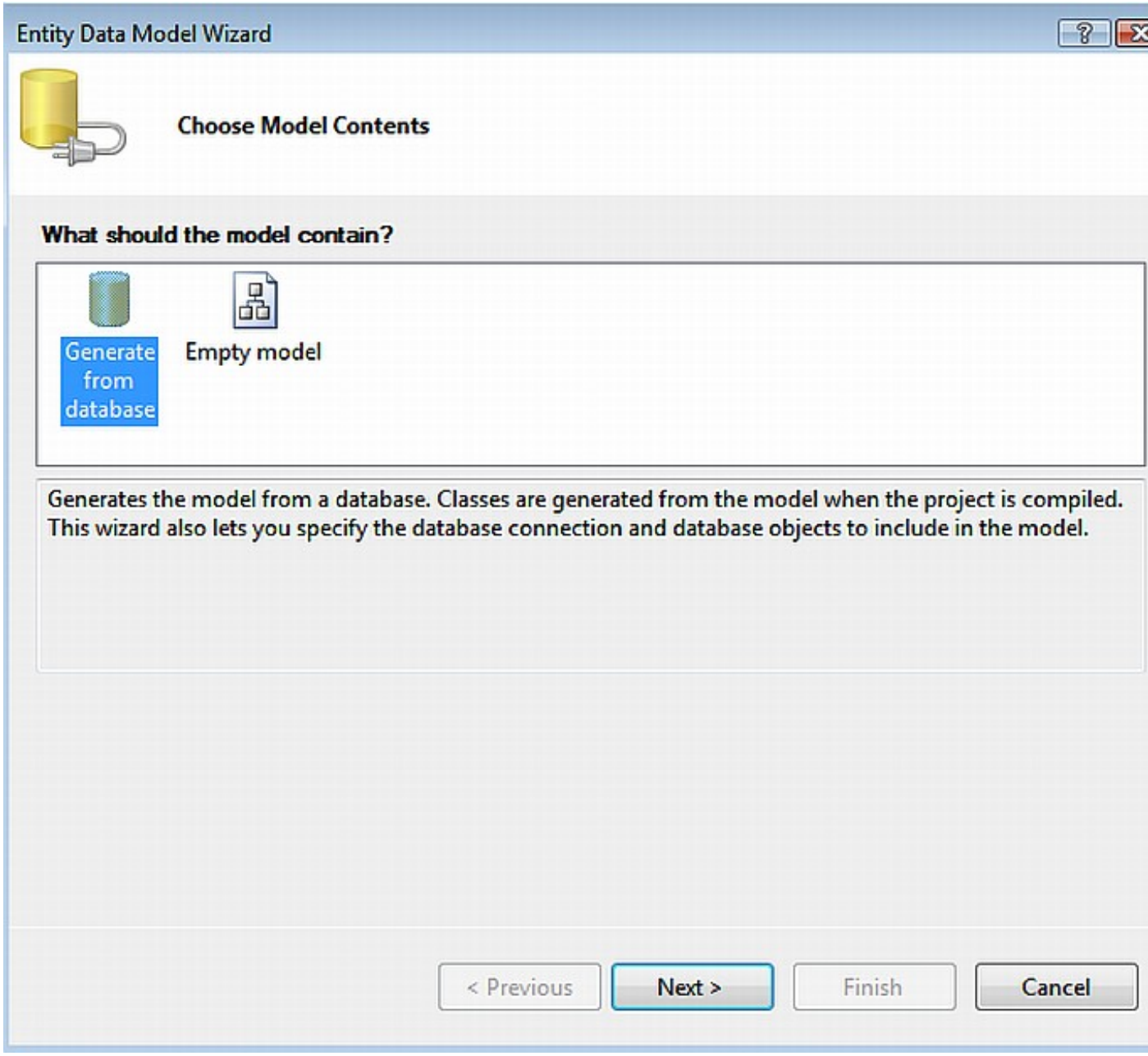
Generate from Database

model type and click

Next

.

Figure 8.164. Choose Model Contents



10. In the

    Entity Data Model Wizard

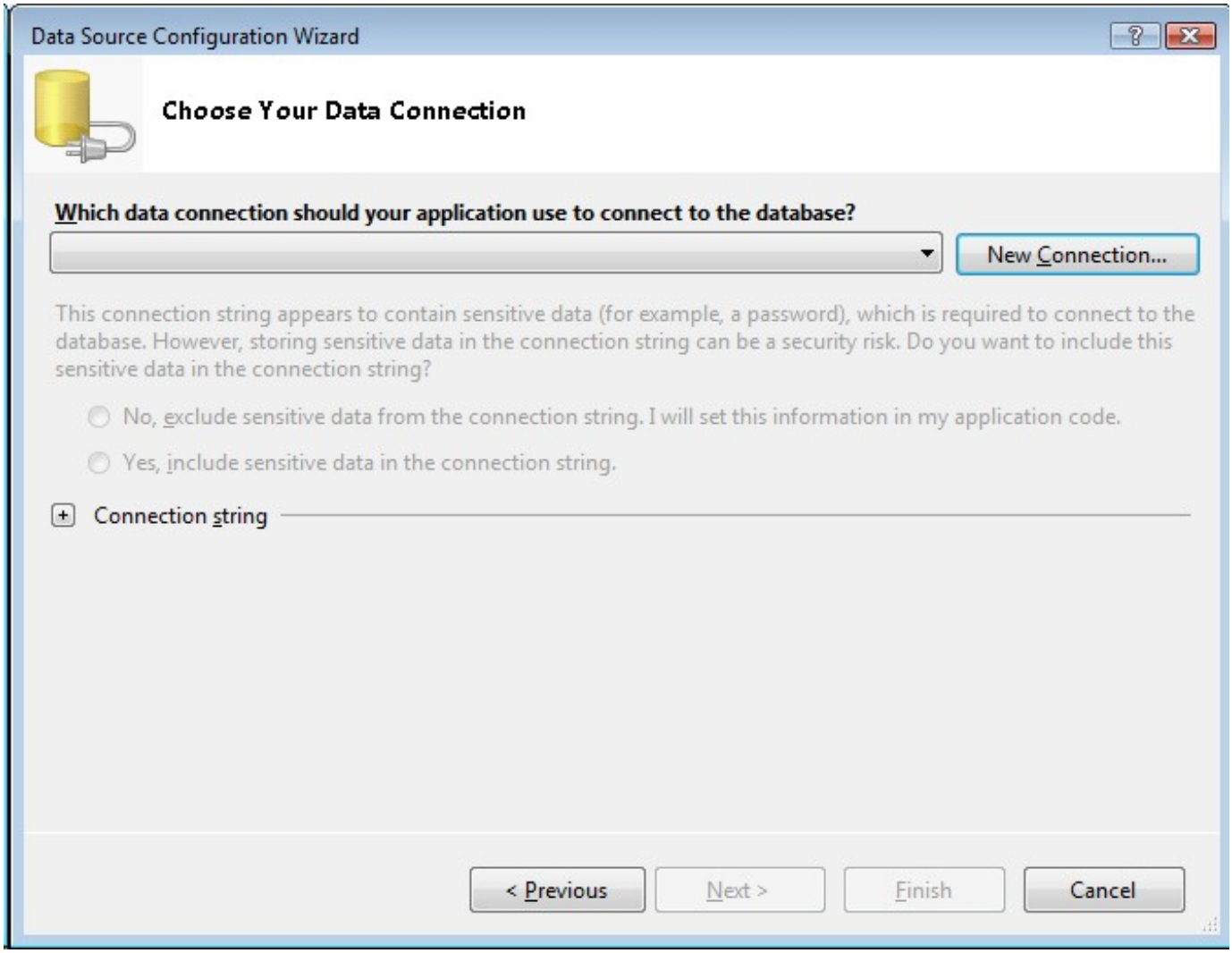
    dialog

    Choose your Data Connection

    page, select the

    New Connection button

Figure 8.165. Entity Data Model Wizard



11. In the

    Choose Data Source

    dialog, select the OpenLink

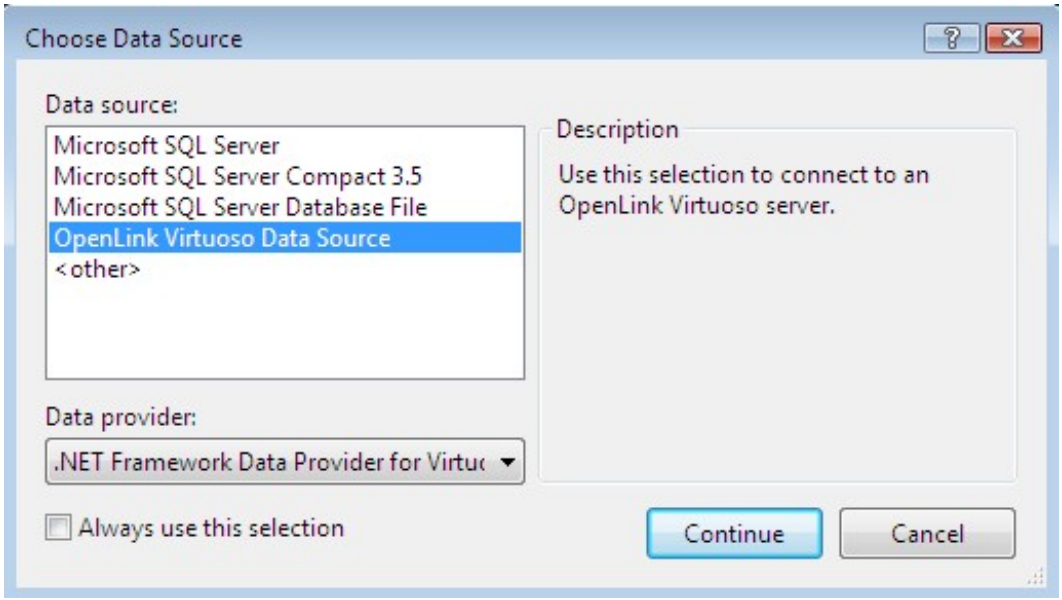
    Virtuoso Data Source

    from the list and click

    Continue

    .

Figure 8.166. Choose Data Source



12. In the

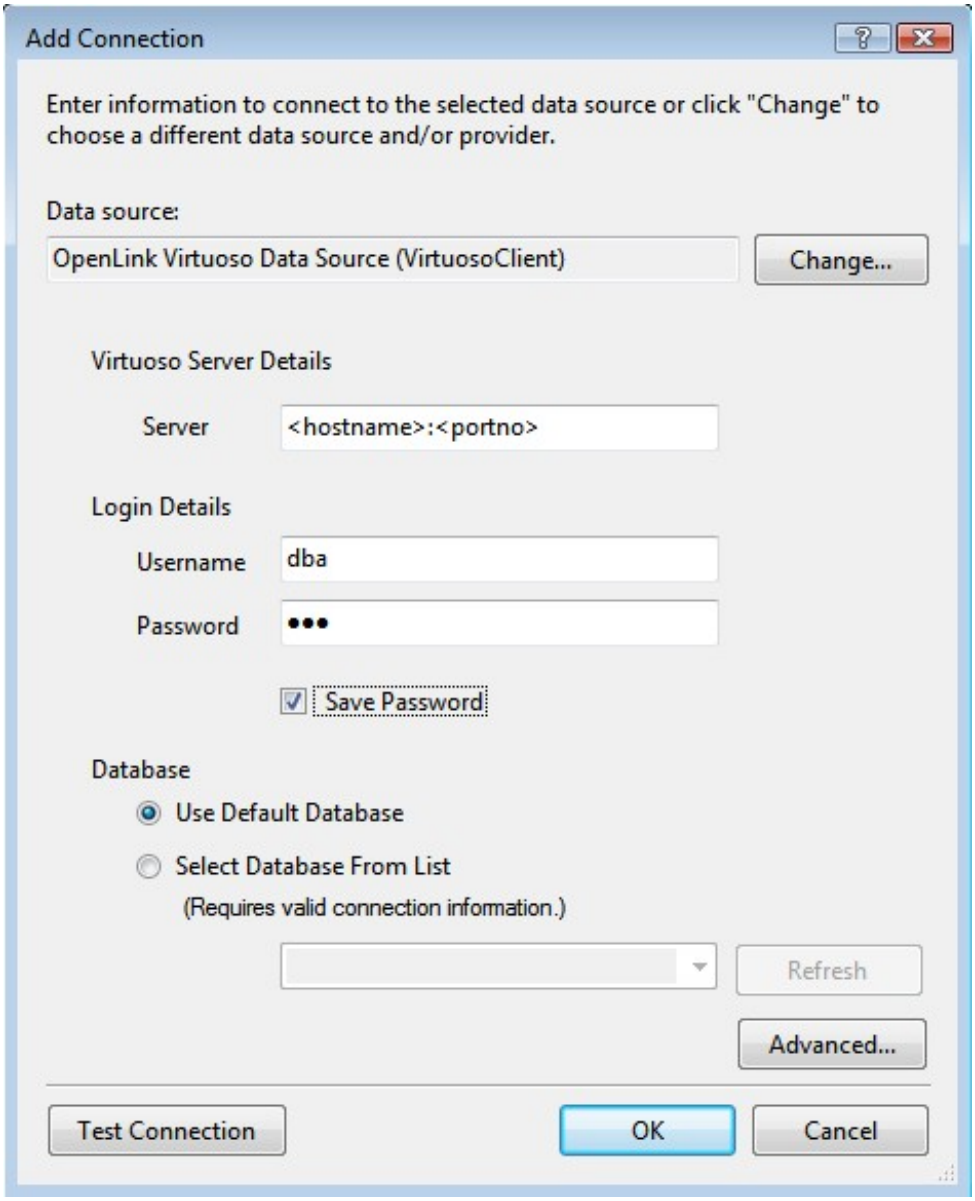
    Add Connection

    dialog, specify the

    hostname, portno, username and password

    for the target Virtuoso Server and check the Save Password check box.

    Figure 8.167. Connection Properties



13. Select the

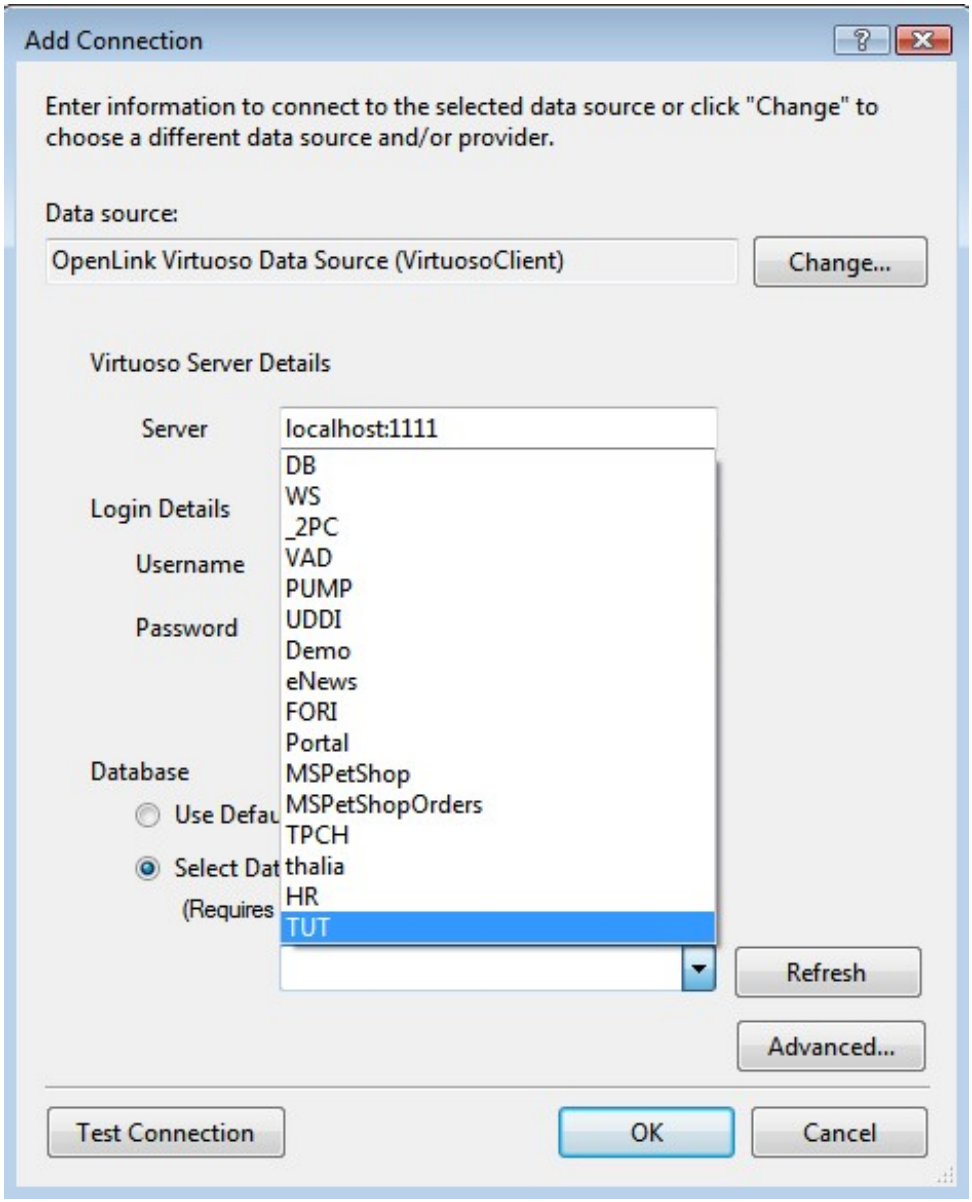
    Select Database From List

    radio button and choose the

    TUT

    database from the drop down list.

Figure 8.168. Add conneciton

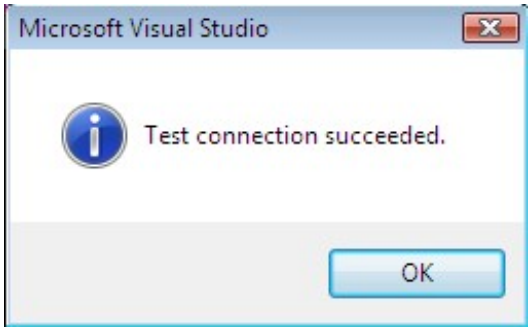


14. Press the

    Test Connection

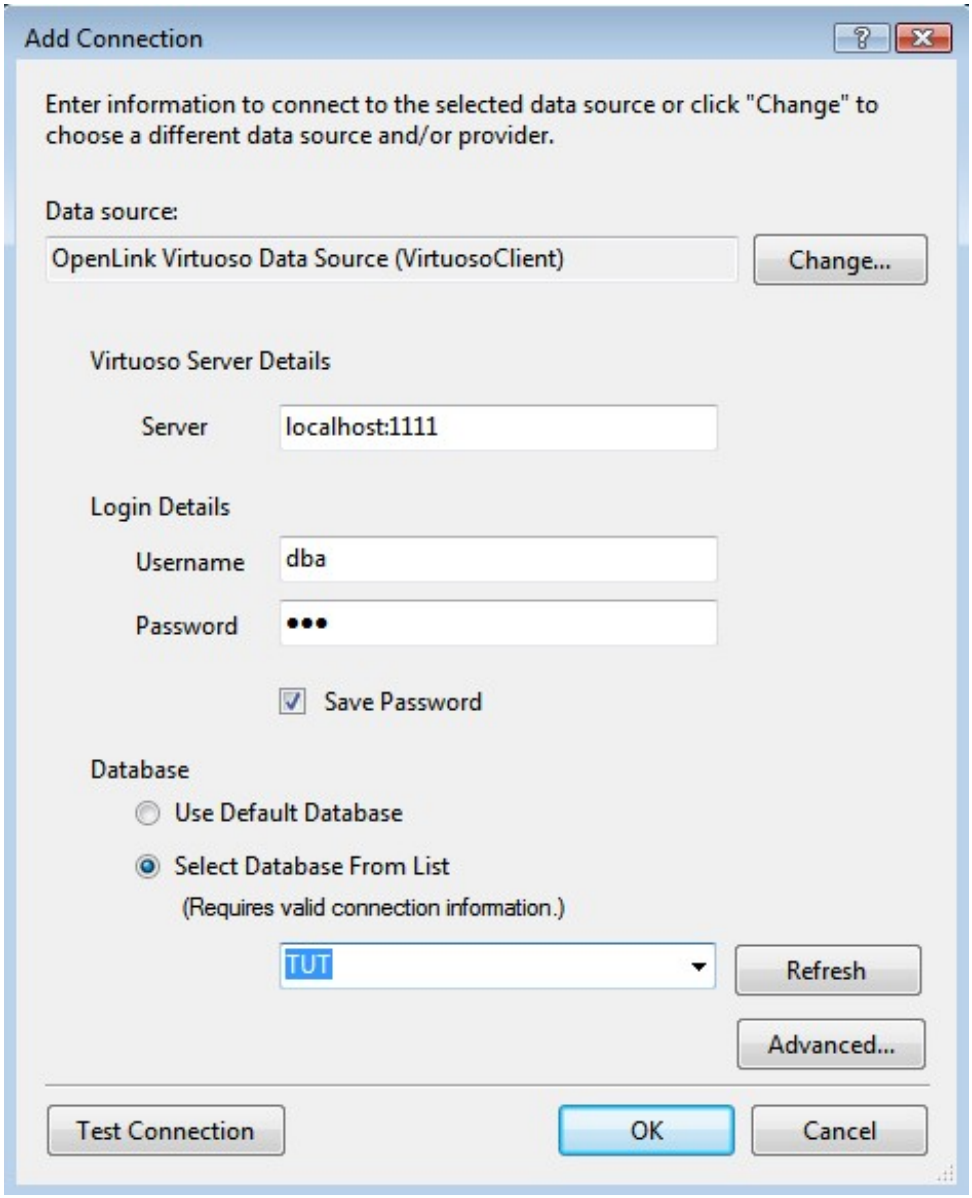
    dialog to verify that the database is accessible.

Figure 8.169. Test Connection



15. Click OK to add the connection.

Figure 8.170. Test Connection



16. Set the

    entity connect string

    name to

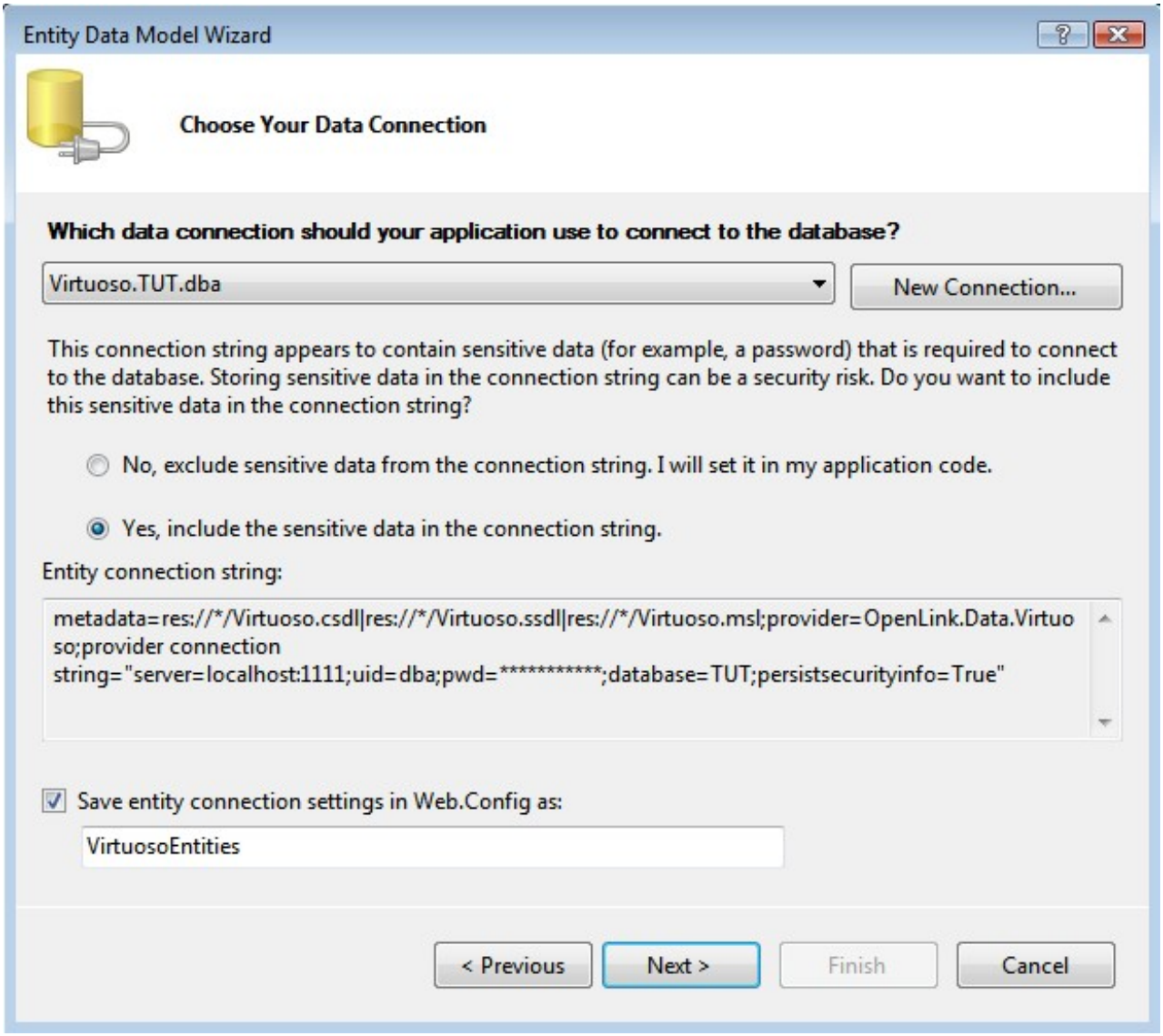
    VirtuosoEntities

    (note this name as it is required in the section on creating and ADO.Net Data Service) and click

    Next

    .

    Figure 8.171. entity connect string



17. In the

    Choose your Database Objects

    page, tick the

    Tables

    check box to select all tables in the TUT catalog for addition to the Entity Data Model. Set the

    Model Namespace

    to

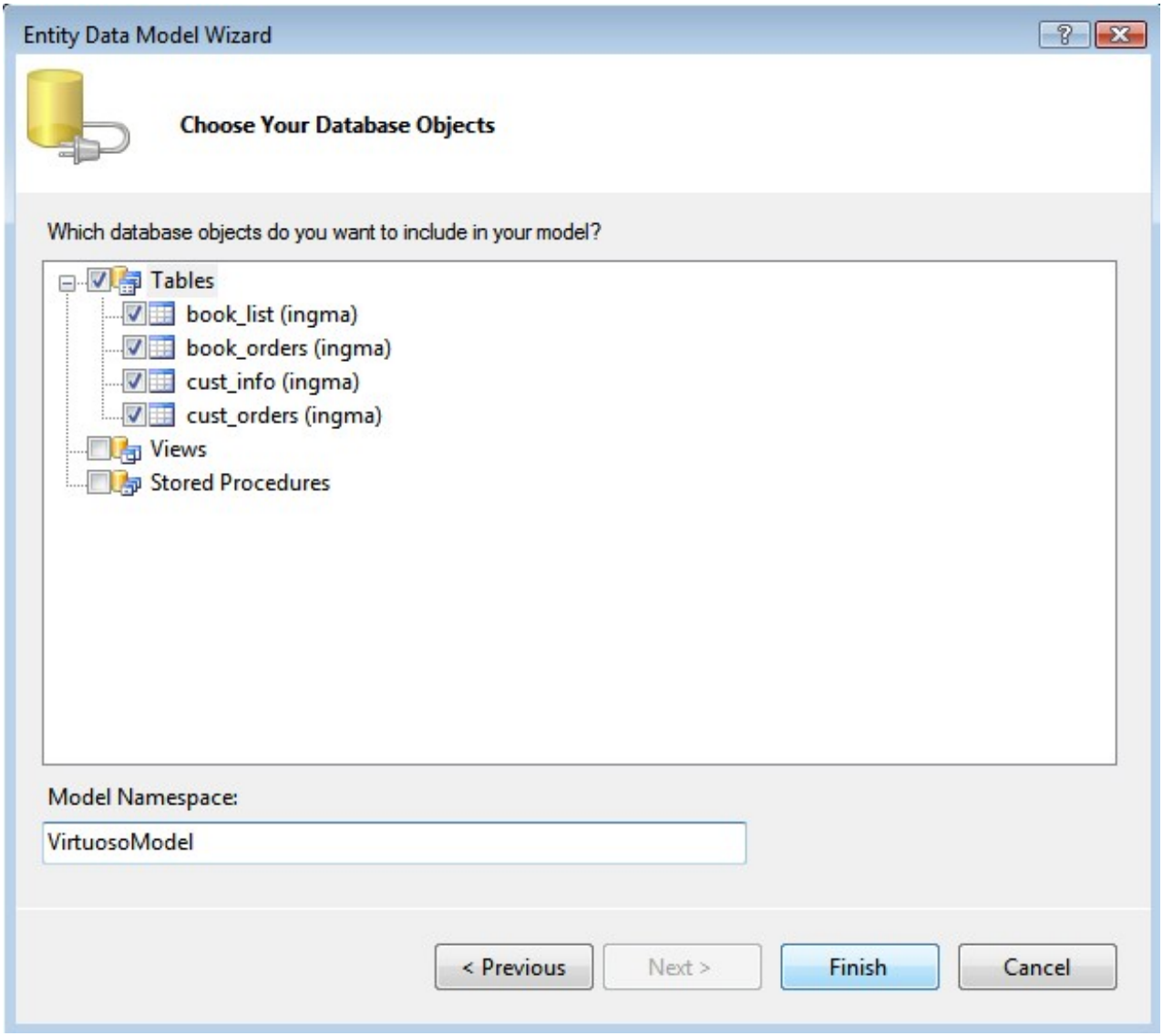
    VirtuosoModel

    and click

    Finish

    .

Figure 8.172. Database Objects

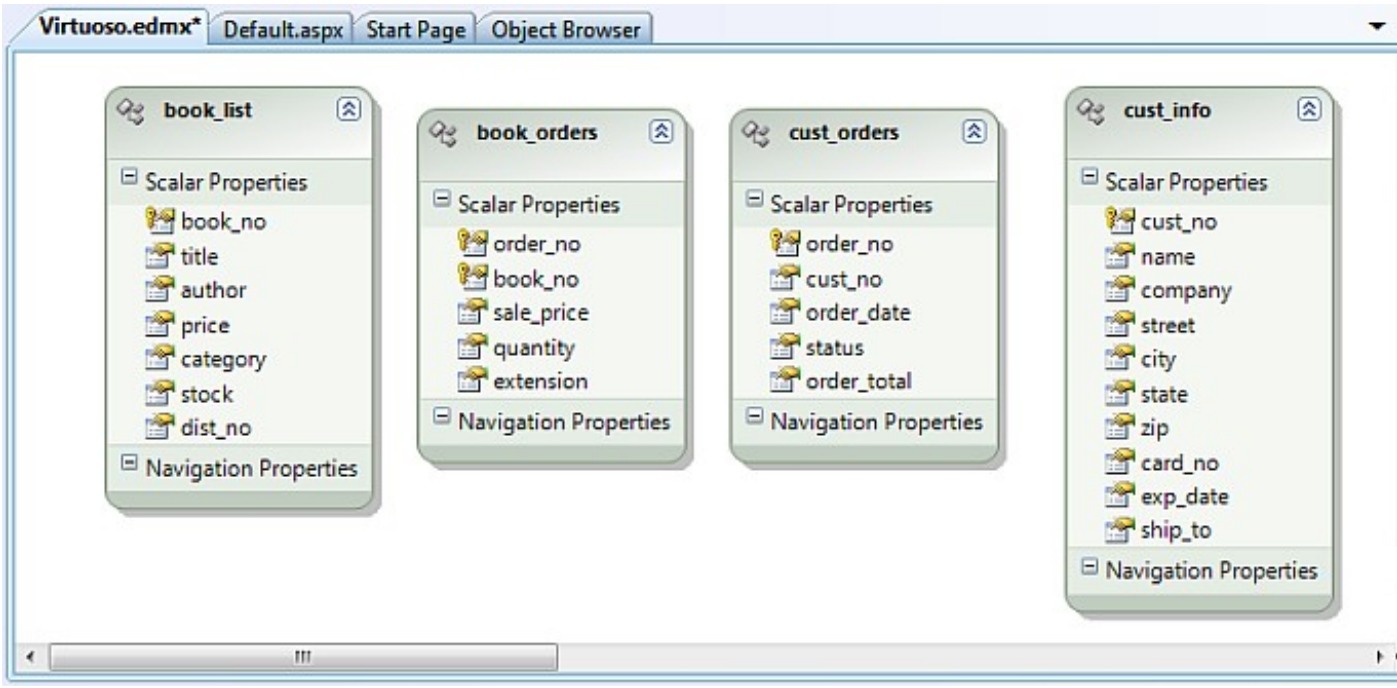


18. The

    Virtuoso.edmx

    EDM will be created with the tables and relationships displayed in the Visual Studio IDE

    Figure 8.173. Virtuoso.edmx



Creation for the Entity Data Model for the Ingres Tutorial database is now complete.

8.4.5. Manually creating EDM Associations (FKs) for the Ingres Tutorial database

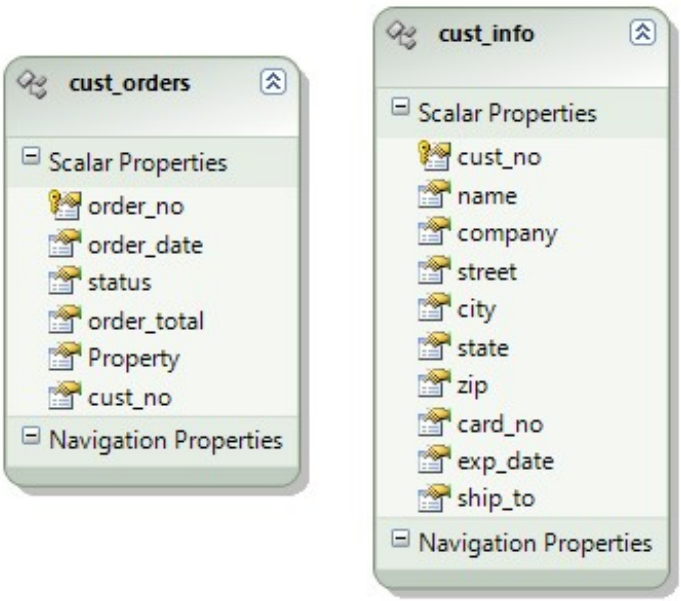
The following steps will detail what is required to manually create "Associations" in your Entity Data Model.

Before commencing the process, you will need to determine where these associations exist and their multiplicity (e.g., one-to-one, one-to-many, etc.).

Note: These steps will need to be repeated for each association.

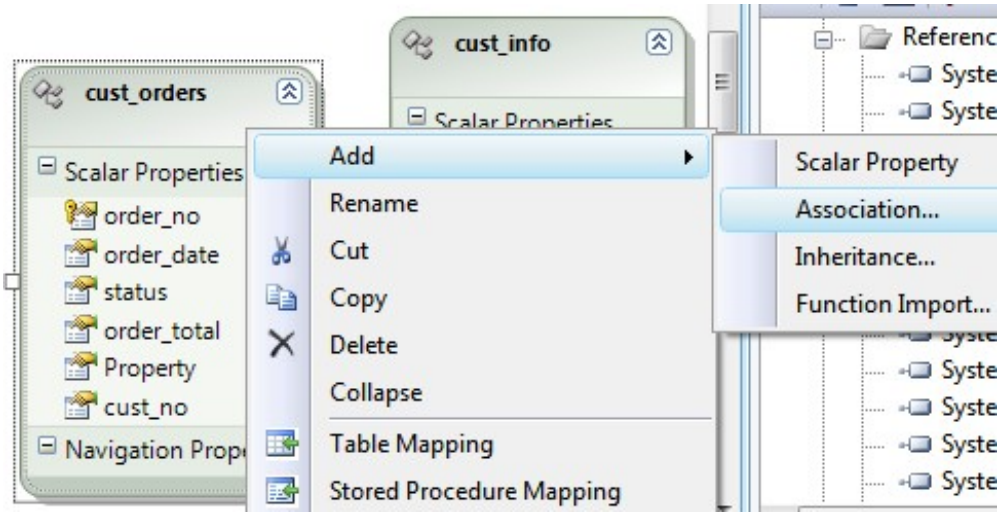
1. The first Association I will deal with is the relationship between cust_orders and cust_info, identified by the presence of the Scalar Property cust_no in both entities. This is a one-to-one mpmprelationship, as an entry in a customer order may only be associated with one customer's information.

   Figure 8.174. Association



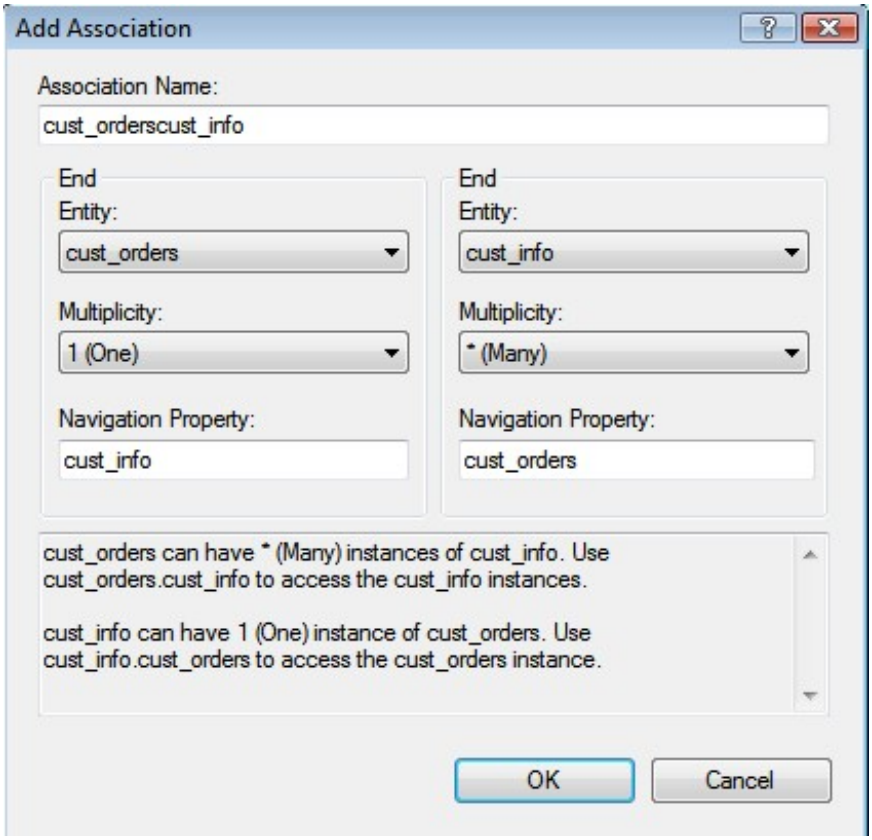
2. To add the Association, right click on the Customer entity then Add -> Association.

Figure 8.175. add the Association



3. You will now see the Add Association dialog.

Figure 8.176. Add Association dialog



4. Once you then hit OK the diagram is refreshed to include the newly created association.
5. You now need to edit the mappings associated with the newly created association, so right-click the association on the diagram and select

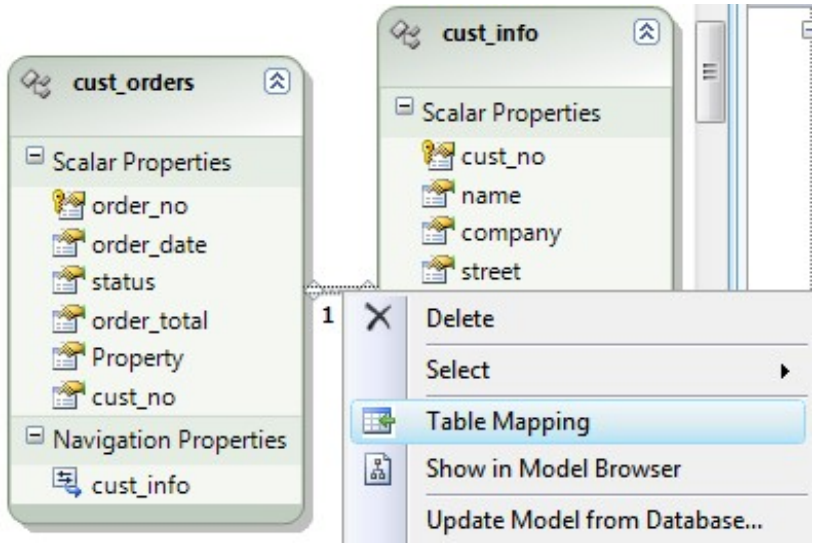
   Table Mapping

   to display the

   Mapping Details

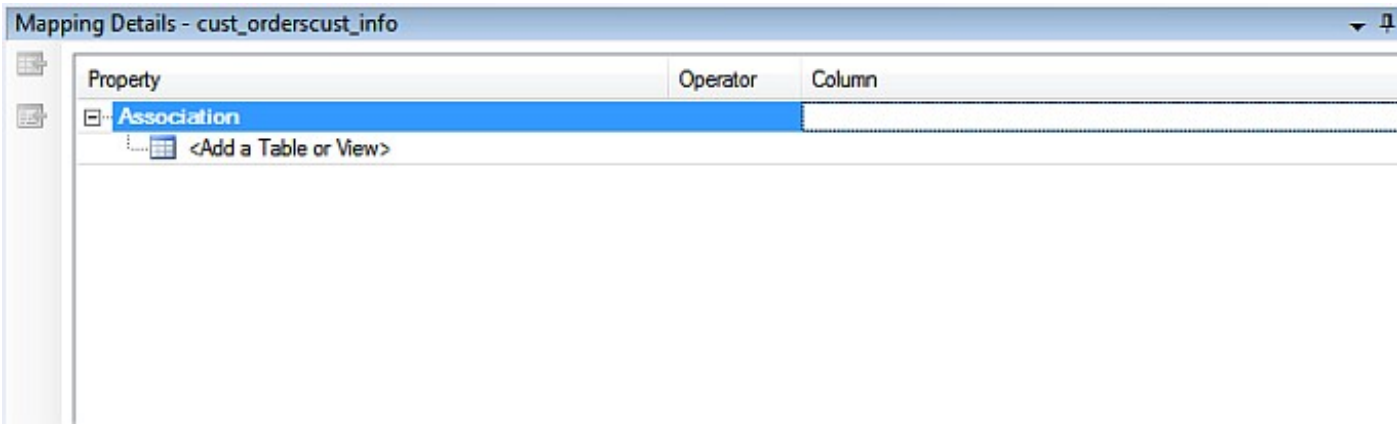
pane.

Figure 8.177. Navigation Property name



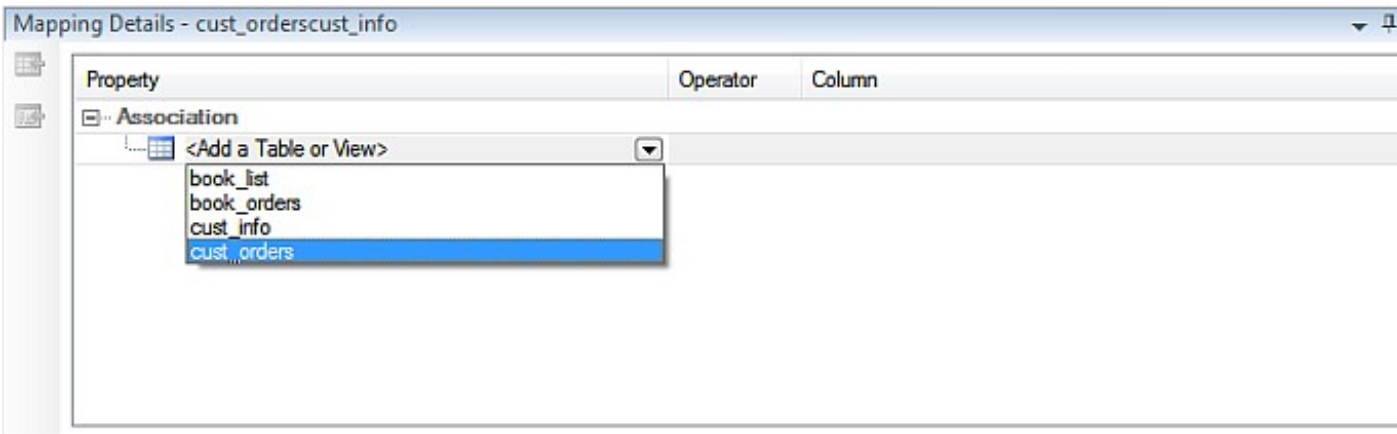
6. Click that line reading <Add a Table or View> to reveal a drop down list of all entities.

Figure 8.178. Add a Table or View



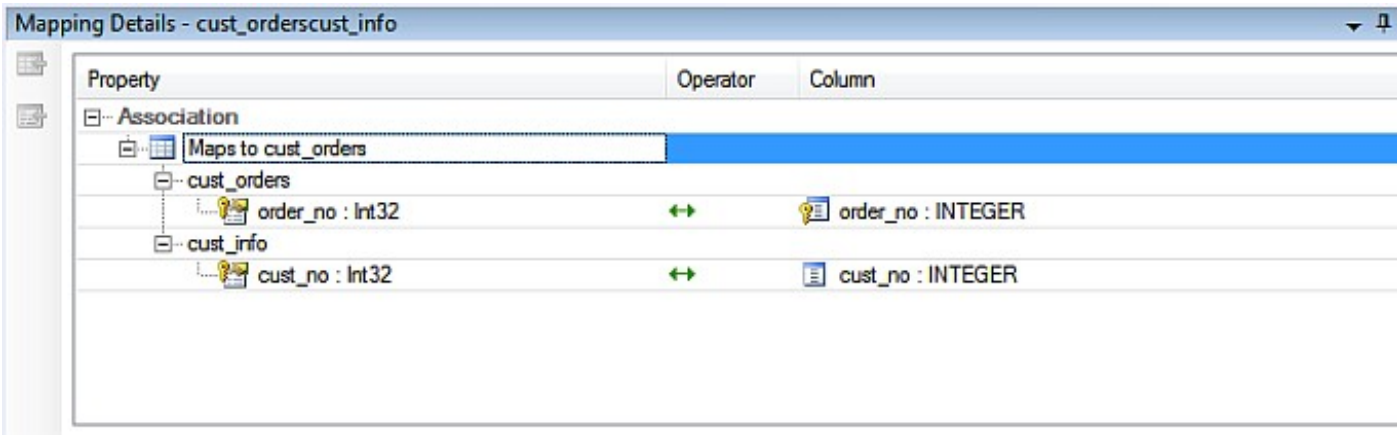
7. Here you need to select the entity on the right/far side of the association (the entity where the foreign key exists). In this example, it is the cust_orders entity.

Figure 8.179. Entity



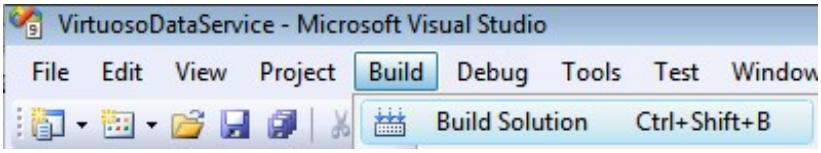
8. The Mapping Details pane now refreshes to display both ends of the association, requiring that you provide relevant target store data types in the Column column for the key fields, as depicted here.

Figure 8.180. Mapping Details



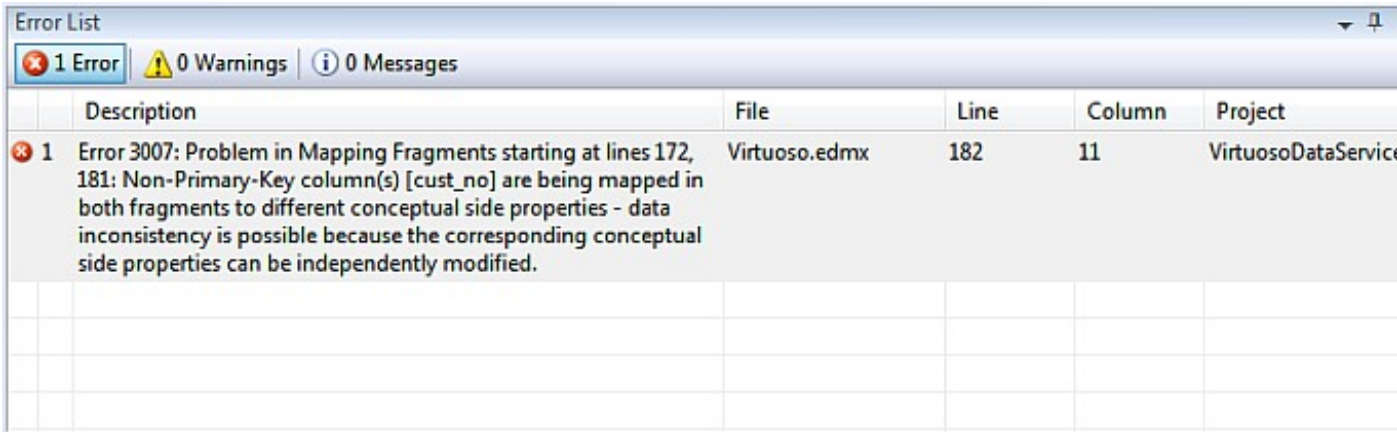
9. Once the mapping is complete, you can build the project using Build -> Build Solution. NOTE: It is worthwhile building as each association is made, since the error messages can be a little confusing.

Figure 8.181. Build the project



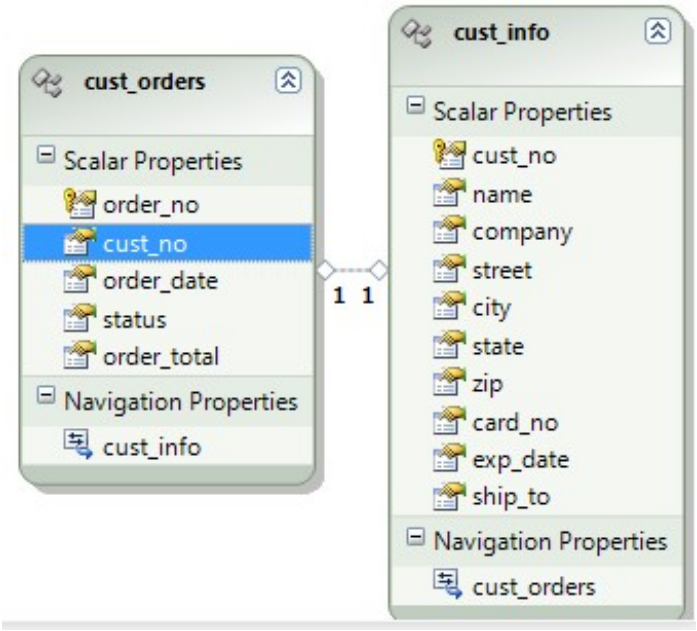
10. This should result in the following error:

Figure 8.182. Error



This error indicates that there are two sources - in this case, the cust_orders entity's Scalar Property cust_no and the Navigation Property cust_info in - which are being mapped to the same target column - the Ingres column cust_orders.cust_no - which is not supported.

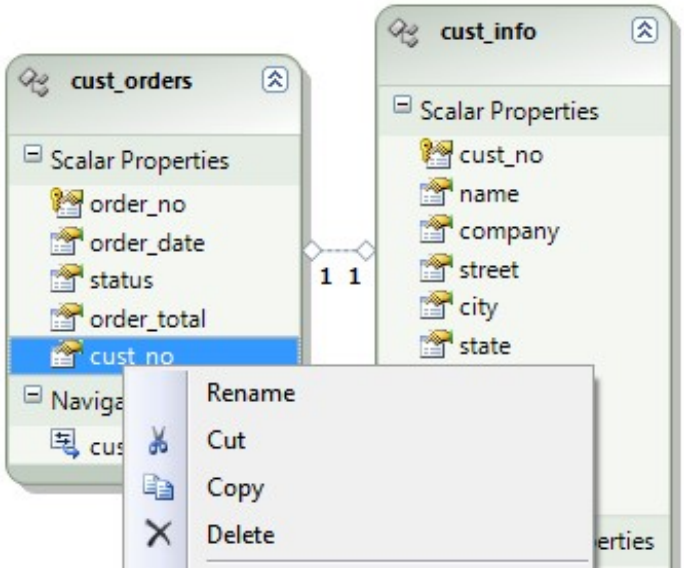
Figure 8.183. Error



11. The solution is simple! Simply delete the mapping for Scalar Property cust_orders.cust_no since its only purpose is to hold data representing a relationship/association (it is a Foreign Key), which has already been represented by the newly-created association and resulting Navigation Property cust_no.

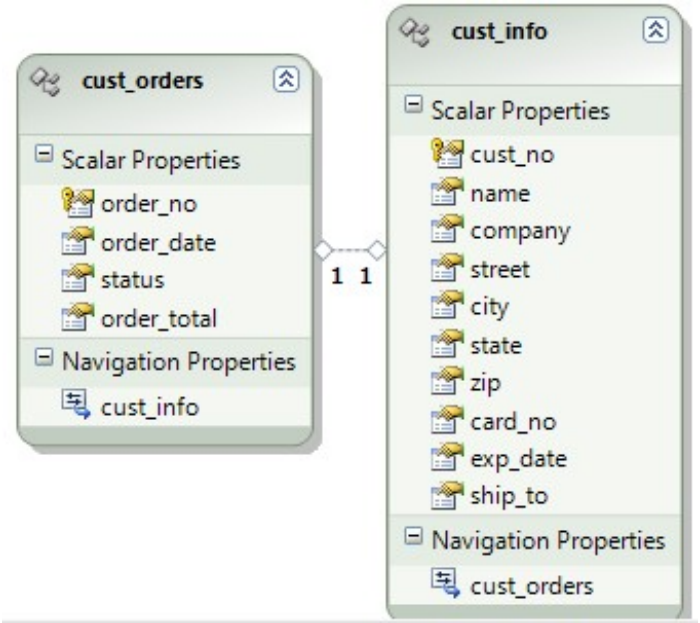
    Right-click on cust_orders.cust_no then Delete.

    Figure 8.184. delete the mapping



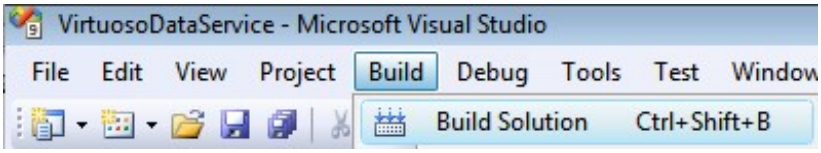
12. The model diagram will refresh to reflect this change.

    Figure 8.185. Model Diagram



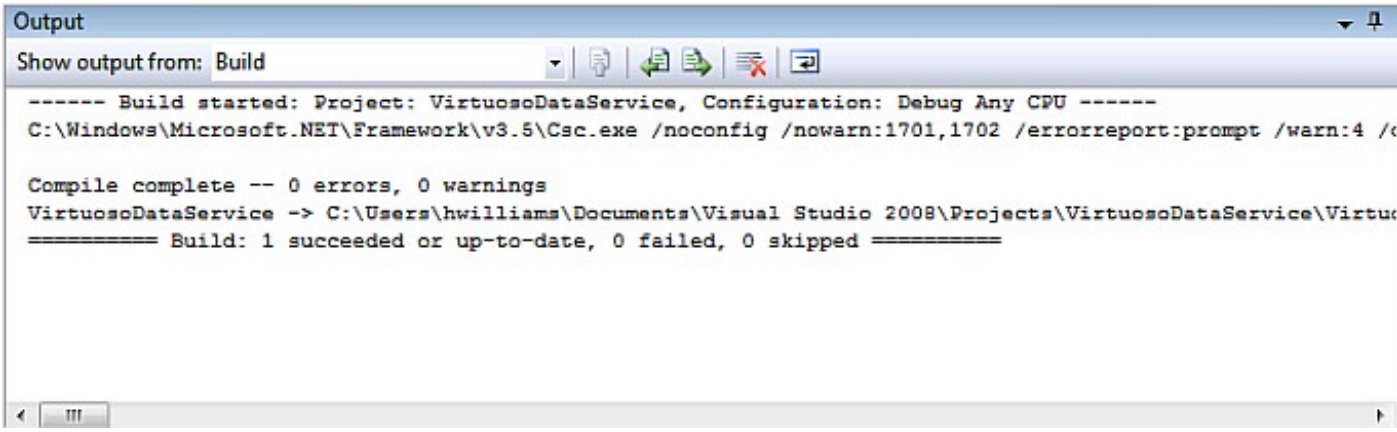
13. Build the project, again, using Build -> Build Solution.

Figure 8.186. Build project



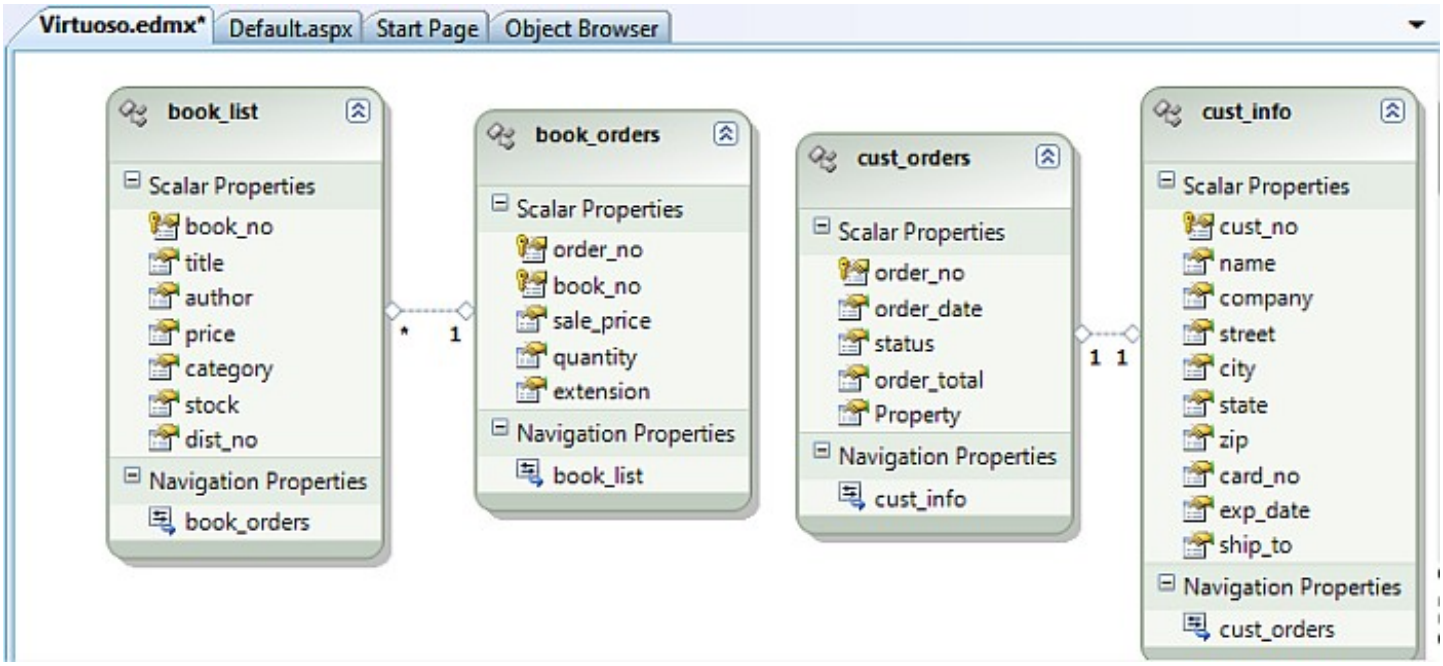
14. The project should now be fine.

Figure 8.187. Build Project



You will need to repeat these steps for each association until you have a completed Entity Data Model.

Figure 8.188. Entity Data Model



8.4.6. Using EDM to create Entity Framework based applications

Now that a Microsoft Entity Data Model has been created for the Ingres Tutorial database, Entity Framework applications can be created to make use of it.

Entity Frameworks based ADO.NET Data Service

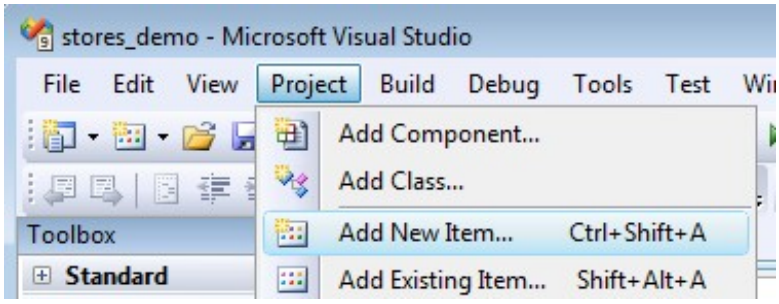
An ADO.Net Data Service for the Ingres tables can be created using the Entity Data Model created in the Creating EDM in Visual Studio 2008 section .

1. Open the

   VirtuosoDataService

   project created in the Creating EDM in Visual Studio 2008 section .
2. Select the Project -> Add New Item menu option.

   Figure 8.189. VirtuosoDataService

3. The

   Add New Item

   dialog will appear. Choose the

   ADO.NET Data Service template

   . Give it the name

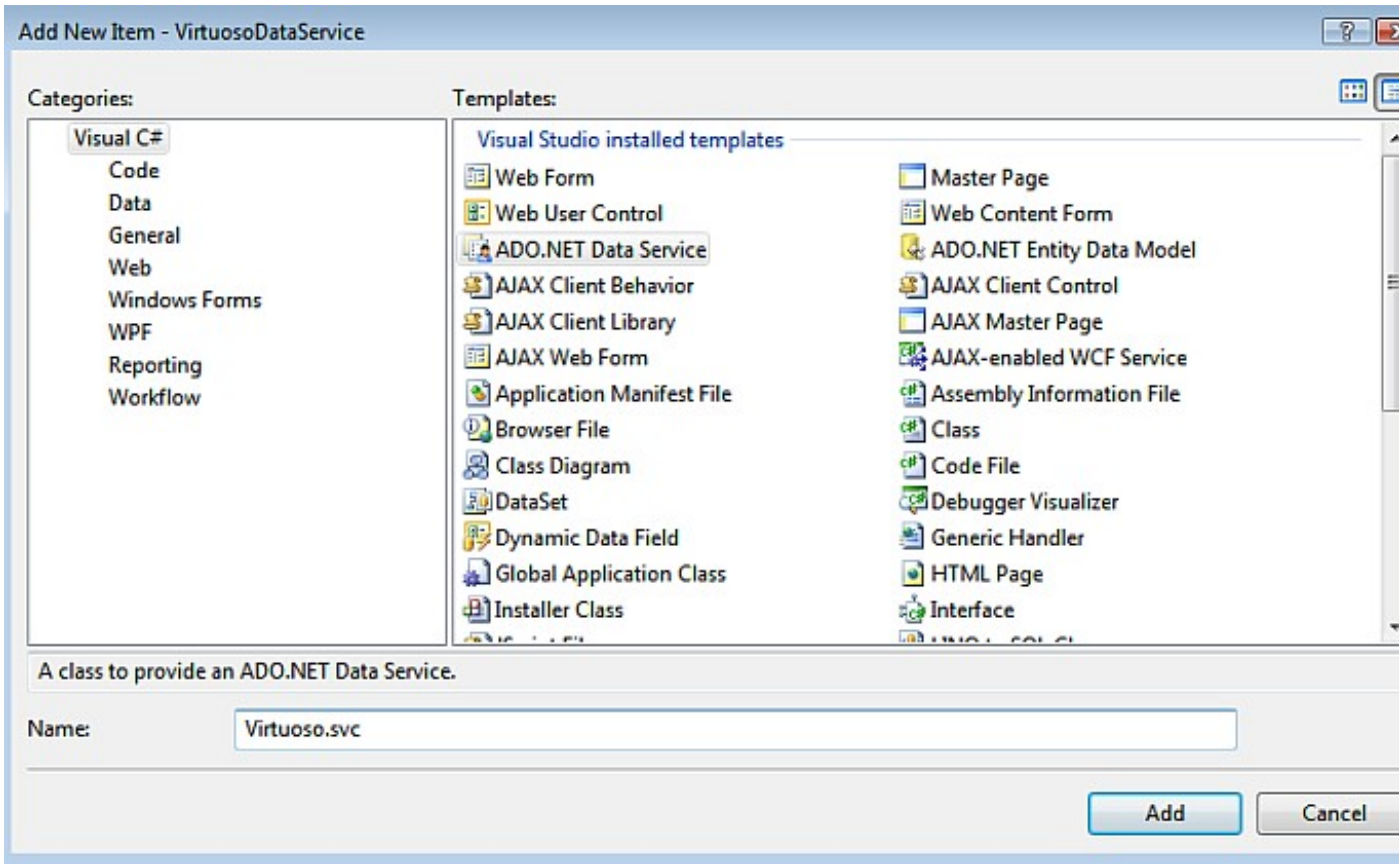
Virtuoso.svc

, and click

Add

to create the ADO.Net Data Service.

Figure 8.190. Add New Item



4. In the newly created

   Virtuoso.svc.cs

   Data Service, add the data source class name of

   VirtuosoEntities

   (note this is the name set in the Creating EDM in Visual Studio 2008 section) as the

   DataService

   name. Enable the access to the Data Service by adding the entry

   config.SetEntitySetAccessRule("", EntitySetRights.All);*

   in the

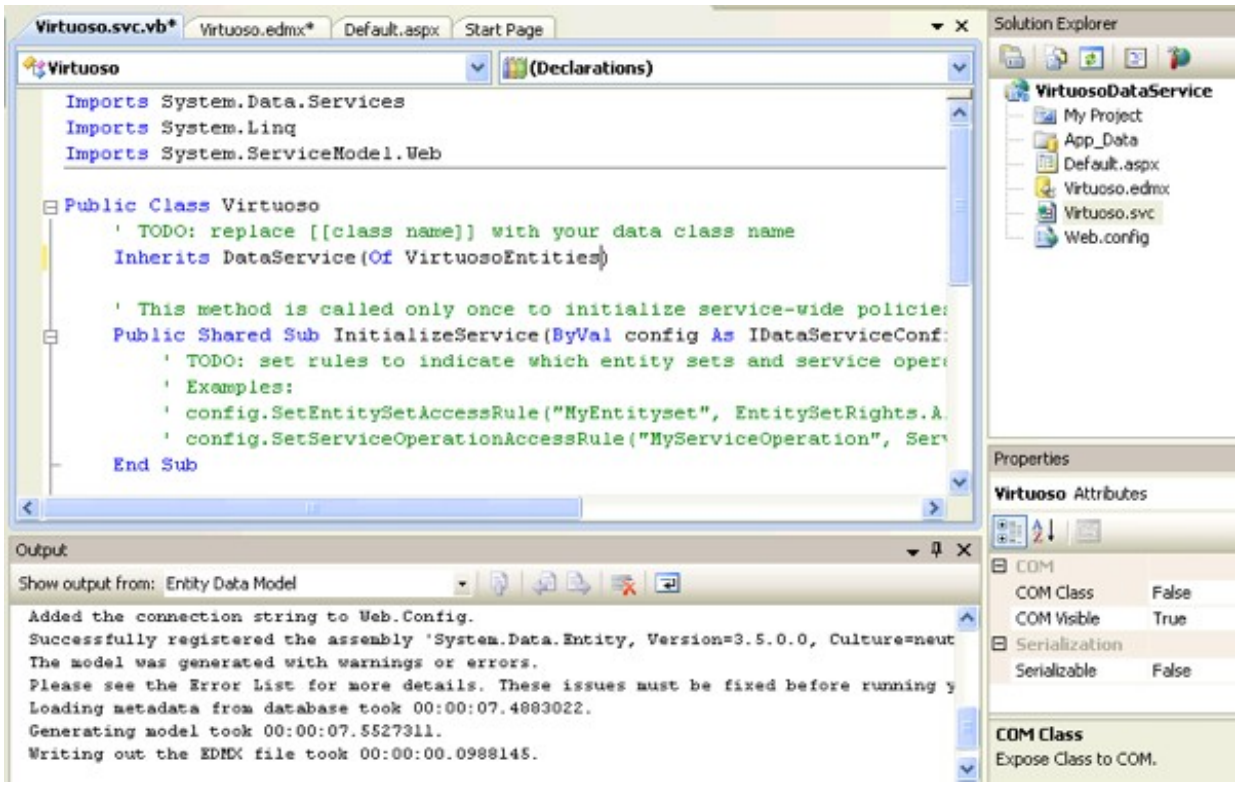
   InitializeService

   method.

   ```
   // C#
   ```

```
using System;
using System.Web;
using System.Collections.Generic;
using System.ServiceModel.Web;
using System.Linq;
using System.Data.Services;

namespace SimpleDataService
{
    public class Northwind : DataService<VirtuosoDemoEntities>
    {
        public static void InitializeService(IDataServiceConfiguration  config)
        {
            config.SetEntitySetAccessRule("*", EntitySetRights.All);
        }
    }
}
```

Figure 8.191. Virtuoso.svc.cs

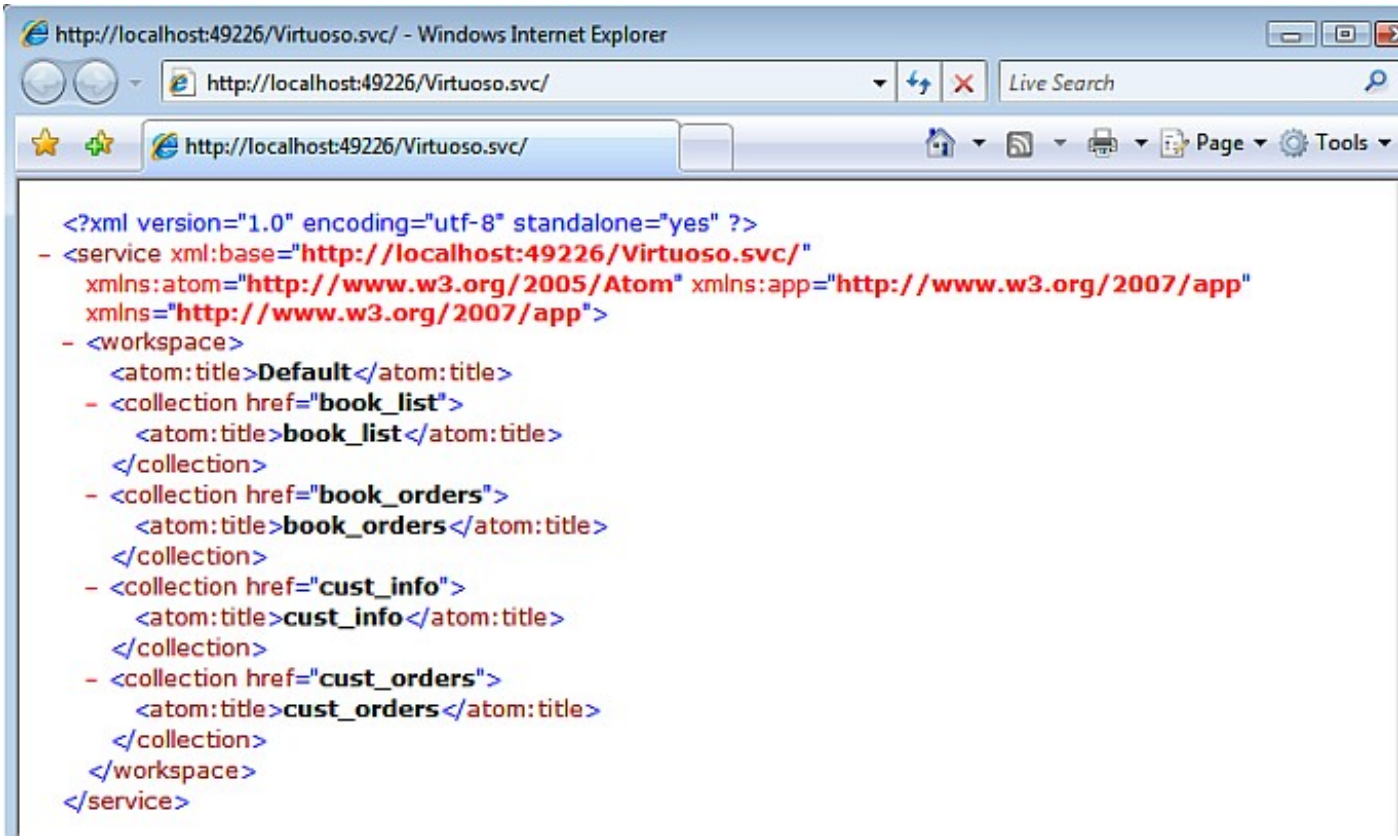


5. To test the Data Service, simply hit

   Ctrl+F5

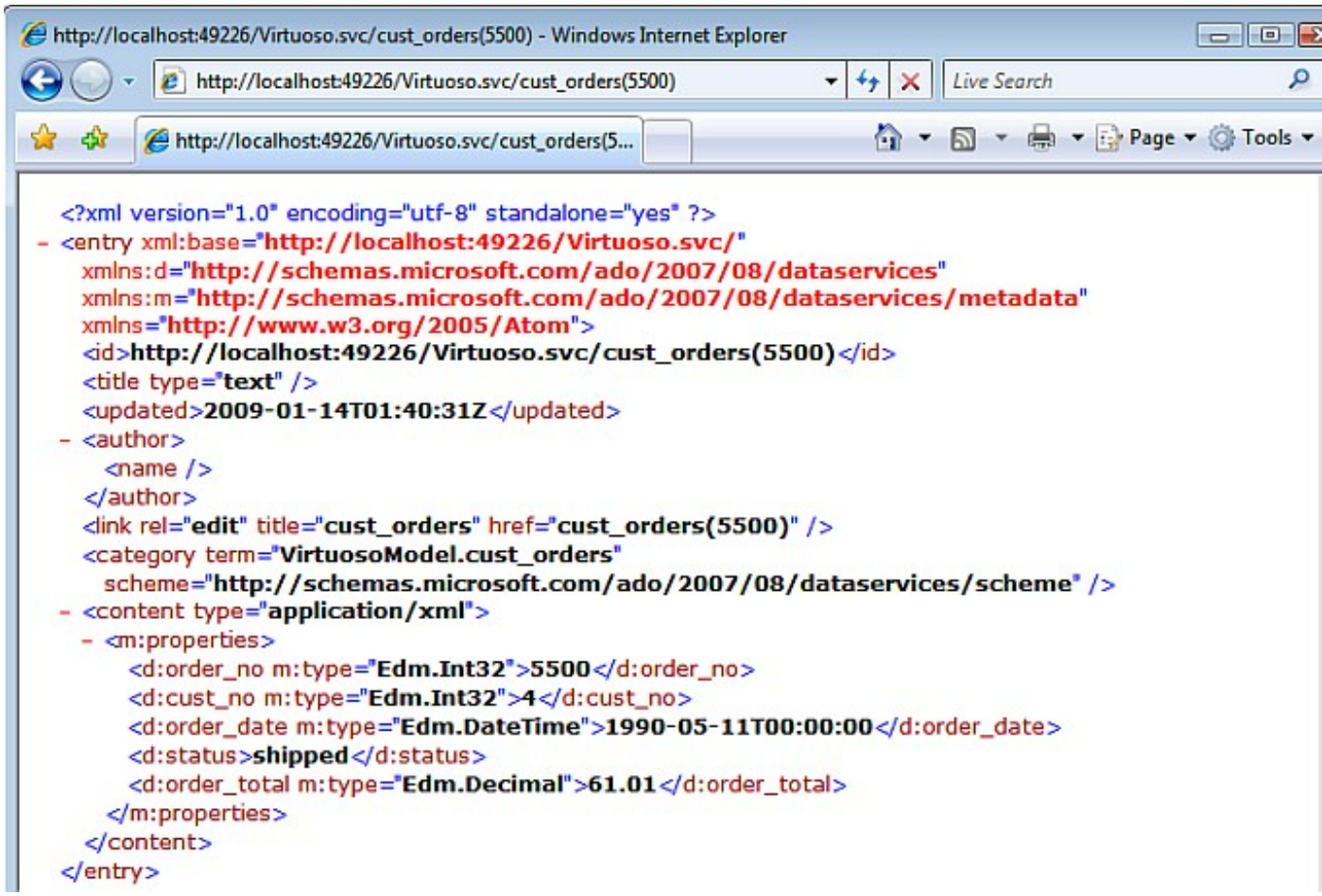
   within Visual Studio. This will start the development web server, run the Data Services server inside, and load a Web browser page displaying the list of available tables/entities for the Ingres Tutorial database catalog.

Figure 8.192. Data Service test



6. To access a specific entity instance like the cust_orders table order number 5500 record, use this convention: http://host/vdir/Virtuoso.svc/cust_orders(5500) .

Figure 8.193. Orders



Notes:

1. *Important*

   - To view

   Atom

   (the default format returned by an ADO.NET Data Service) in Internet Explorer, you must first ensure that

   Feed Reading View

   is turned

   off

   . This can be done on the

   Content tab

   of

   Tools in Internet Options

   .
2. If a Data Services entity instance URI page fails to load you can turn

   Verbose

   errors on by adding

   config.UseVerboseErrors = true;

   in the

   virtuoso.svc.cs InitializeService

   method to obtain more detailed information from the server as to why the page failed to load:

```
public static void InitializeService(IDataServiceConfiguration config)
{
config.UseVerboseErrors = true;
config.SetEntitySetAccessRule("*", EntitySetRights.All);
}
```

Visual Studio Windows DataGrid Form Application

This section details the steps required to create a simple Visual Studio 2008 Windows Form application, with associated DataGridView control for displaying data in selected tables from the target database.

1. Launch the Visual Studio 2008 SP1 IDE.

   Figure 8.194. Visual Studio 2008 SP1 IDE

2. Create a

   Web Application

   project by going to the

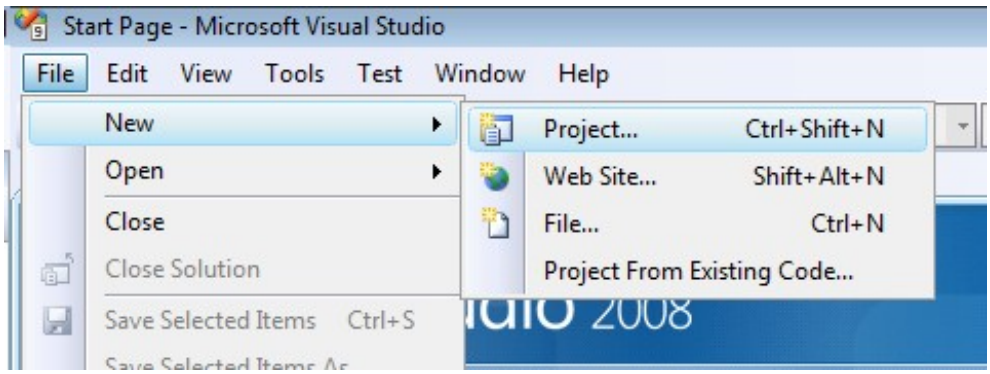
   File

   menu in Visual Studio and choosing

   New Project

   .

   Figure 8.195. Web Application



3. When the New Project window appears, choose either

   Visual Basic

   or

   Visual C#

   as your programming language.
4. Within the language category, click on

   Windows

   and select

   Windows Form Application

from the right-hand panel.

5. Choose a name for the project, for example

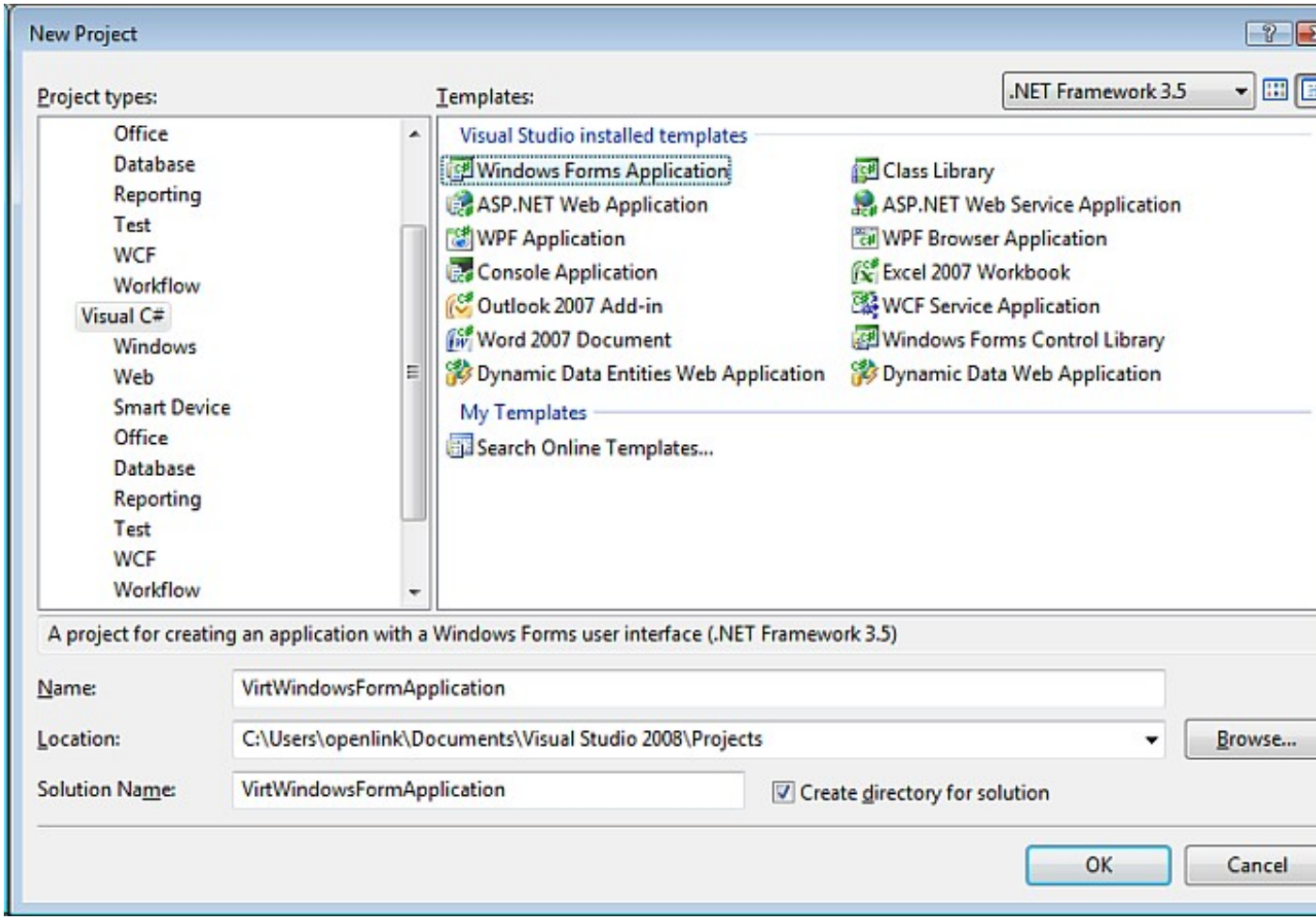
   VirtWindowsFormApplication

   , and click

   OK

   .

   Figure 8.196. Web Application



6. In the

   Toolbox

   , expand

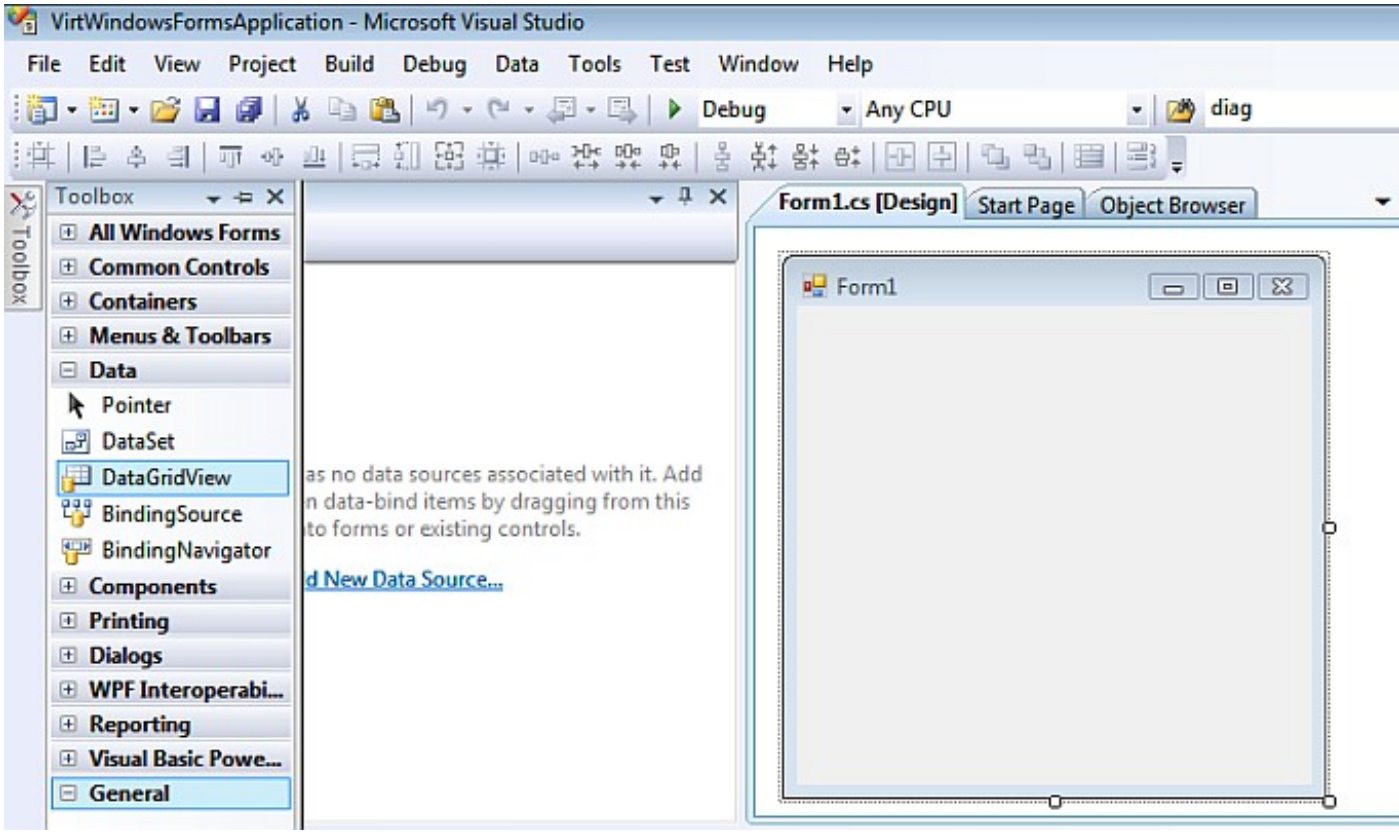
   Data Controls

   , and drag the

   DataGridView

   control onto the form.

   Figure 8.197. Toolbox



7. Click on the little

   arrow

   in the top right of the

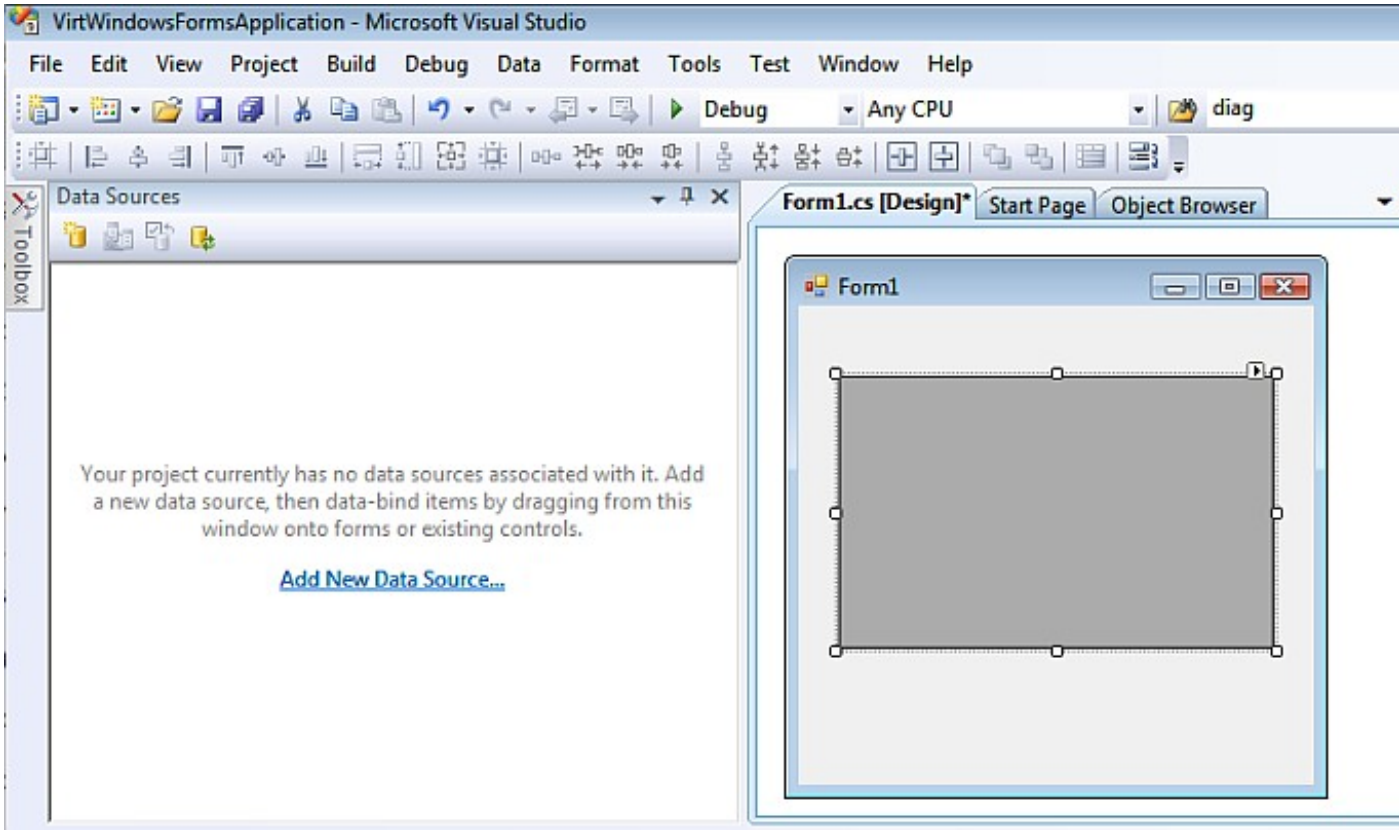
   DataGridView

   control. This loads the

   DataGridView Task

   menu.

Figure 8.198. DataGridView Task

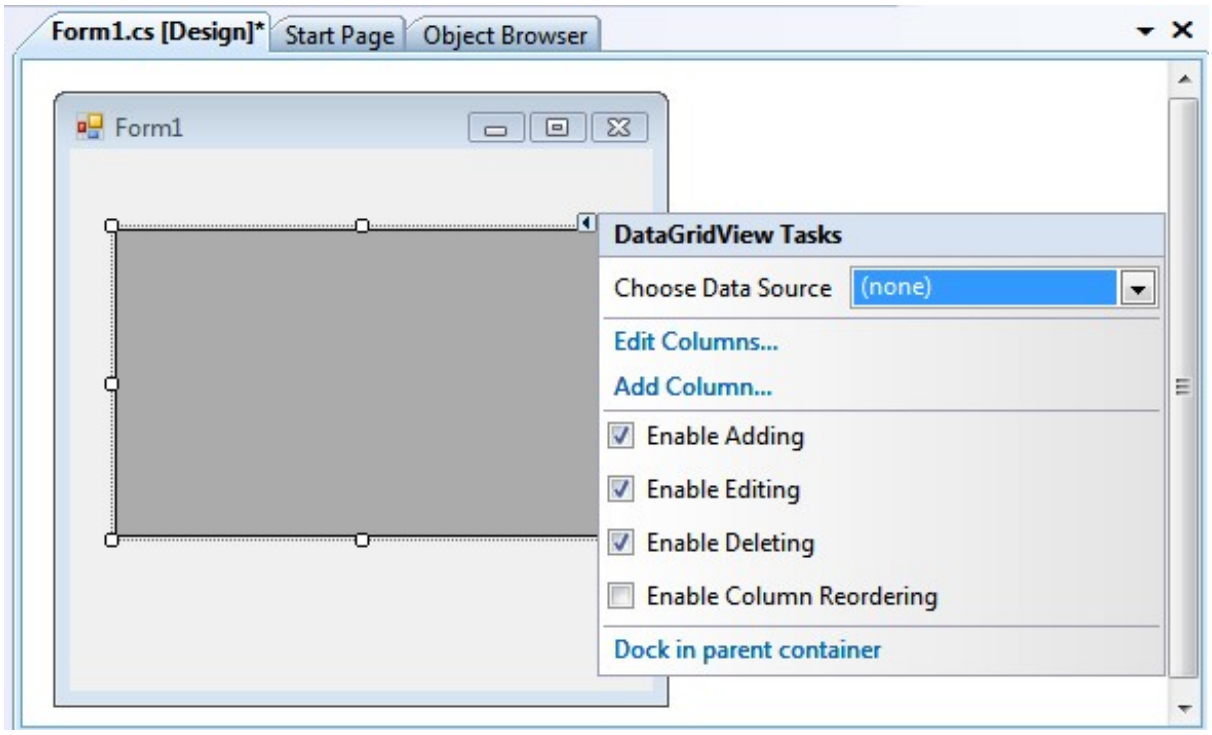


8. Click on the

   Choose Data Source

   list box.

   Figure 8.199. Choose Data Source

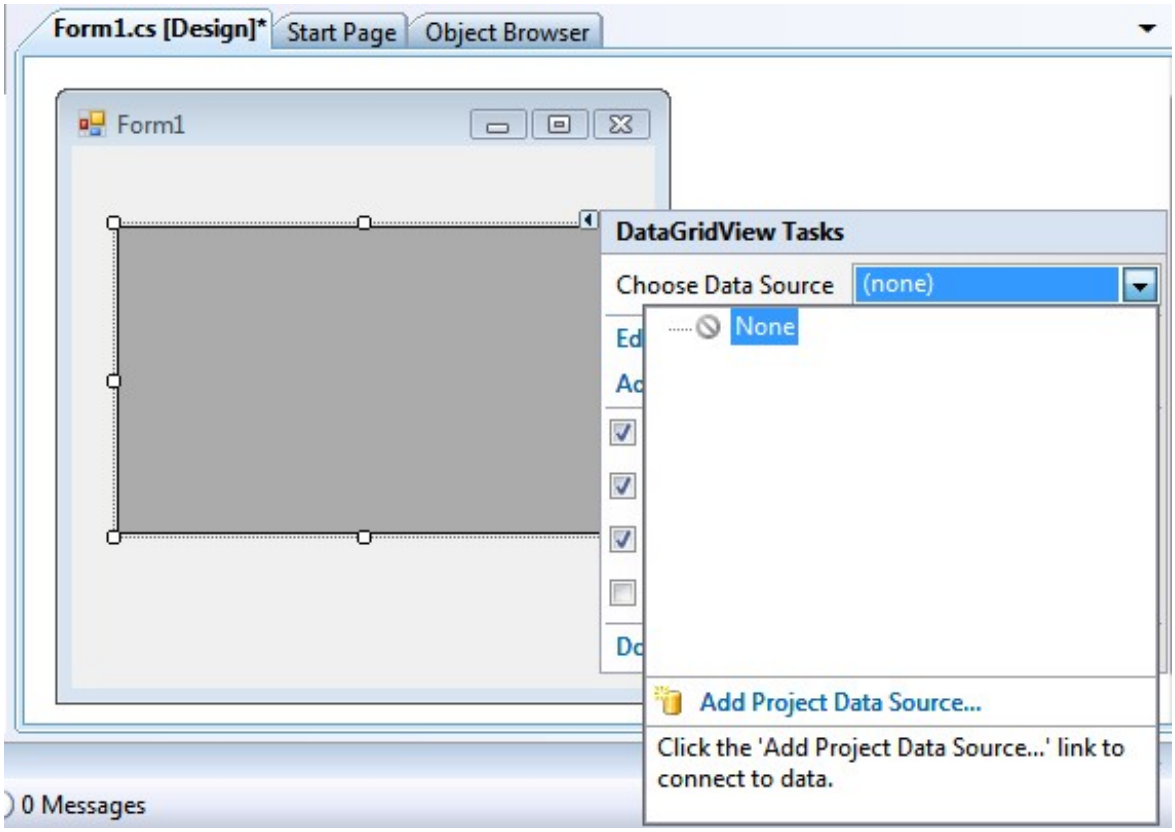


9. Click on the

   Add Project Data Source

link to connect to a data source.

Figure 8.200. Add Project Data Source



10. In the

    Data Source Configuration Wizard

    dialog

    Choose Data Source Type

    page select the

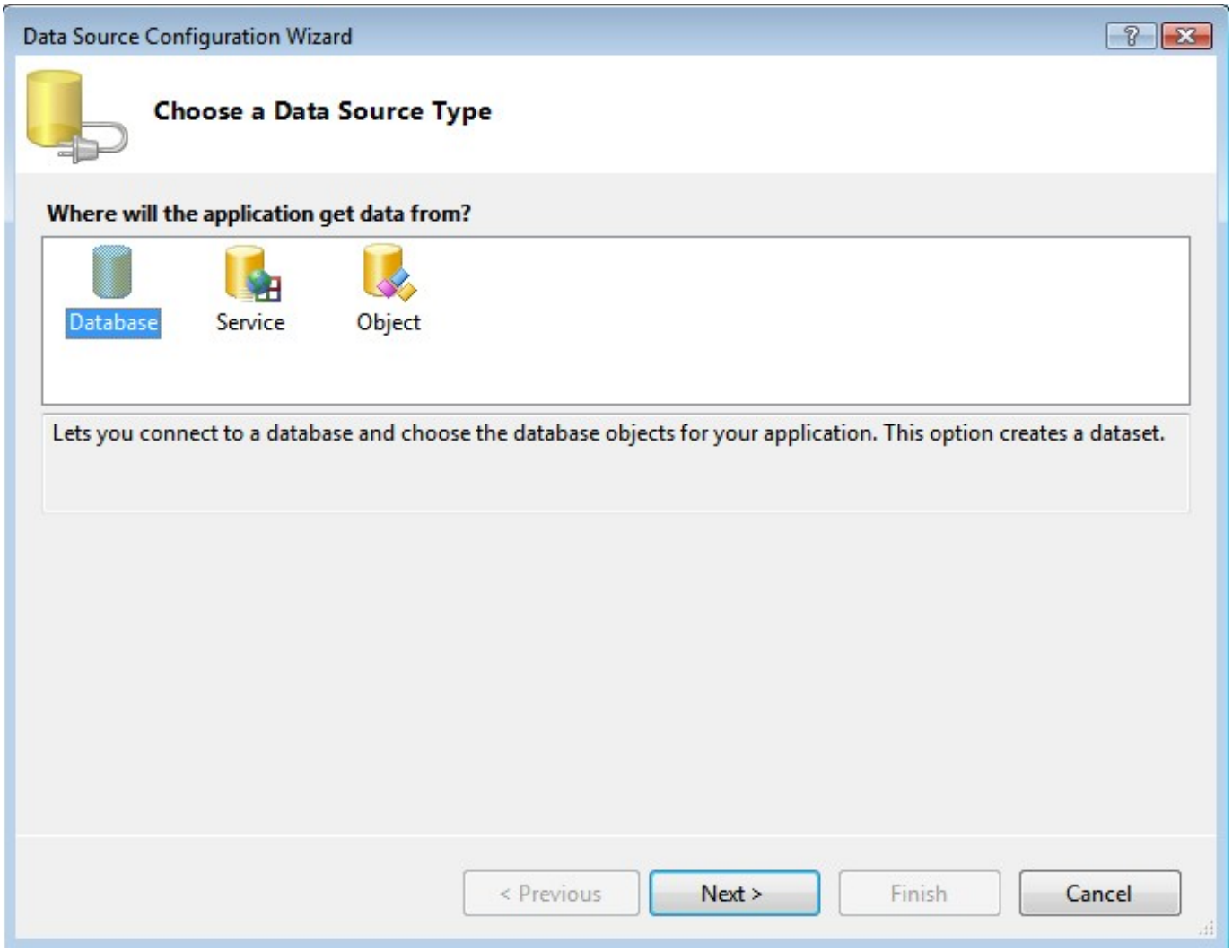
    Database

    data source type and click

    Next

    .

Figure 8.201. Data Source Type



11. In the

    Data Source Configuration Wizard

    dialog

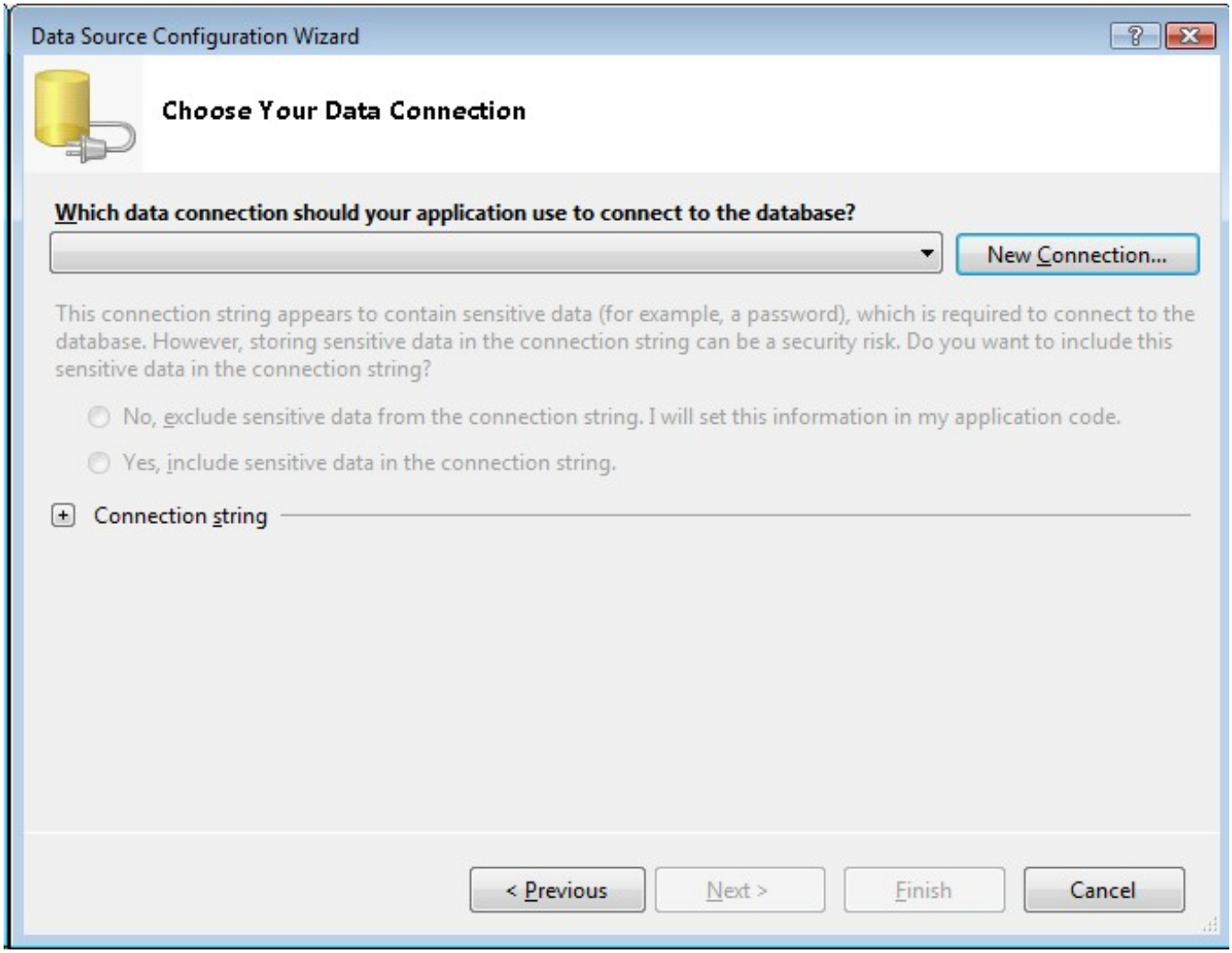
    Choose your Data Connection

    page, select the

    New Connection

    button

Figure 8.202. Data Source Configuration Wizard



12. In the

    Choose Data Source

    dialog, select the OpenLink

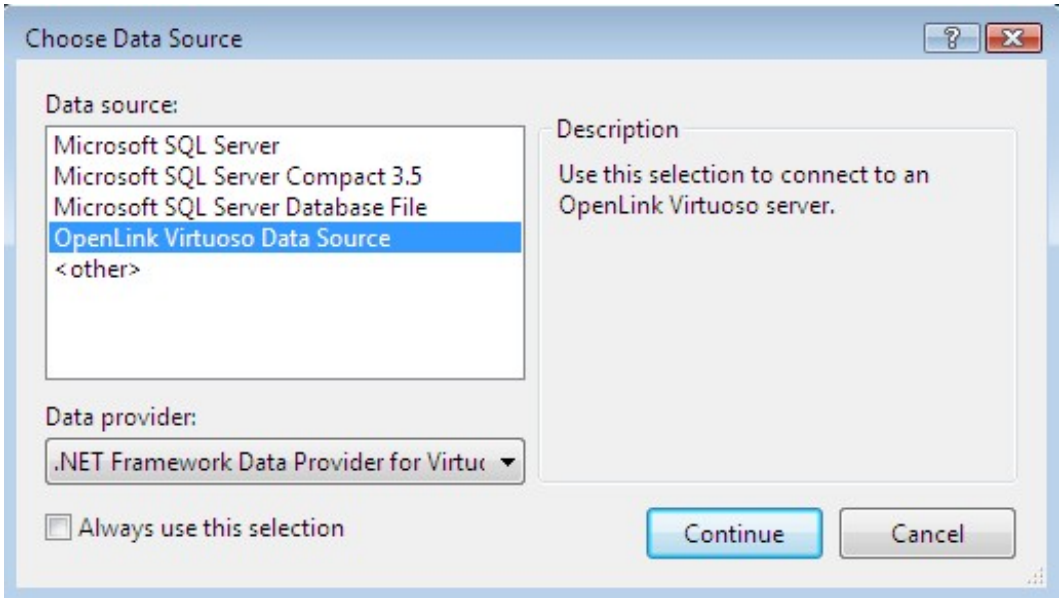
    Virtuoso Data Source

    from the list and click

    Continue

    .

Figure 8.203. Data Source



13. In the

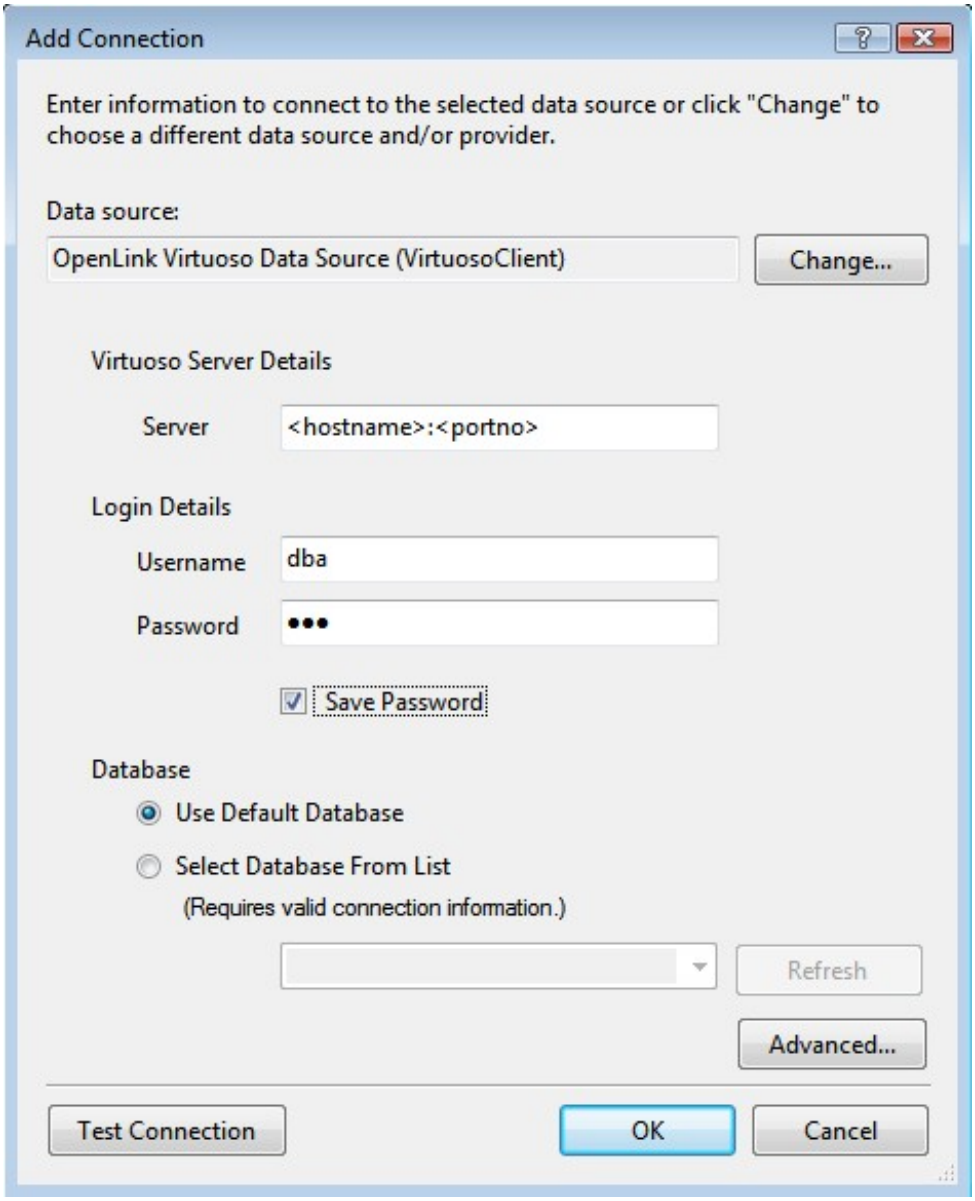
    Add Connection

    dialog, specify the

    hostname, portno, username and password

    for the target Virtuoso Server and check the Save Password check box.

    Figure 8.204. Connection Properties



14. Select the

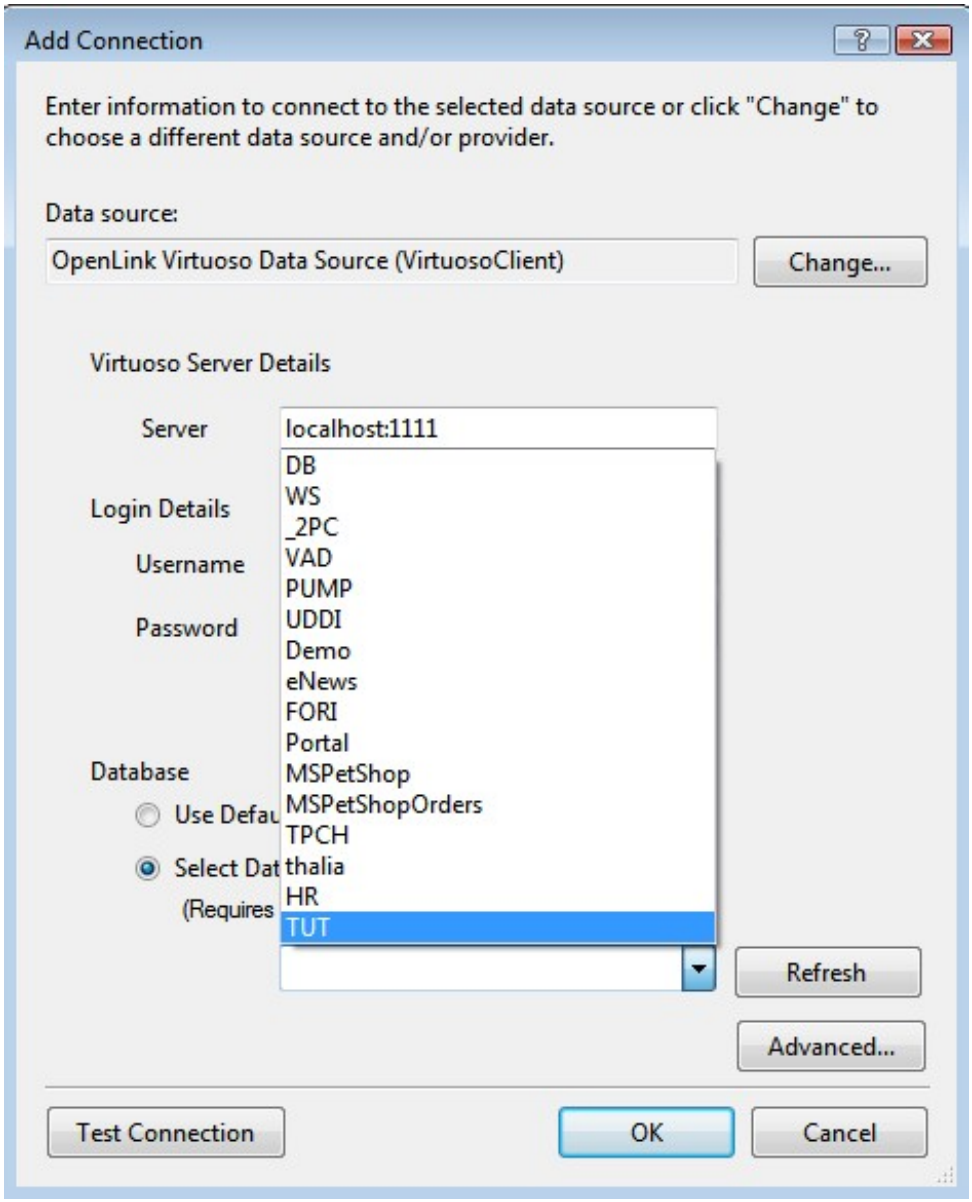
    Select Database From List

    radio button and choose the

    TUT

    database from the drop down list.

Figure 8.205. Add connection

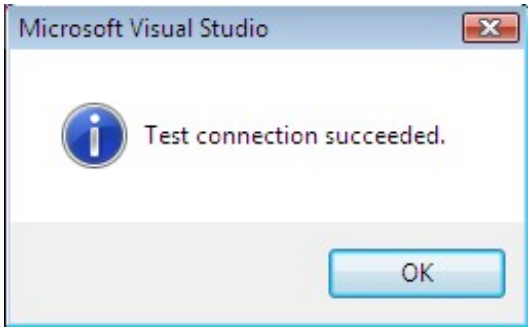


15. Press the

    Test Connection

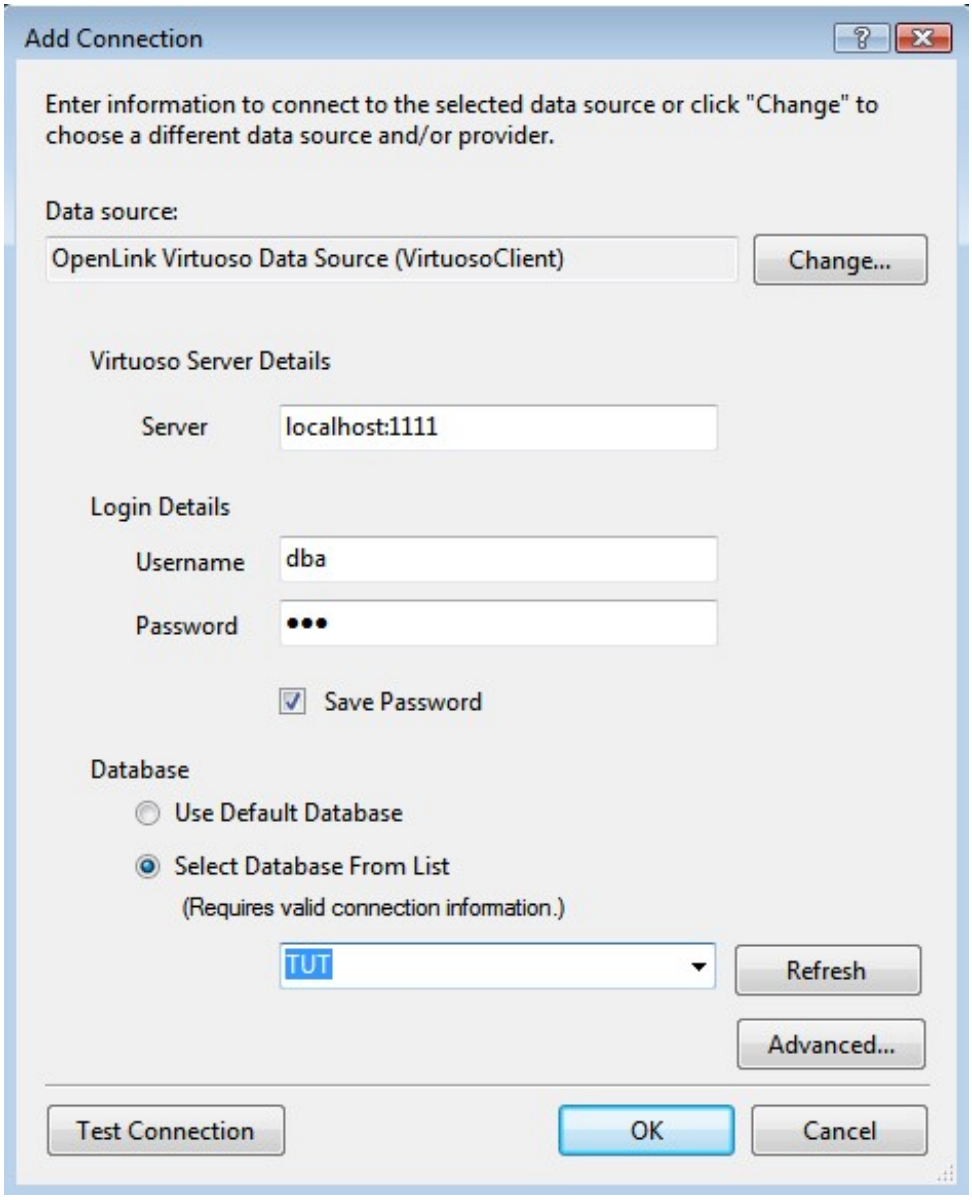
    dialog to verify that the database is accessible.

Figure 8.206. Test Connection



16. Click OK to add the connection.

Figure 8.207. Test Connection



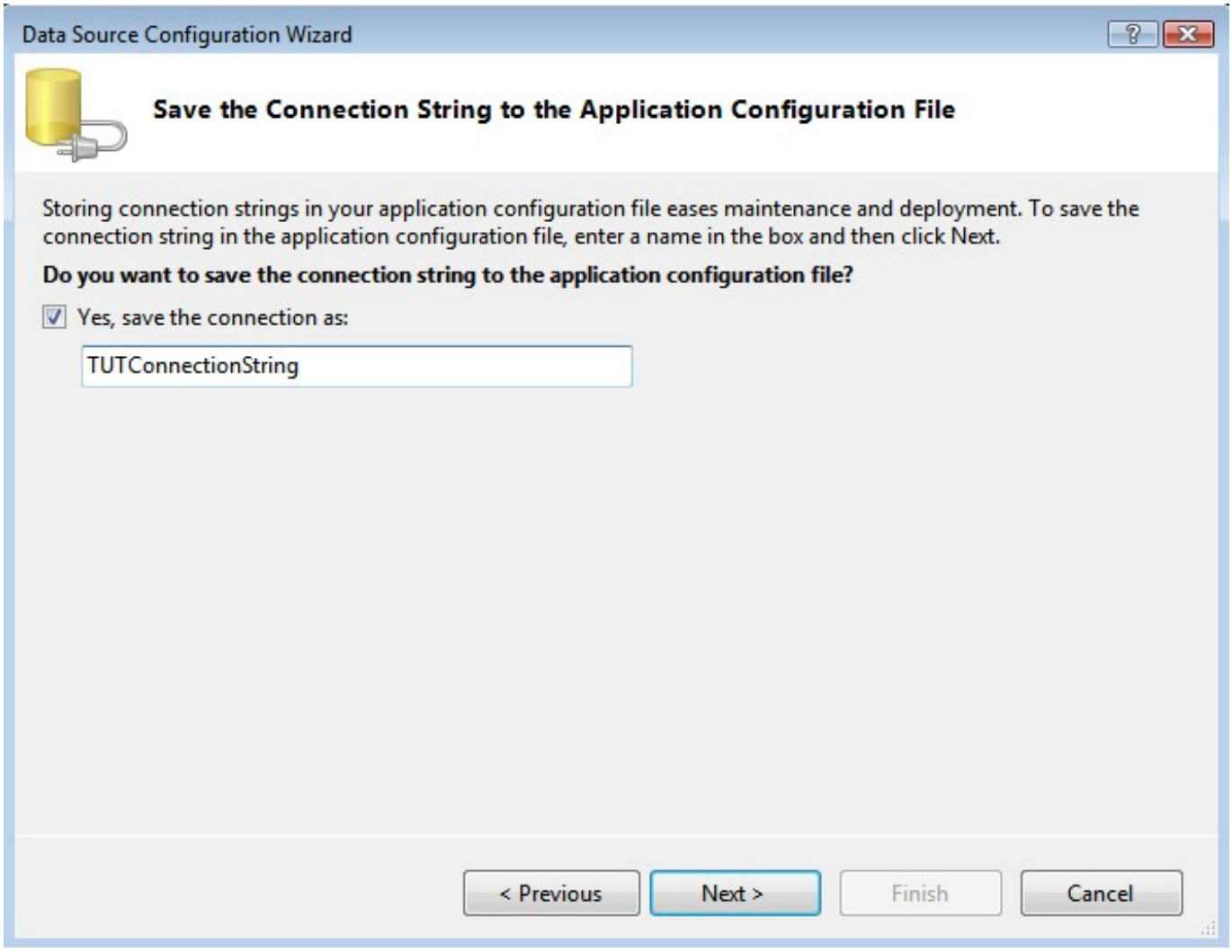
17. Leave the name of the connect string to the default of

    TUTConnectionString

    and click

    Next

    Figure 8.208. TUTConnectionString



18. From the list of available tables returned for the TUT database, select the

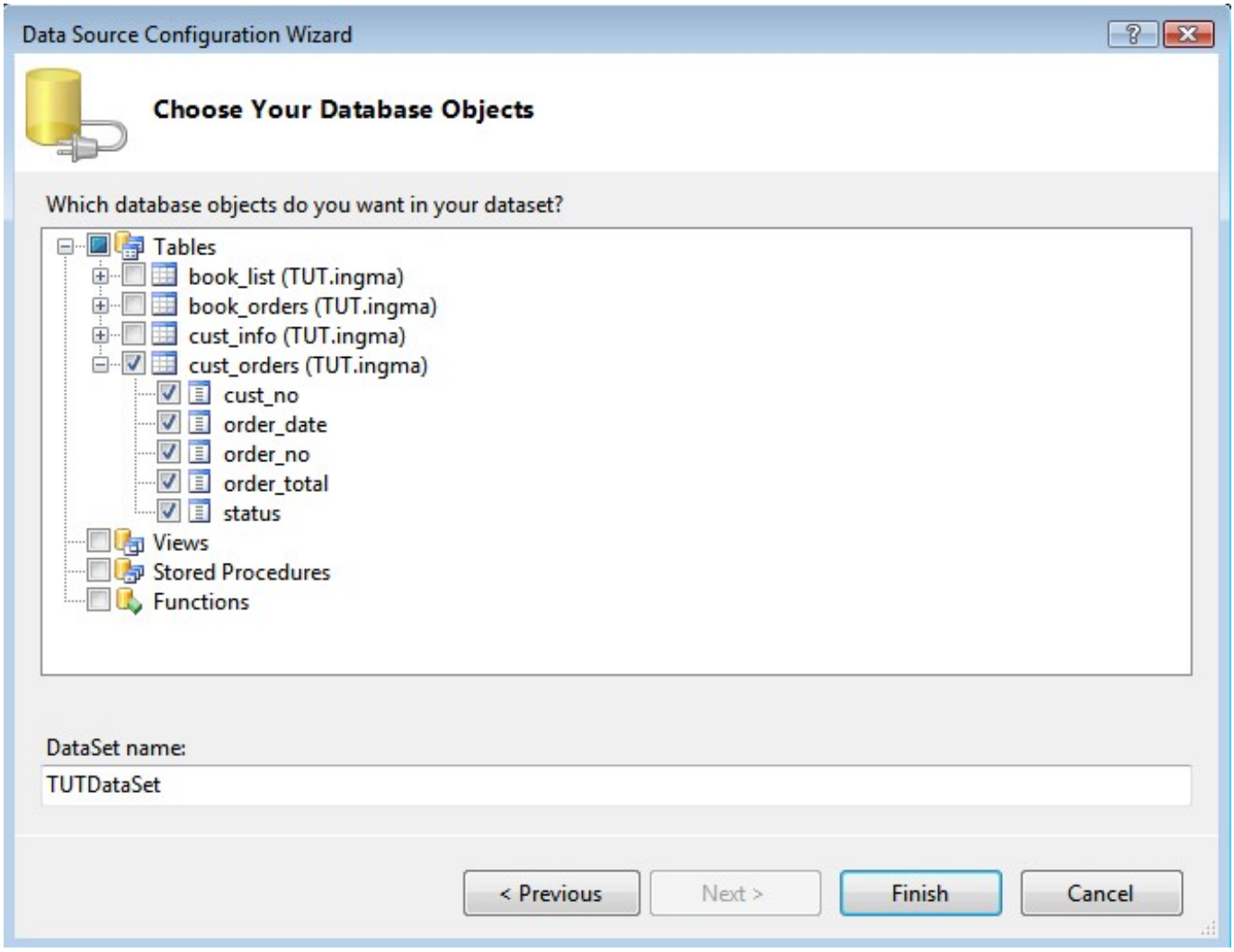
    cust_orders

    table to be associated with the

    DataGridView

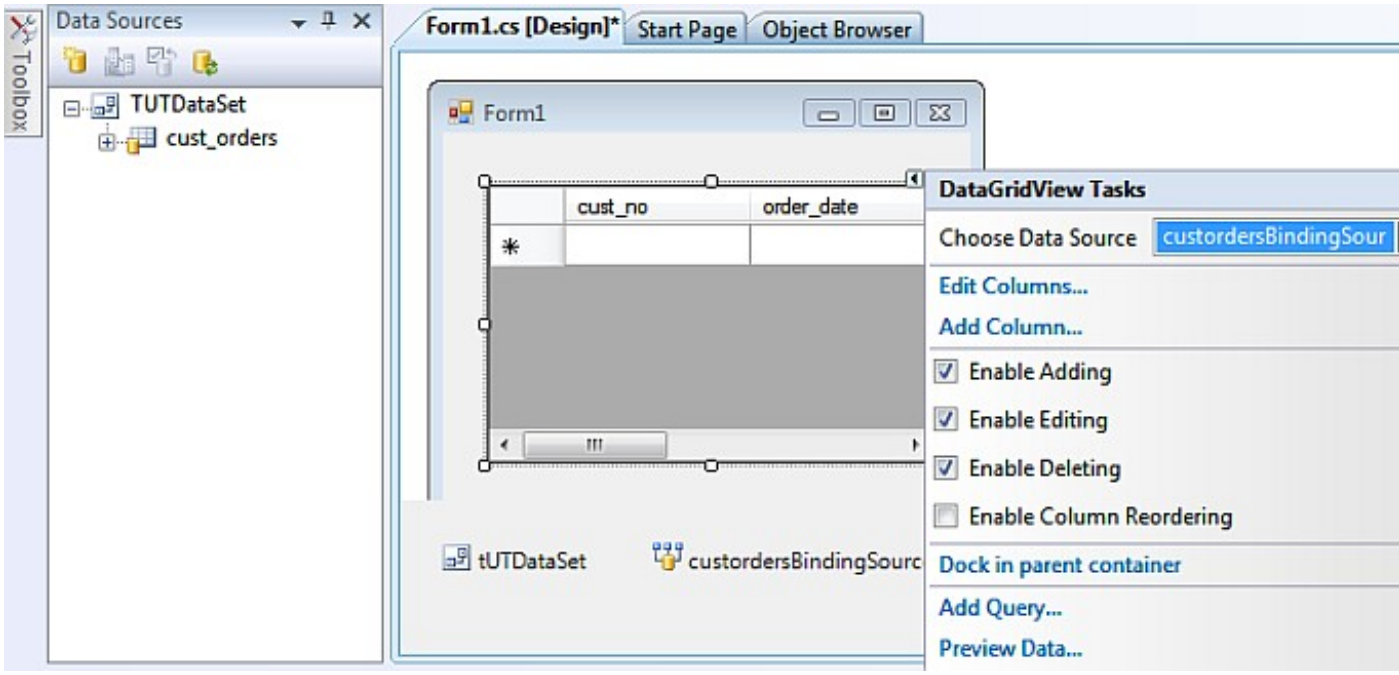
    control.

    Figure 8.209. TUT database



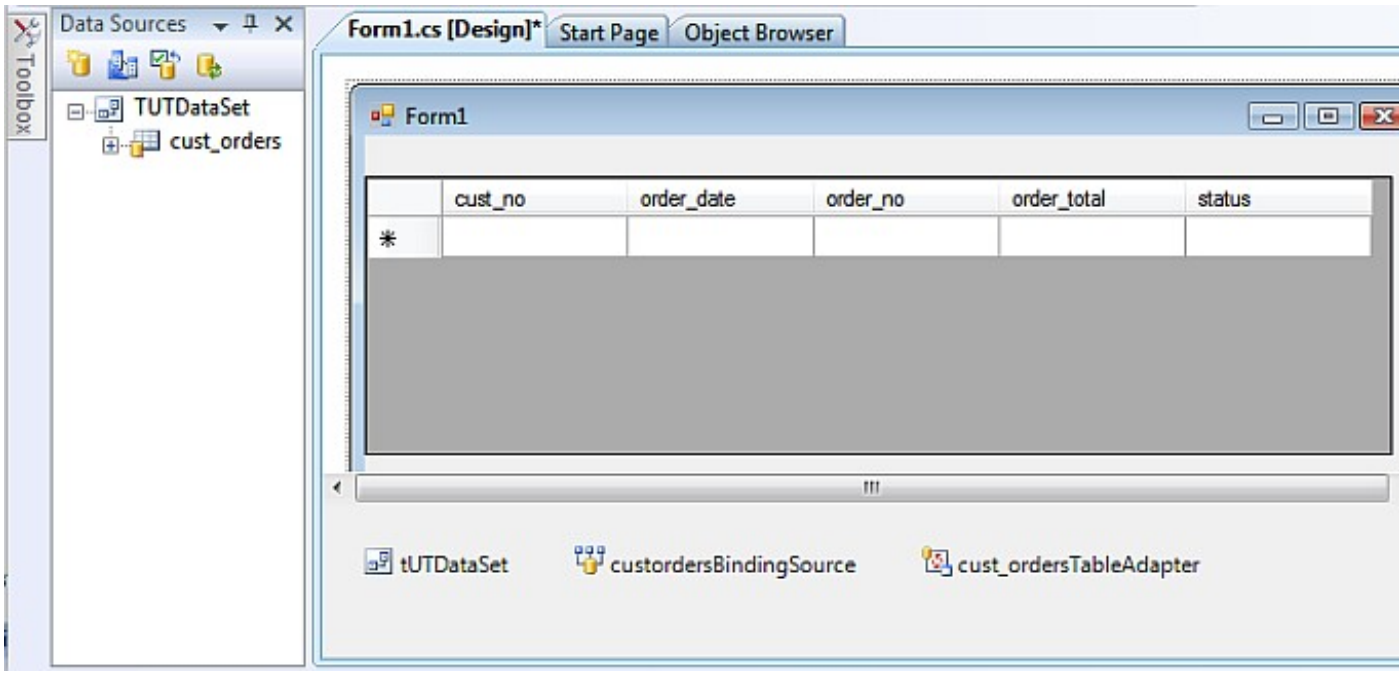
19. The columns names of the select table will be displayed in the DataGridView.

Figure 8.210. DataGridView



20. Resize the Form and DataGridView to allow all columns to be visible, if possible.

Figure 8.211. Resize the Form and DataGridView



21. To test the application, simply hit

    Ctrl+F5

    within Visual Studio or select

    Start Debugging

    from the

    Debug

    menu.

    Figure 8.212. Start Debugging

22. The data from the

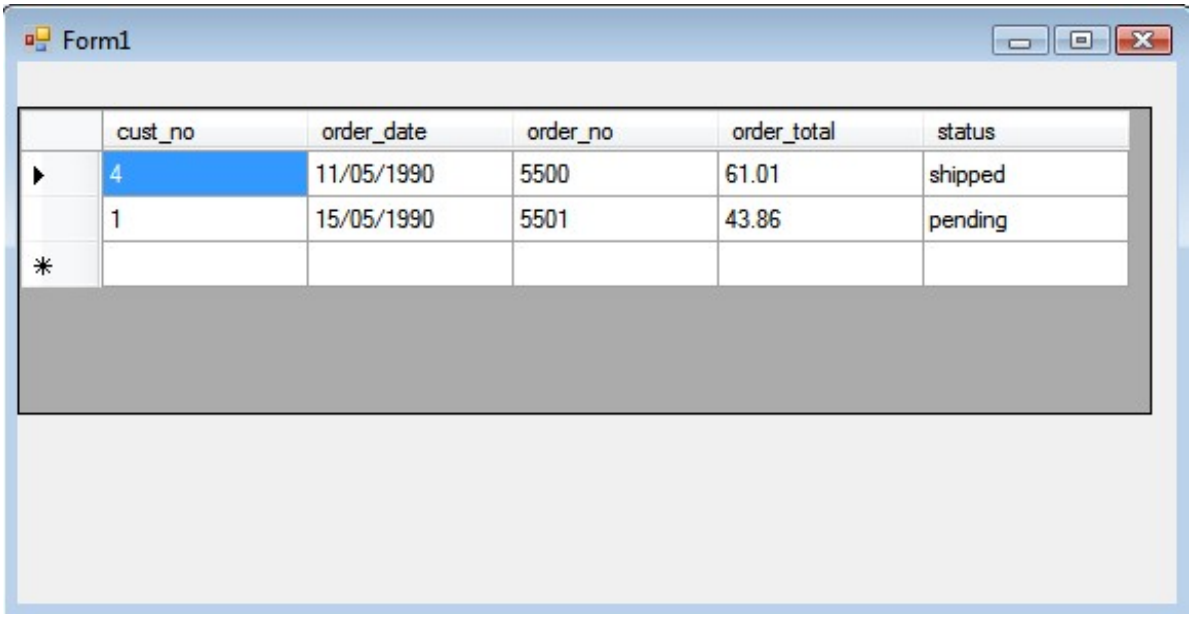
    cust_orders

    table will be displayed in the

    DataGrid

    .

    Figure 8.213. DataGrid



The task is now complete.

8.5. Using Microsoft Entity Frameworks to Access Informix Schema Objects with Virtuoso

Abstract

This section details the steps required to provide Microsoft Entity Framework access to Informix Schema Objects using the OpenLink Virtuoso Universal Server. This is achieved by linking the required Informix Schema objects into Virtuoso using its built in Virtual Database engine, and then using the Virtuoso ADO.NET Entity Framework provider to query the remote Informix Schema objects linked into the Virtuoso Server.

Prerequisites. The following prerequisites must be in place for this solution to be possible.

Informix DBMS. An Informix DBMS hosting the required Schema Objects needs to be available. In this section the *Informix stores_demo* sample database will be used to demonstrate the process.

An Informix DBMS hosting the required Schema Objects needs to be available. In this section the *Informix stores_demo* sample database will be used to demonstrate the process.

ODBC Driver for Informix. An Informix ODBC Driver is required for Linking the Informix Schema Objects into the Virtuoso Server. The OpenLink *ODBC Driver for Informix* will be used in this section, for which a functional ODBC Data source name of "inf10ma" will be assumed to exist on the machine hosting the Virtuoso Server.

An Informix ODBC Driver is required for Linking the Informix Schema Objects into the Virtuoso Server. The OpenLink *ODBC Driver for Informix* will be used in this section, for which a functional ODBC Data source name of "inf10ma" will be assumed to exist on the machine hosting the Virtuoso Server.

Virtuoso Universal Server. An Virtuoso installation including the Virtuoso Universal Server and ADO.NET Entity Framework Provider is required. The Virtuoso components used must be Release 5.10.x or above, this being the minimum version containing support for Microsoft Entity Frameworks

An Virtuoso installation including the Virtuoso Universal Server and ADO.NET Entity Framework Provider is required. The Virtuoso components used must be Release 5.10.x or above, this being the minimum version containing support for Microsoft Entity Frameworks

Microsoft Visual Studio 2008 SP1. Microsoft Visual Studio 2008 Service Pack 1 is required, this being the only version containing the necessary Entity Framework support available at the time of writing.

Microsoft Visual Studio 2008 Service Pack 1 is required, this being the only version containing the necessary Entity Framework support available at the time of writing.

Tasks

Ensure Informix Primary Keys (PKs) are not nullable. The Visual Studio 2008 Entity Data Model (EDM) requires that all primary keys are *NOT* Nullable, and will fail to generate an EDM if any are. Thus ensure any tables to be used are defined as not nullable in the Informix database schema before attempting to generate an EDM. In the case of the Informix stores_demo database many tables have primary keys that are nullable so these will need to be redeclared.

The Visual Studio 2008 Entity Data Model (EDM) requires that all primary keys are *NOT* Nullable, and will fail to generate an EDM if any are. Thus ensure any tables to be used are defined as not nullable in the Informix database schema before attempting to generate an EDM. In the case of the Informix stores_demo database many tables have primary keys that are nullable so these will need to be redeclared.

This is possible using dbaccess however care needs to be taken, particularly with the following

- Amending a PK field to be not nullable can results in the primary key constraint being removed.
- Amending a PK field to be not nullable when the primary key is referenced by a foreign key constraint results in the foreign key constraint being removed.

In both of the cases above the primary key and foreign key constraints that are removed will have to be manually recreated.

8.5.1. Install and configure OpenLink ODBC Driver for Informix

The Virtuoso Virtual Database engine uses ODBC as the connectivity mechanism for linking remote database objects into its local schema. Thus an Informix ODBC Driver must be available with a suitably configured DSN for connecting to the target database. The OpenLink ODBC Drivers for Informix have been used in this section, although in theory any Informix ODBC Driver can be used.

Installation instructions for the OpenLink ODBC Driver for Informix are available from:

- Product Installation & Basic Configuration (ODBC)

8.5.2. Install and configure OpenLink Virtuoso Universal Server

- Install and configure OpenLink Virtuoso Universal Server

8.5.3. Linking Informix tables into OpenLink Virtuoso

1. Start the Virtuoso Web User Interface

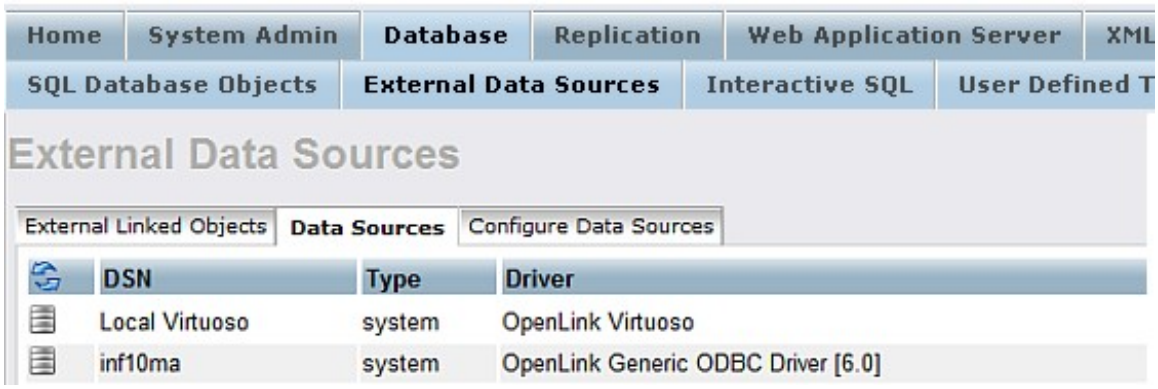
   Figure 8.214. Start

2. Select the "Conductor" link in the Left Frame and Login as the "dba" user.

Figure 8.215. Conductor

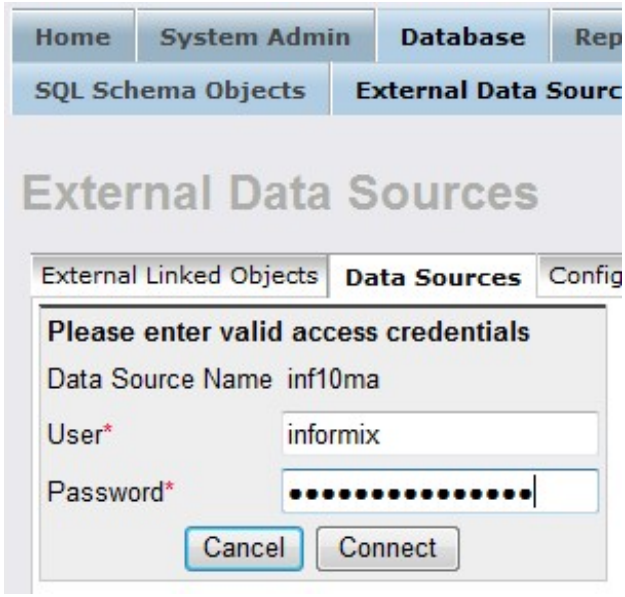
3. Select the "Databases" -> "External Data Source" -> "Data Sources" tab menu items

Figure 8.216. Databases



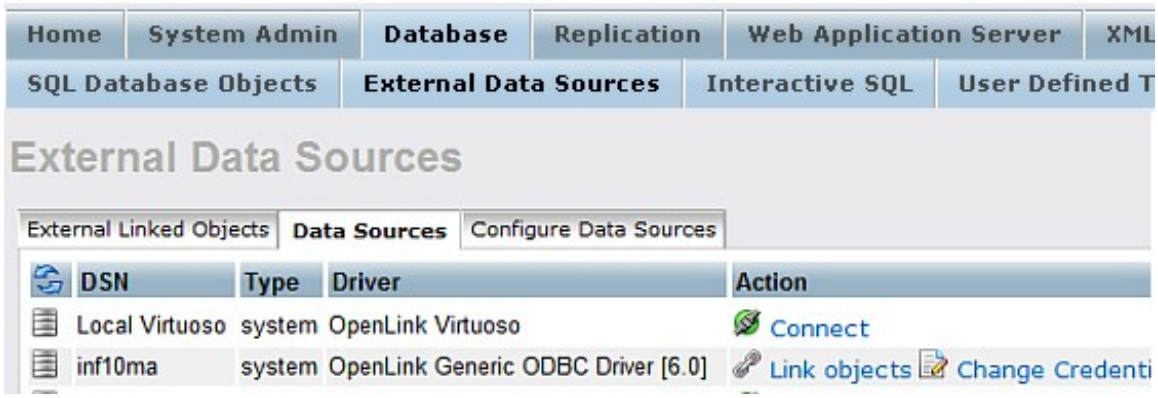
4. Select the "Connect" button for the "inf10ma" Informix DSN.

Figure 8.217. Connect



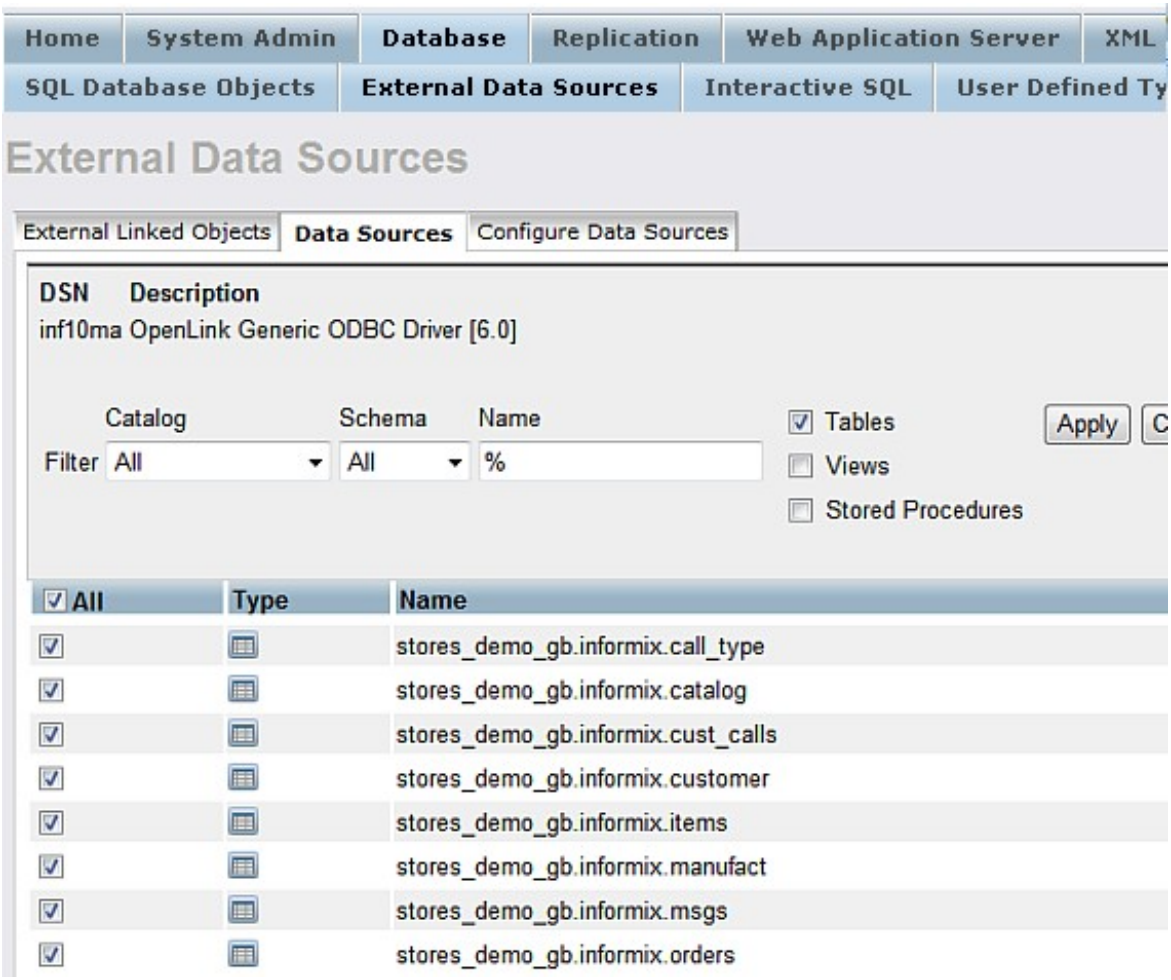
5. On successful connection Select the "Link Objects" button to obtain a list of available tables

Figure 8.218. Link Objects



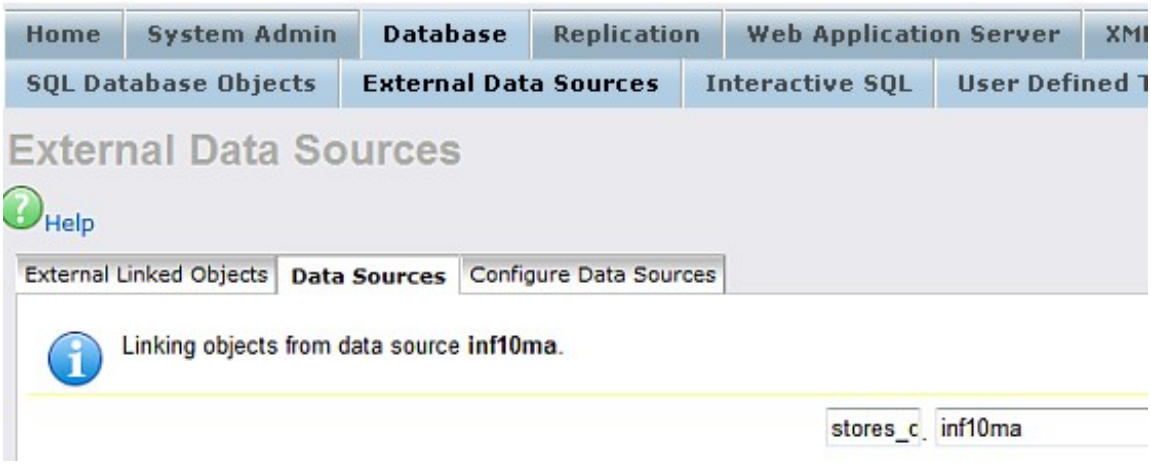
6. Select all the tables that are part of the "stores_demo" catalog.

Figure 8.219. Select all tables



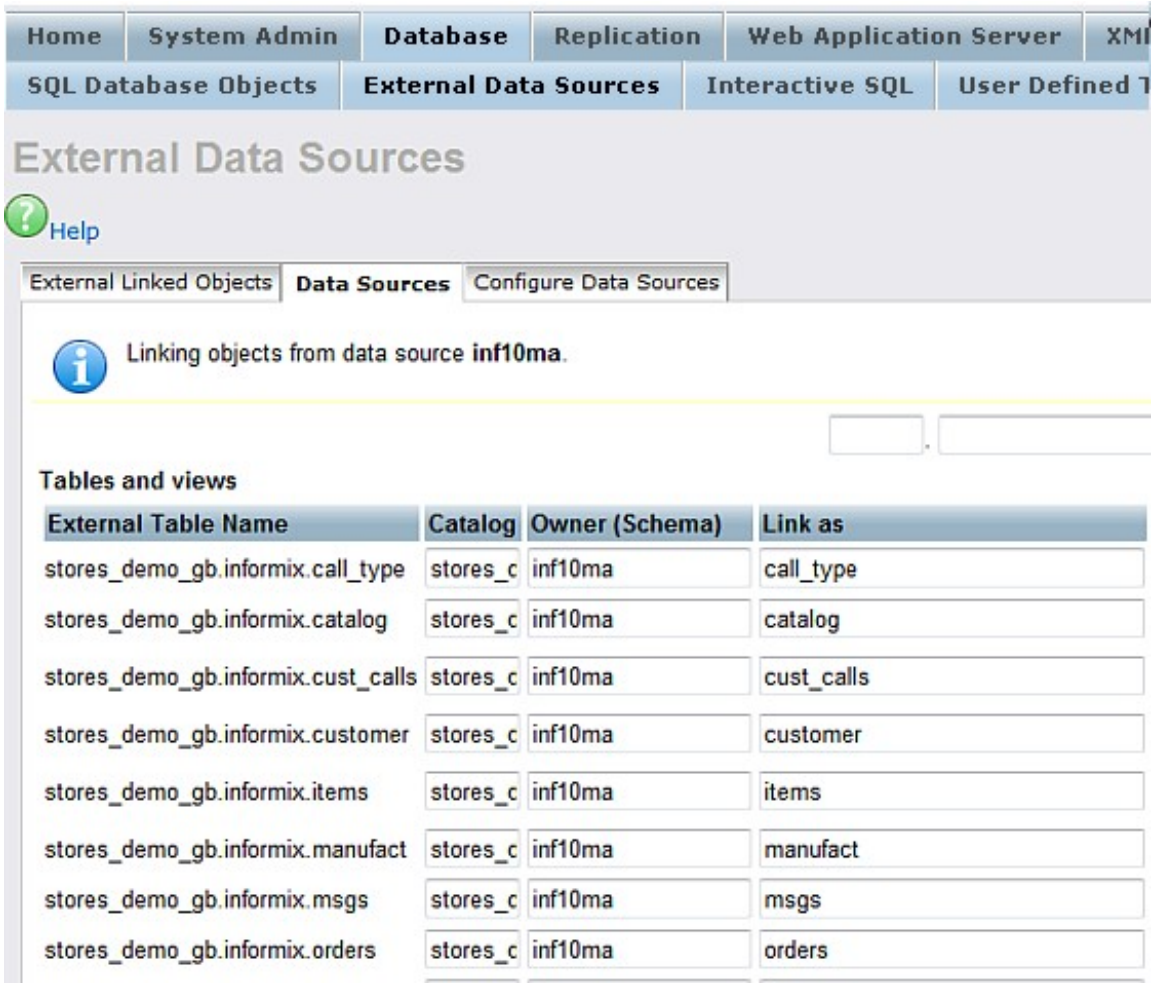
7. Change the Catalog for each table to be "stores_demo" using the "Set All" button.

Figure 8.220. Catalog



8. All the catalog names are changed to be "stores_demo".

Figure 8.221. catalog names

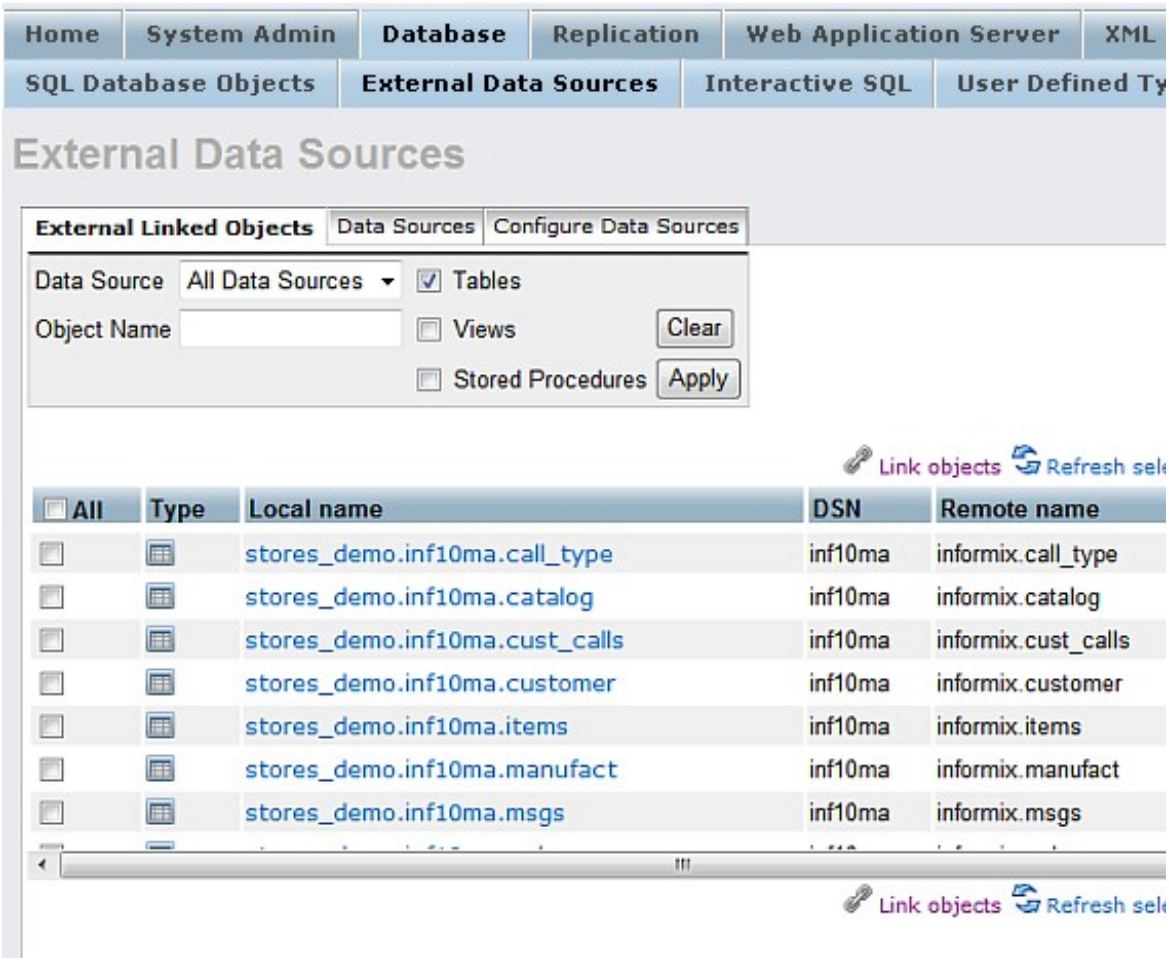


9. Select the "Link" button to link the selected tables into Virtuoso

Figure 8.222. "Link" button

10. On completion of the Link process the tables will be displayed in the "External Linked Objects" tab.

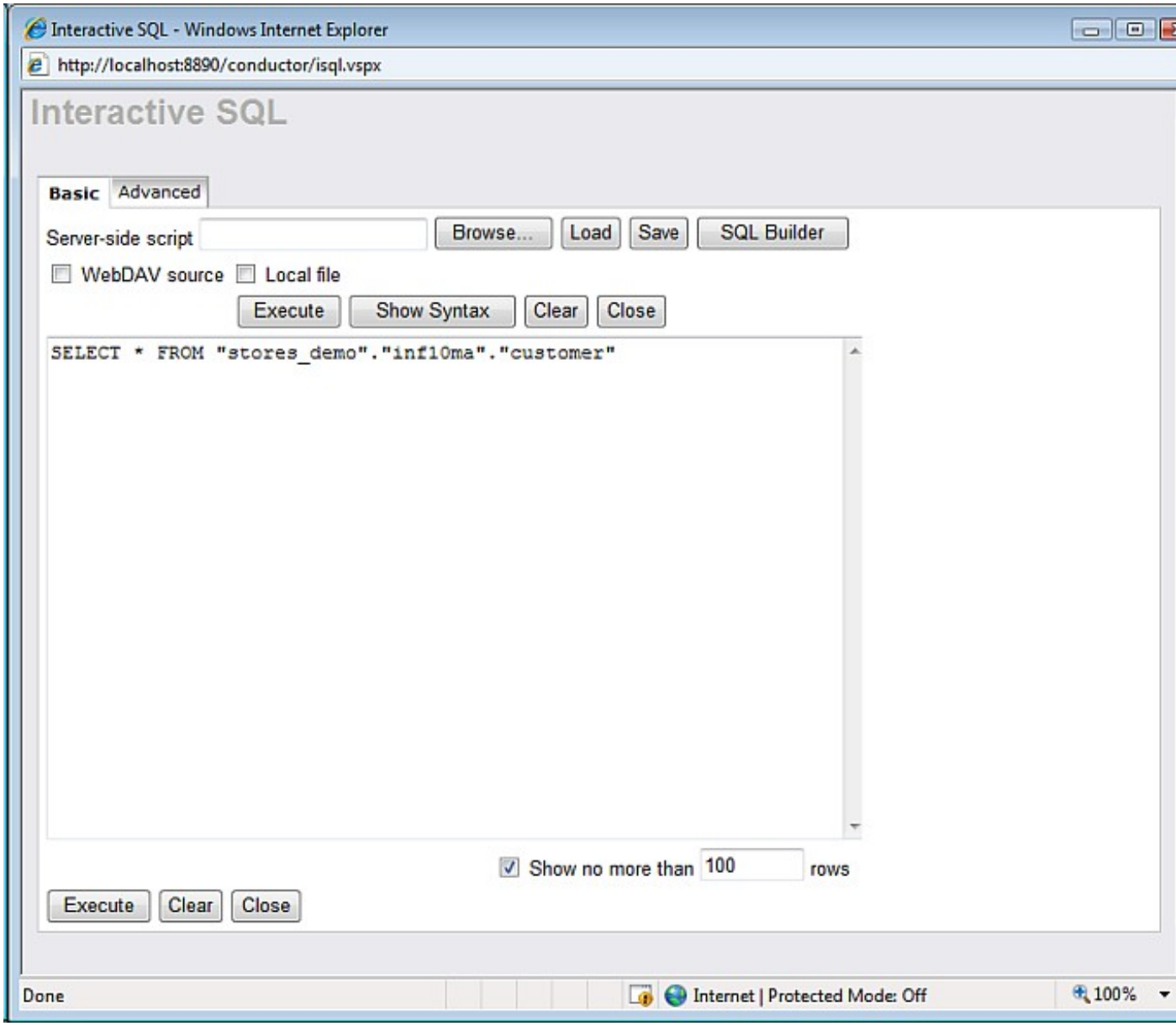
Figure 8.223. Completion



11. At this point you can test the remotely linked tables by clicking on the link that accompanies each table. e.g. stores_demo.inf10ma.customer.

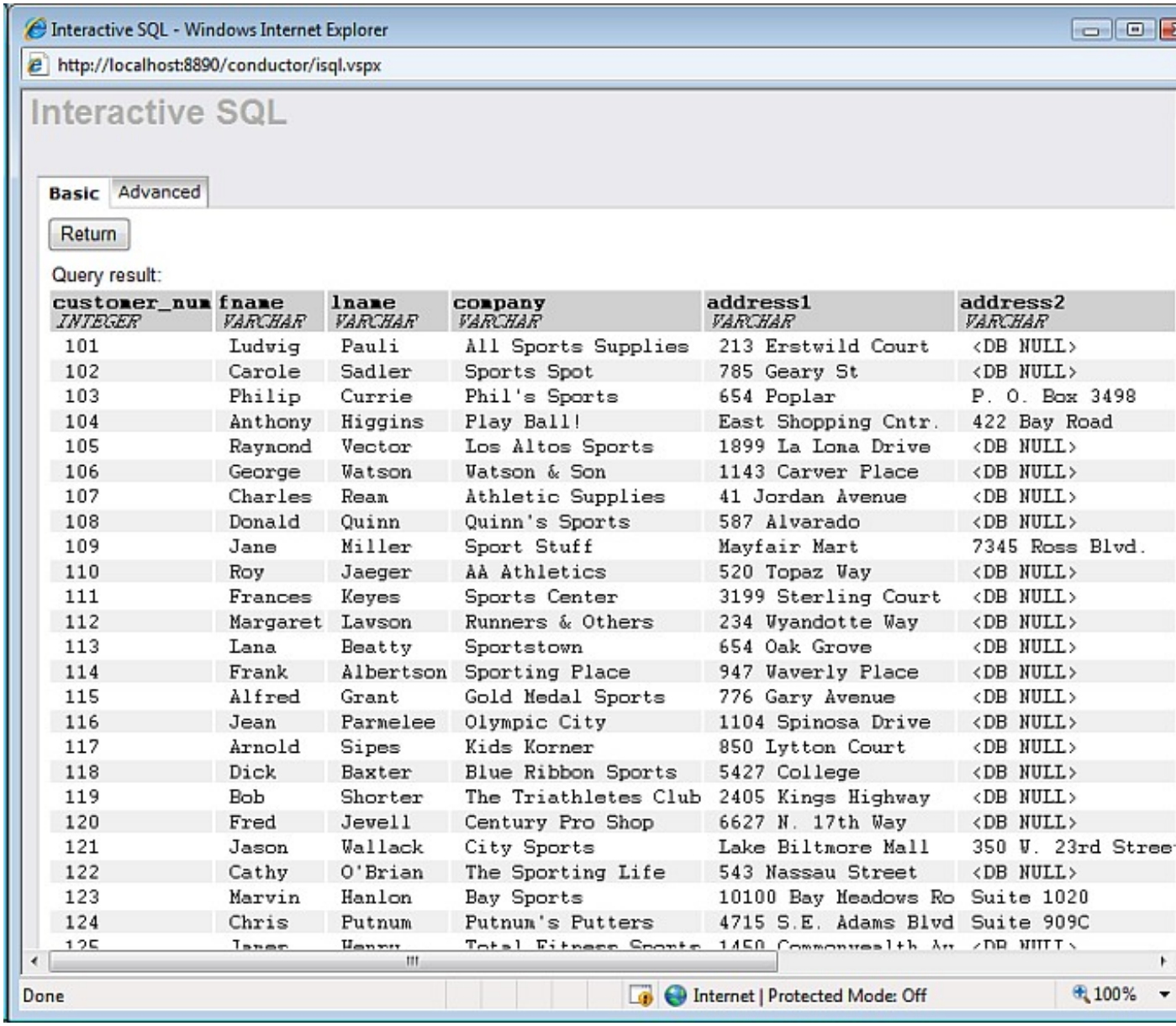
    This will display the Interactive ISQL interface which will have been already populated with a suitable SQL statement.

Figure 8.224. Querying



12. Select Execute to see data from the remotely linked table.

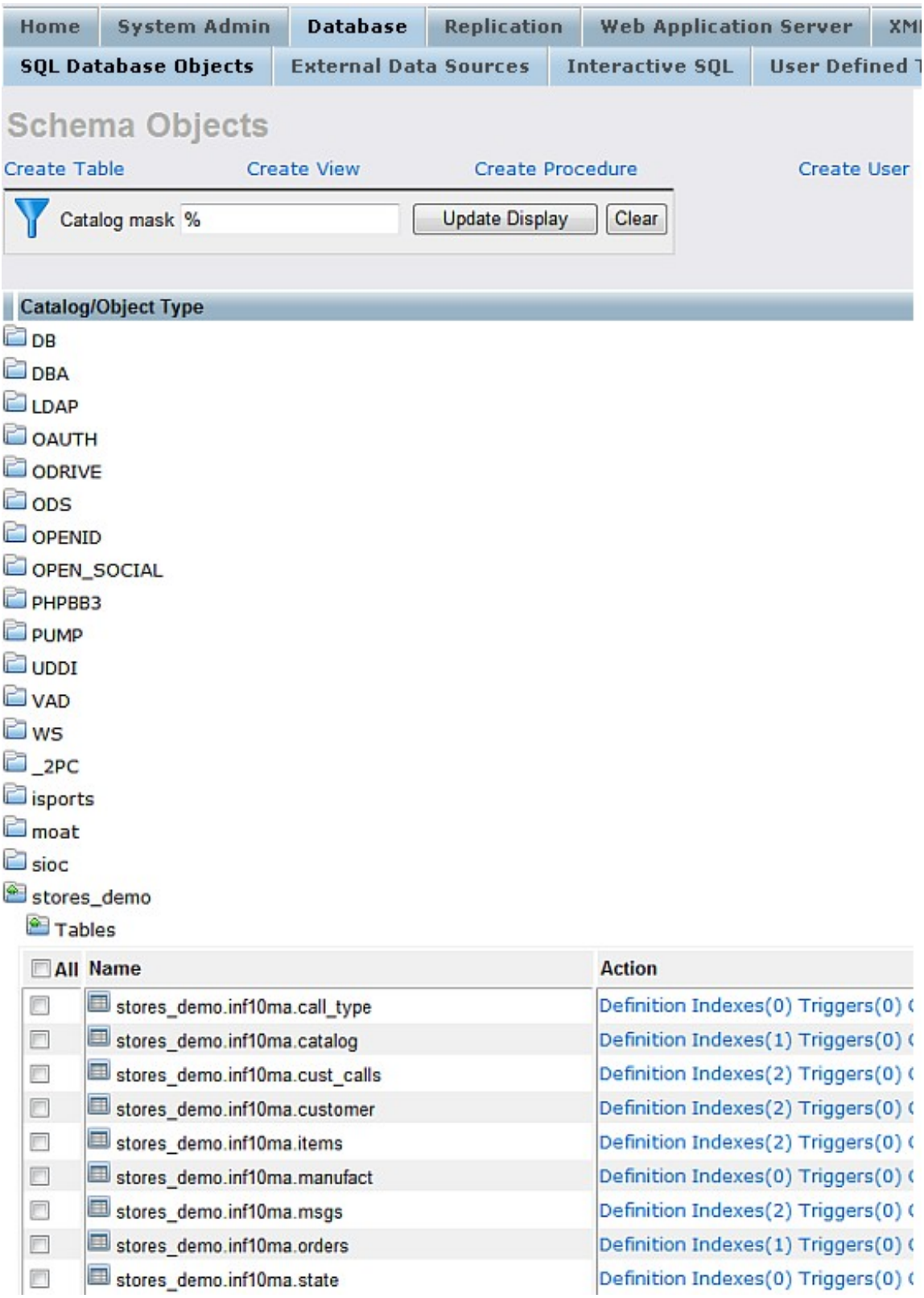
Figure 8.225. Execute



customer_num INTEGER	fname VARCHAR	lname VARCHAR	company VARCHAR	address1 VARCHAR	address2 VARCHAR
101	Ludvig	Pauli	All Sports Supplies	213 Erstwild Court	<DB NULL>
102	Carole	Sadler	Sports Spot	785 Geary St	<DB NULL>
103	Philip	Currie	Phil's Sports	654 Poplar	P. O. Box 3498
104	Anthony	Higgins	Play Ball!	East Shopping Cntr.	422 Bay Road
105	Raymond	Vector	Los Altos Sports	1899 La Loma Drive	<DB NULL>
106	George	Watson	Watson & Son	1143 Carver Place	<DB NULL>
107	Charles	Ream	Athletic Supplies	41 Jordan Avenue	<DB NULL>
108	Donald	Quinn	Quinn's Sports	587 Alvarado	<DB NULL>
109	Jane	Miller	Sport Stuff	Mayfair Mart	7345 Ross Blvd.
110	Roy	Jaeger	AA Athletics	520 Topaz Way	<DB NULL>
111	Frances	Keyes	Sports Center	3199 Sterling Court	<DB NULL>
112	Margaret	Lawson	Runners & Others	234 Wyandotte Way	<DB NULL>
113	Lana	Beatty	Sportstown	654 Oak Grove	<DB NULL>
114	Frank	Albertson	Sporting Place	947 Waverly Place	<DB NULL>
115	Alfred	Grant	Gold Medal Sports	776 Gary Avenue	<DB NULL>
116	Jean	Parmelee	Olympic City	1104 Spinosa Drive	<DB NULL>
117	Arnold	Sipes	Kids Korner	850 Lytton Court	<DB NULL>
118	Dick	Baxter	Blue Ribbon Sports	5427 College	<DB NULL>
119	Bob	Shorter	The Triathletes Club	2405 Kings Highway	<DB NULL>
120	Fred	Jewell	Century Pro Shop	6627 N. 17th Way	<DB NULL>
121	Jason	Wallack	City Sports	Lake Biltmore Mall	350 W. 23rd Stree
122	Cathy	O'Brian	The Sporting Life	543 Nassau Street	<DB NULL>
123	Marvin	Hanlon	Bay Sports	10100 Bay Meadows Ro	Suite 1020
124	Chris	Putnum	Putnum's Putters	4715 S.E. Adams Blvd	Suite 909C

13. The tables can also be viewed as part of the Virtuoso "SQL Schema Objects" under the "stores_demo" catalog name.

Figure 8.226. SQL Schema Objects



The Link process is now complete enabling the tables to be queried as if part of the Virtuoso Schema.

8.5.4. Creating EDM in Visual Studio 2008

The following steps can be used to create an Entity Data Model (EDM) for the Informix stores_demo database:

1. Launch the Visual Studio 2008 SP1 IDE.

Figure 8.227. Visual Studio 2008 SP1 IDE

2. Create a

   Web Application

   project by going to the

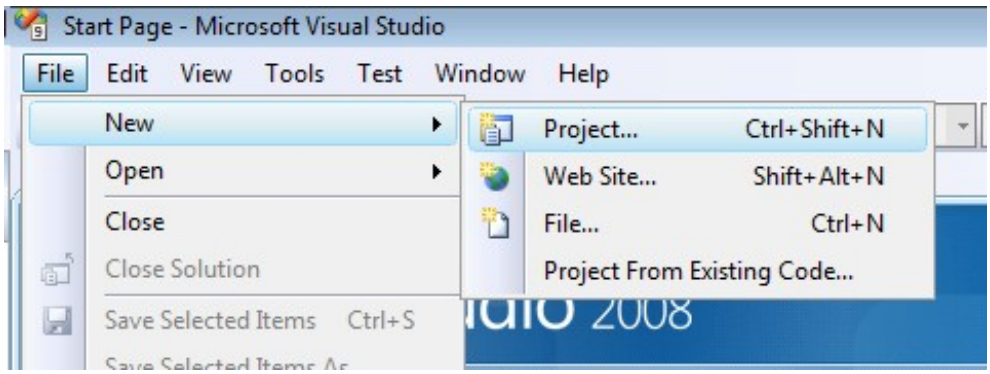
   File

   menu in Visual Studio and choosing

   New Project

   .

   Figure 8.228. Web Application



3. When the New Project window appears, choose either

   Visual Basic

   or

   Visual C#

   as your programming language.
4. Within the language category, click on

   Web

   and select

   ASP.NET Web Application

from the right-hand panel.

5. Choose a name for the project, for example

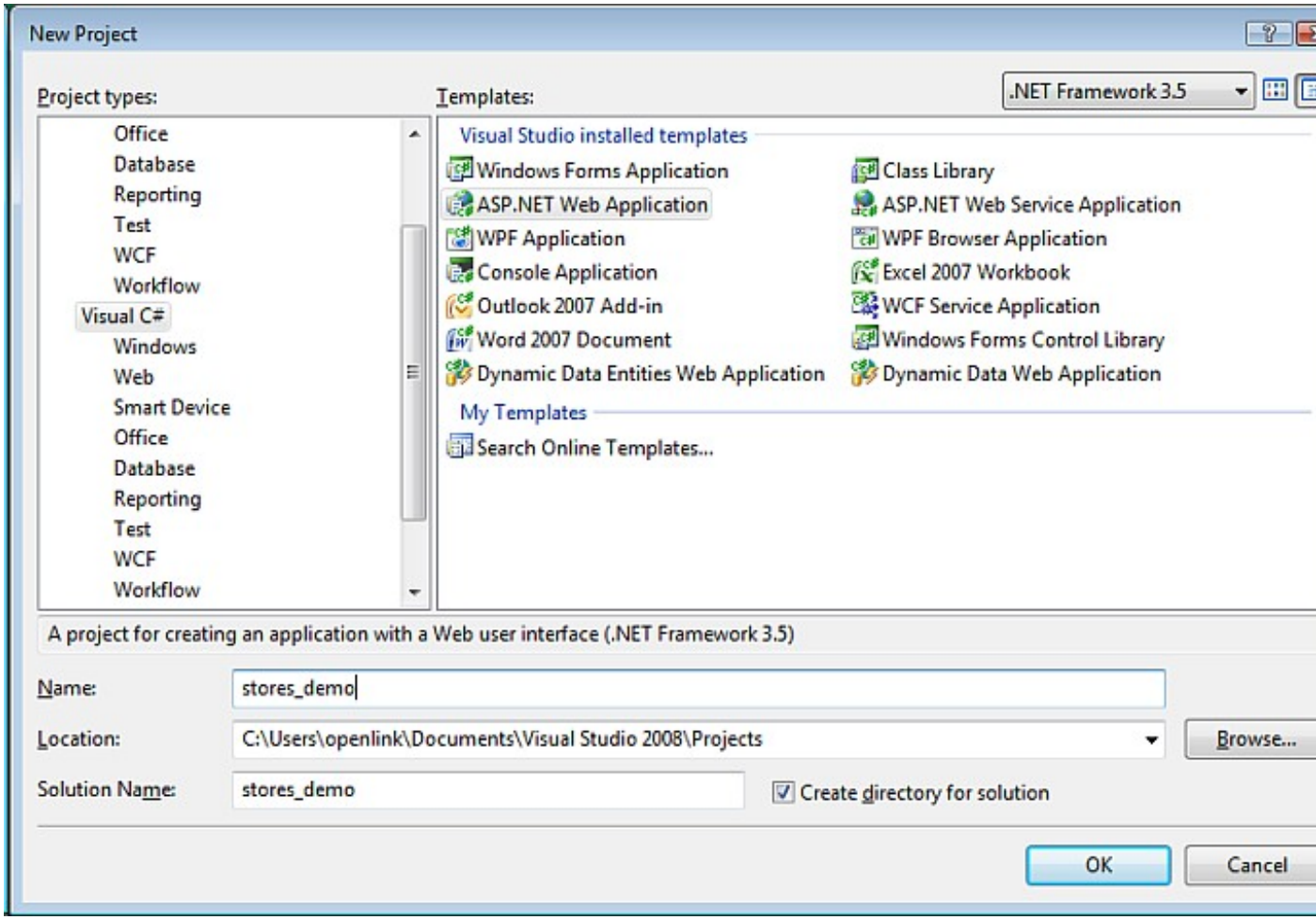
   stores_demo

   , and click

   OK

   .

   Figure 8.229. name for the project

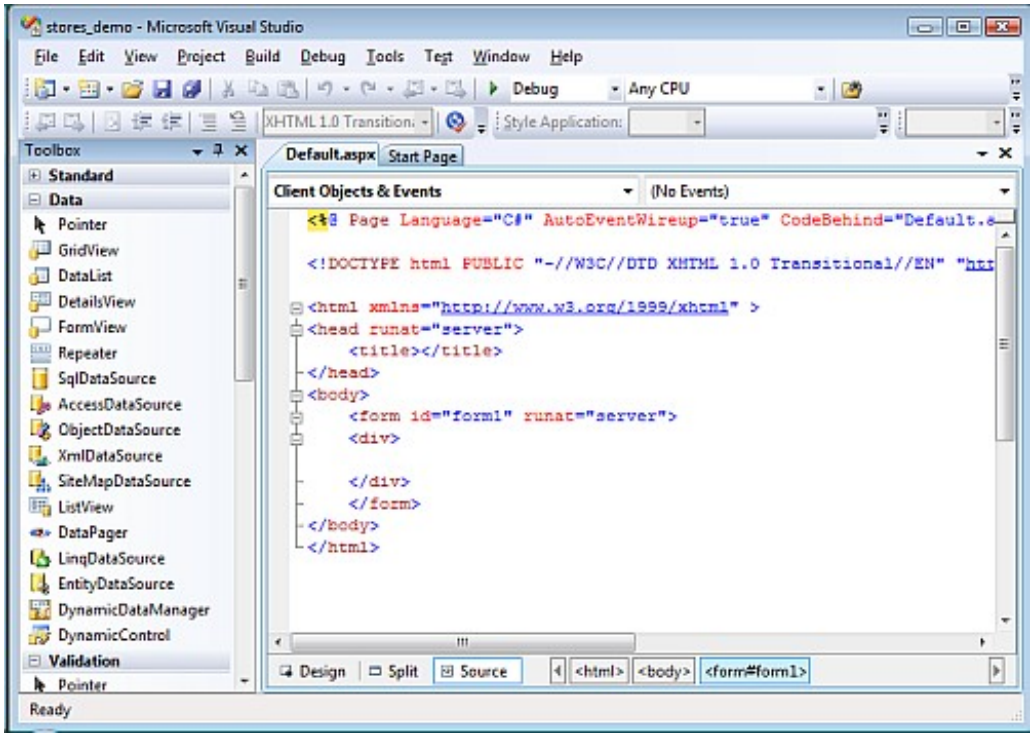


6. This will create a new project called

   stores_demo

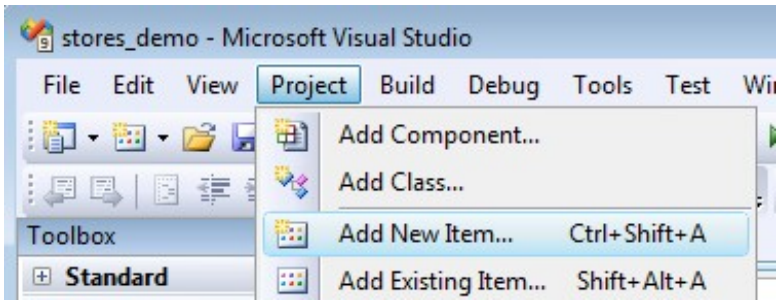
   .

   Figure 8.230. create a new project



7. Select the Project -> Add New Item menu option.

Figure 8.231. New Item



8. The

Add New Item

dialog will appear. Choose the

ADO.NET Entity Data Model

template. Give it the name

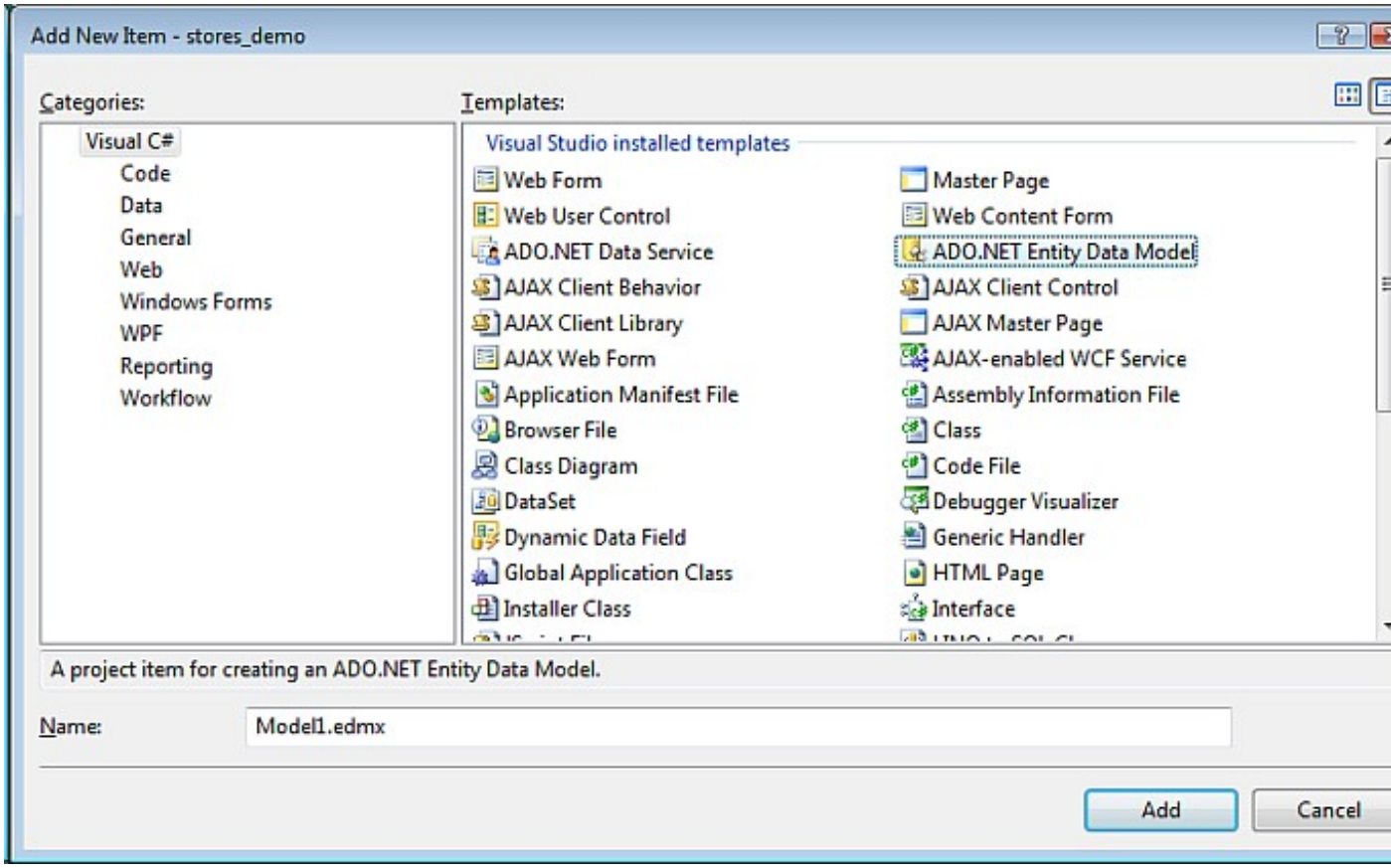
Model1.edmx

e and click

Add

to start the creation of the ADO.Net Entity Data Model.

Figure 8.232. Add New Item



9. In the

   Entity Data Model Wizard

   dialog

   Choose Model Contents

   page select the

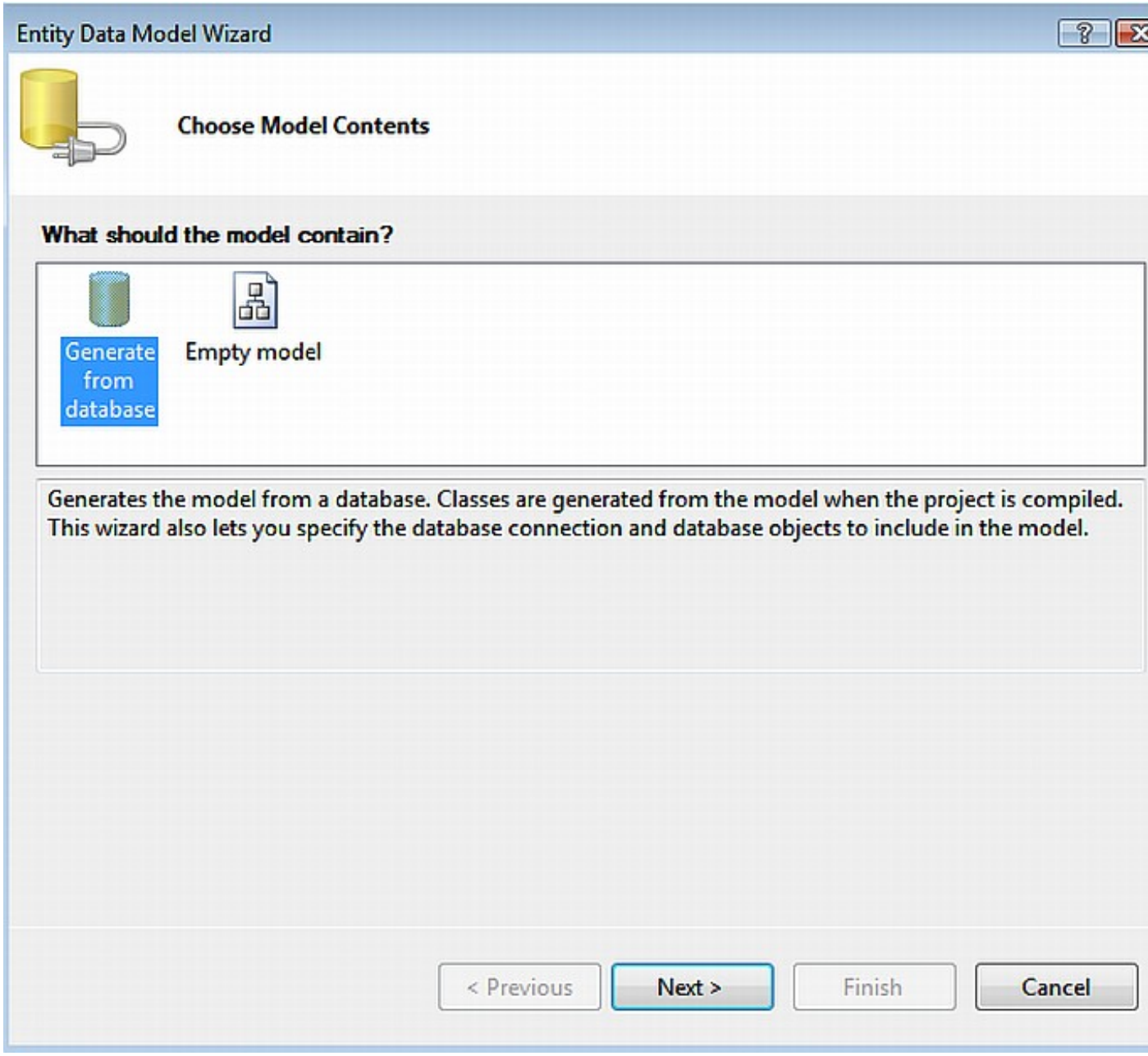
   Generate from Database

   model type and click

   Next

   .

Figure 8.233. Choose Model Contents



10. In the

    Entity Data Model Wizard

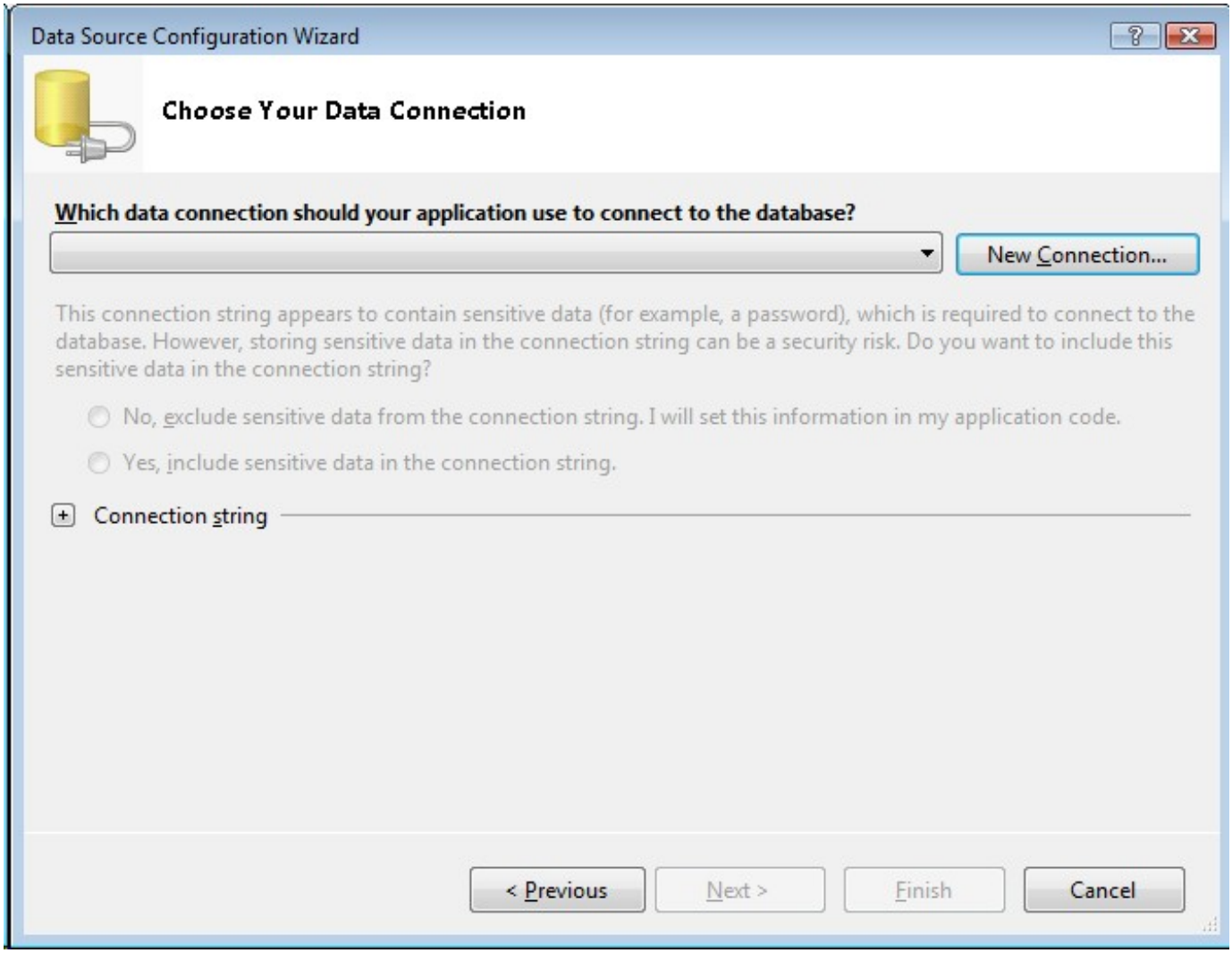
    dialog

    Choose your Data Connection

    page, select the

    New Connection button

Figure 8.234. Entity Data Model Wizard



11. In the

    Choose Data Source

    dialog, select the OpenLink

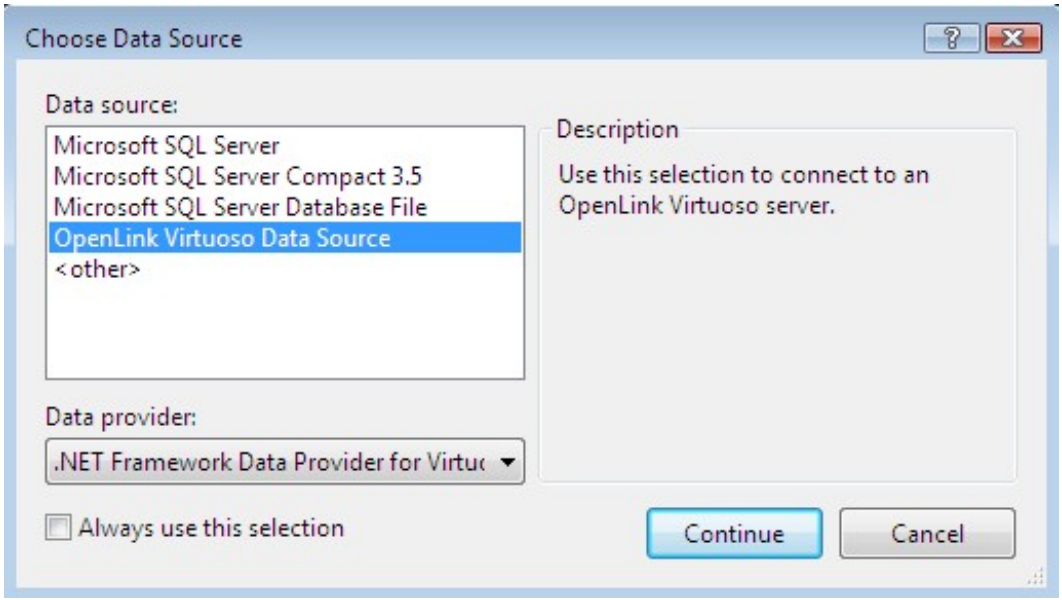
    Virtuoso Data Source

    from the list and click

    Continue

    .

Figure 8.235. Choose Data Source



12. In the

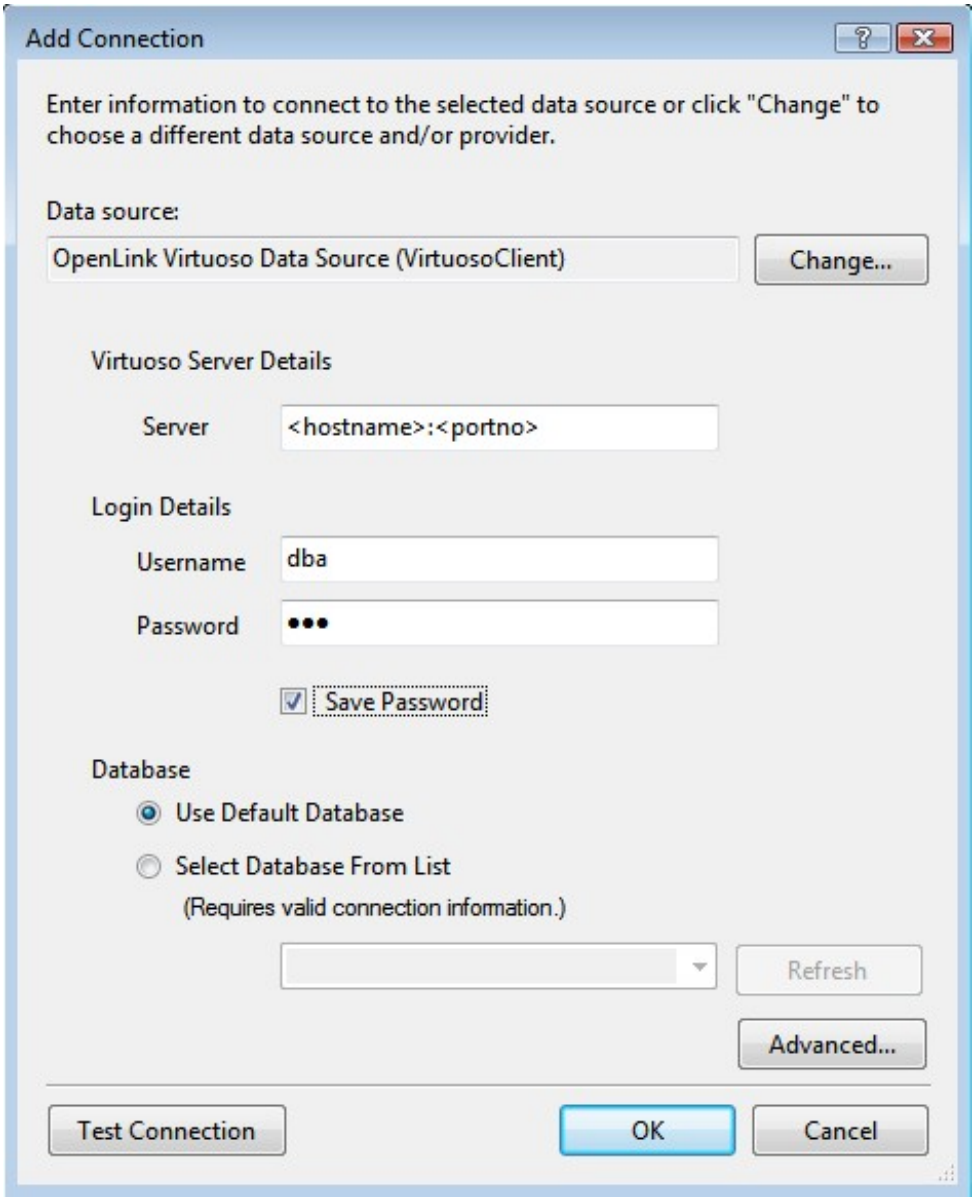
    Add Connection

    dialog, specify the

    hostname, portno, username and password

    for the target Virtuoso Server and check the Save Password check box.

    Figure 8.236. Connection Properties



13. Select the

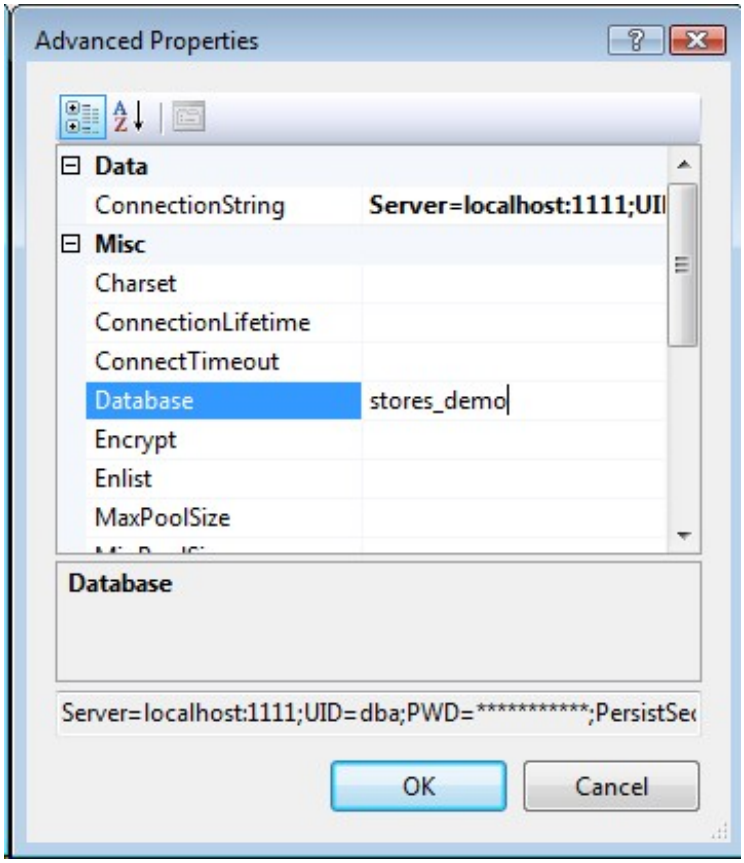
    Select Database From List

    radio button and choose the

    stores_demo

    database from the drop down list.

Figure 8.237. Add connection

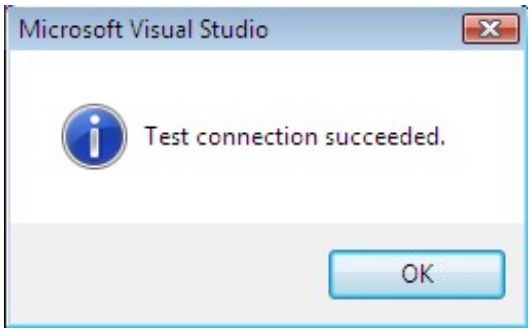


14. Press the

    Test Connection

    dialog to verify that the database is accessible.

    Figure 8.238. Test Connection



15. Set the

    entity connect string

    name to

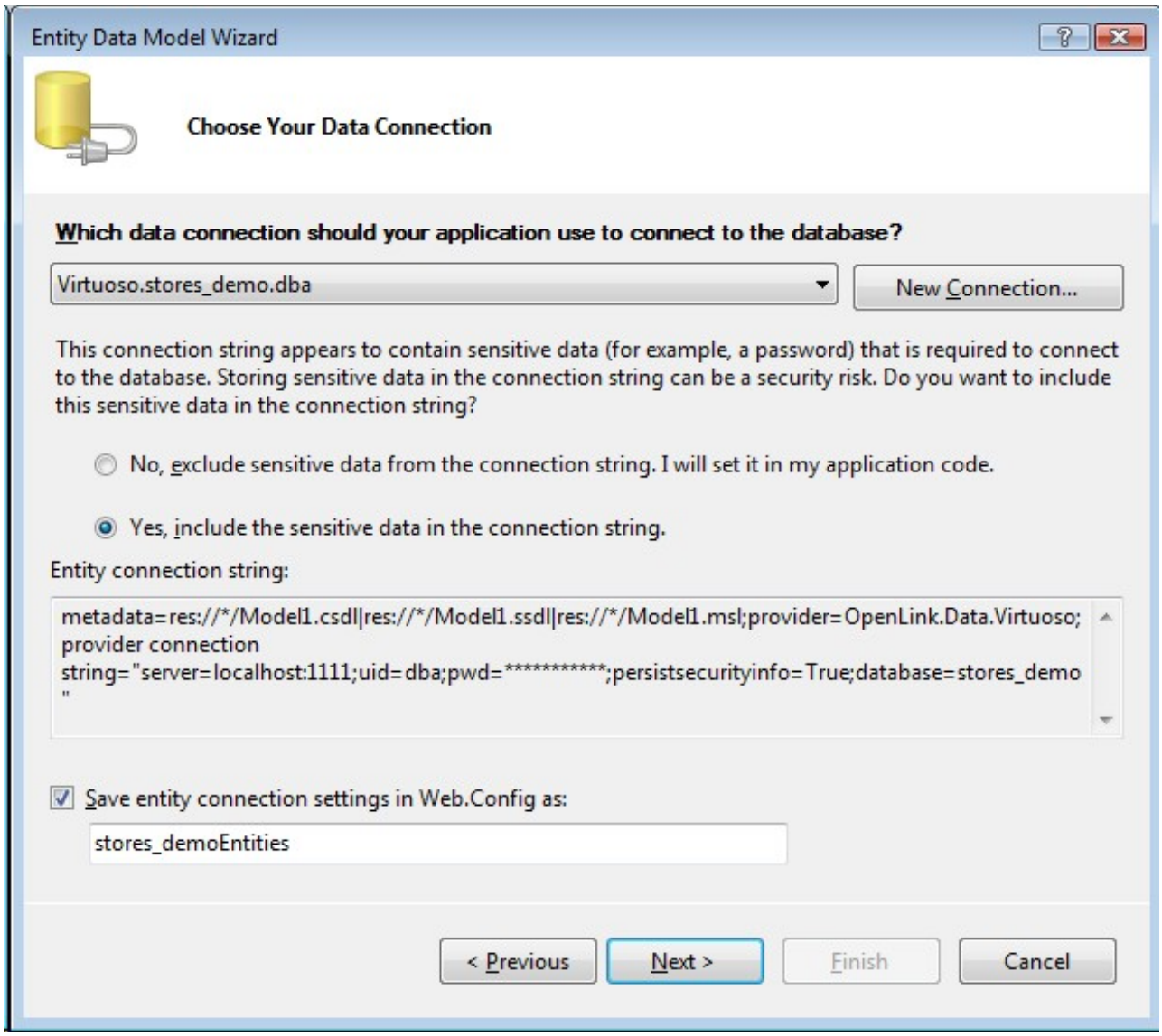
    stores_demoEntities

    (note this name as it is required in the section on creating and ADO.Net Data Service) and click

    Next

    .

Figure 8.239. entity connect string



16. In the

    Choose your Database Objects

    page tick the

    Tables

    check box to select all tables in the stores_demo catalog for addition to the Entity Data Model. Set the

    Model Namespace

    to

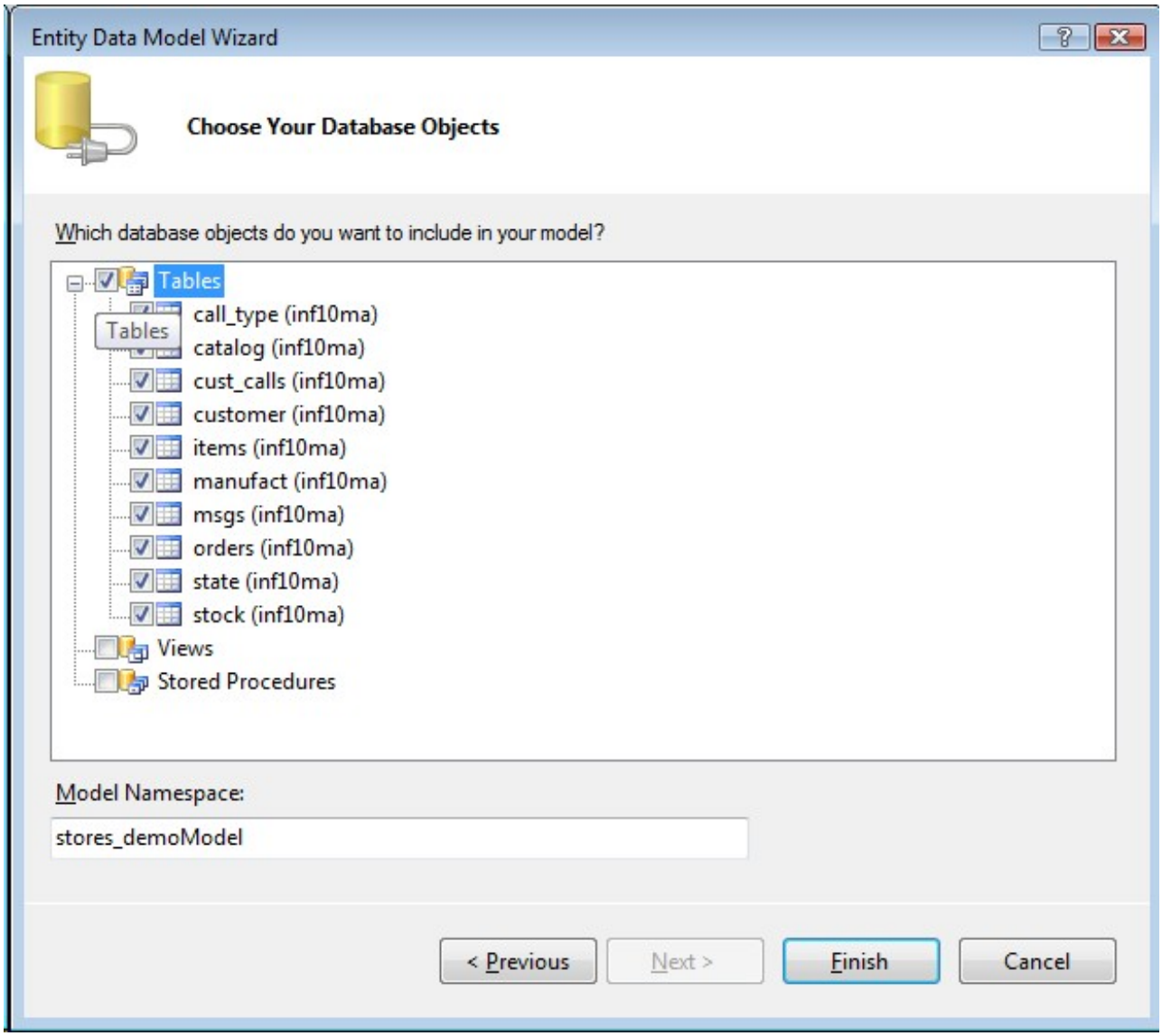
    stores_demoModel

    and click

    Finish

    .

Figure 8.240. Database Objects

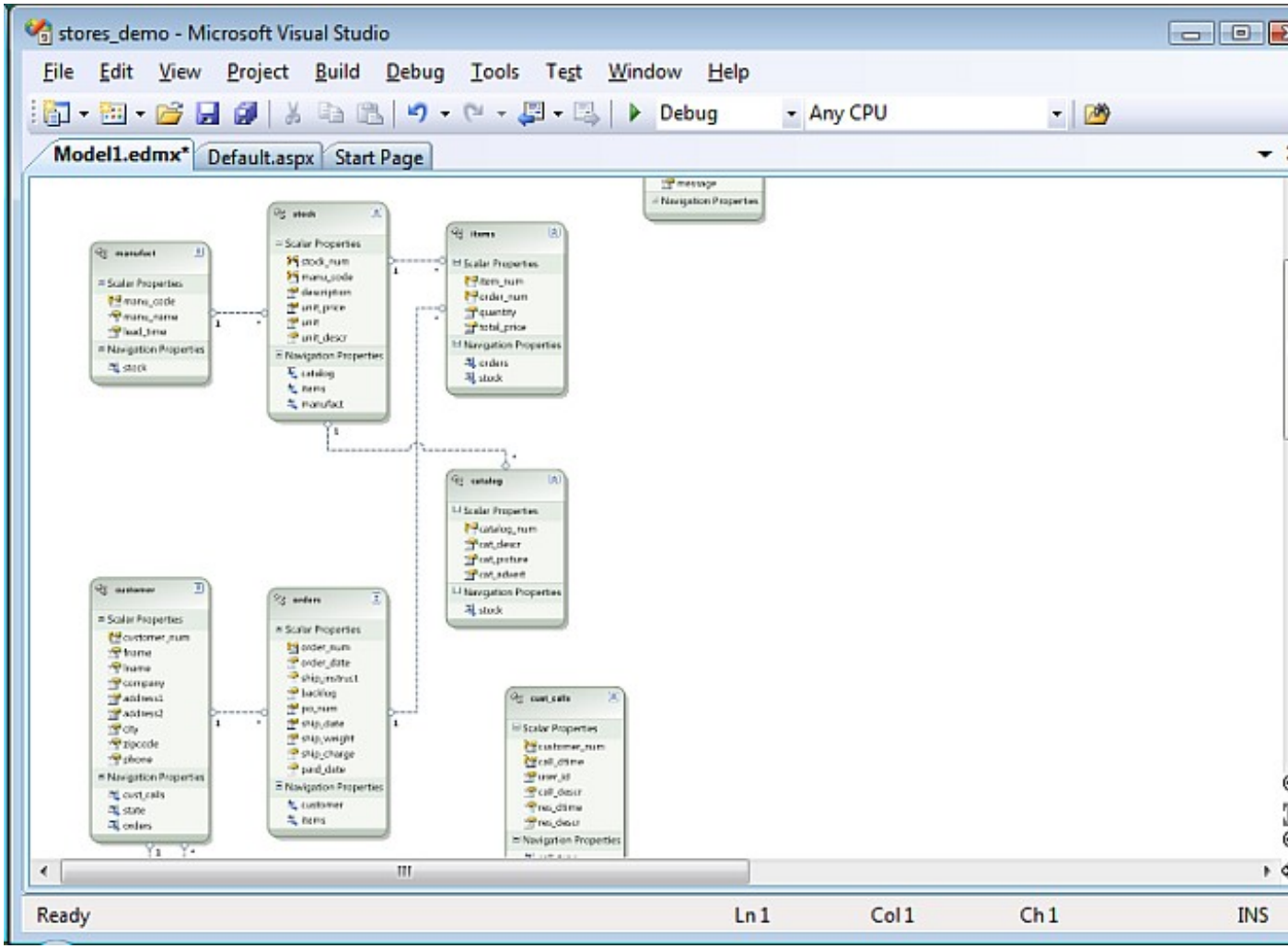


17. The

    Model1.edmx

    EDM will be created with the tables and relationships displayed in the Visual Studio IDE

Figure 8.241. Virtuoso.edmx



Creation for the Entity Data Model for the Informix stores_demo database is now complete.

8.5.5. Using EDM to create Entity Framework based applications

Now that a Microsoft Entity Data Model has been created for the Informix stores_demo database, Entity Framework applications can be created to make use of it.

Entity Frameworks based finf

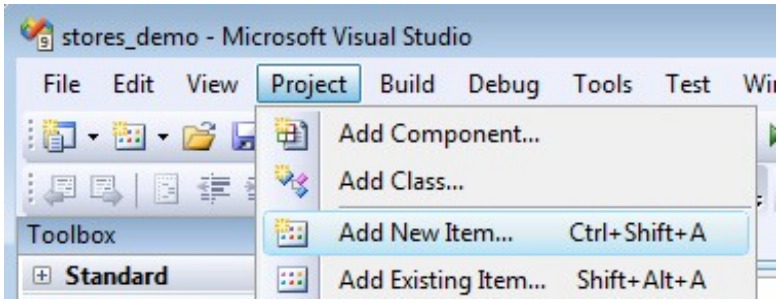
An ADO.Net Data Service for the Informix tables can be created using the Entity Data Model created in the Creating EDM in Visual Studio 2008 section .

1. Open the

   stores_demo

   project created in the Creating EDM in Visual Studio 2008 section .
2. Select the Project -> Add New Item menu option.

Figure 8.242. VirtuosoDataService



3. The

   Add New Item

   dialog will appear. Choose the

   ADO.NET Data Service template

   . Give it the name

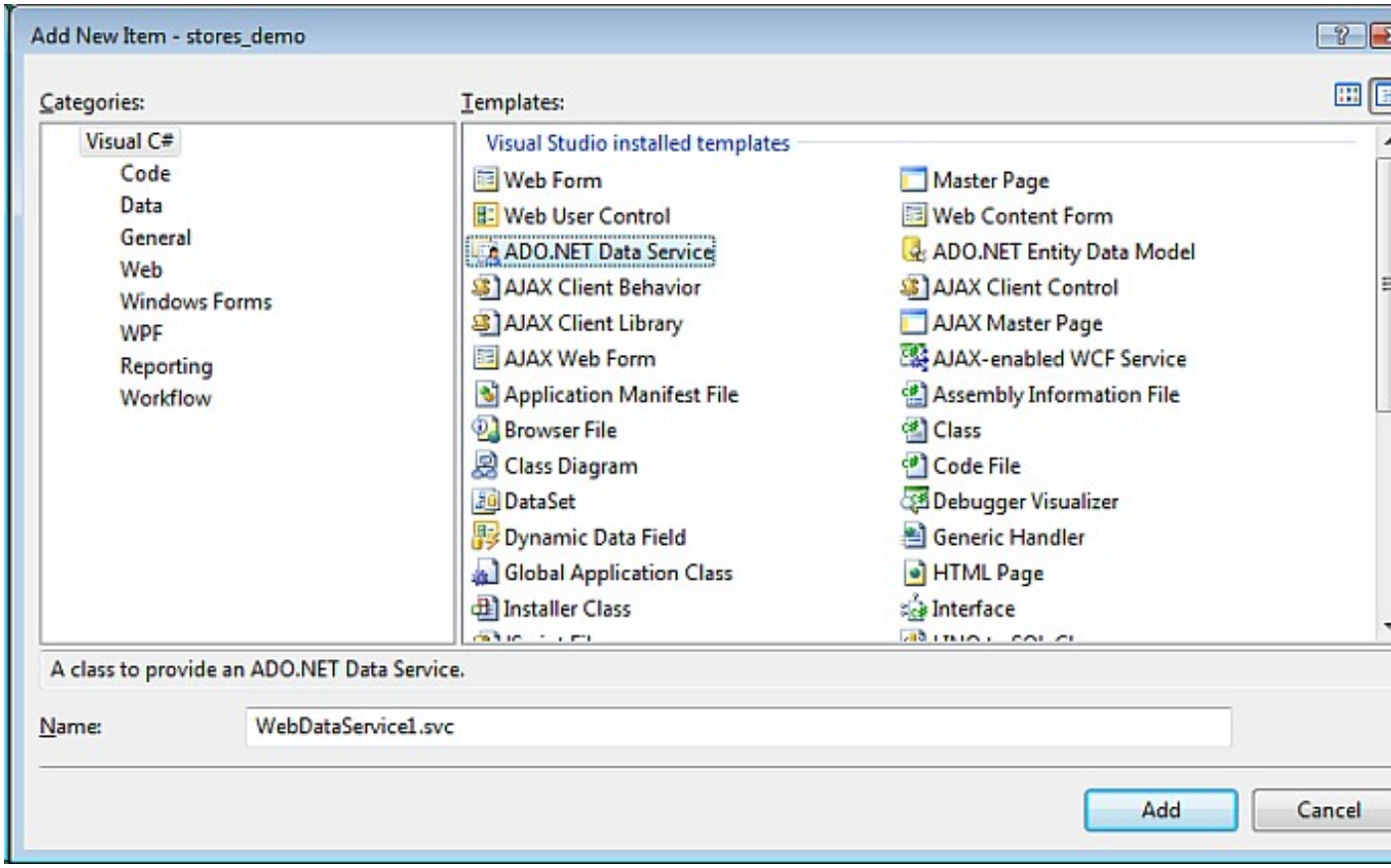
   WebDataService.svc

   and click

   Add

   to create the ADO.Net Data Service.

   Figure 8.243. Add New Item



4. In the newly created

   WebDataService1.svc.cs

Data Service file, add the data source class name of

stores_demoEntities

(note this is the name set in the Creating EDM in Visual Studio 2008 section) as the

DataService

name. Enable the access to the Data Service by adding the entry

config.SetEntitySetAccessRule("", EntitySetRights.All);*

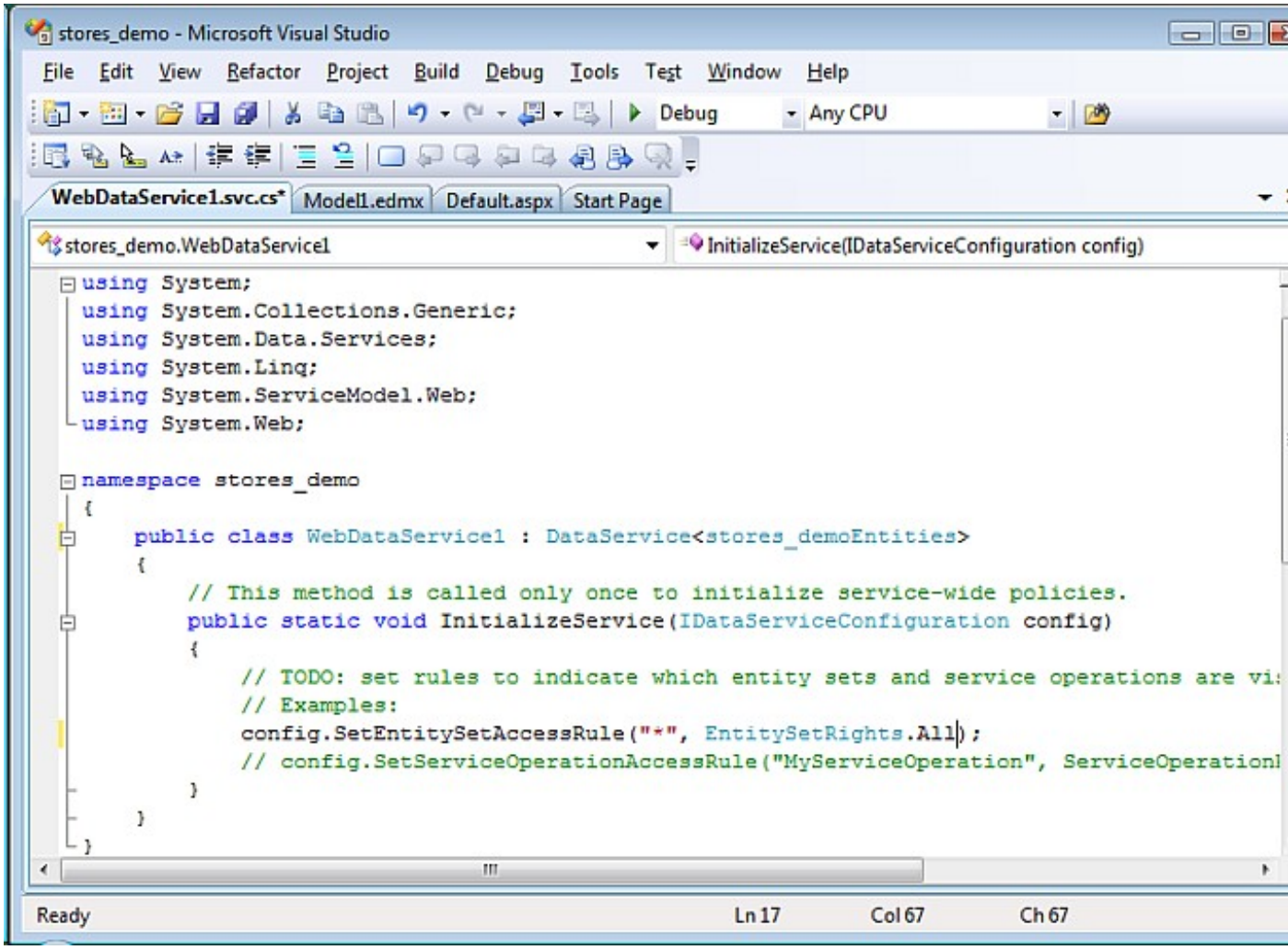
in the

InitializeService

method.

```
// C#

using System;
using System.Web;
using System.Collections.Generic;
using System.ServiceModel.Web;
using System.Linq;
using System.Data.Services;

namespace SimpleDataService
{
    public class Northwind : DataService<stores_demoDemoEntities>
    {
        public static void InitializeService(IDataServiceConfiguration  config)
        {
            config.SetEntitySetAccessRule("*", EntitySetRights.All);
        }
    }
}
```

Figure 8.244. WebDataService1.svc.cs

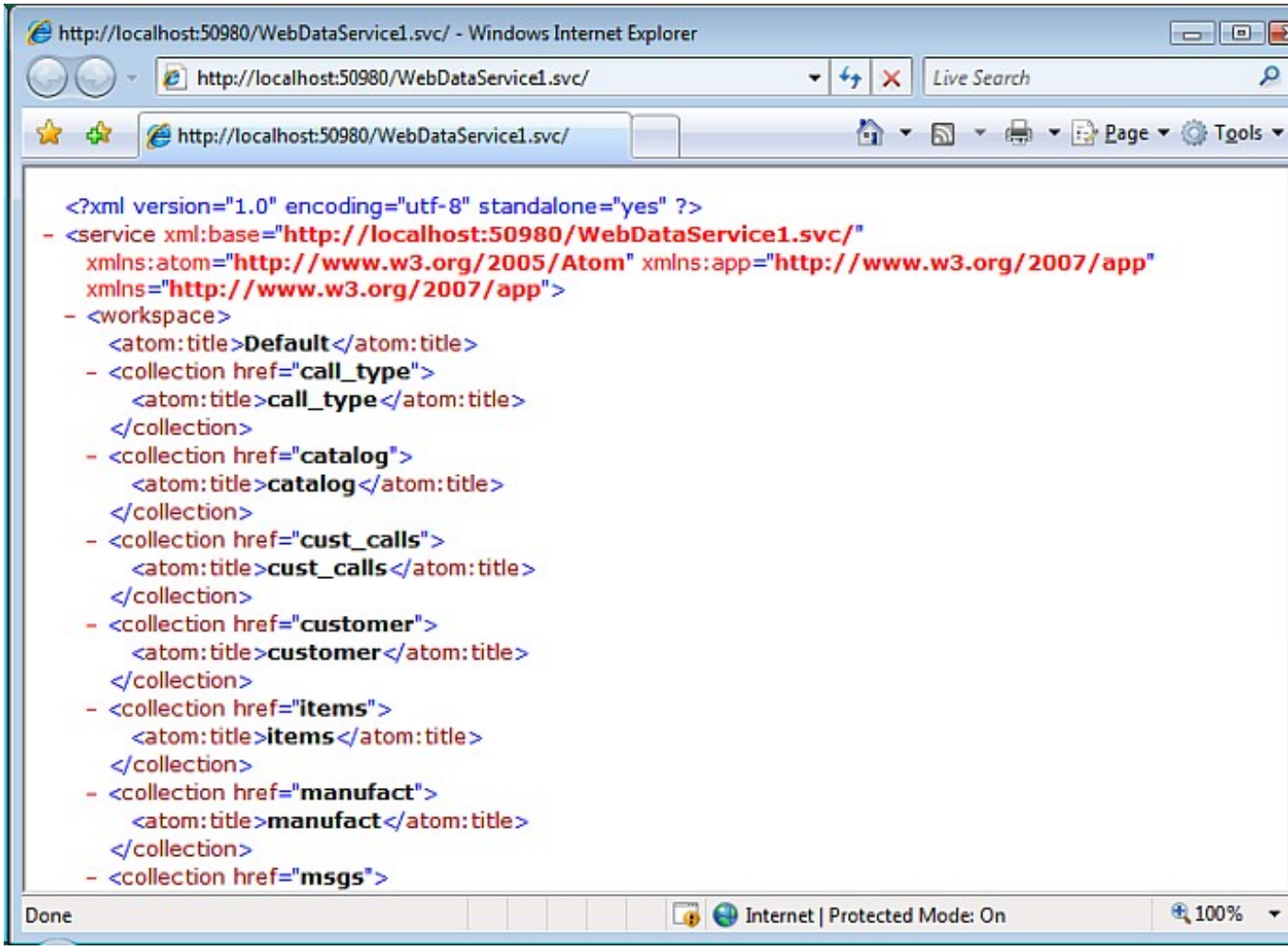


5. To test the Data Service, simply hit

   Ctrl+F5

   within Visual Studio. This will start the development web server, run the Data Services server inside, and load a Web browser page displaying the list of available tables/entities for the stores_demo database catalog.

Figure 8.245. Data Service test



6. To access a specific entity instance like the

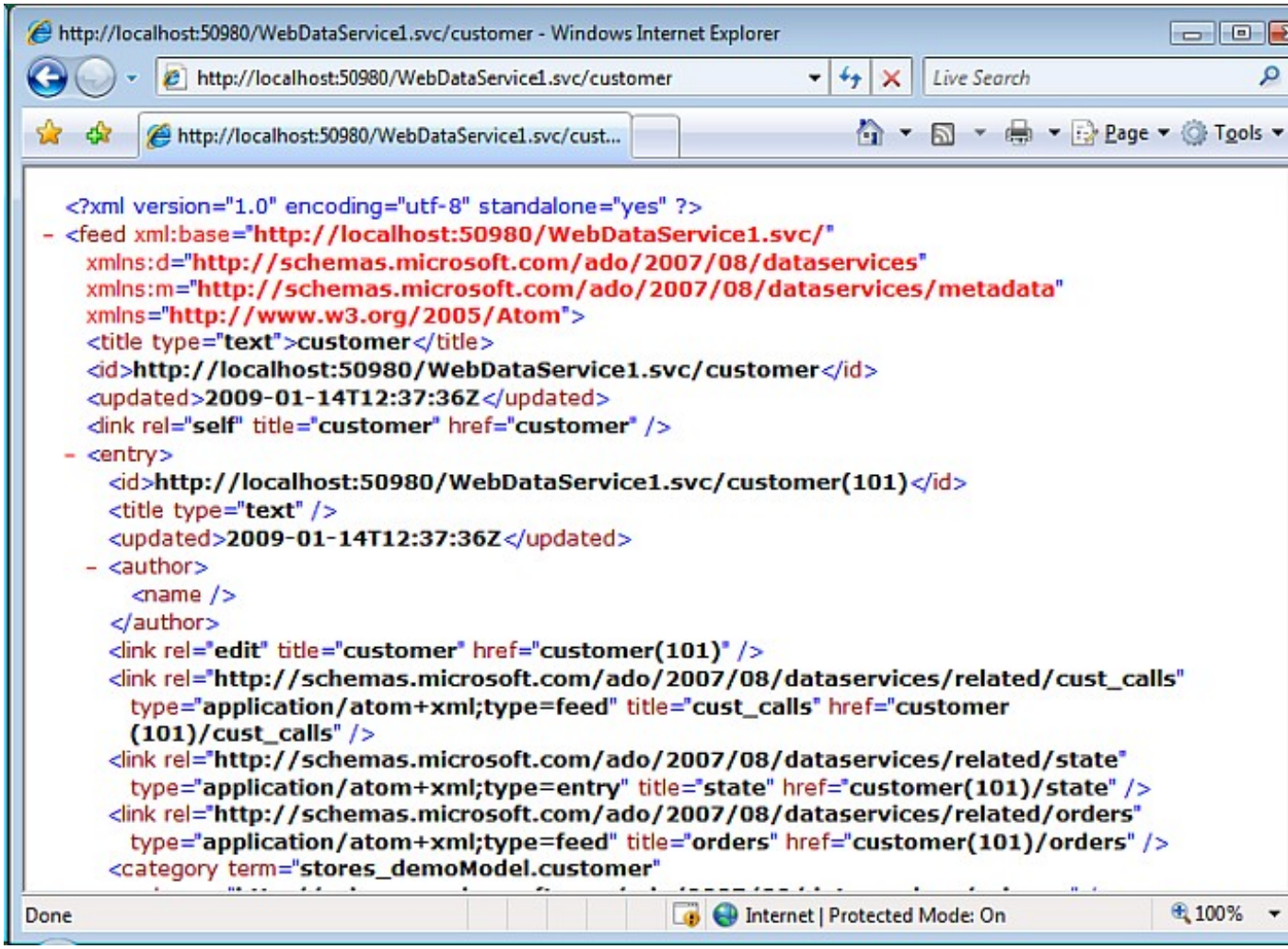
   CUSTOMERS

   table customer number

   101

   record, use this convention http://host/vdir/WebDataService1.svc/customer(101).

   Figure 8.246. custmers



Notes:

1. *Important*

   - To view

   Atom

   (the default format returned by an ADO.NET Data Service) in Internet Explorer, you must first ensure that

   Feed Reading View

   is turned

   off

   . This can be done on the

   Content tab

   of

   Tools in Internet Options

   .
2. If a Data Services entity instance URI page fails to load you can turn

Verbose

errors on by adding

config.UseVerboseErrors = true;

in the

virtuoso.svc.cs InitializeService

method to obtain more detailed information from the server as to why the page failed to load:

```
public static void InitializeService(IDataServiceConfiguration config)
{
config.UseVerboseErrors = true;
config.SetEntitySetAccessRule("*", EntitySetRights.All);
}
```

Visual Studio Windows DataGrid Form Application

This sectiond details the steps required to create a simple Visual Studio 2008 Windows Form application, with associated DataGridView control for displaying data in selected tables from the target database.

1. Launch the Visual Studio 2008 SP1 IDE.

   Figure 8.247. Visual Studio 2008 SP1 IDE

2. Create a

   Web Application

   project by going to the

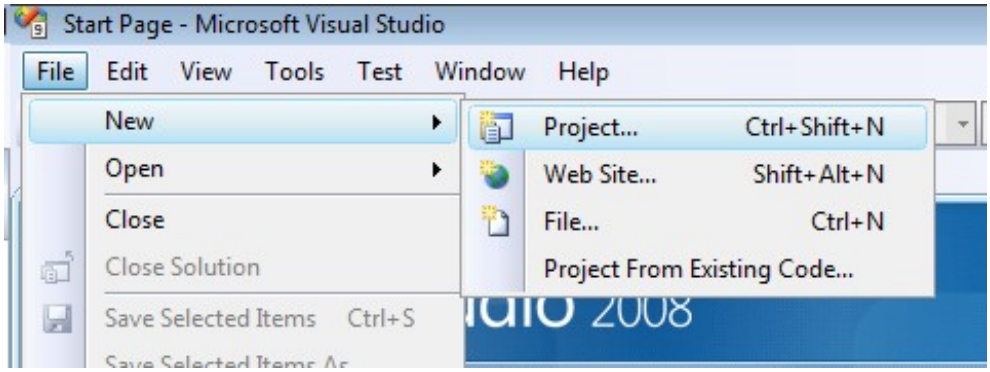
   File

   menu in Visual Studio and choosing

   New Project

   .

   Figure 8.248. Web Application



3. When the New Project window appears, choose either

   Visual Basic

   or

   Visual C#

   as your programming language.
4. Within the language category, click on

   Windows

   and select

   Windows Form Application

   from the right-hand panel.
5. Choose a name for the project, for example

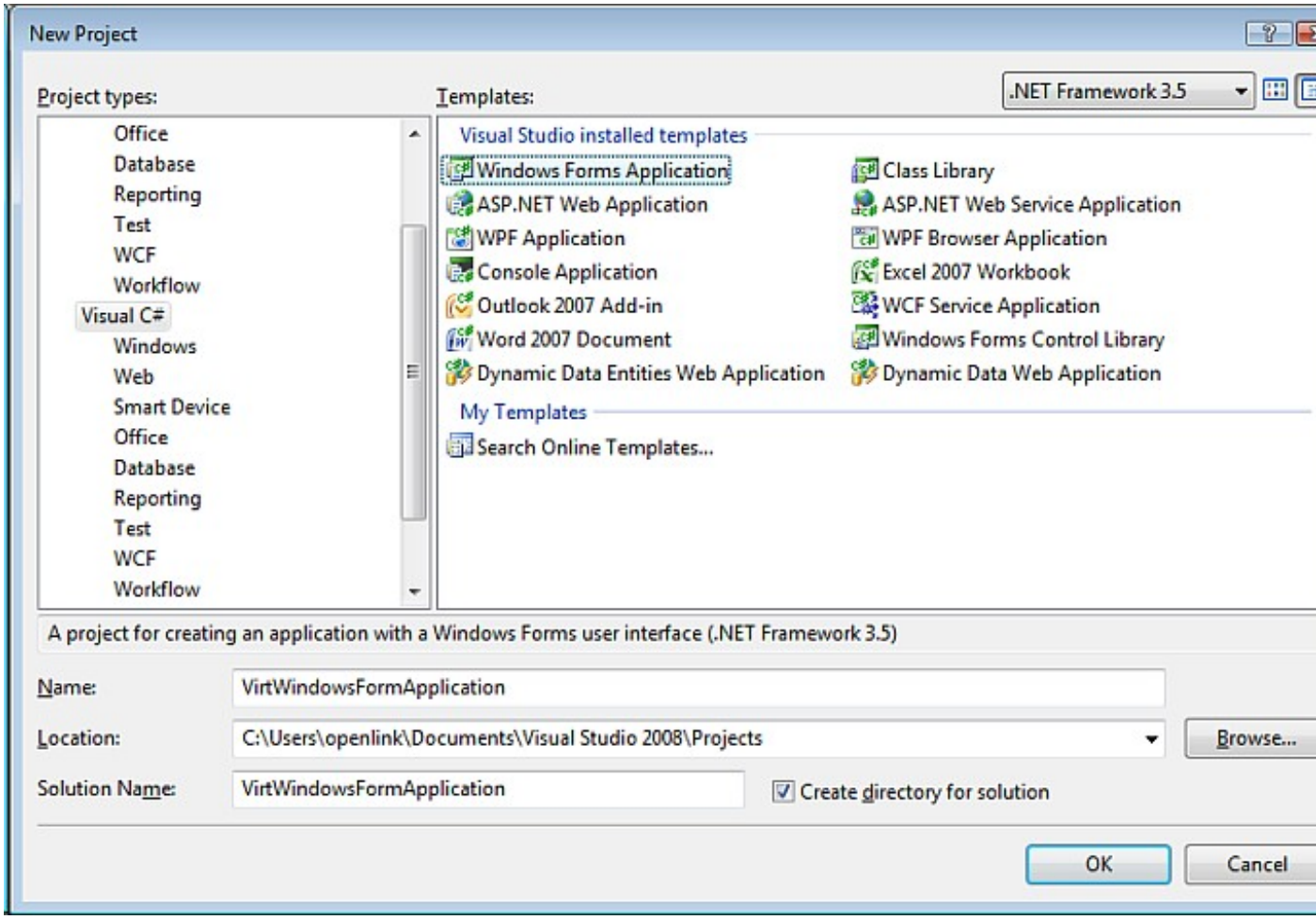
   VirtWindowsFormApplication

   , and click

   OK

   .

   Figure 8.249. Web Application



6. In the

   Toolbox

   , expand

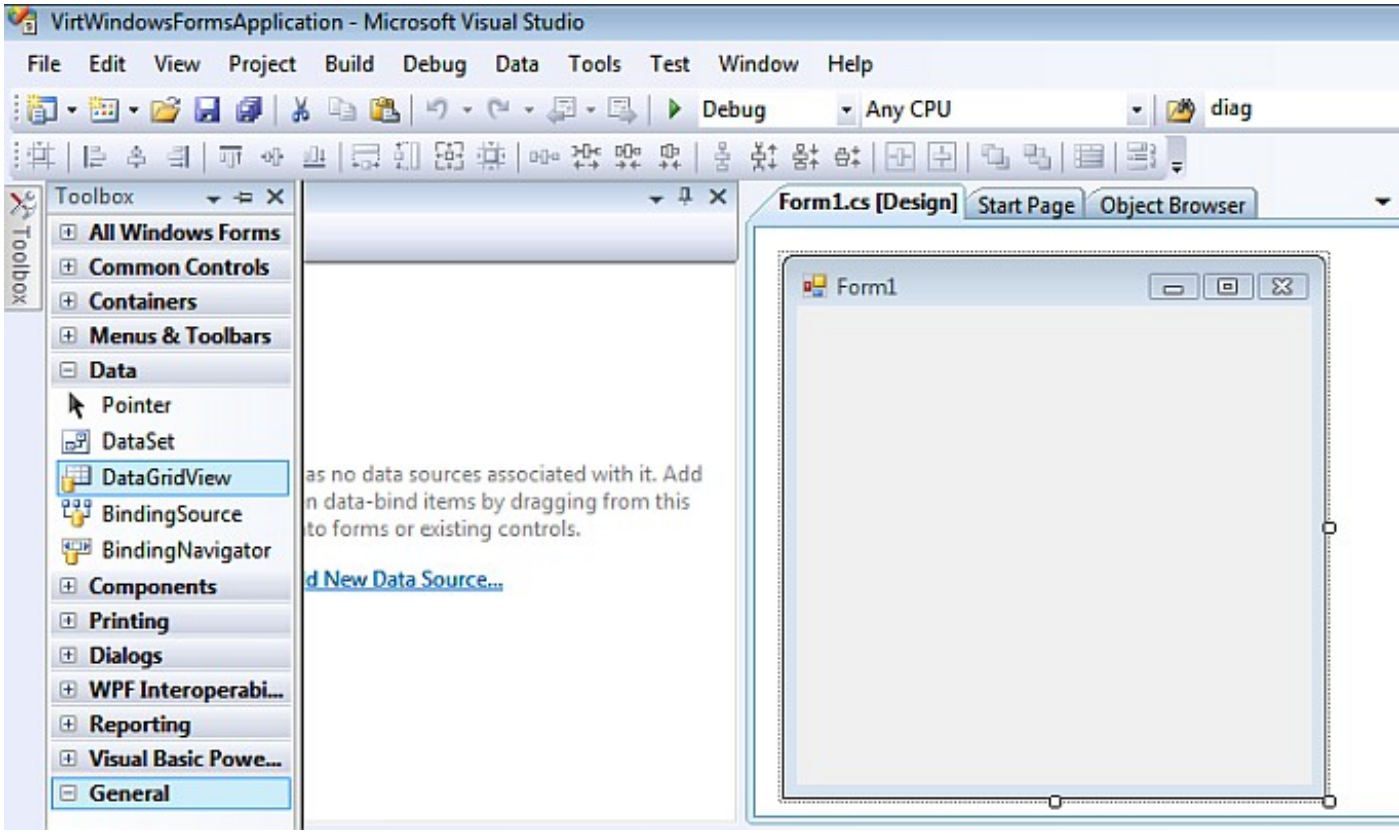
   Data Controls

   , and drag the

   DataGridView

   control onto the form.

   Figure 8.250. Toolbox



7. Click on the little

   arrow

   in the top right of the

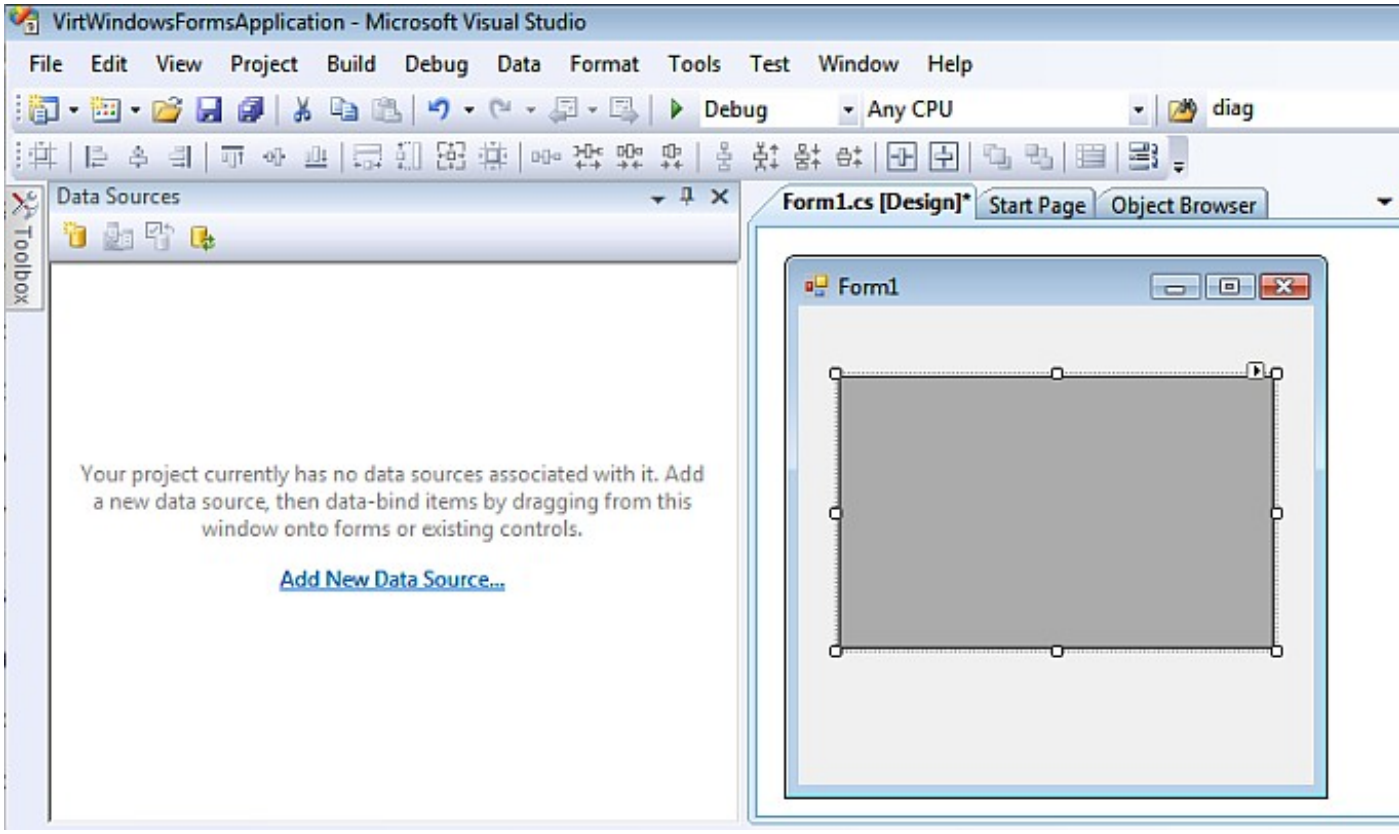
   DataGridView

   control. This loads the

   DataGridView Task

   menu.

Figure 8.251. DataGridView Task

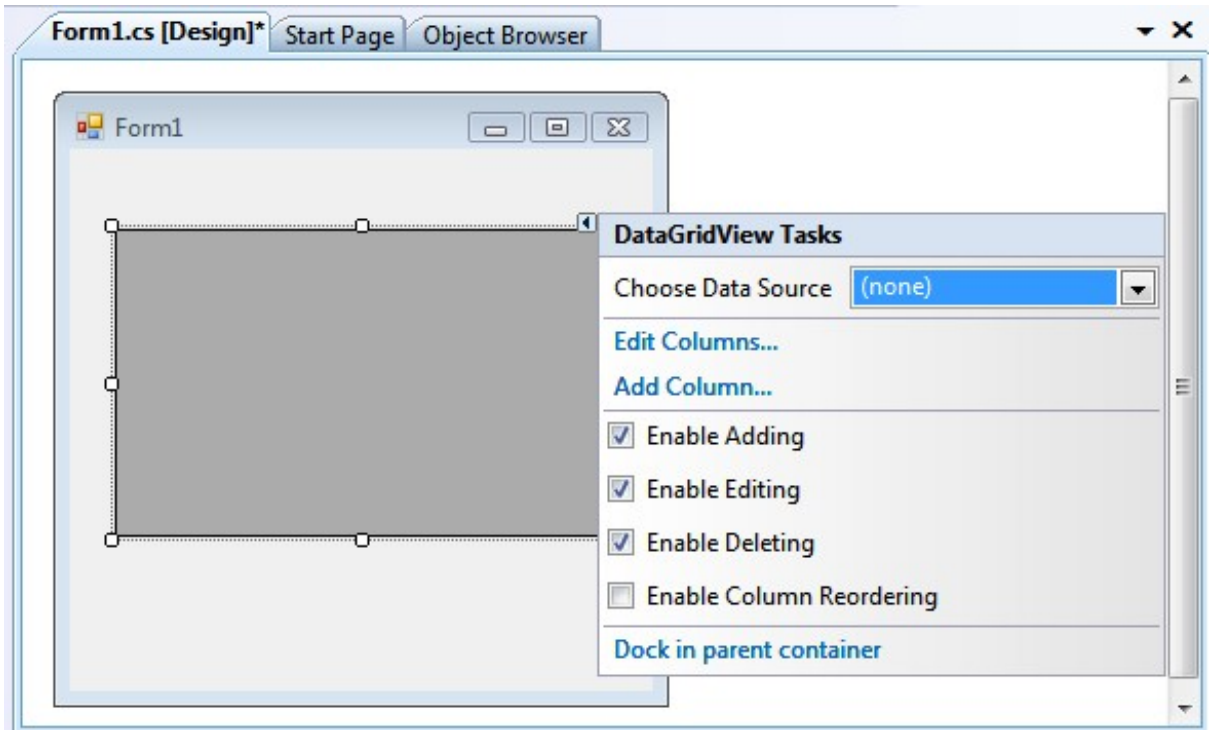


8. Click on the

   Choose Data Source

   list box.

   Figure 8.252. Choose Data Source

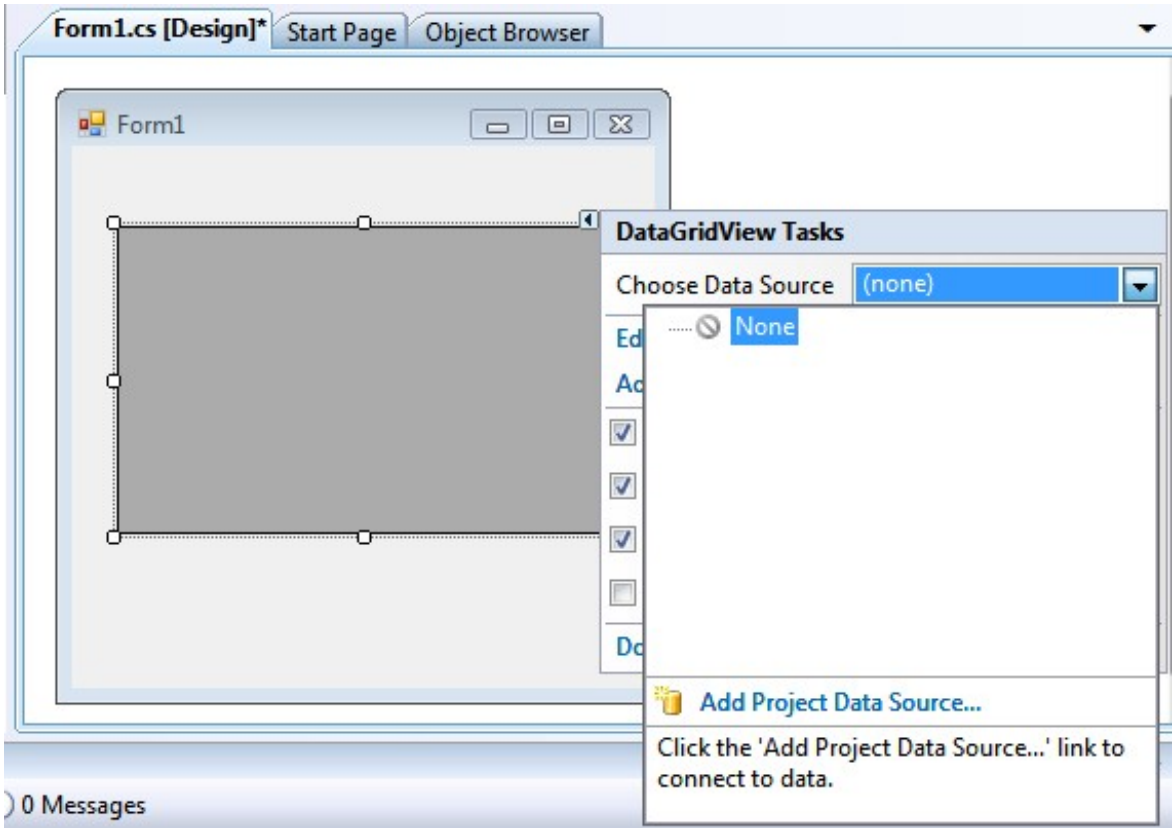


9. Click on the

   Add Project Data Source

link to connect to a data source.

Figure 8.253. Add Project Data Source



10. In the

    Data Source Configuration Wizard

    dialog

    Choose Data Source Type

    page select the

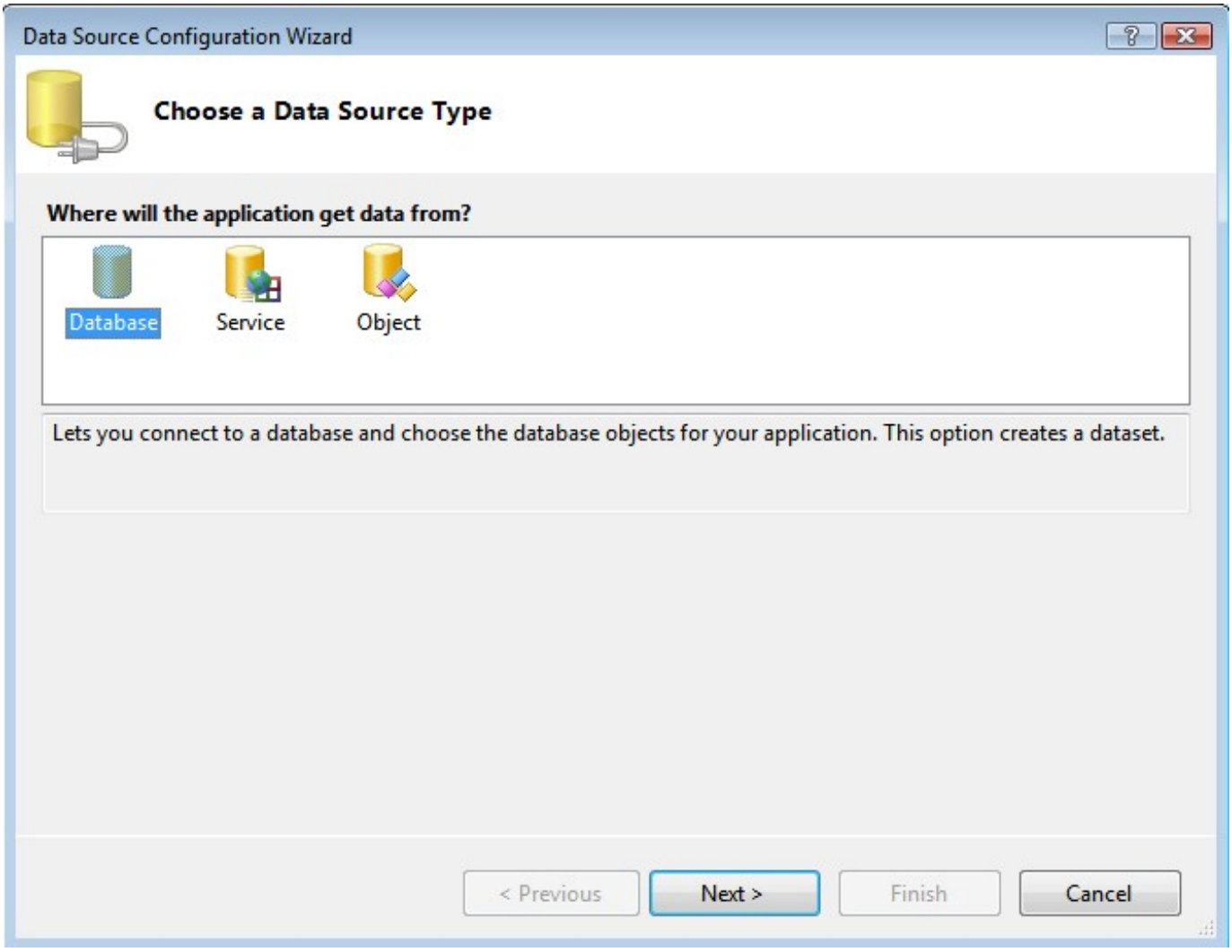
    Database

    data source type and click

    Next

    .

Figure 8.254. Data Source Type



11. In the

    Data Source Configuration Wizard

    dialog

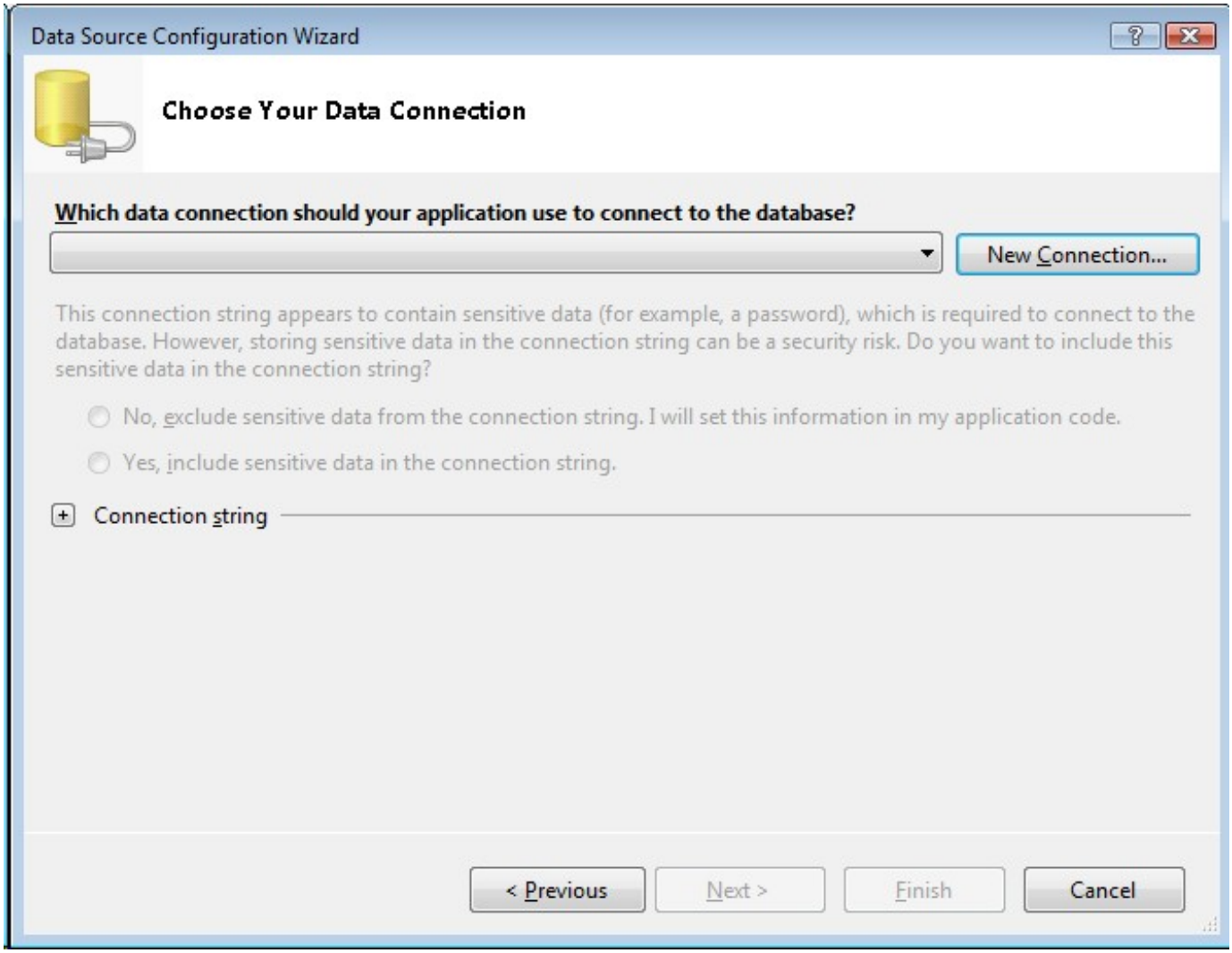
    Choose your Data Connection

    page, select the

    New Connection

    button

Figure 8.255. Data Source Configuration Wizard



12. In the

    Choose Data Source

    dialog, select the OpenLink

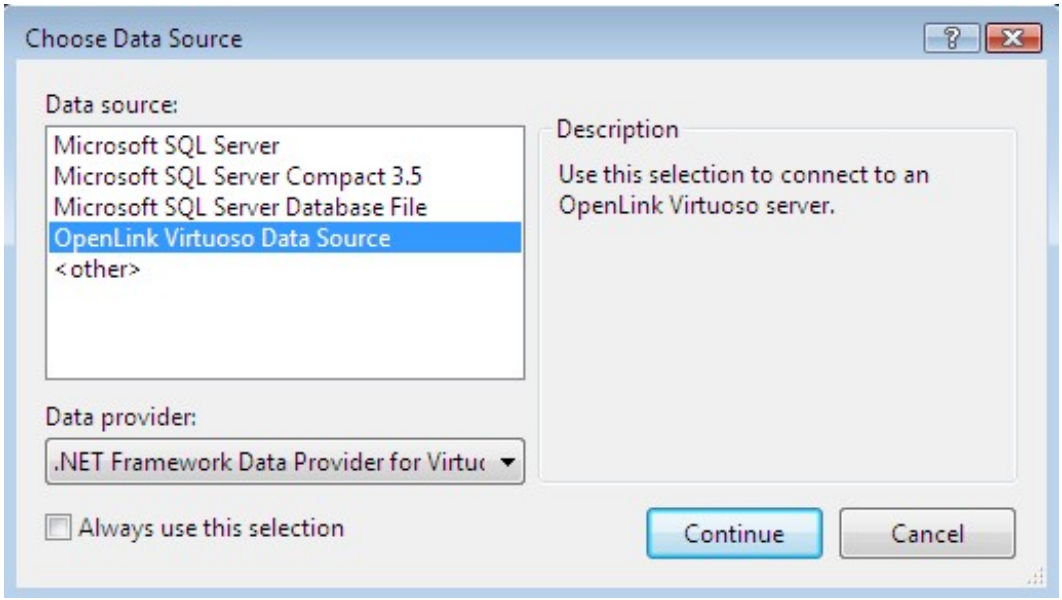
    Virtuoso Data Source

    from the list and click

    Continue

    .

Figure 8.256. Data Source



13. In the

    Add Connection

    dialog, specify the

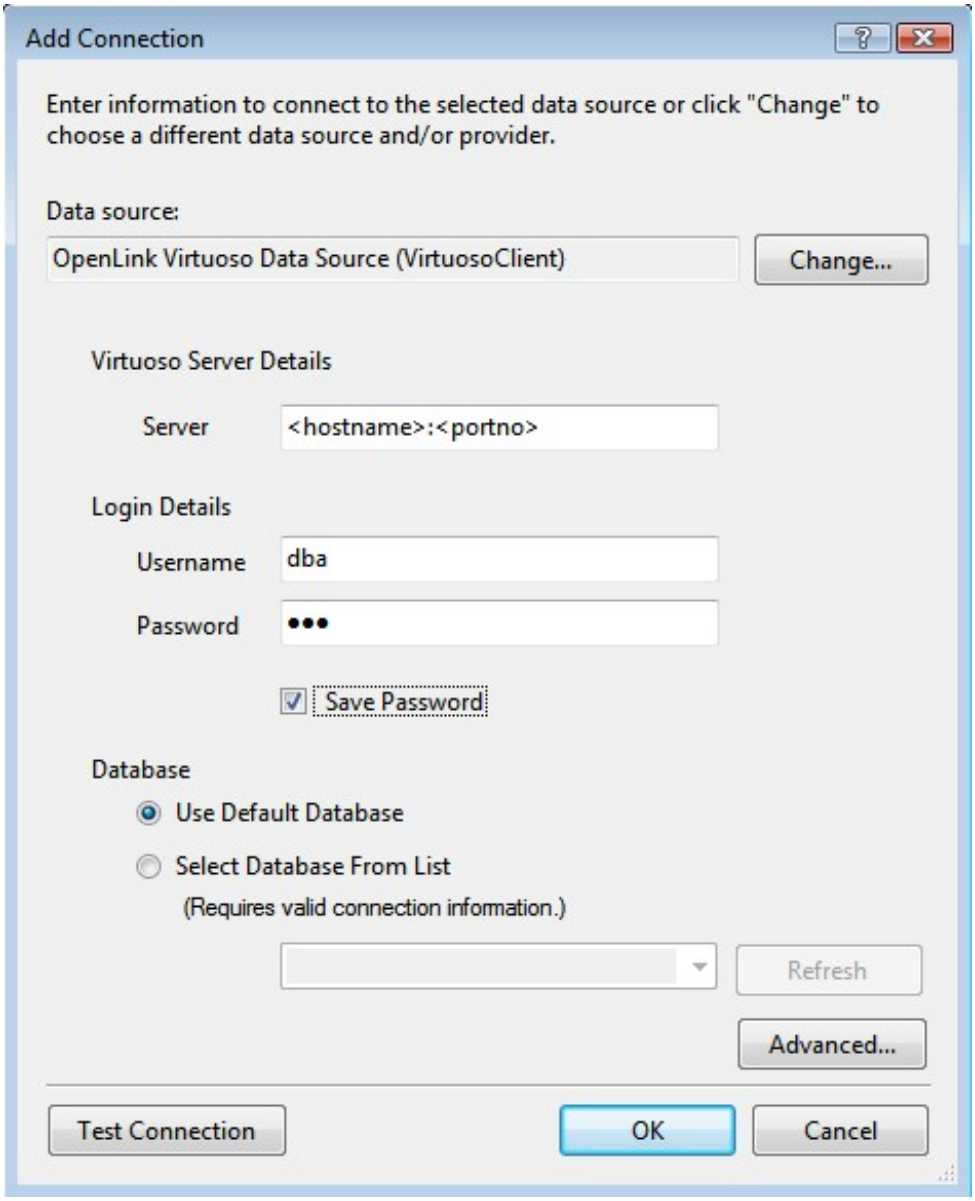
    hostname, portno, username and password

    for the target Virtuoso Server and check the

    Save Password

    check box.

    Figure 8.257. Connection Properties

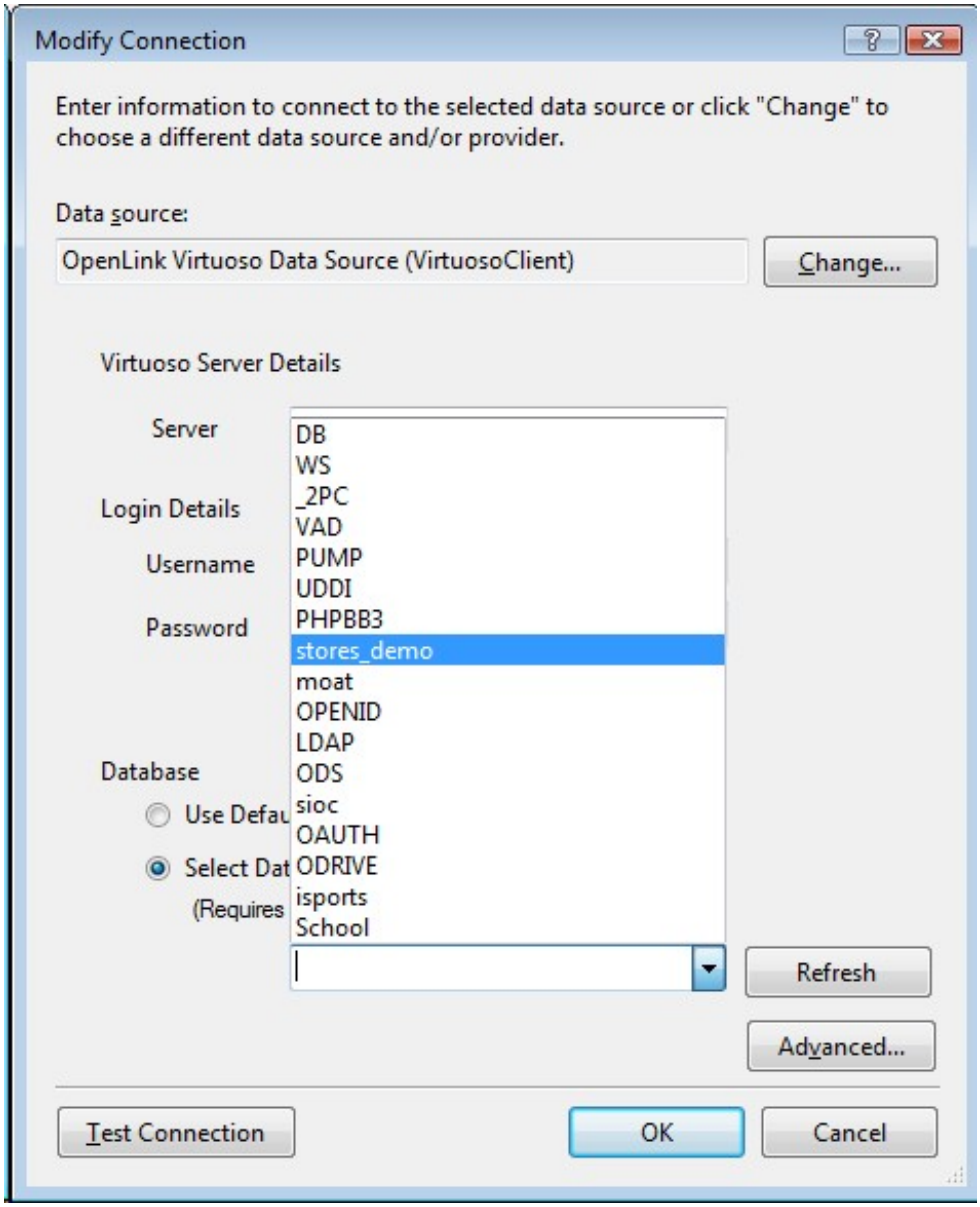


14. Select the Select Database From List radio button and choose the

    stores_demo

    from the drop down list.

Figure 8.258. Add connection

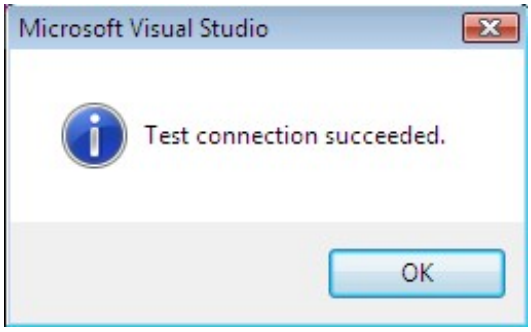


15. Press the

    Test Connection

    dialog to verify that the database is accessible.

    Figure 8.259. Test Connection



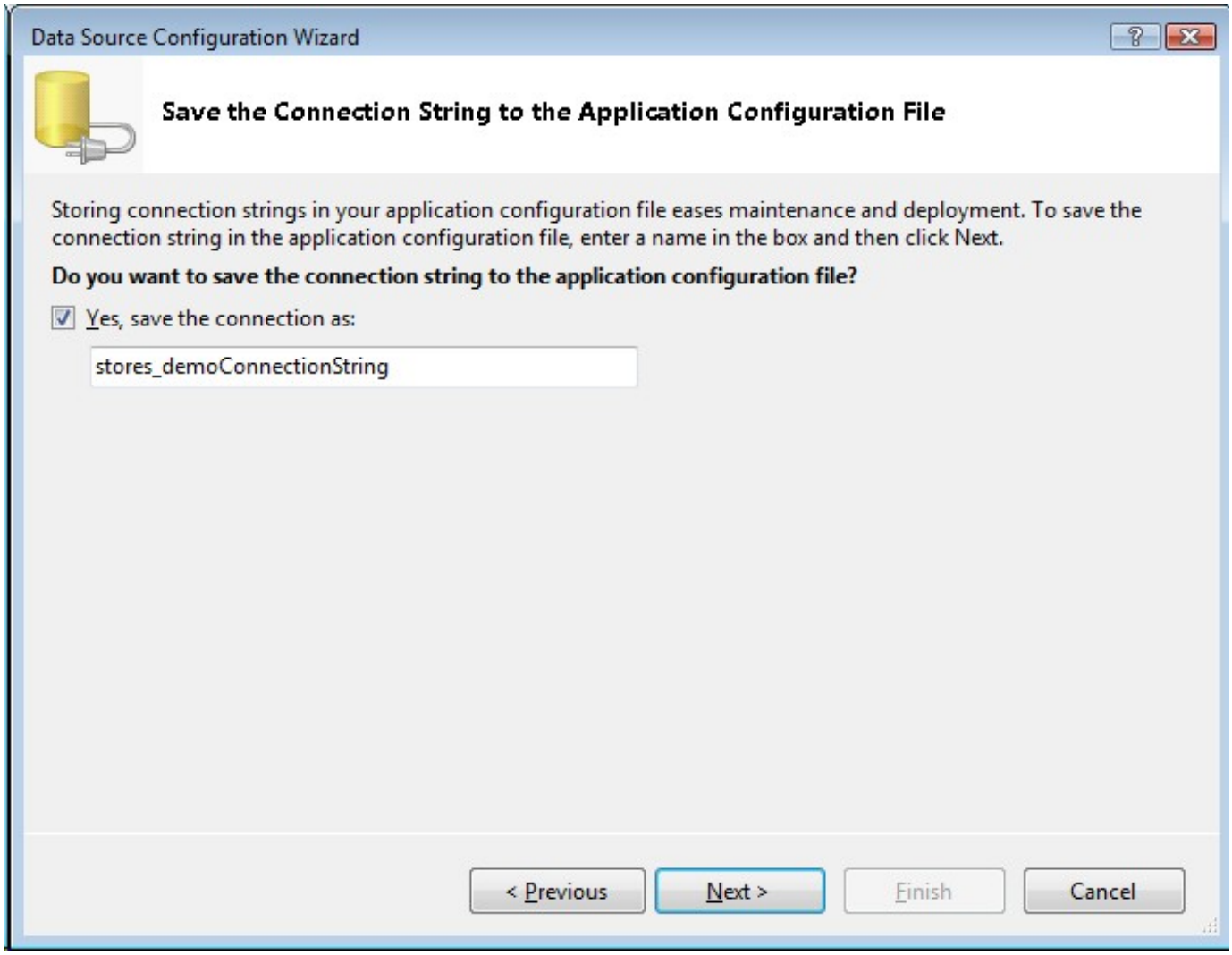
16. Leave the default connect string

    stores_demoConnectionString

and click

Next

Figure 8.260. stores_demoConnectionString



17. From the list of available tables returned for the stores_demo database, select the

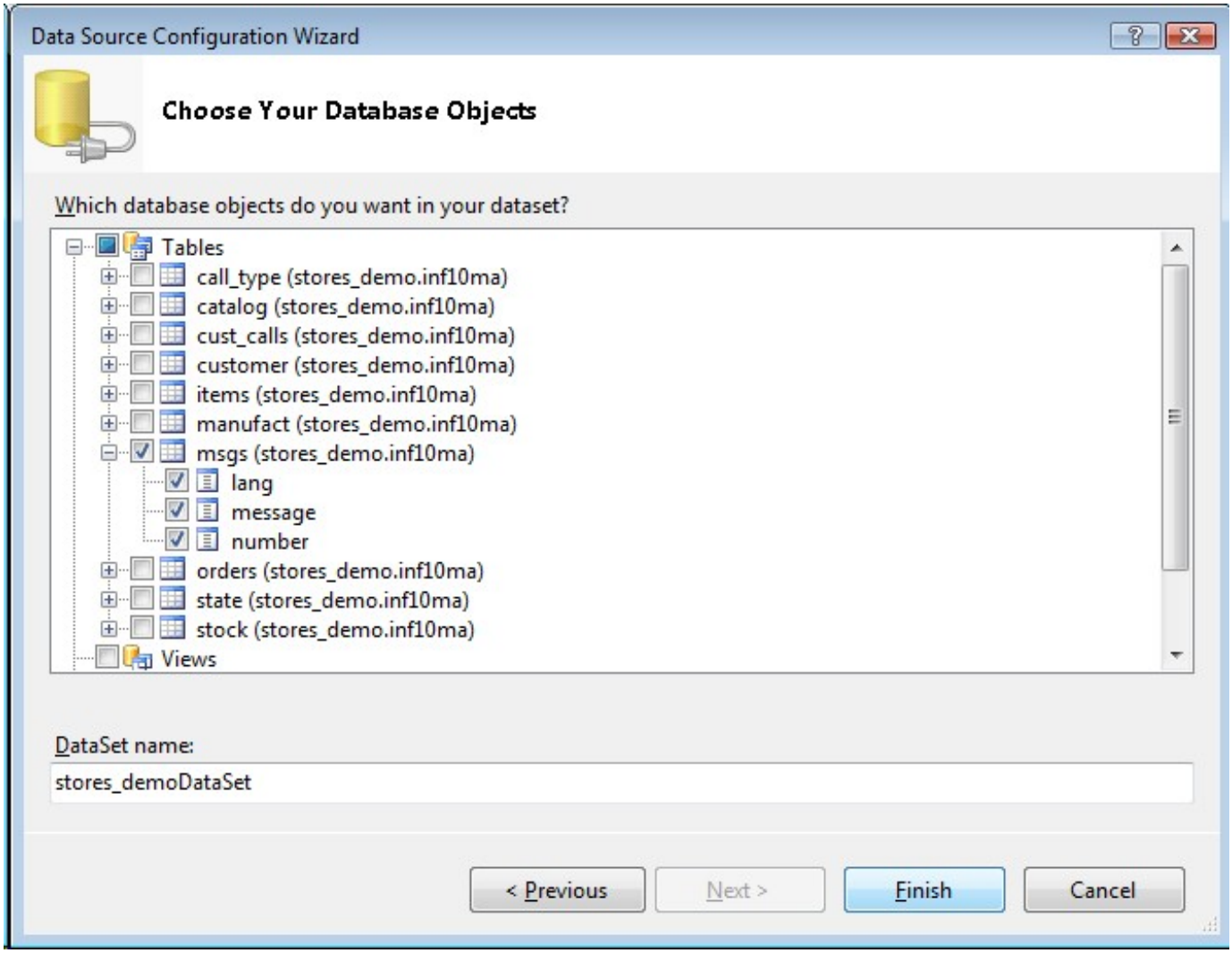
    msgs

    table to be associated with the

    DataGridView

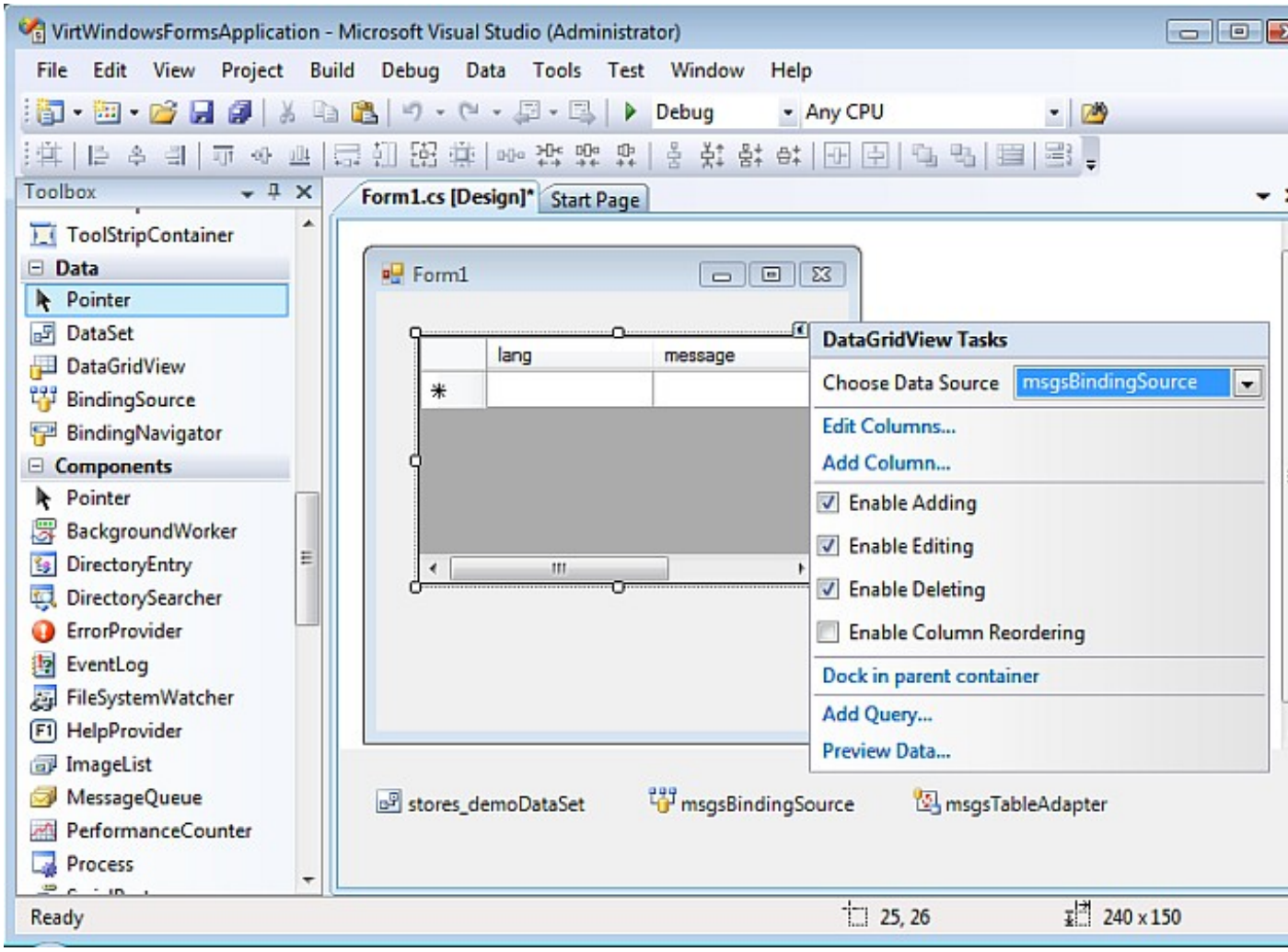
    control.

Figure 8.261. stores_demo database



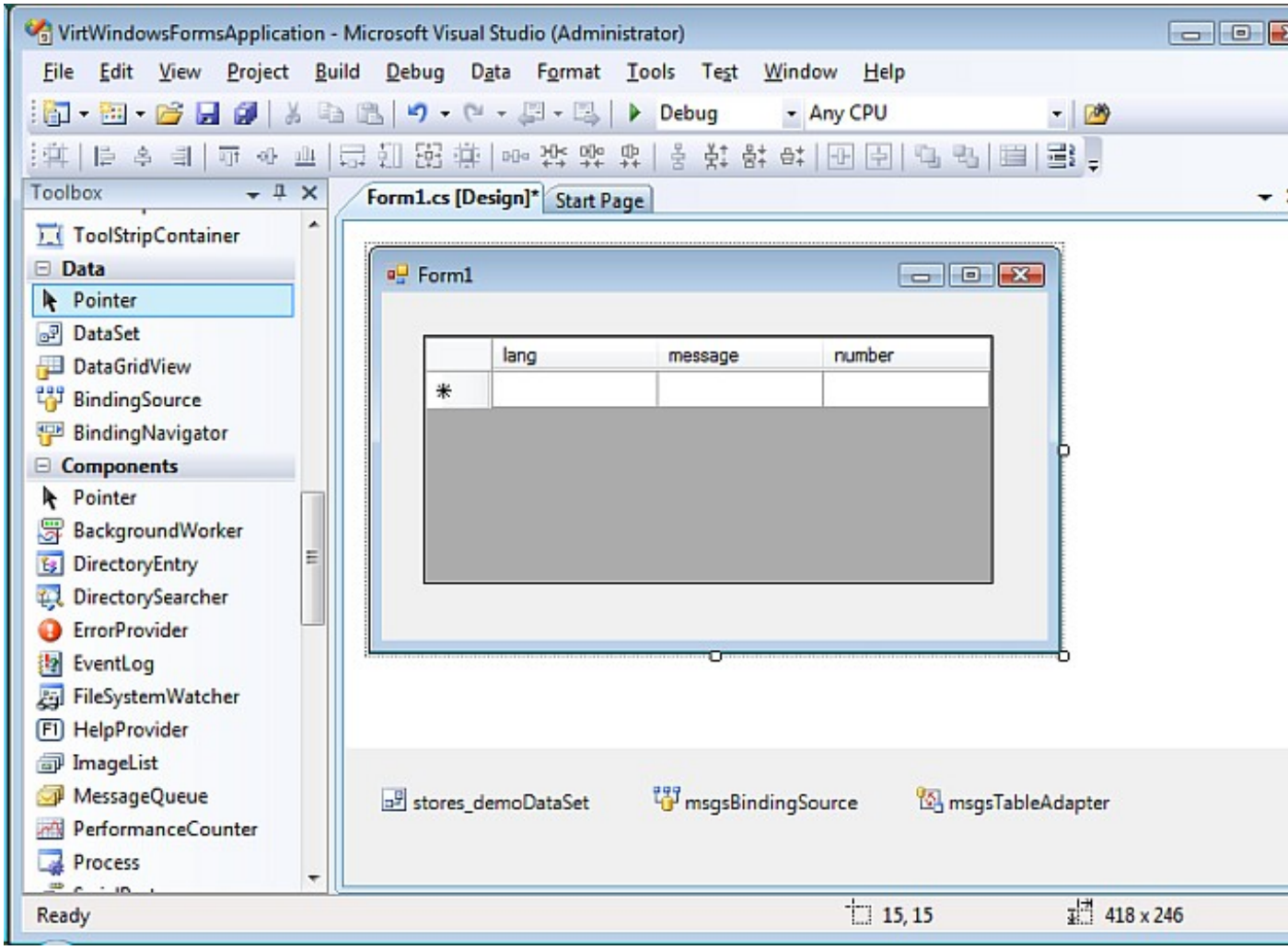
18. The columns names of the select table will be displayed in the DataGridView.

Figure 8.262. DataGridView



19. Resize the Form and DataGridView to allow all columns to be visible, if possible.

Figure 8.263. Resize the Form and DataGridView



20. To test the application, simply hit

    Ctrl+F5

    within Visual Studio or select

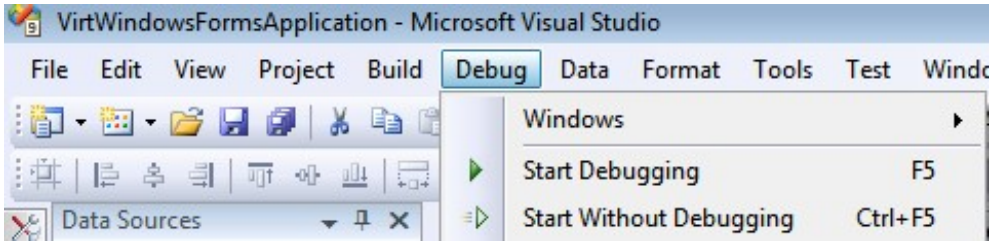
    Start Debugging

    from the

    Debug

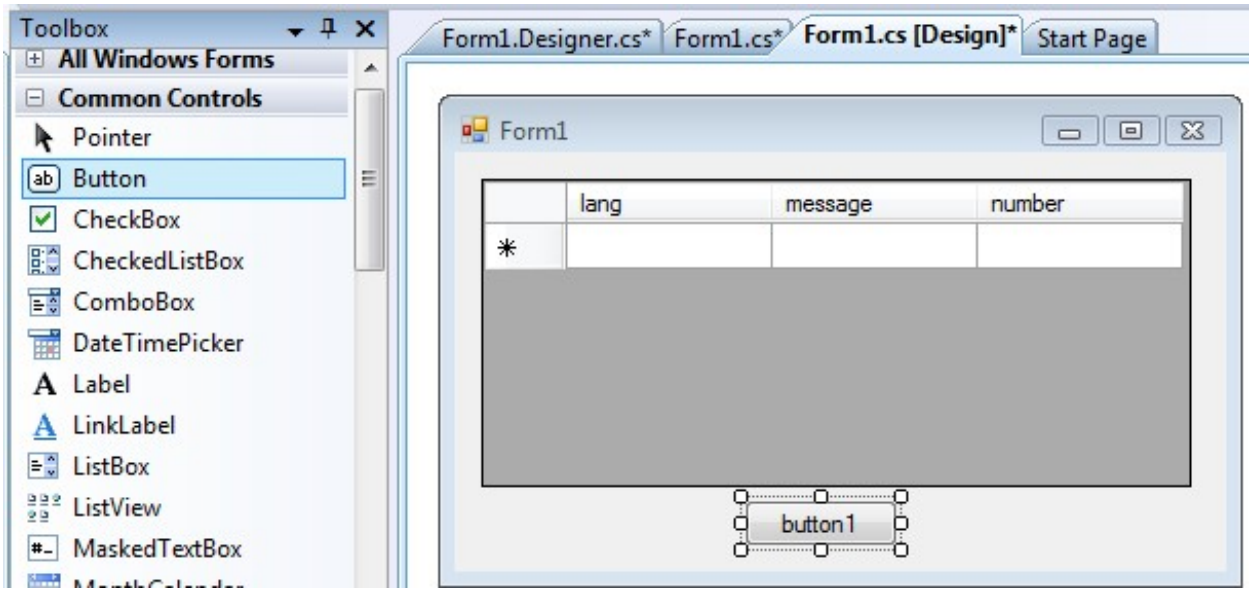
    menu.

Figure 8.264. Start Debugging



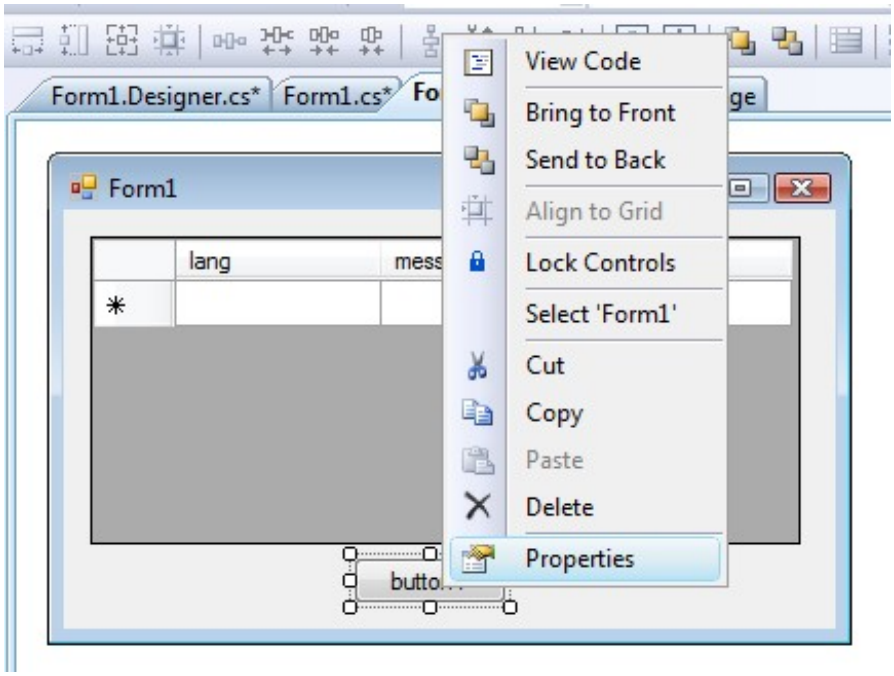
21. To make the DataGridView updateable, you will need to manually add some code to the project along with a suitable control to invoke the code. Drag a Button control onto the form.

Figure 8.265. DataGrid



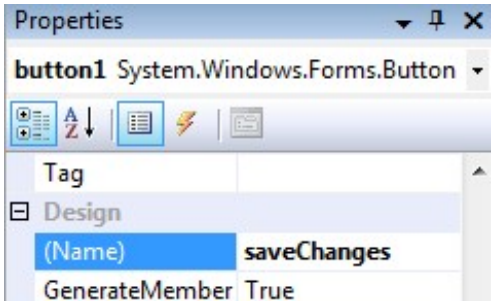
22. Right click on the Button and select Properties.

Figure 8.266. DataGrid



23. In the Properties view, edit the buttons Text property to read Save Changes and its (Name) property to read saveChanges.

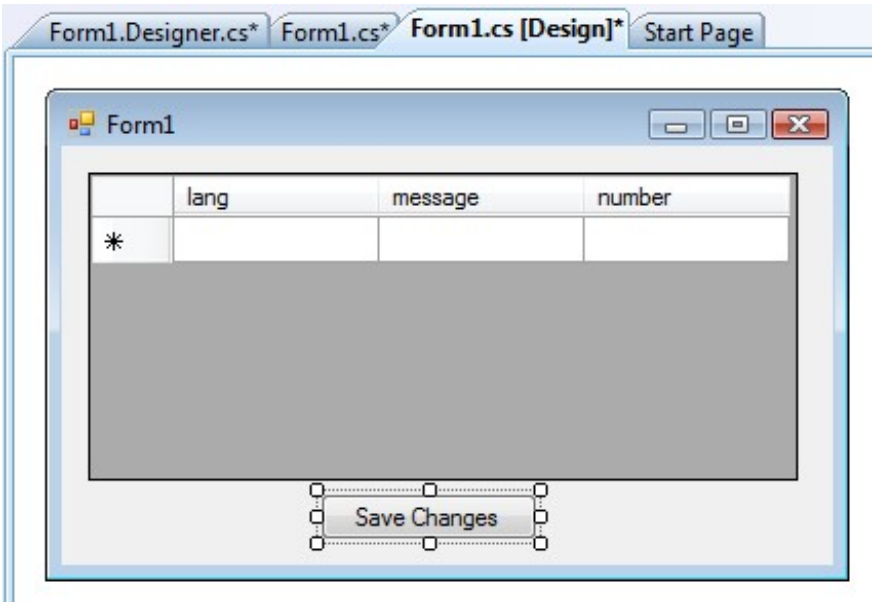
Figure 8.267. DataGrid



24. The button will now update to reflect these changes.

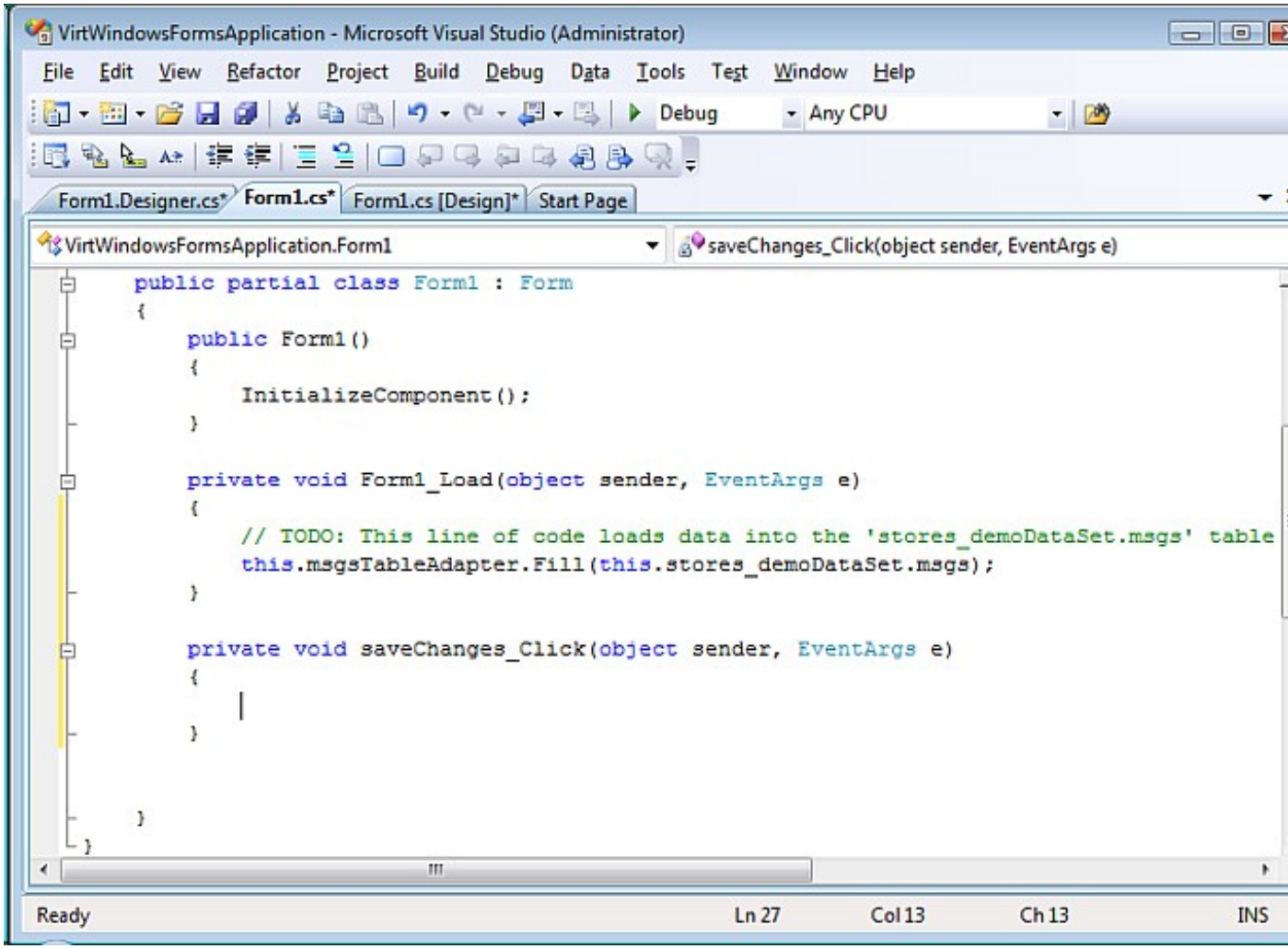
NOTE: You will need to resize the button to make the new text visible.

Figure 8.268. DataGrid



25. Double click the new button to generate the required event handler. It should take you directly to the area of code that will execute when the button is clicked.

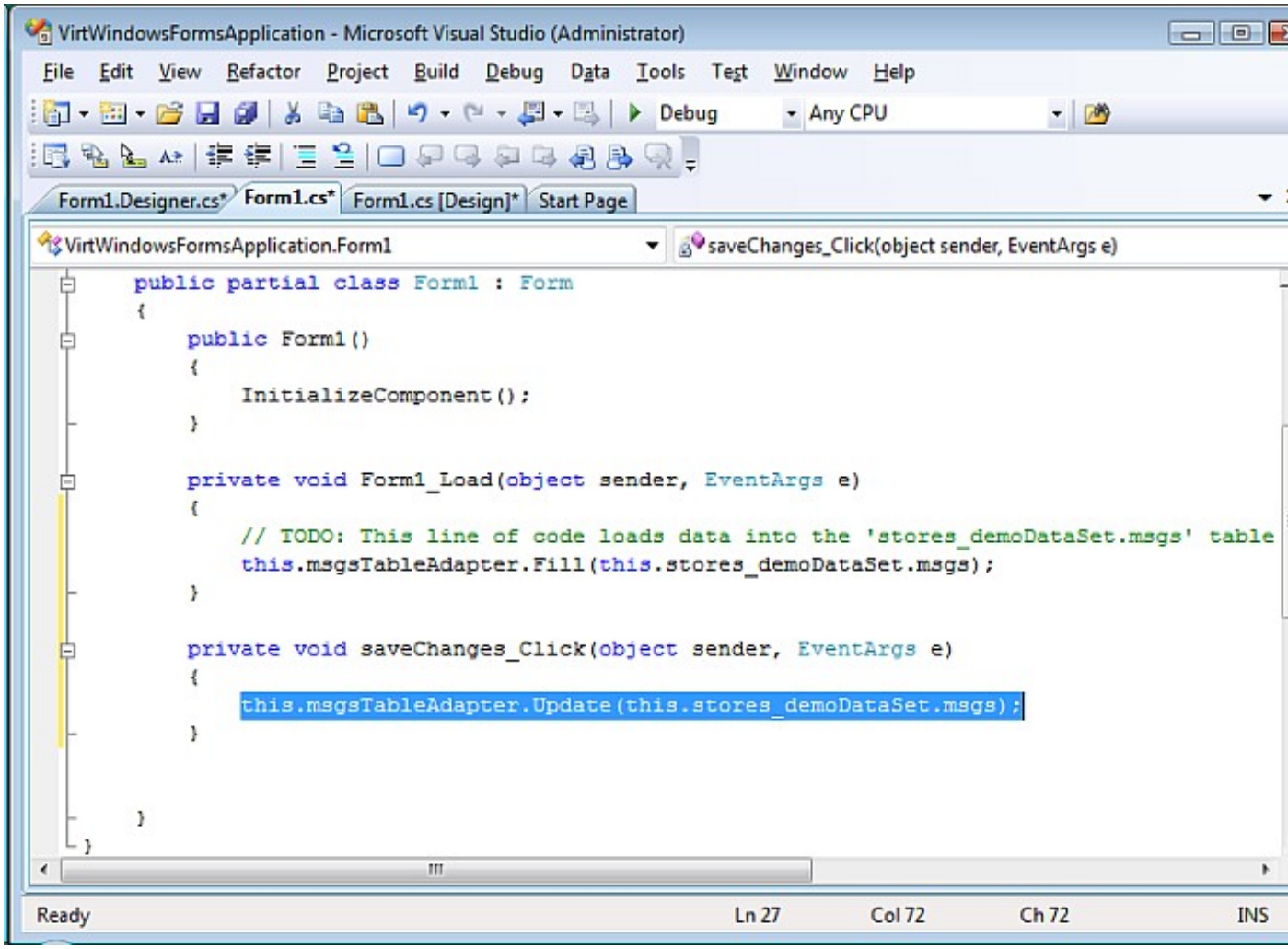
Figure 8.269. DataGrid



26. Edit the saveChanges_Click event handler code to include the following line.

```
private void saveChanges_Click(object sender, EventArgs e)
{
  this.msgsTableAdapter.Update(this.stores_demoDataSet.msgs);
}
```

Figure 8.270. DataGrid

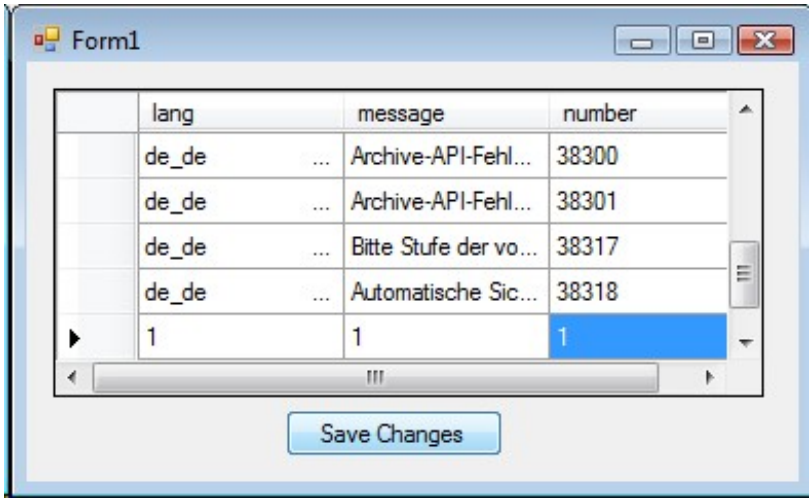


27. Now test the application again by hitting Ctrl+F5.

    Scroll to the empty row, at the bottom, and enter data for a new row. Then, select Save Changes which will write the new row back to the database.

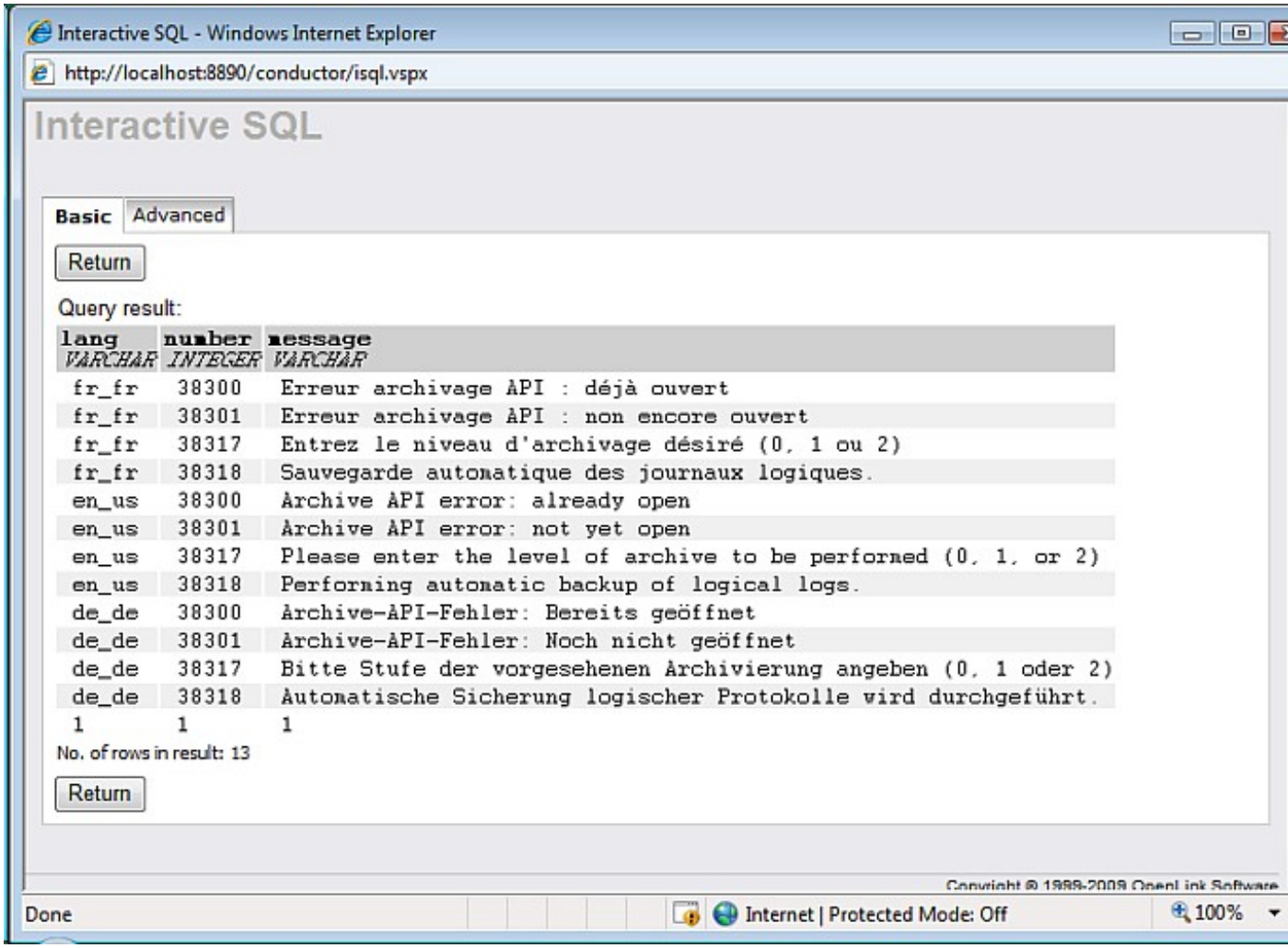
    Updates and deletes can be performed similarly.

Figure 8.271. DataGrid



28. You can use Interactive ISQL to test that the changes that have been written. Interactive ISQL Interface is detailed in the Linking Informix tables section.

Figure 8.272. DataGrid



The task is now complete.

8.6. Using Microsoft Entity Frameworks to Access DB2 Schema Objects with Virtuoso

Abstract

This section details the steps required to provide Microsoft Entity Framework access to DB2 Schema Objects using the OpenLink Virtuoso Universal Server. This is achieved by linking the required DB2 Schema objects into Virtuoso using its built in Virtual Database engine, and then using the Virtuoso ADO.NET Entity Framework provider to query the remote DB2 Schema objects linked into the Virtuoso Server.

Prerequisites. The following prerequisites must be in place for this solution to be possible.

DB2 DBMS. An DB2 DBMS hosting the required Schema Objects needs to be available. In this section the *DB2 sample* database will be used to demonstrate the process.

An DB2 DBMS hosting the required Schema Objects needs to be available. In this section the *DB2 sample* database will be used to demonstrate the process.

ODBC Driver for DB2. An DB2 ODBC Driver is required for Linking the DB2 Schema Objects into the Virtuoso Server. The OpenLink *ODBC Driver for DB2* will be used in this section, for which a functional ODBC Data source name of "db2v8ma" will be assumed to exist on the machine hosting the Virtuoso Server.

An DB2 ODBC Driver is required for Linking the DB2 Schema Objects into the Virtuoso Server. The OpenLink *ODBC Driver for DB2* will be used in this section, for which a functional ODBC Data source name of "db2v8ma" will be assumed to exist on the machine hosting the Virtuoso Server.

Virtuoso Universal Server. An Virtuoso installation including the Virtuoso Universal Server and ADO.NET Entity Framework Provider is required. The Virtuoso components used must be Release 5.10.x or above, this being the minimum version containing support for Microsoft Entity Frameworks

An Virtuoso installation including the Virtuoso Universal Server and ADO.NET Entity Framework Provider is required. The Virtuoso components used must be Release 5.10.x or above, this being the minimum version containing support for Microsoft Entity Frameworks

Microsoft Visual Studio 2008 SP1. Microsoft Visual Studio 2008 Service Pack 1 is required, this being the only version containing the necessary Entity Framework support available at the time of writing.

Microsoft Visual Studio 2008 Service Pack 1 is required, this being the only version containing the necessary Entity Framework support available at the time of writing.

Tasks

Ensure DB2 Primary Keys (PKs) are not nullable. The Visual Studio 2008 Entity Data Model (EDM) requires that all primary keys are *NOT* Nullable, and will fail to generate an EDM if any are. Thus ensure any tables to be used are defined as not nullable in the DB2 database schema before attempting to generate an EDM. In the case of the DB2 sample database all tables are not nullable, thus this should not be an issue in this case.

The Visual Studio 2008 Entity Data Model (EDM) requires that all primary keys are *NOT* Nullable, and will fail to generate an EDM if any are. Thus ensure any tables to be used are defined as not nullable in the DB2 database schema before attempting to generate an EDM. In the case of the DB2 sample database all tables are not nullable, thus this should not be an issue in this case.

8.6.1. Install and configure OpenLink ODBC Driver for DB2

The Virtuoso Virtual Database engine uses ODBC as the connectivity mechanism for linking remote database objects into its local schema. Thus an DB2 ODBC Driver must be available with a suitably configured DSN for connecting to the target database. The OpenLink ODBC Drivers for DB2 have been used in this section, although in theory any DB2 ODBC Driver can be used.

Installation instructions for the OpenLink ODBC Driver for DB2 are available from:

- Product Installation & Basic Configuration (ODBC)

8.6.2. Install and configure OpenLink Virtuoso Universal Server

- Install and configure OpenLink Virtuoso Universal Server

8.6.3. Linking DB2 tables into OpenLink Virtuoso

1. Start the Virtuoso Web User Interface

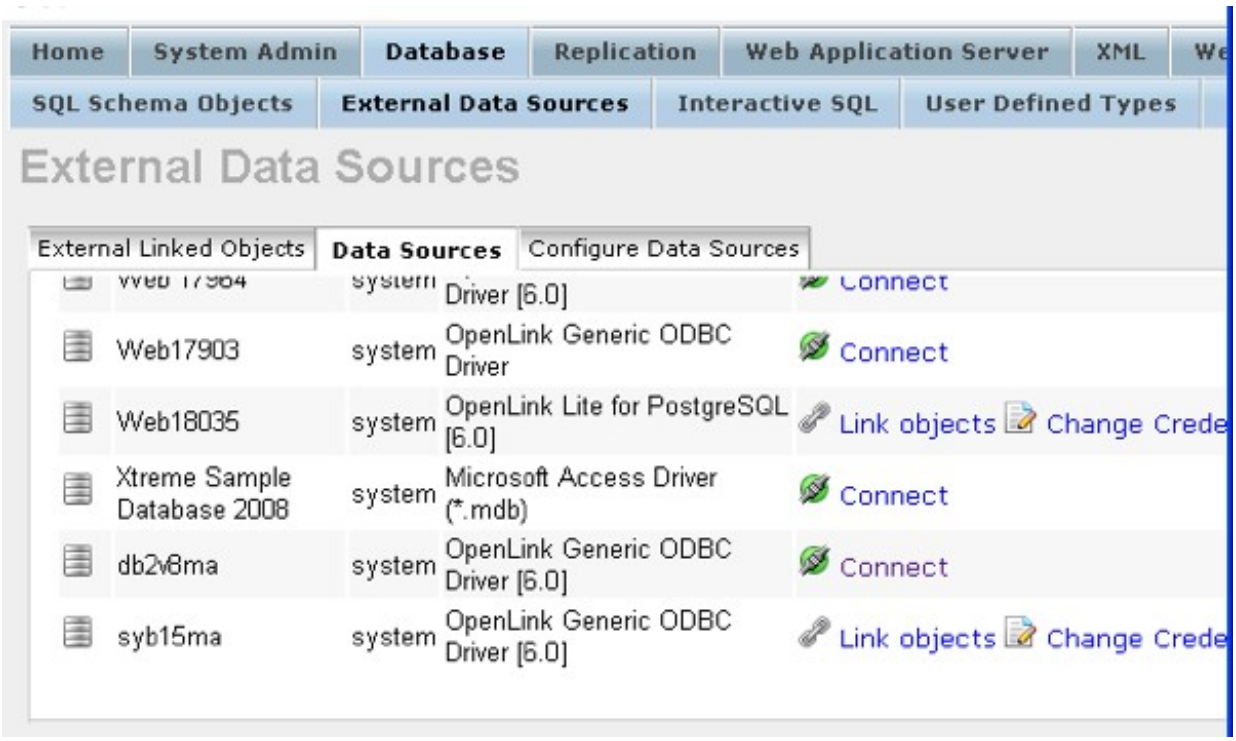
Figure 8.273. Start

2. Select the "Conductor" link in the Left Frame and Login as the "dba" user.

Figure 8.274. Conductor

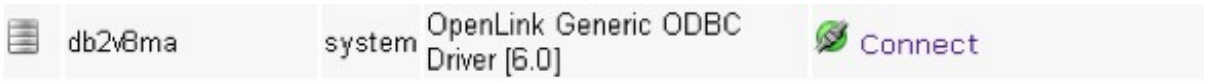
3. Select the "Databases" -> "External Data Source" -> "Data Sources" tab menu items

Figure 8.275. Databases



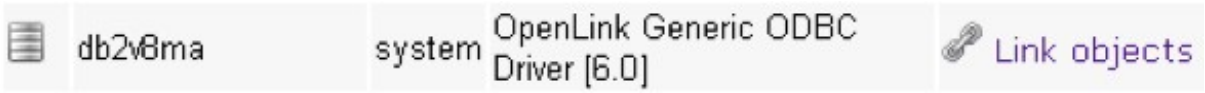
4. Select the "Connect" button for the "db2v8ma" DB2 DSN.

Figure 8.276. Connect



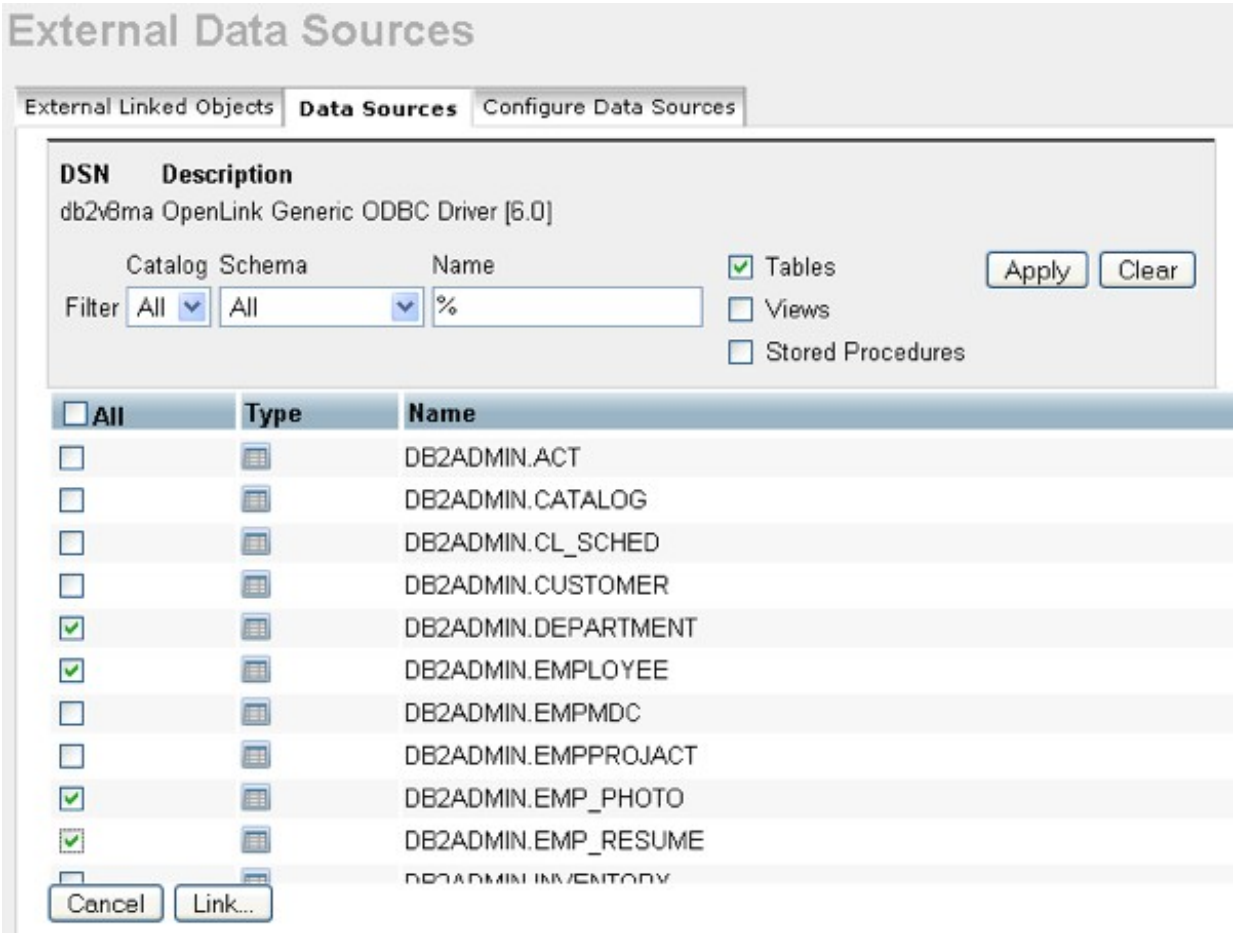
5. On successful connection Select the "Link Objects" button to obtain a list of available tables

Figure 8.277. Link Objects



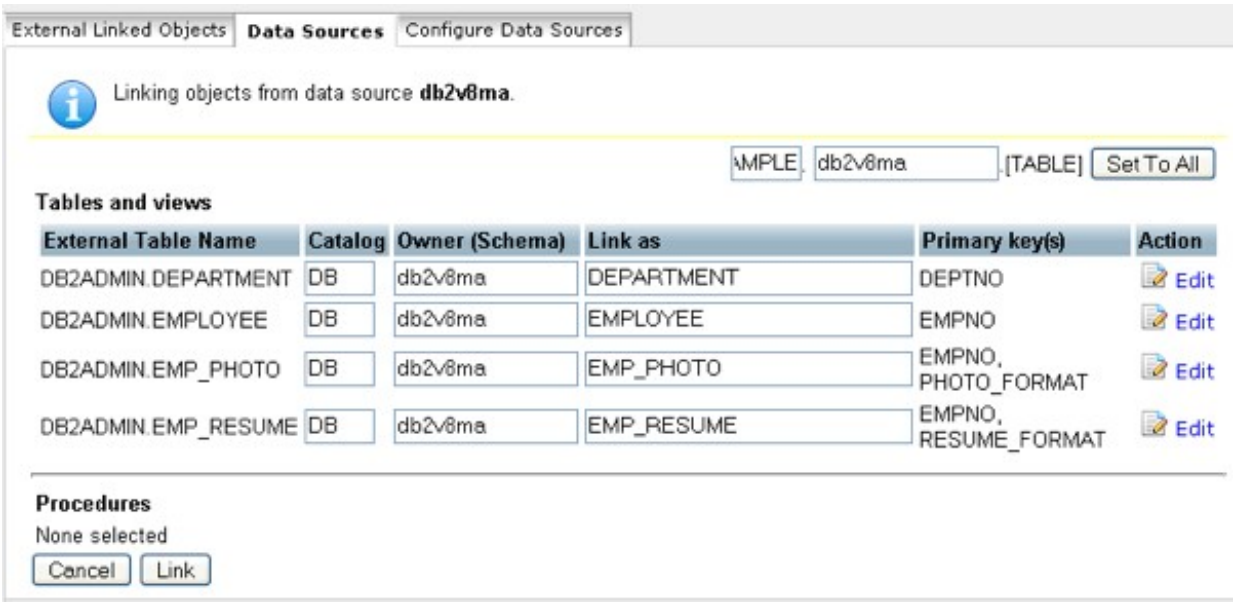
6. Select the tables that you would like to link.

Figure 8.278. Select tables



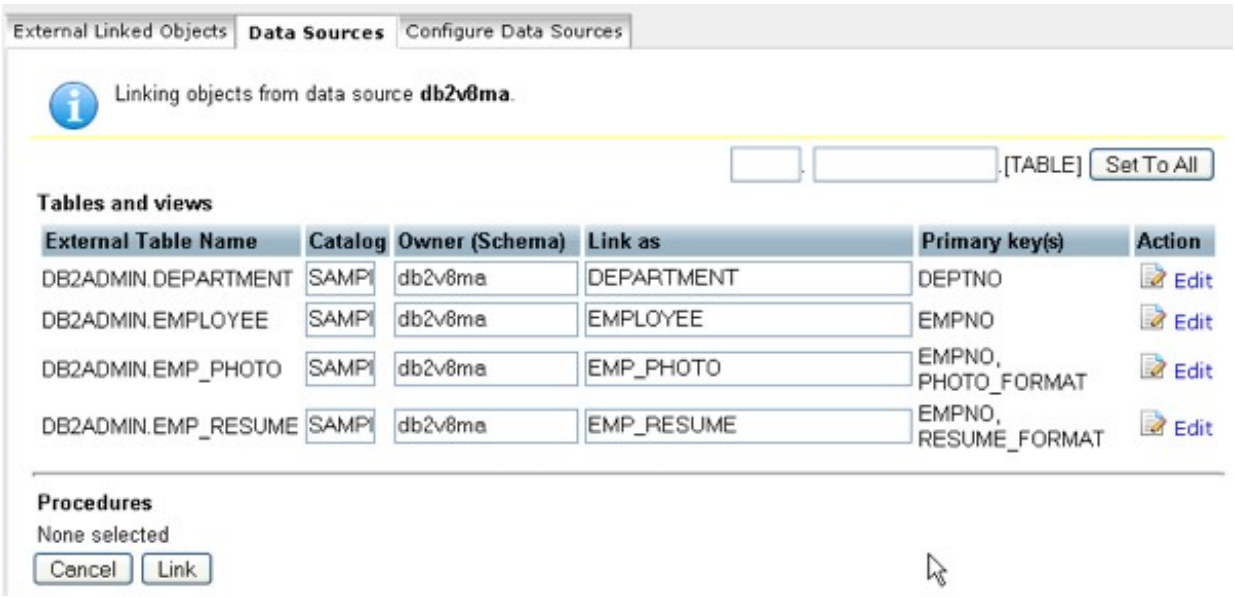
7. Change the Catalog for each table to be "SAMPLE" using the "Set All" button.

Figure 8.279. Catalog



8. All the catalog names are changed to be "SAMPLE".

Figure 8.280. catalog names

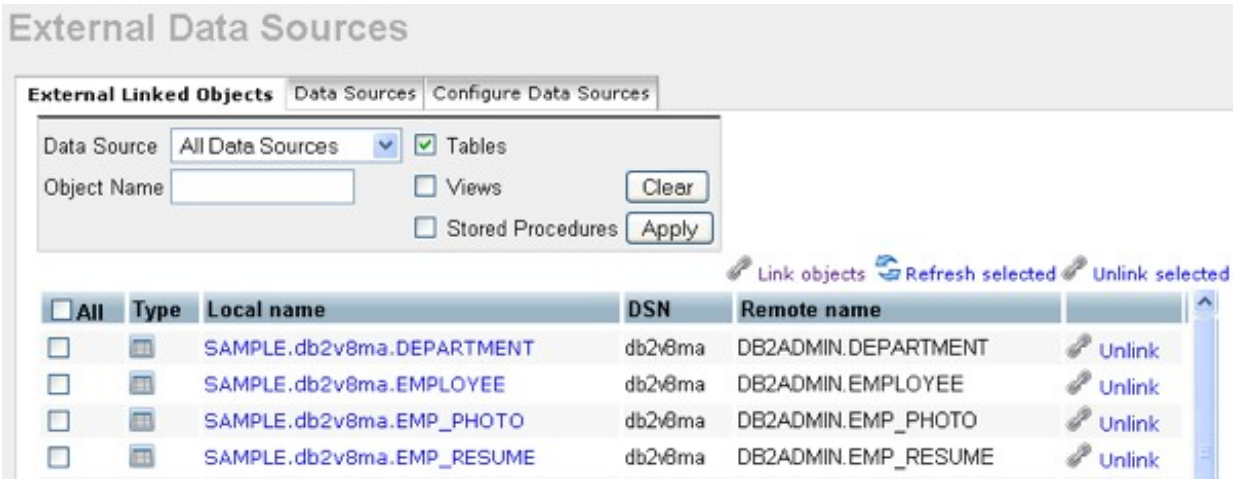


9. Select the "Link" button to link the selected tables into Virtuoso

Figure 8.281. "Link" button

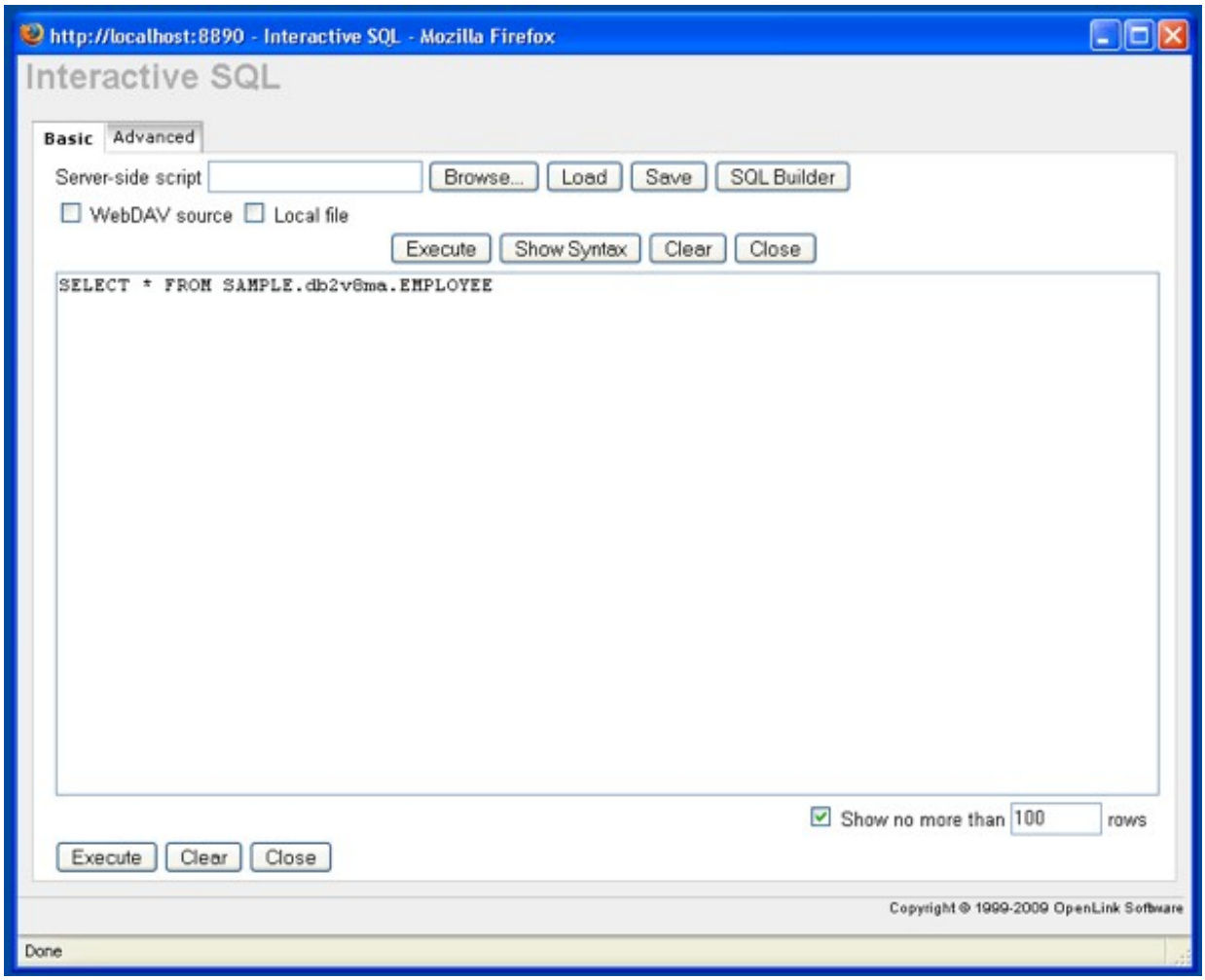
10. On completion of the Link process the tables will be displayed in the "External Linked Objects" tab.

Figure 8.282. Completion



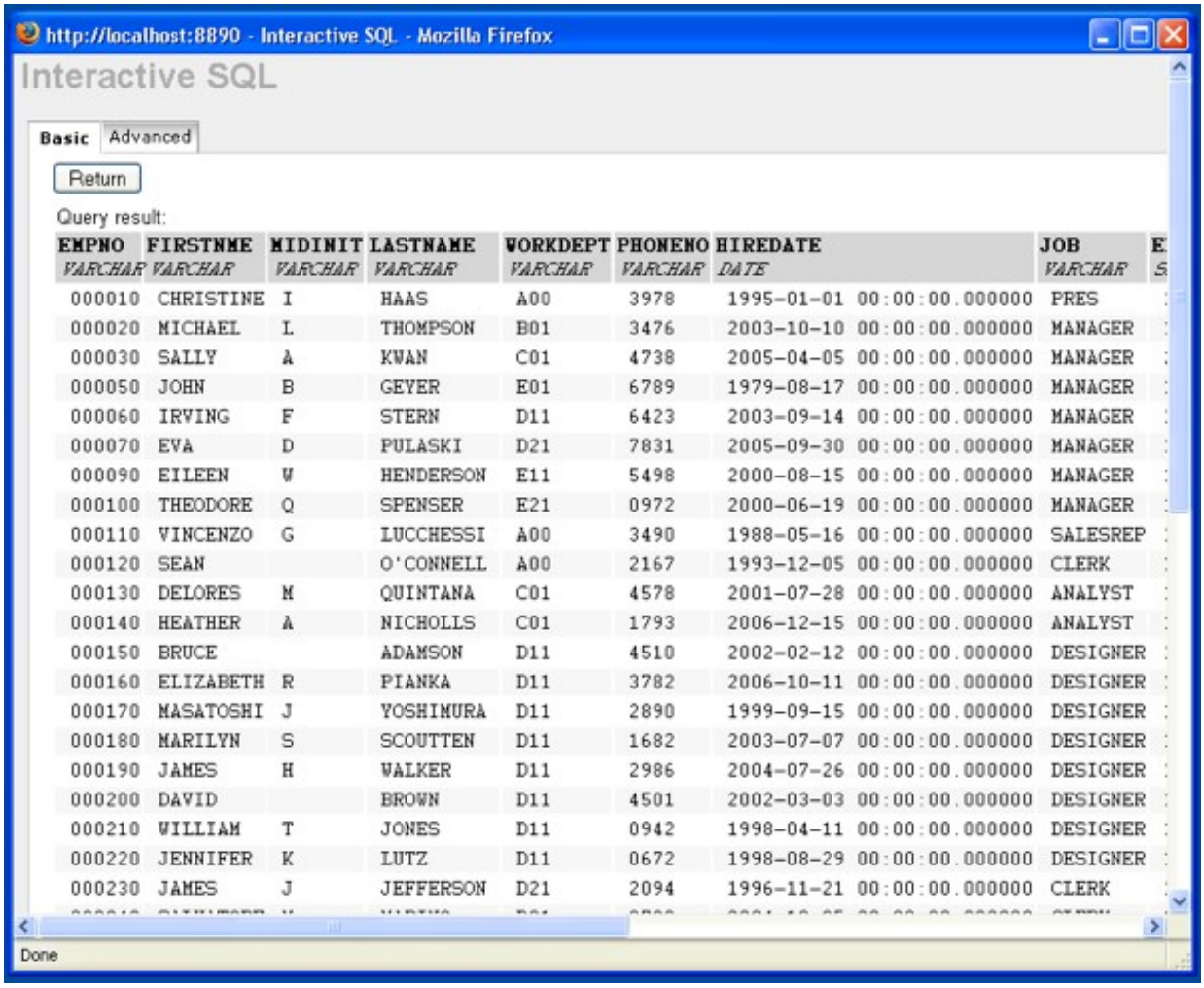
11. The linked tables can be queried by clicking on the hyperlink in the Local Name column of the External Linked Objects tab above. This loads the Virtuoso Interactive SQL interface with the required SQL Select for retrieving the remote table data . We shall use the SAMPLE.db2v8ma.EMPLOYEE table to demonstrate this.

Figure 8.283. view tables



12. Click the Execute button to run the query and retrieve the results from the remote table.

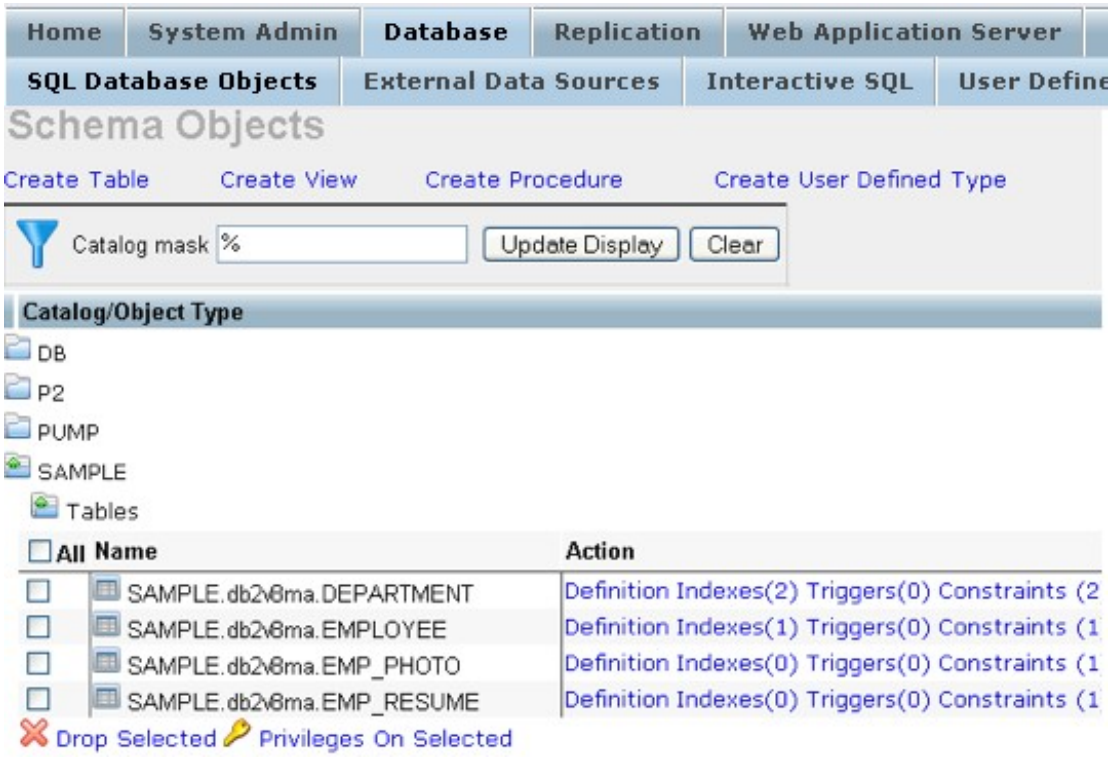
Figure 8.284. Execute



EMPNO	FIRSTNME	MIDINIT	LASTNAME	WORKDEPT	PHONENO	HIREDATE	JOB
VARCHAR	VARCHAR	VARCHAR	VARCHAR	VARCHAR	VARCHAR	DATE	VARCHAR
000010	CHRISTINE	I	HAAS	A00	3978	1995-01-01 00:00:00.000000	PRES
000020	MICHAEL	L	THOMPSON	B01	3476	2003-10-10 00:00:00.000000	MANAGER
000030	SALLY	A	KWAN	C01	4738	2005-04-05 00:00:00.000000	MANAGER
000050	JOHN	B	GEYER	E01	6789	1979-08-17 00:00:00.000000	MANAGER
000060	IRVING	F	STERN	D11	6423	2003-09-14 00:00:00.000000	MANAGER
000070	EVA	D	PULASKI	D21	7831	2005-09-30 00:00:00.000000	MANAGER
000090	EILEEN	W	HENDERSON	E11	5498	2000-08-15 00:00:00.000000	MANAGER
000100	THEODORE	Q	SPENSER	E21	0972	2000-06-19 00:00:00.000000	MANAGER
000110	VINCENZO	G	LUCCHESSI	A00	3490	1988-05-16 00:00:00.000000	SALESREP
000120	SEAN		O'CONNELL	A00	2167	1993-12-05 00:00:00.000000	CLERK
000130	DELORES	M	QUINTANA	C01	4578	2001-07-28 00:00:00.000000	ANALYST
000140	HEATHER	A	NICHOLLS	C01	1793	2006-12-15 00:00:00.000000	ANALYST
000150	BRUCE		ADAMSON	D11	4510	2002-02-12 00:00:00.000000	DESIGNER
000160	ELIZABETH	R	PIANKA	D11	3782	2006-10-11 00:00:00.000000	DESIGNER
000170	MASATOSHI	J	YOSHIMURA	D11	2890	1999-09-15 00:00:00.000000	DESIGNER
000180	MARILYN	S	SCOUTTEN	D11	1682	2003-07-07 00:00:00.000000	DESIGNER
000190	JAMES	H	WALKER	D11	2986	2004-07-26 00:00:00.000000	DESIGNER
000200	DAVID		BROWN	D11	4501	2002-03-03 00:00:00.000000	DESIGNER
000210	WILLIAM	T	JONES	D11	0942	1998-04-11 00:00:00.000000	DESIGNER
000220	JENNIFER	K	LUTZ	D11	0672	1998-08-29 00:00:00.000000	DESIGNER
000230	JAMES	J	JEFFERSON	D21	2094	1996-11-21 00:00:00.000000	CLERK

13. The tables can also be viewed as part of the Virtuoso SQL Schema Objects under the SAMPLE catalog name.

Figure 8.285. SQL Schema Objects



The Link process is now complete enabling the tables to be queried as if part of the Virtuoso Schema.

8.6.4. Creating EDM in Visual Studio 2008

The following steps can be used to create an Entity Data Model (EDM) for the DB2 Samples database:

1. Launch the Visual Studio 2008 SP1 IDE.

   Figure 8.286. Visual Studio 2008 SP1 IDE

2. Create a

   Web Application

   project by going to the

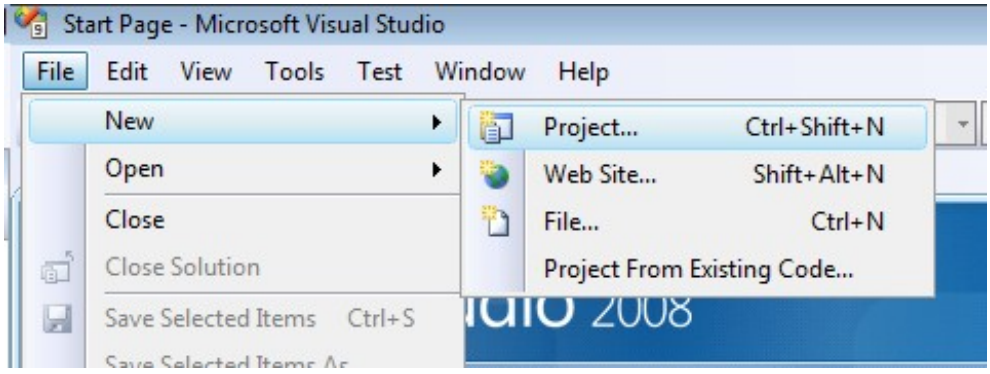
   File

   menu in Visual Studio and choosing

   New Project

   .

   Figure 8.287. Web Application

3. When the New Project window appears, choose either

   Visual Basic

   or

Visual C#

as your programming language.

4. Within the language category, click on

   Web

   and select

   ASP.NET Web Application

   from the right-hand panel.
5. Choose a name for the project, for example

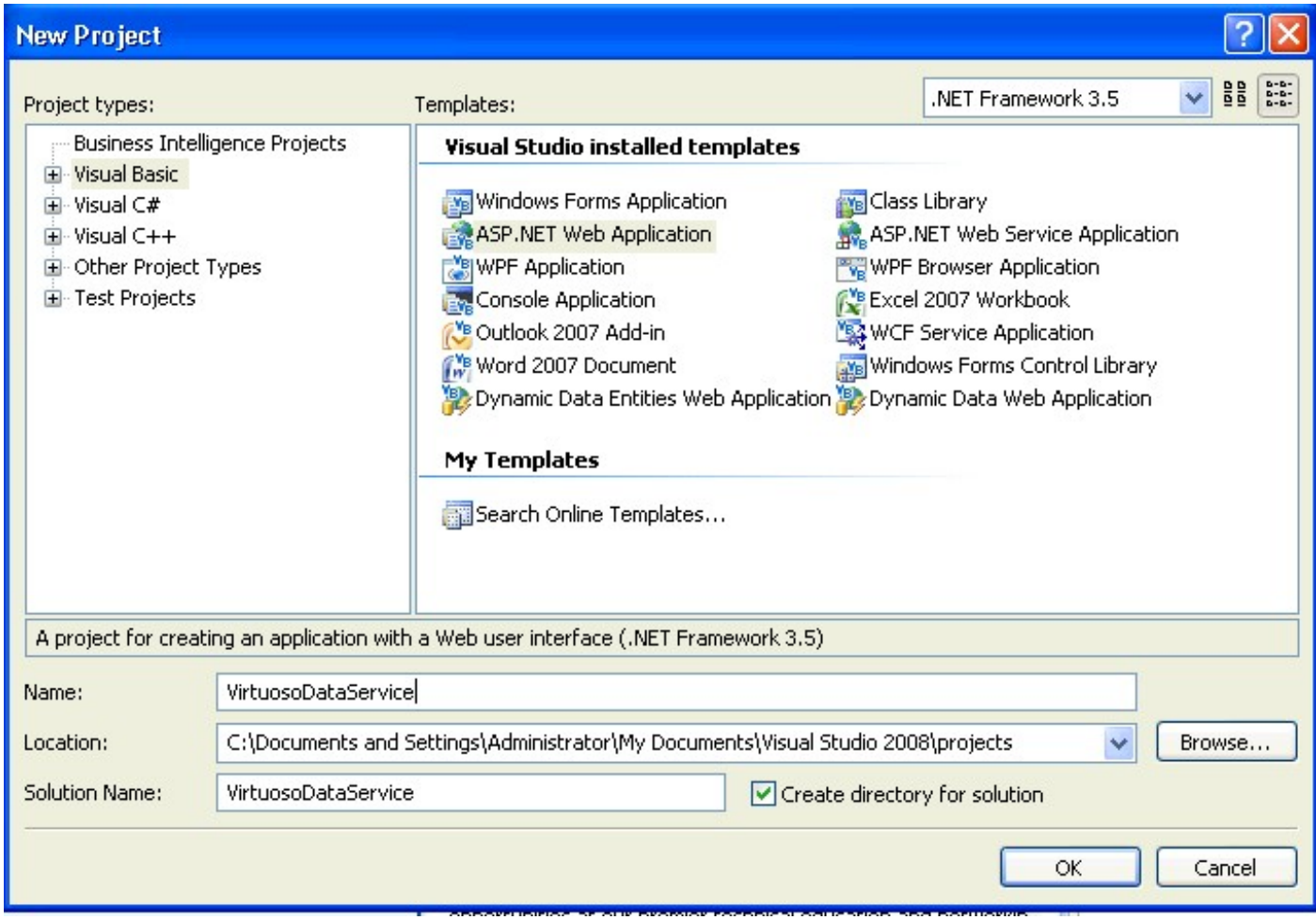
   VirtuosoDataService

   , and click

   OK

   .

   Figure 8.288. name for the project

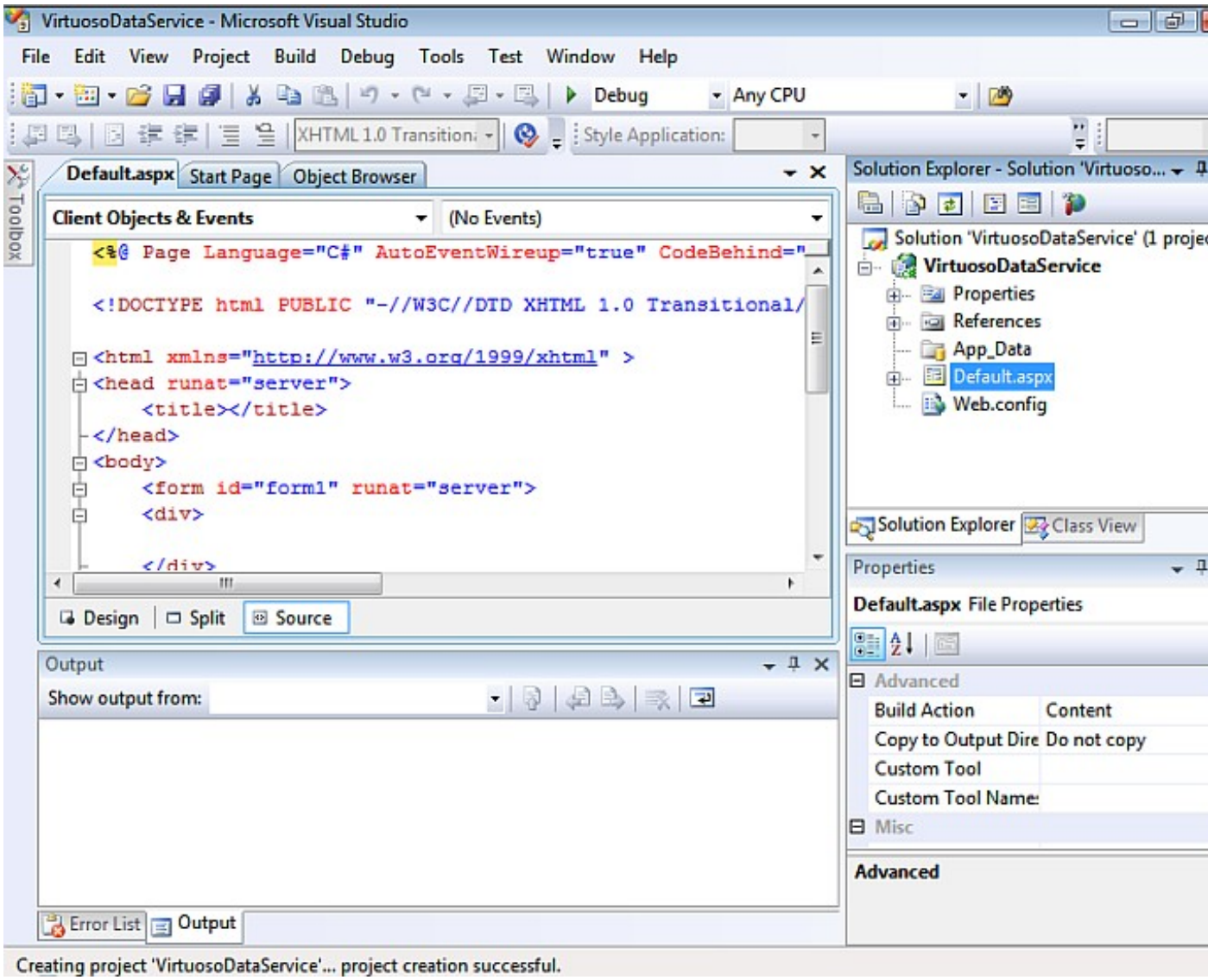


6. This will create a new project called

   VirtuosoDataService

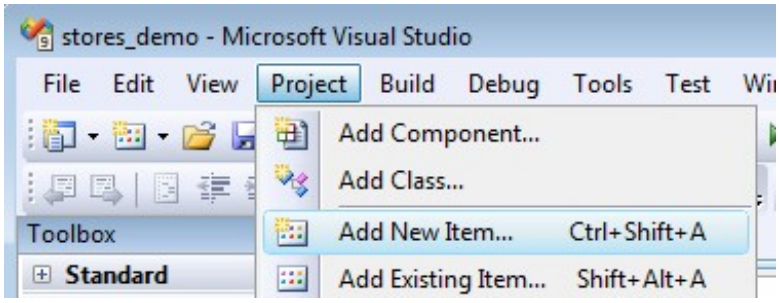
   .

Figure 8.289. create a new project



7. Select the Project -> Add New Item menu option.

Figure 8.290. VirtuosoDataService



8. The

Add New Item

dialog will appear. Choose the

ADO.NET Entity Data Model

template. Give it the name

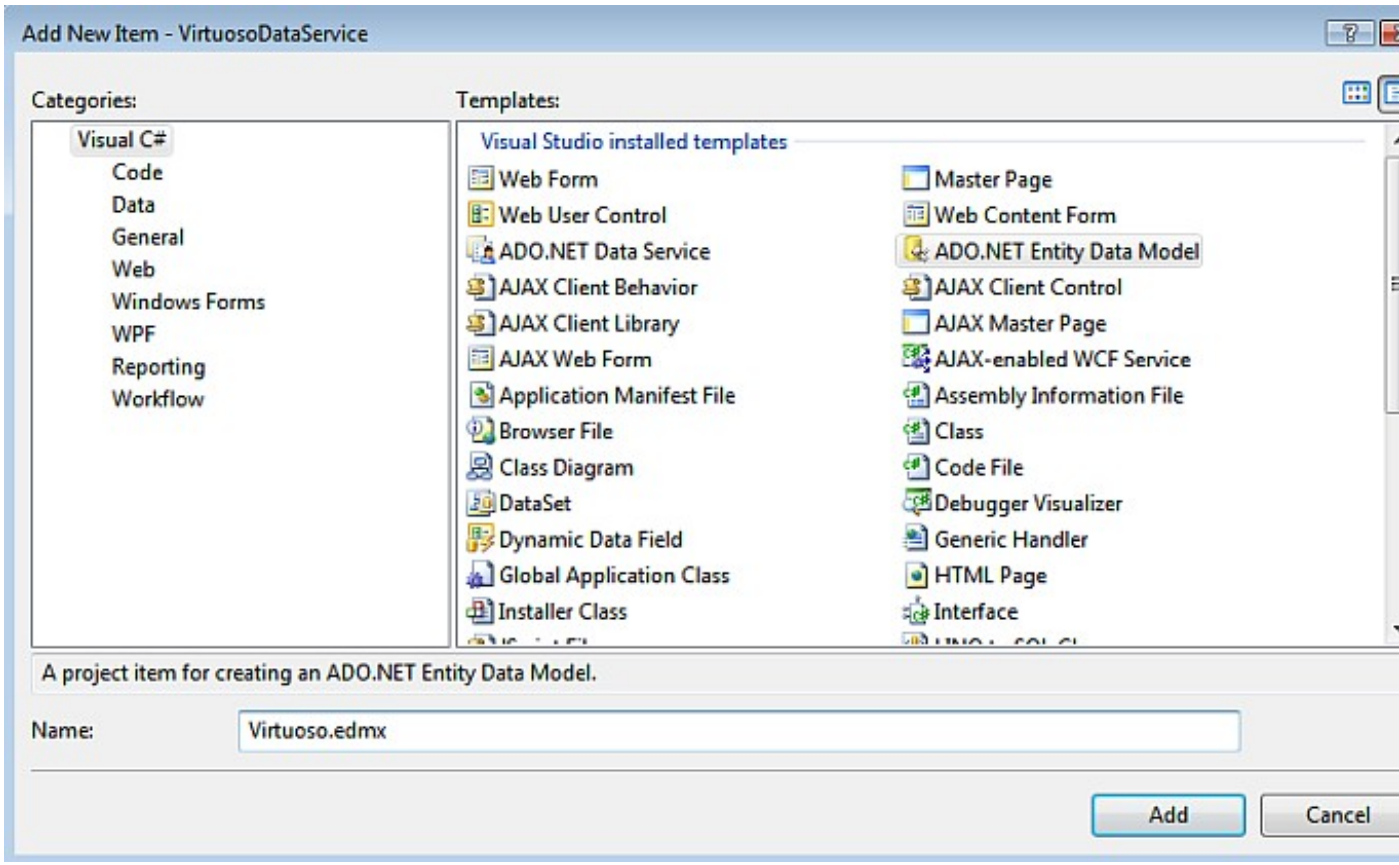
Virtuoso.edmx

and click

Add

to start the creation of the ADO.Net Entity Data Model.

Figure 8.291. Add New Item



9. In the

Entity Data Model Wizard

dialog

Choose Model Contents

page select the

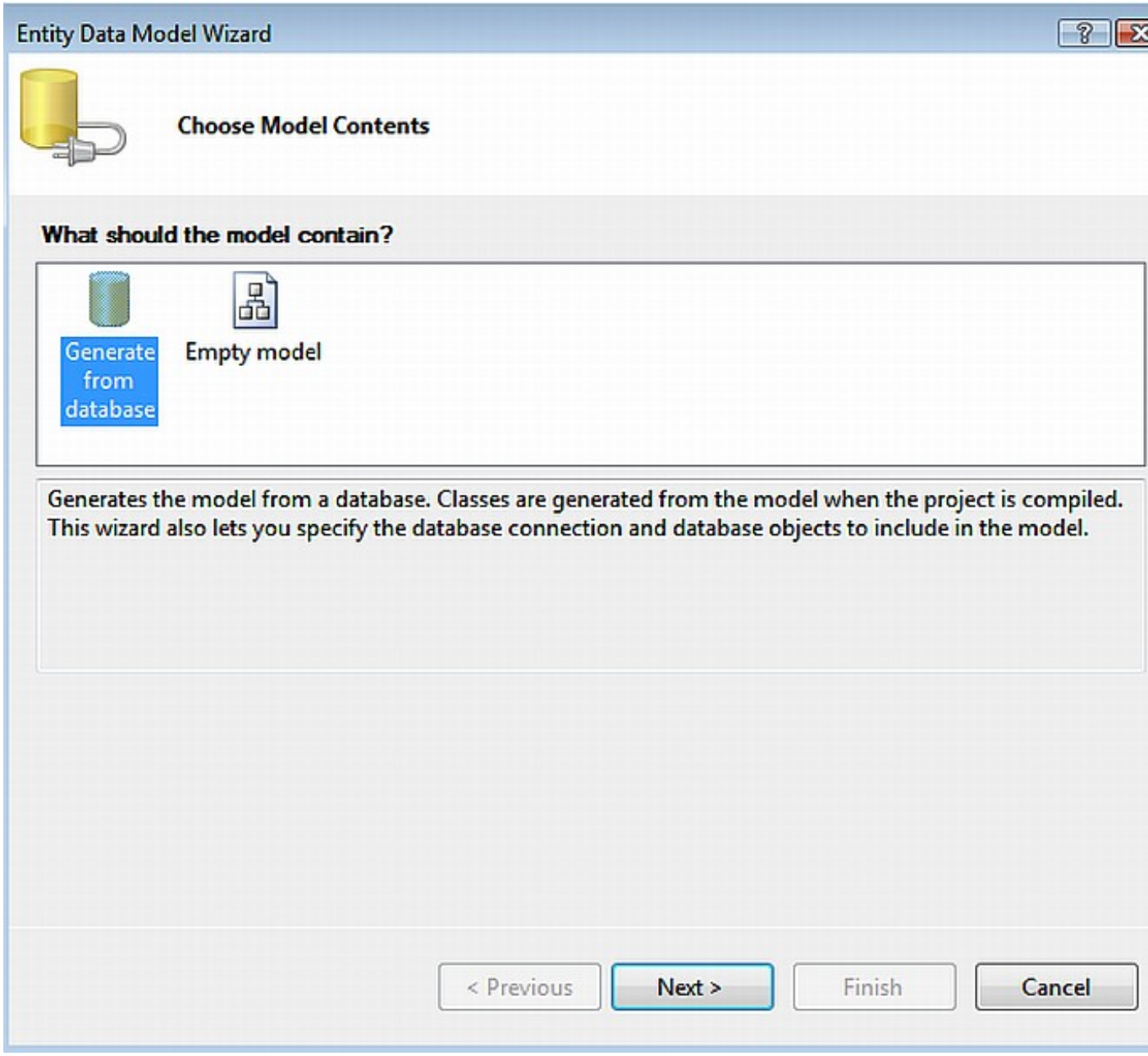
Generate from Database

model type and click

Next

.

Figure 8.292. Choose Model Contents



10. In the

    Entity Data Model Wizard

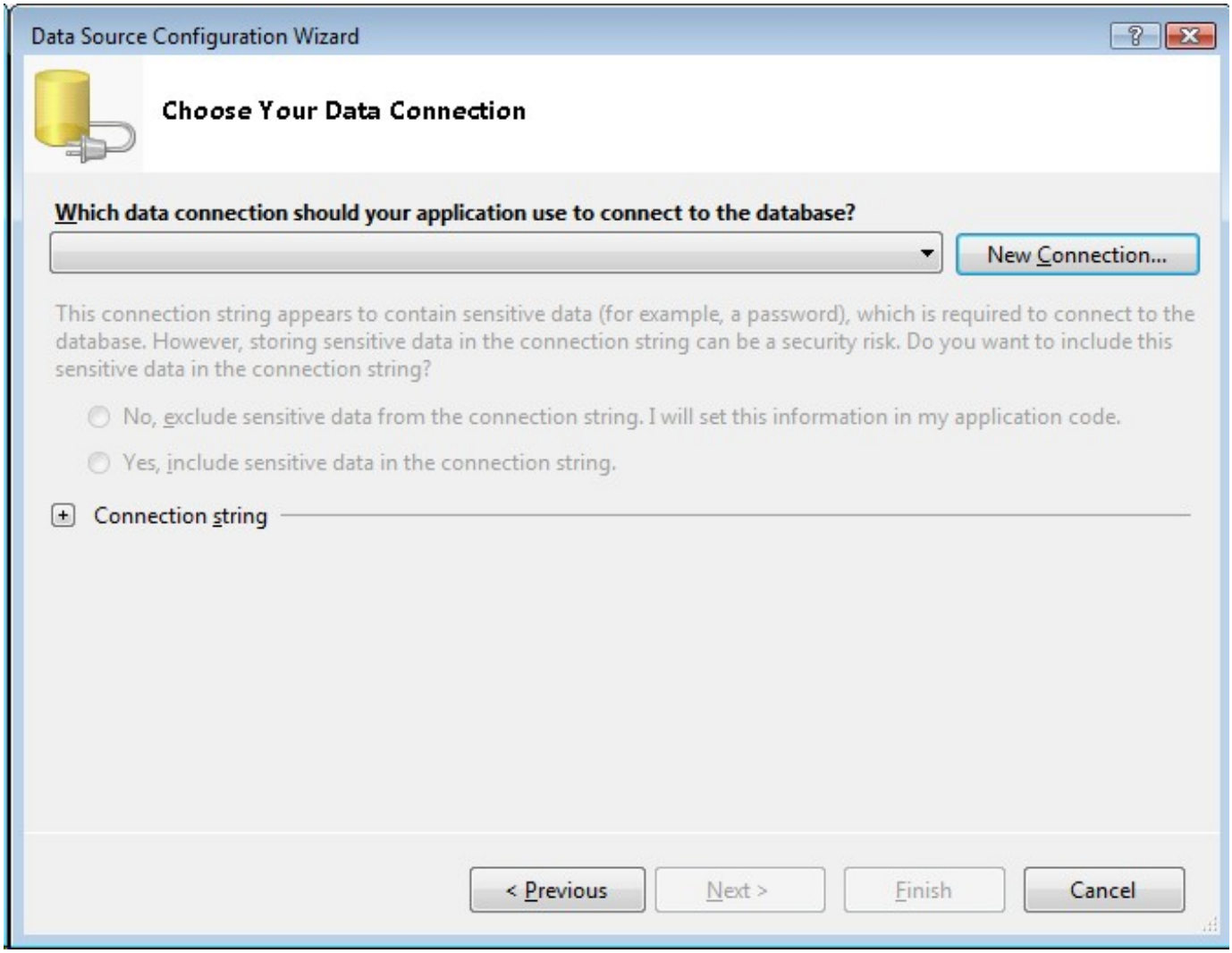
    dialog

    Choose your Data Connection

    page, select the

    New Connection button

Figure 8.293. Entity Data Model Wizard



11. In the

    Choose Data Source

    dialog, select the OpenLink

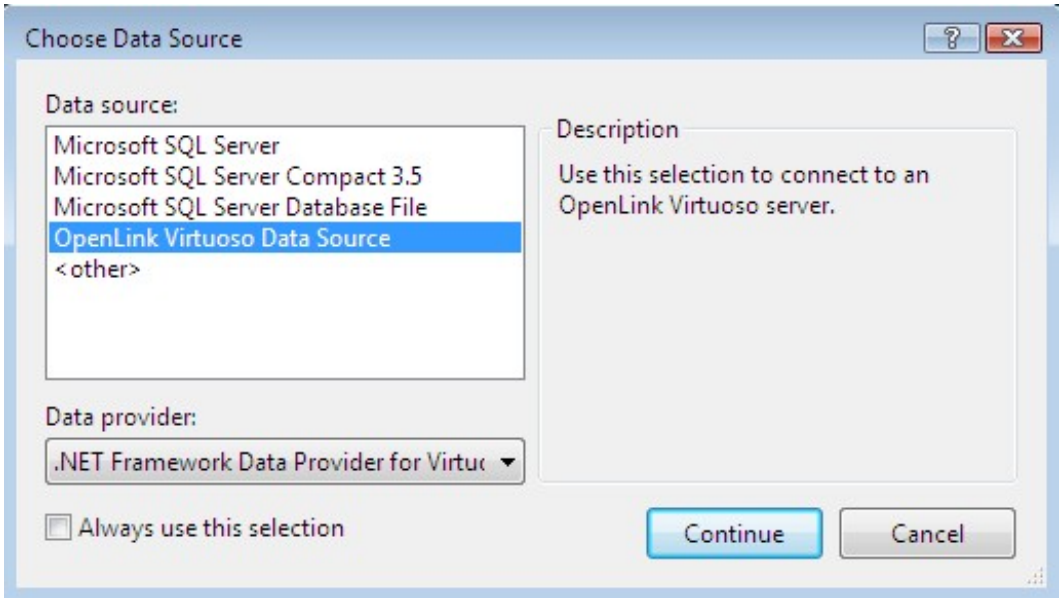
    Virtuoso Data Source

    from the list and click

    Continue

    .

Figure 8.294. Choose Data Source



12. In the

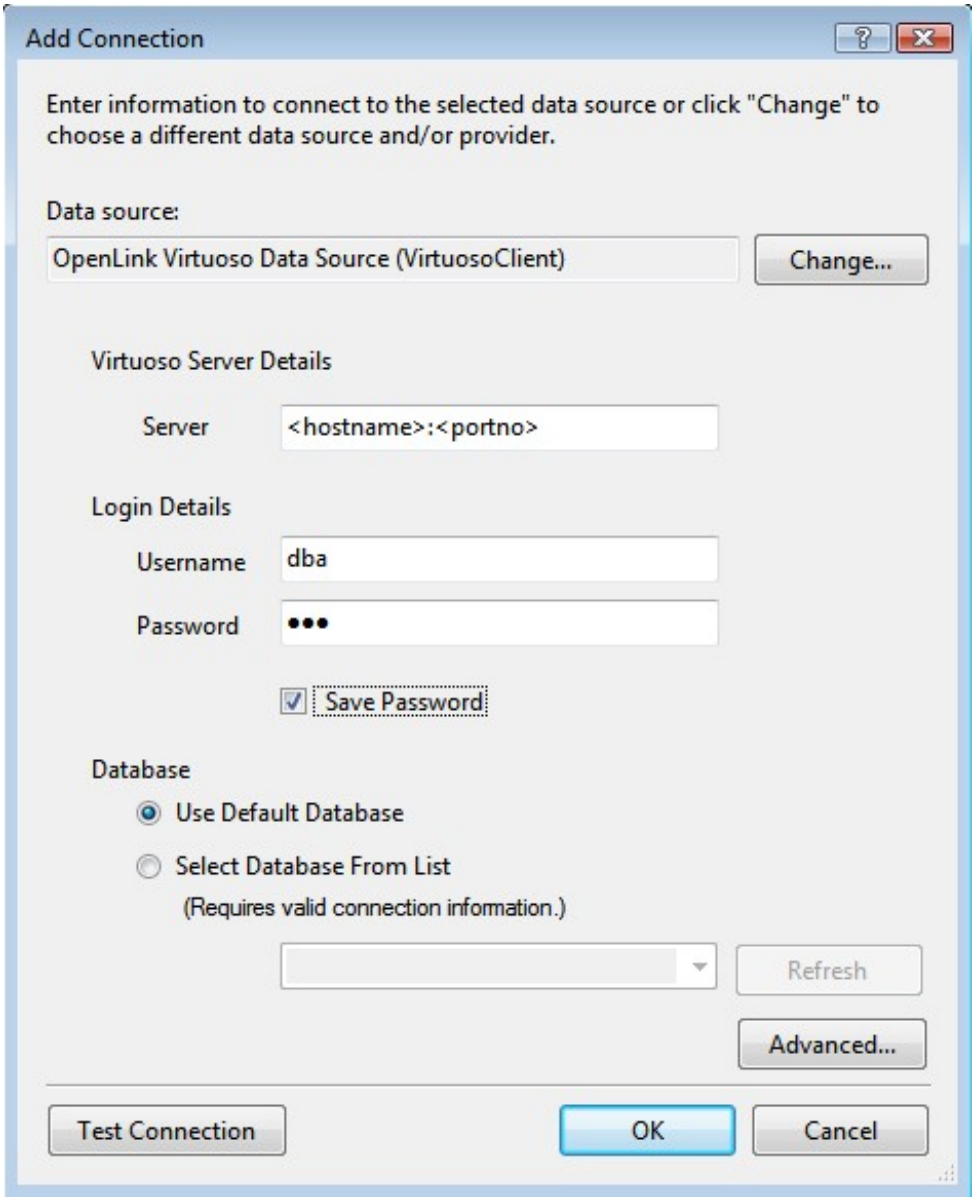
    Add Connection

    dialog, specify the

    hostname, portno, username and password

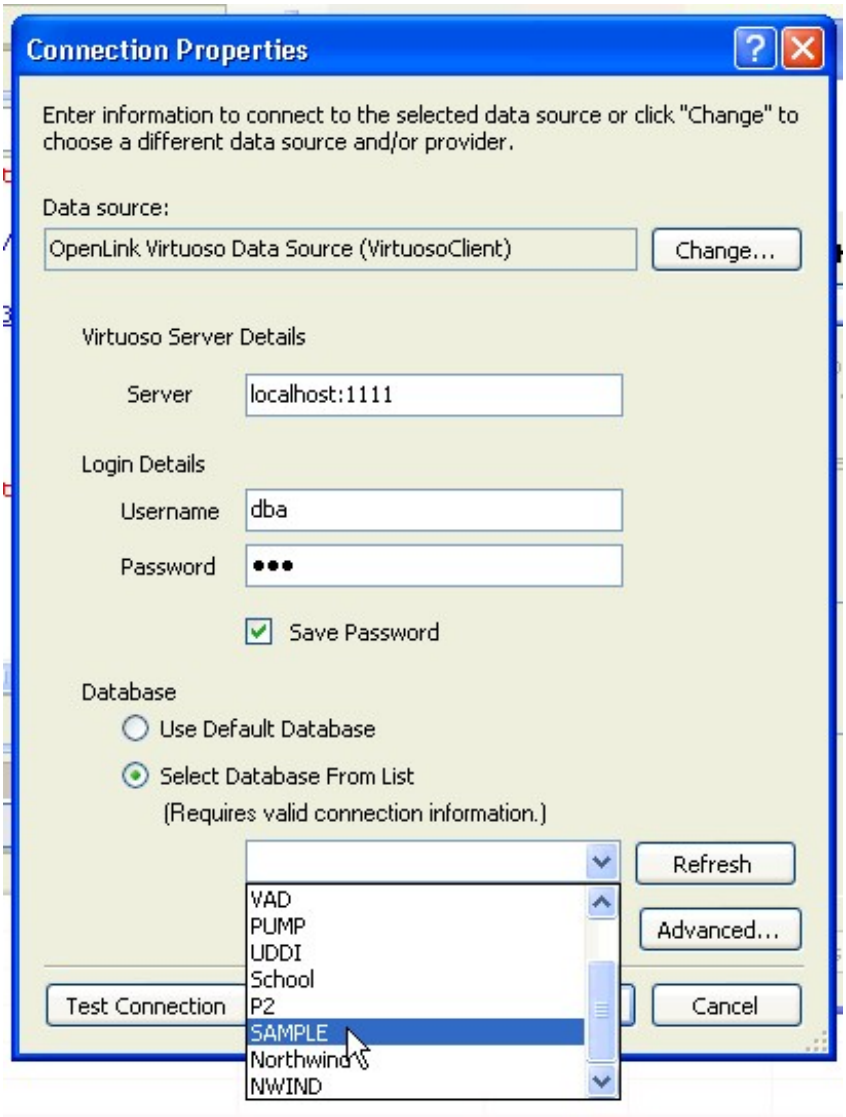
    for the target Virtuoso Server and check the Save Password check box.

    Figure 8.295. Connection Properties



13. Select the Select Database From List radio button and choose SAMPLE from the drop down list. Click OK.

Figure 8.296. Add connection

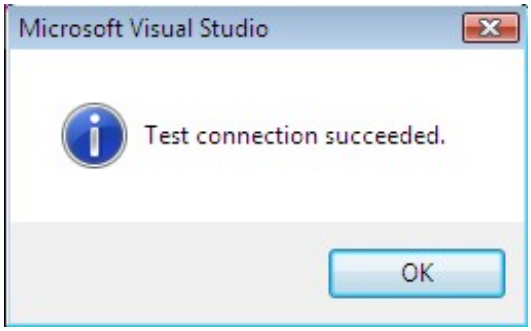


14. Press the

    Test Connection

    dialog to verify that the database is accessible.

    Figure 8.297. Test Connection



15. Set the

    entity connect string

    name to

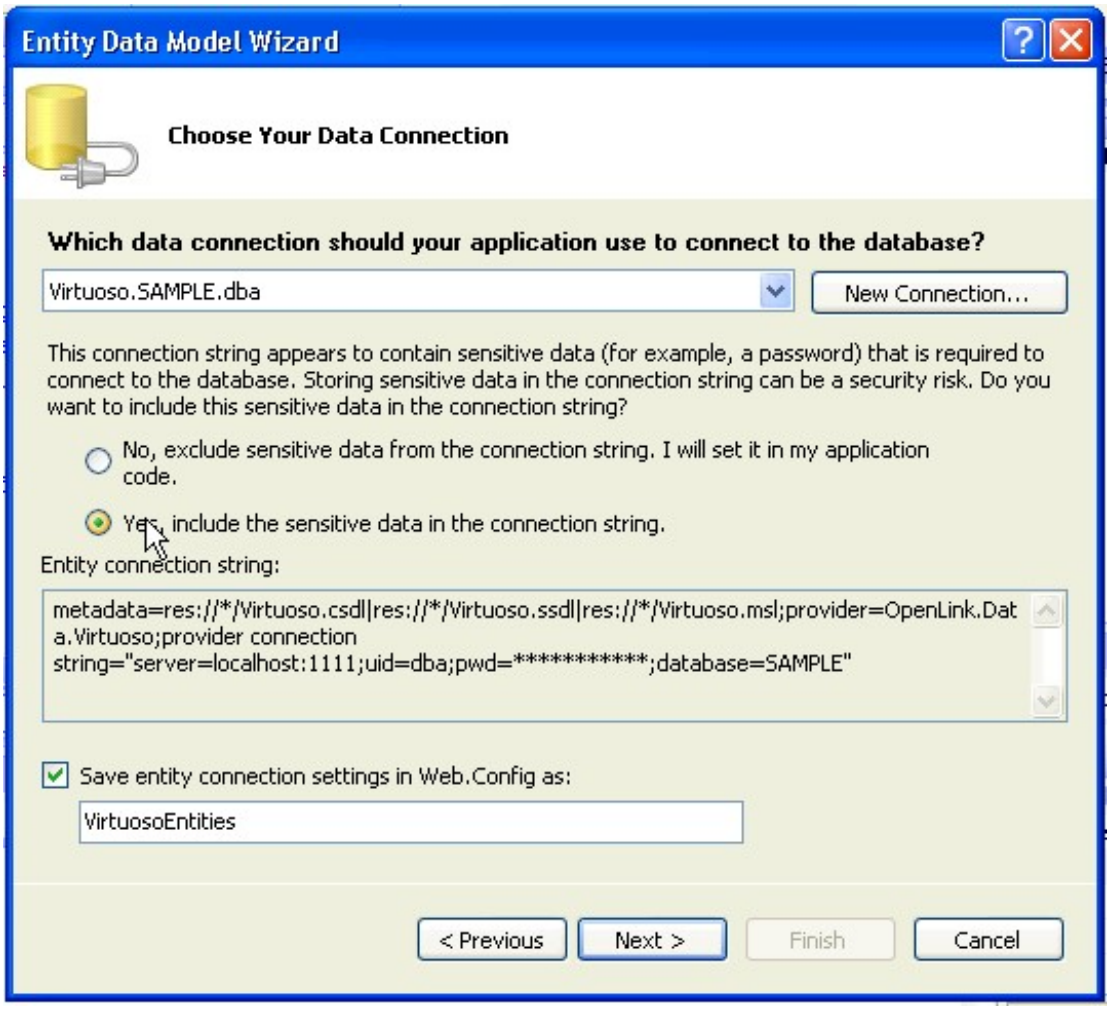
VirtuosoEntities

(note this name as it is required in the section on creating and ADO.Net Data Service) and click

Next

.

Figure 8.298. entity connect string



16. In the

    Choose your Database Objects

    page, tick the

    Tables

    check box to select all tables in the SAMPLE catalog for addition to the Entity Data Model. Set the

    Model Namespace

    to

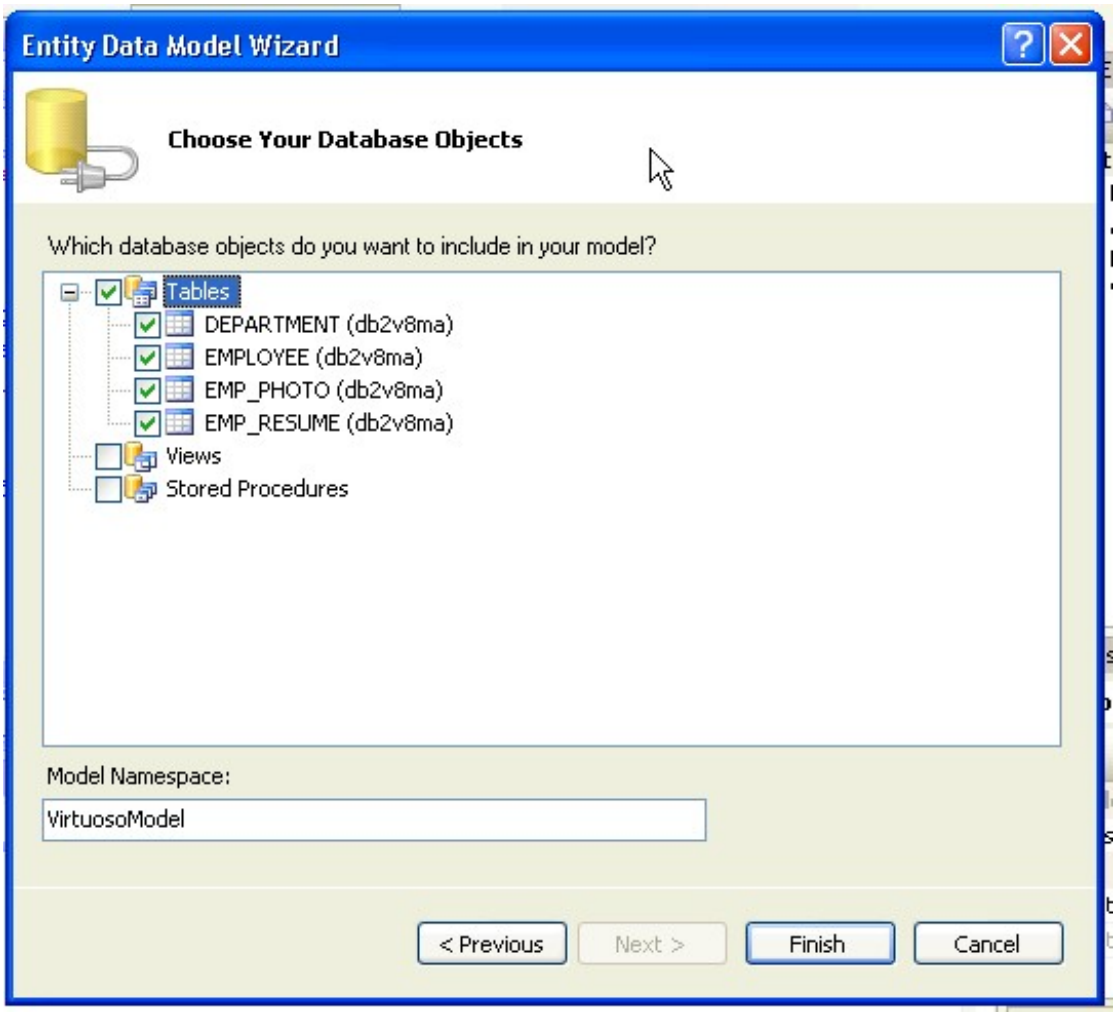
    VirtuosoModel

    and click

    Finish

.

Figure 8.299. Database Objects

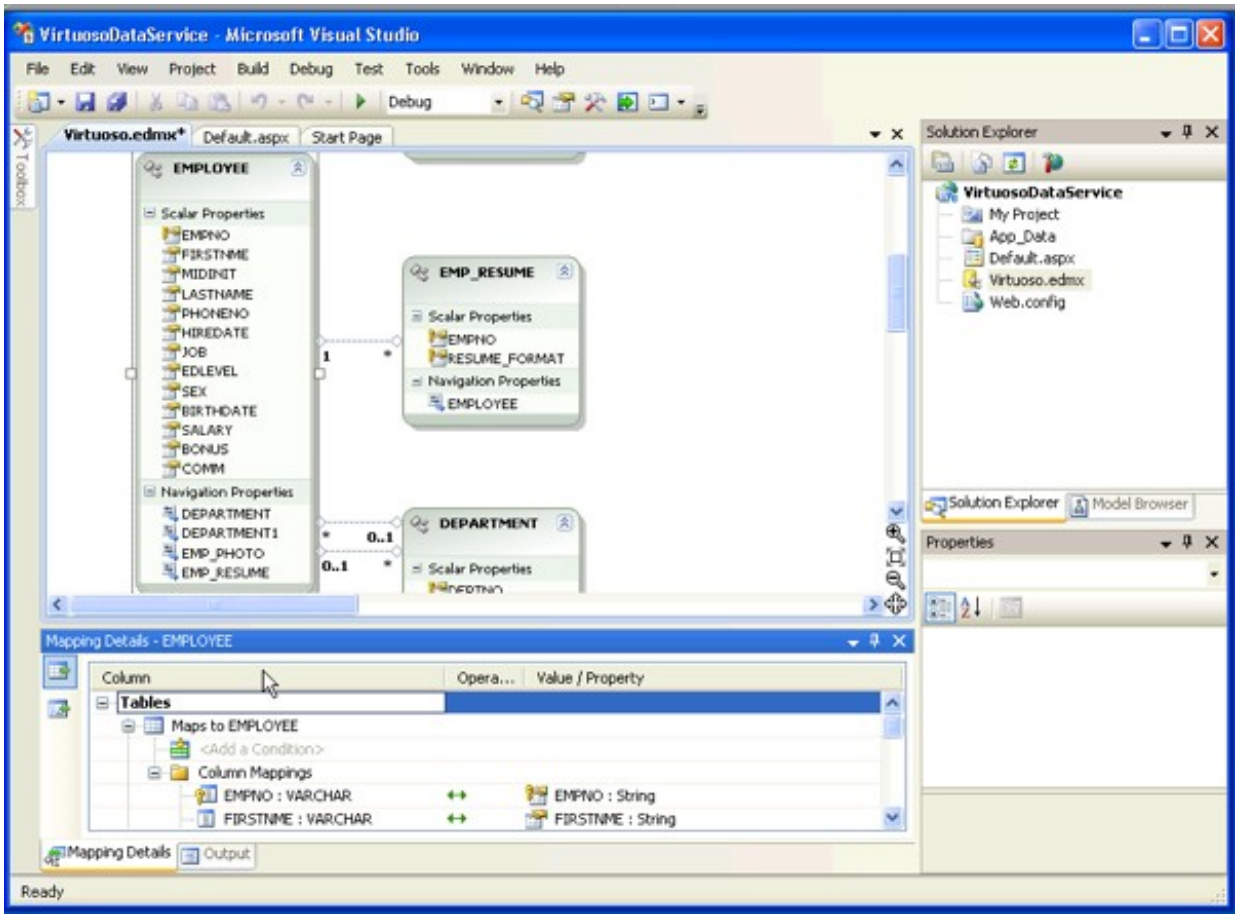


17. The

    Virtuoso.edmx

    EDM will be created with the tables and relationships displayed in the Visual Studio IDE

    Figure 8.300. Virtuoso.edmx



Creation for the Entity Data Model for the DB2 Samples database is now complete.

8.6.5. Using EDM to create Entity Framework based applications

Now that a Microsoft Entity Data Model has been created for the DB2 Samplef database, Entity Framework applications can be created to make use of it.

Entity Frameworks based ADO.NET Data Service

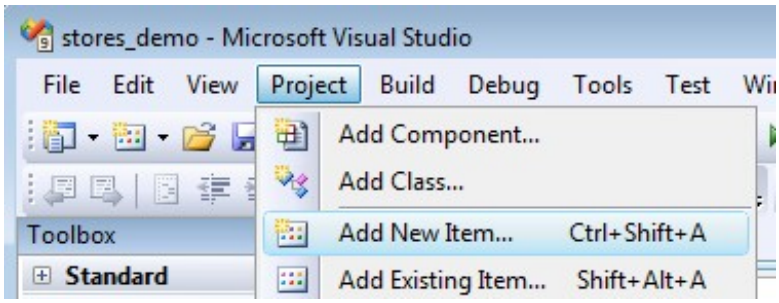
An ADO.Net Data Service for the DB2 tables can be created using the Entity Data Model created in the Creating EDM in Visual Studio 2008 section .

1. Open the

   VirtuosoDataService

   project created in the Creating EDM in Visual Studio 2008 section .
2. Select the Project -> Add New Item menu option.

Figure 8.301. VirtuosoDataService



3. The

   Add New Item

   dialog will appear. Choose the

   ADO.NET Data Service template

   . Give it the name

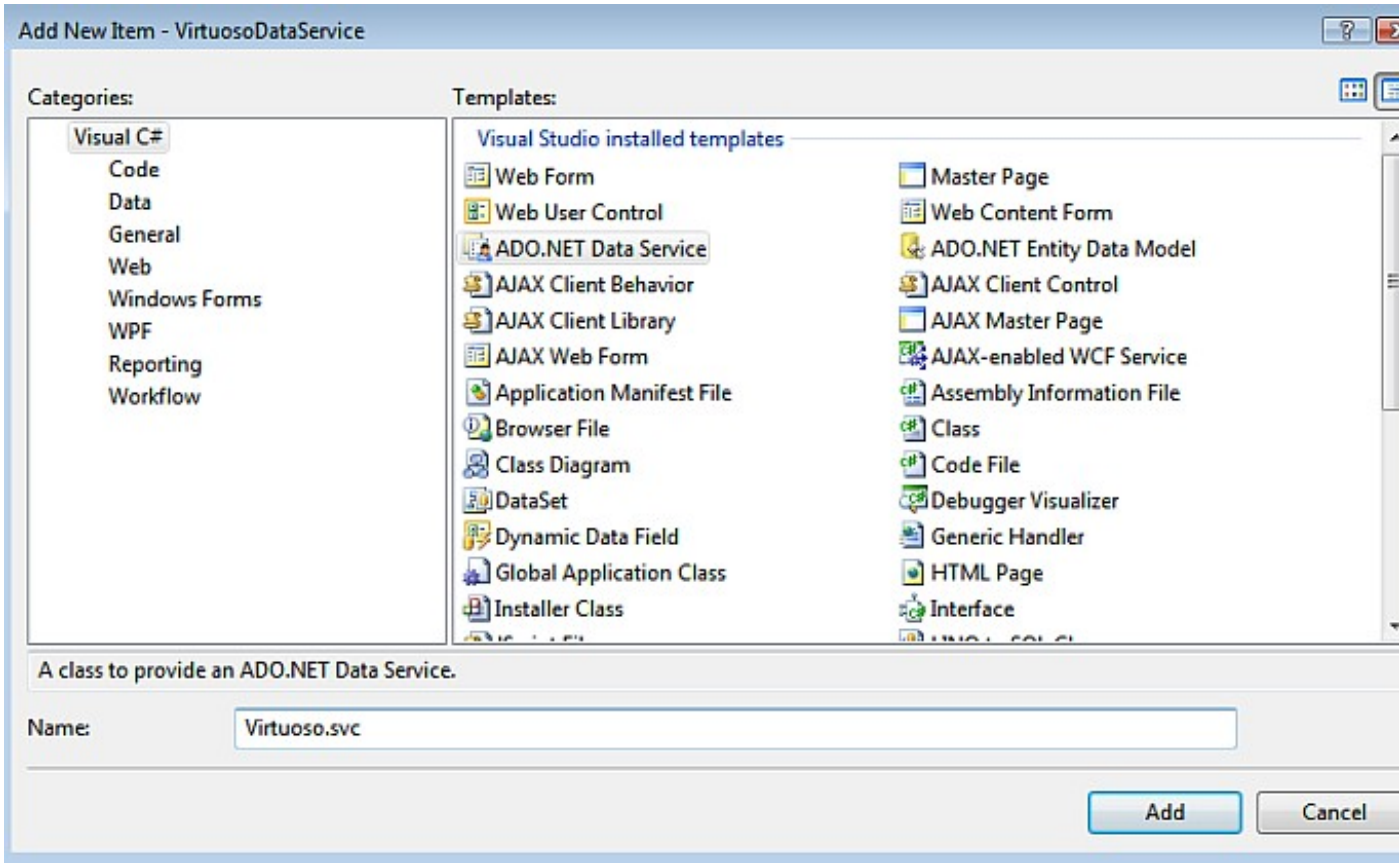
   Virtuoso.svc

   and click

   Add

   to create the ADO.Net Data Service.

   Figure 8.302. Add New Item



4. In the newly created

   Virtuoso.svc.cs

   Data Service file, add the data source class name of

   VirtuosoEntities

   (note this is the name set in the Creating EDM in Visual Studio 2008 section) as the

   DataService

name. Enable the access to the Data Service by adding the entry

config.SetEntitySetAccessRule("", EntitySetRights.All);*

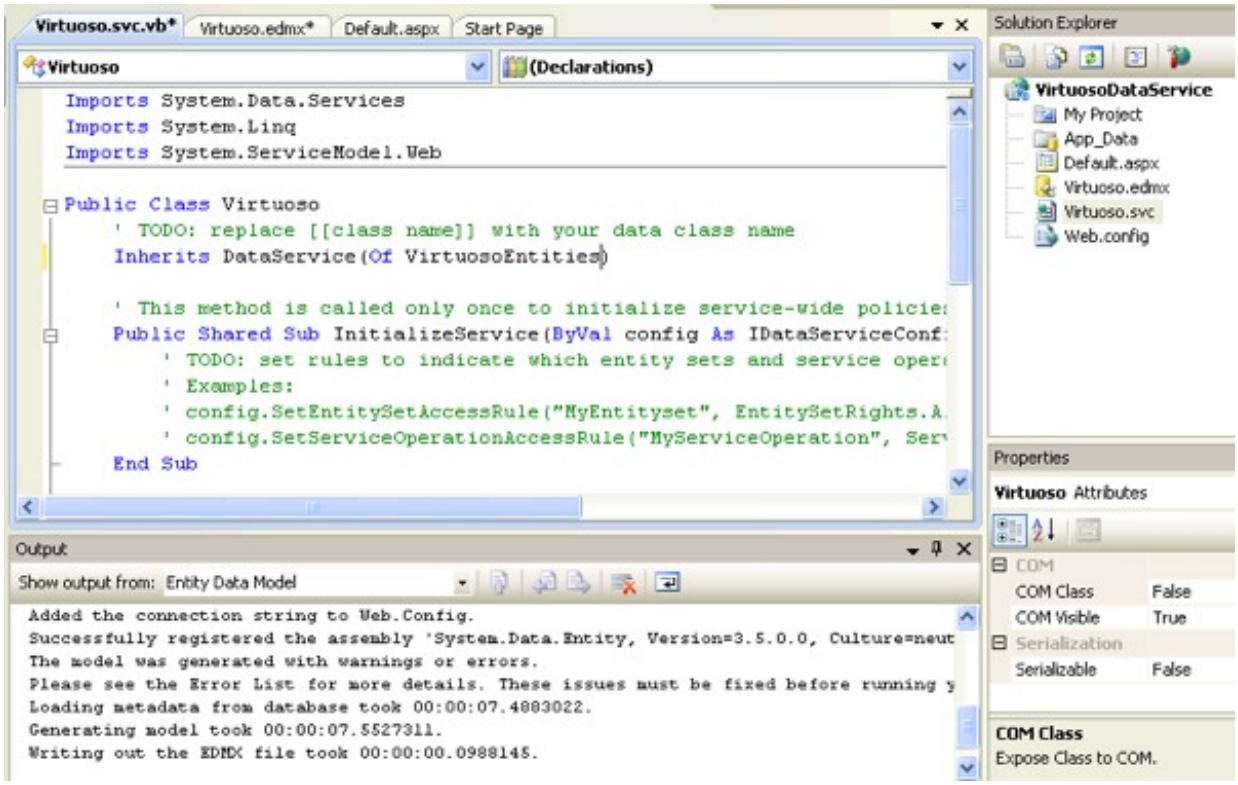
to the

InitializeService

method.

```
// C#

using System;
using System.Web;
using System.Collections.Generic;
using System.ServiceModel.Web;
using System.Linq;
using System.Data.Services;

namespace SimpleDataService
{
    public class Northwind : DataService<VirtuosoDemoEntities>
    {
        public static void InitializeService(IDataServiceConfiguration  config)
        {
            config.SetEntitySetAccessRule("*", EntitySetRights.All);
        }
    }
}
```

Figure 8.303. Virtuoso.svc.cs

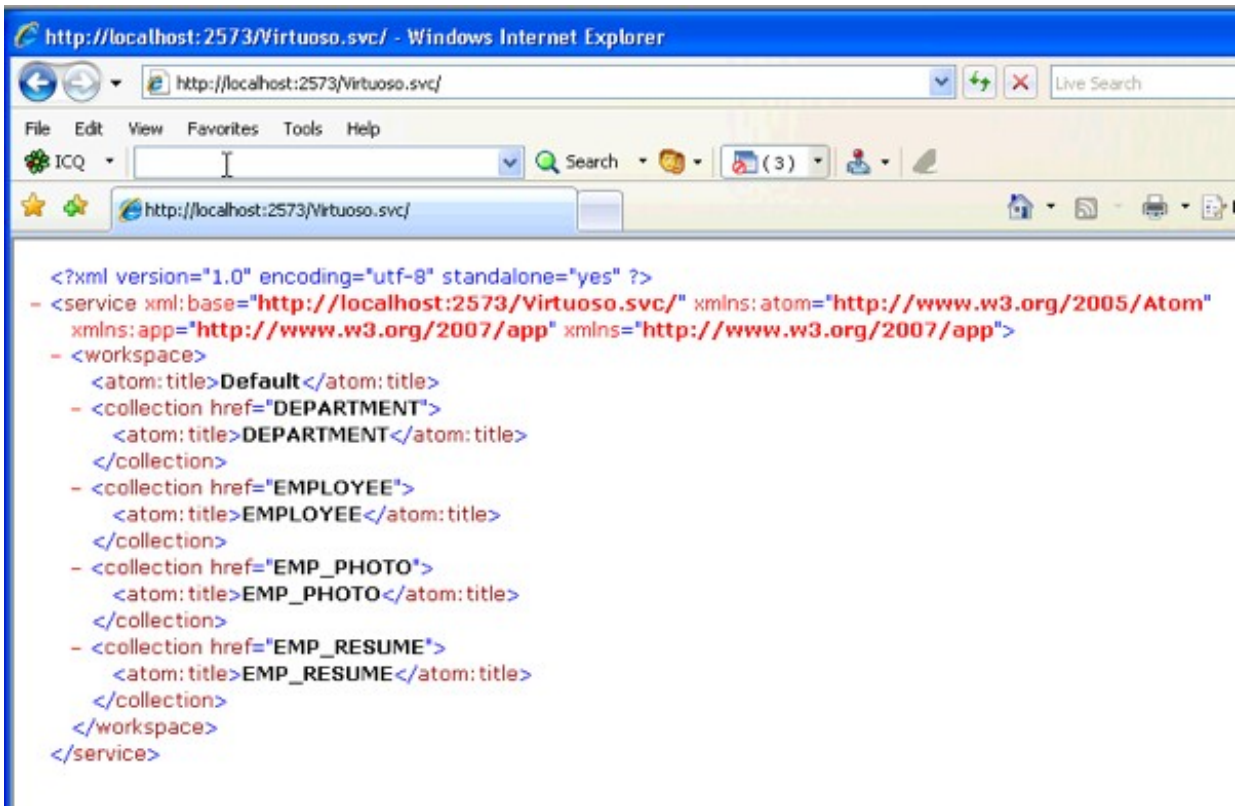


5. To test the Data Service, simply hit

   Ctrl+F5

   within Visual Studio. This will start the development web server, run the Data Services server inside, and load a Web browser page displaying the list of available tables/entities for the SAMPLE database catalog.

Figure 8.304. Data Service test



6. To access a specific entity instance like the

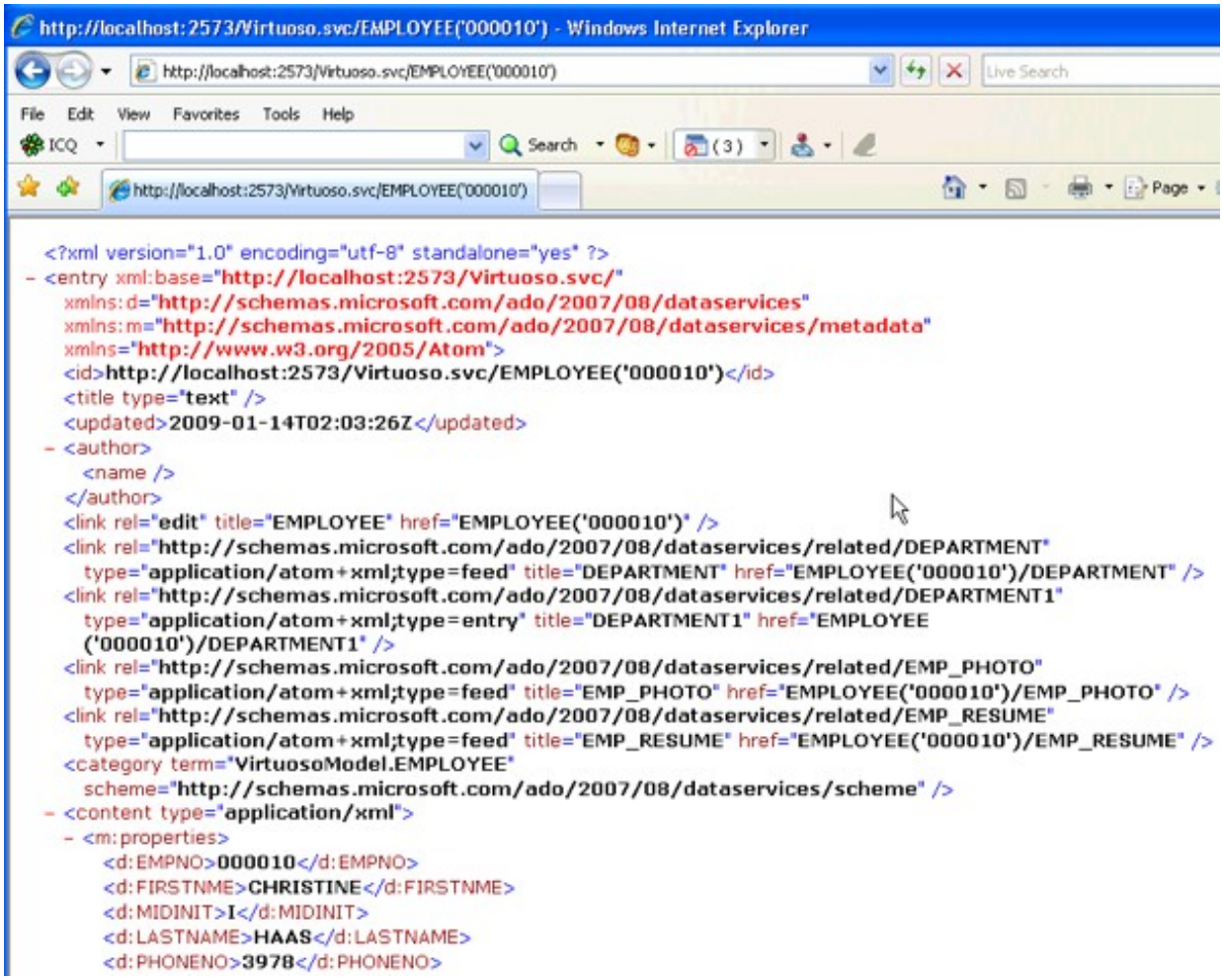
   EMPLOYEES

   table employee number

   000010

   record, use this convention http://host/vdir/Virtuoso.svc/EMPLOYEES("000010") .

   Figure 8.305. EMPLOYEES



Notes:

1. *Important*

   - To view

   Atom

   (the default format returned by an ADO.NET Data Service) in Internet Explorer, you must first ensure that

   Feed Reading View

   is turned

   off

   . This can be done on the

   Content tab

   of

   Tools in Internet Options

   .
2. If a Data Services entity instance URI page fails to load you can turn

   Verbose

errors on by adding

config.UseVerboseErrors = true;

in the

virtuoso.svc.cs InitializeService

method to obtain more detailed information from the server as to why the page failed to load:

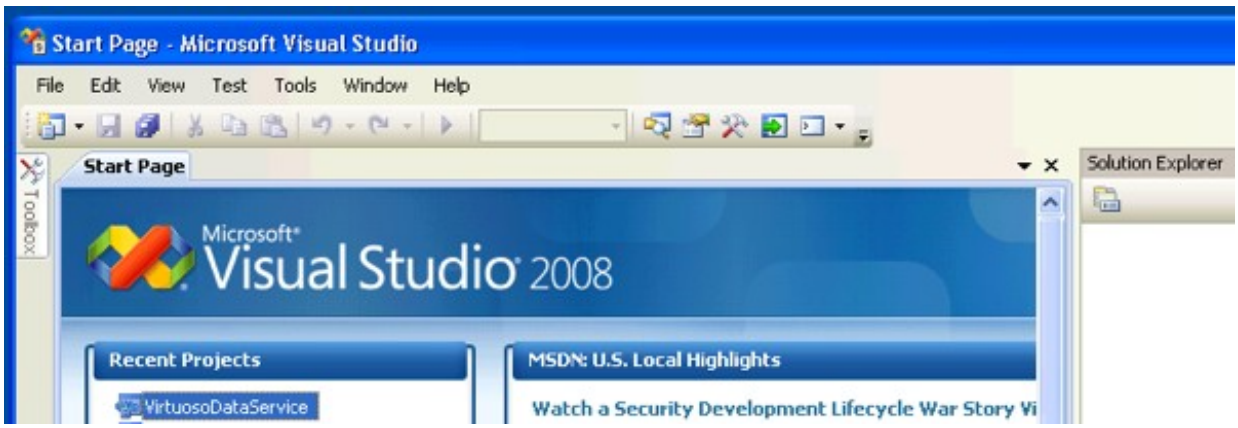
```
public static void InitializeService(IDataServiceConfiguration config)
{
config.UseVerboseErrors = true;
config.SetEntitySetAccessRule("*", EntitySetRights.All);
}
```

Visual Studio Windows DataGrid Form Application

This section details the steps required to create a simple Visual Studio 2008 Windows Form application, with associated DataGridView control for displaying data in selected tables from the target database.

1. Launch the Visual Studio 2008 SP1 IDE.

   Figure 8.306. Visual Studio 2008 SP1 IDE

2. Create a

   Web Application

   project by going to the

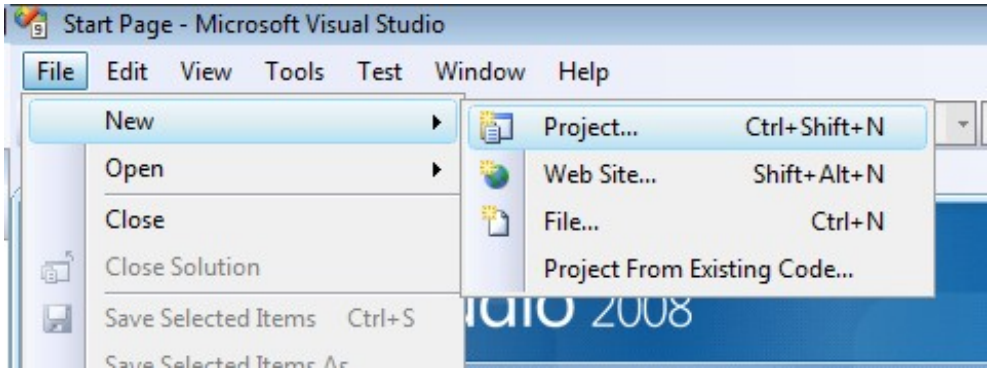
   File

   menu in Visual Studio and choosing

   New Project

   .

   Figure 8.307. Web Application



3. When the New Project window appears, choose either

   Visual Basic

   or

   Visual C#

   as your programming language.
4. Within the language category, click on

   Windows

   and select

   Windows Form Application

   from the right-hand panel.
5. Choose a name for the project, for example

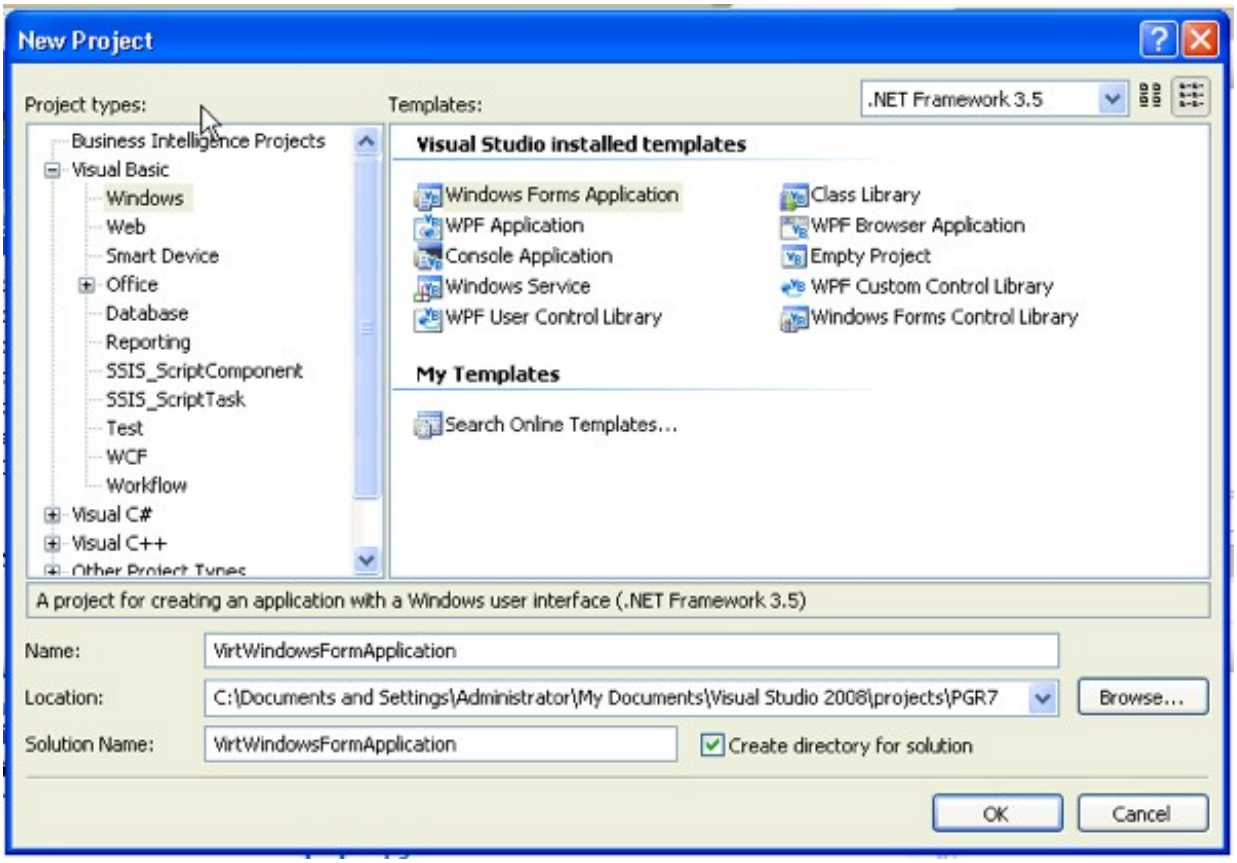
   VirtWindowsFormApplication

   , and click

   OK

   .

   Figure 8.308. Web Application



6. In the

   Toolbox

   , expand

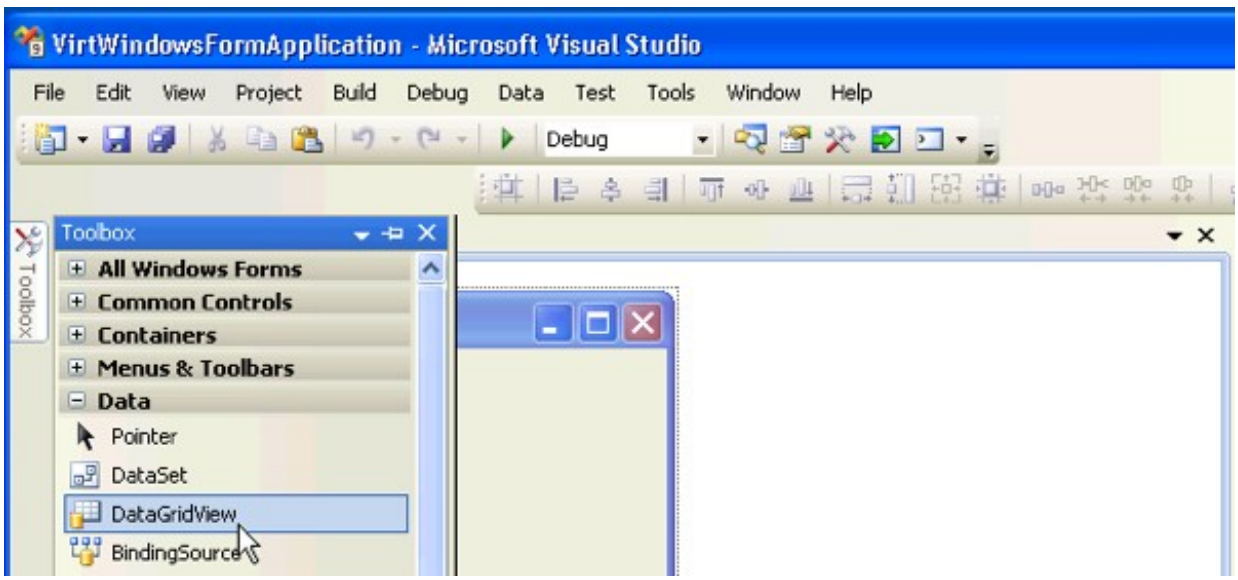
   Data Controls

   , and drag the

   DataGridView

   control onto the form.

Figure 8.309. Toolbox



7. Click on the little

   arrow

   in the top right of the

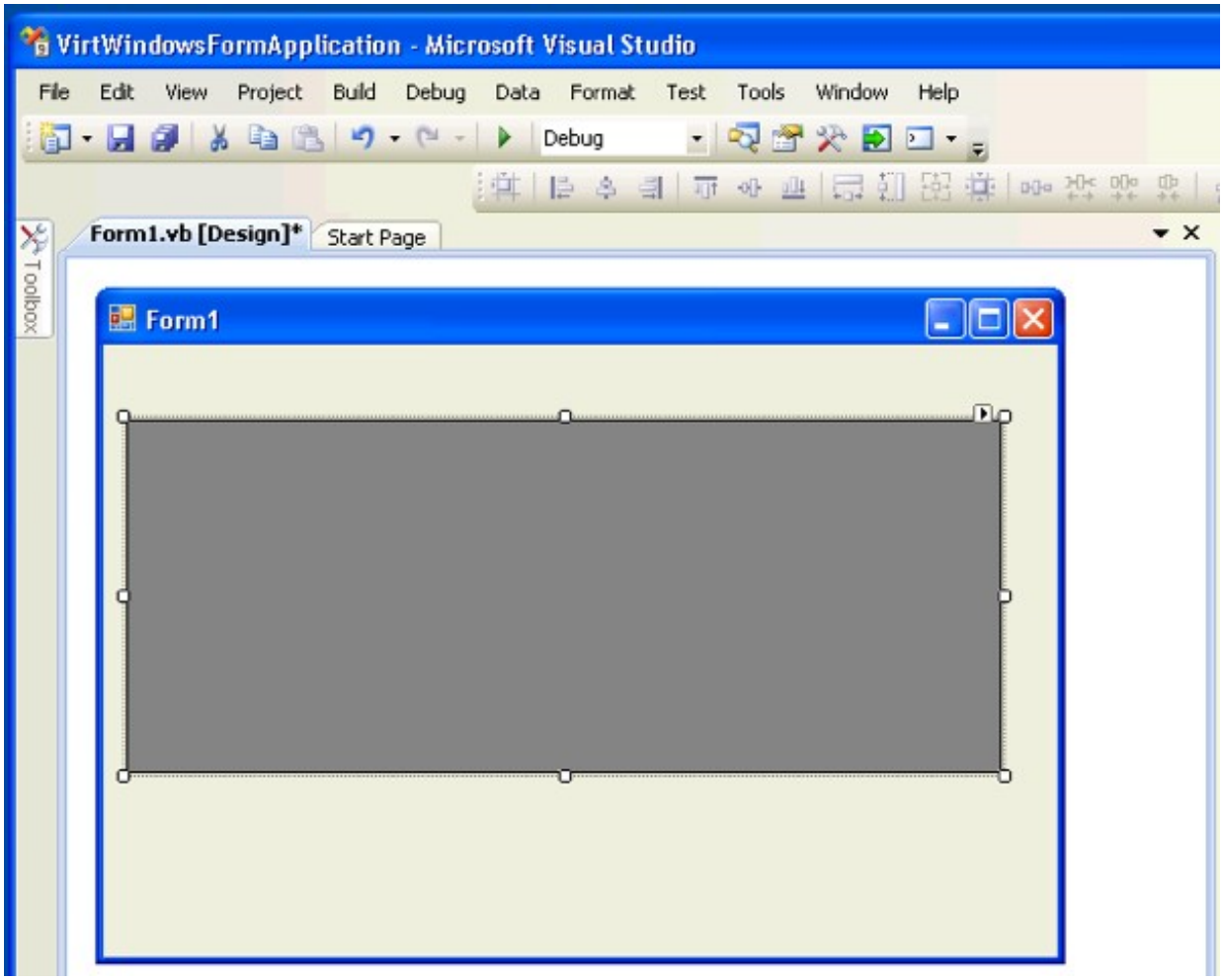
   DataGridView

   control. This loads the

   DataGridView Task

   menu.

   Figure 8.310. DataGridView Task

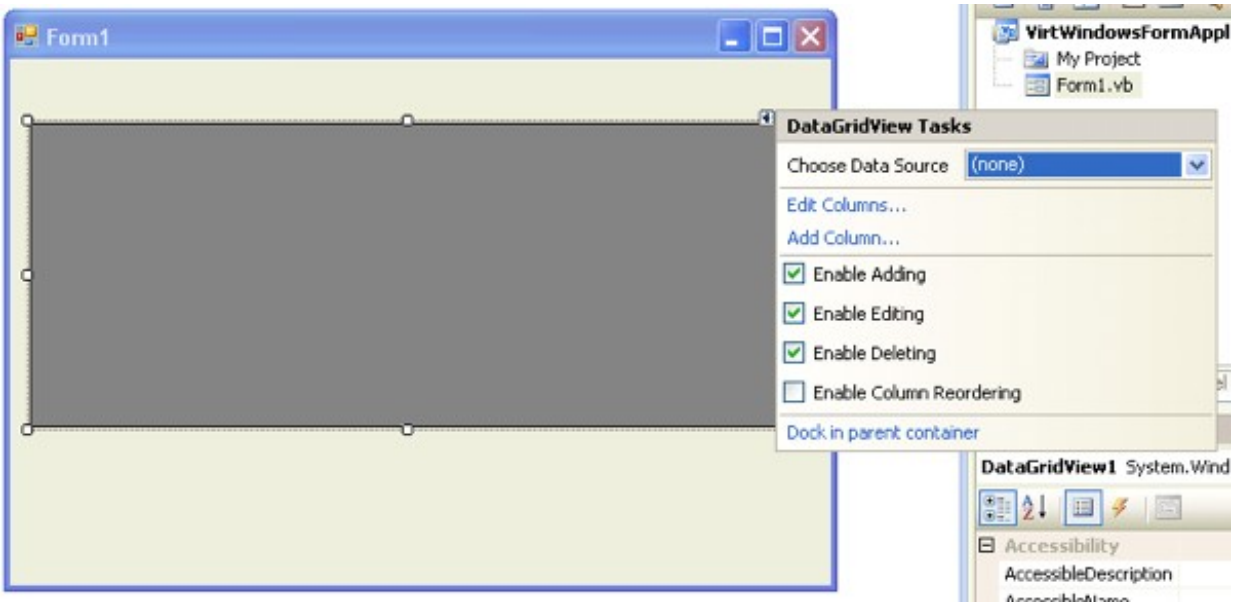


8. Click on the

   Choose Data Source

   list box.

   Figure 8.311. Choose Data Source

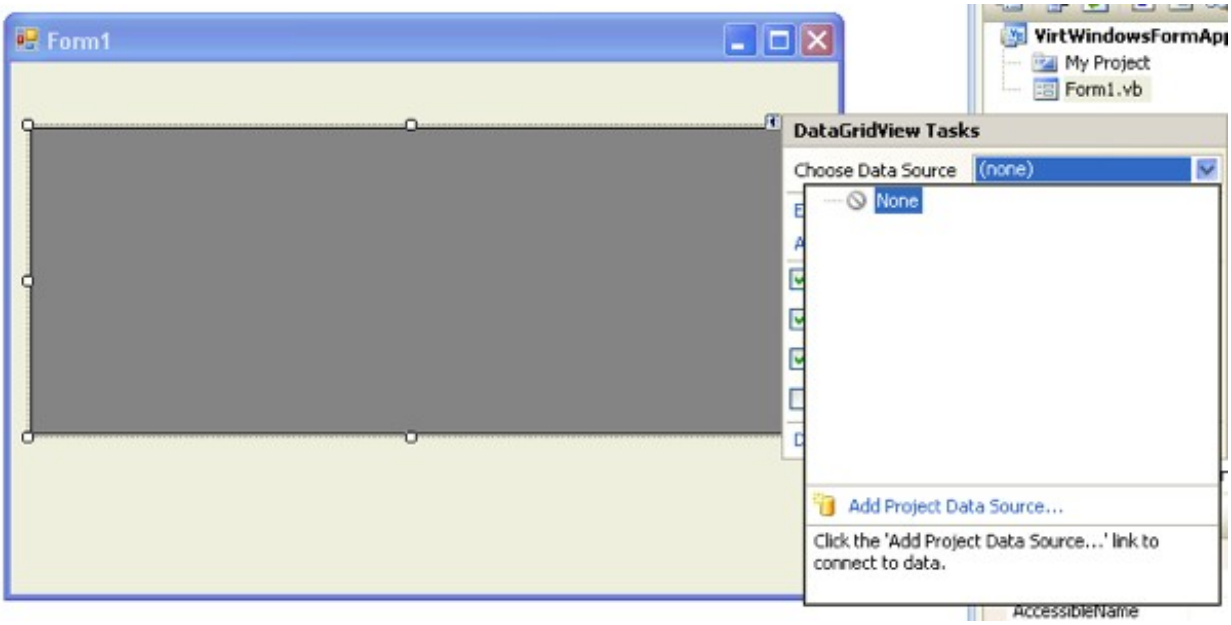


9. Click on the

   Add Project Data Source

   link to connect to a data source.

Figure 8.312. Add Project Data Source



10. In the

    Data Source Configuration Wizard

    dialog

    Choose Data Source Type

    page select the

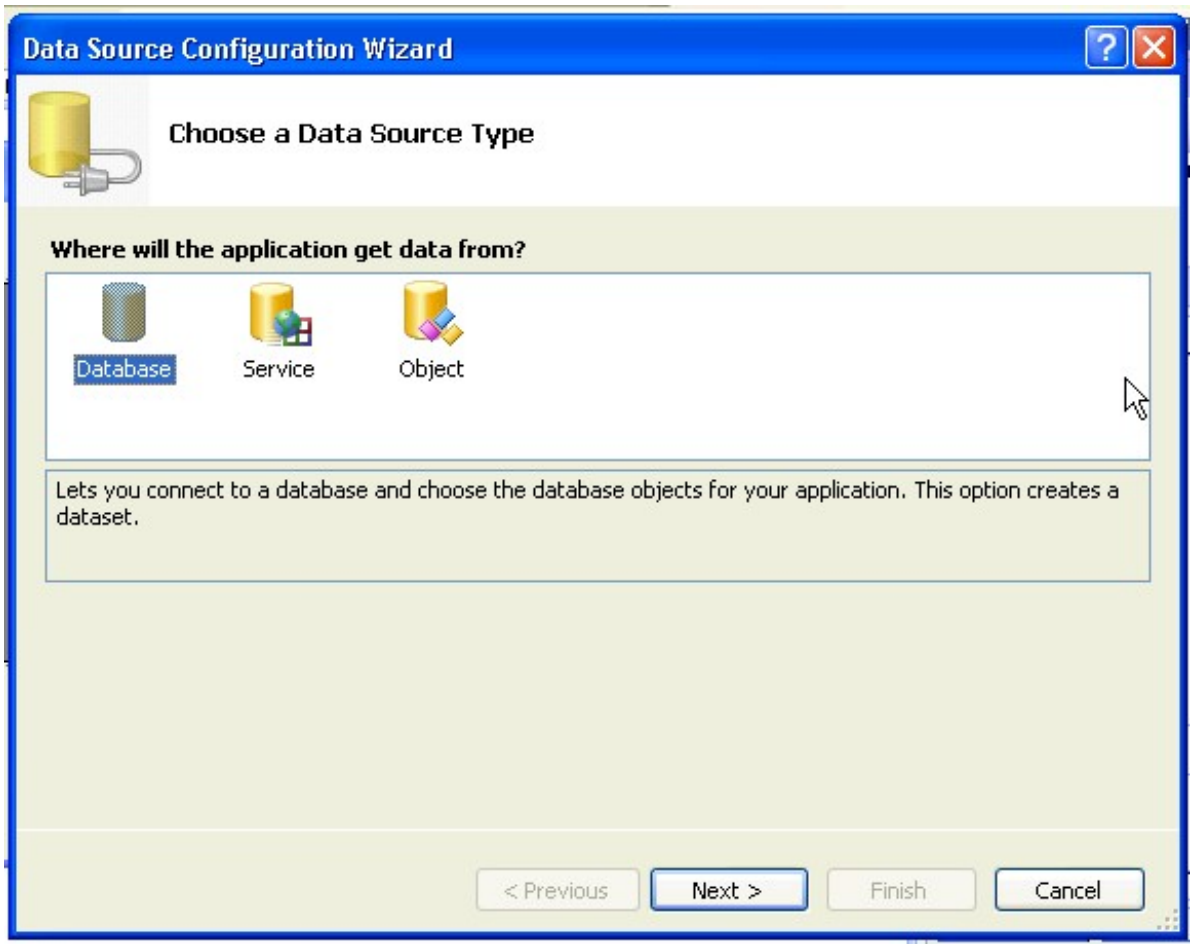
    Database

    data source type and click

Next

.

Figure 8.313. Data Source Type



11. In the

    Data Source Configuration Wizard

    dialog

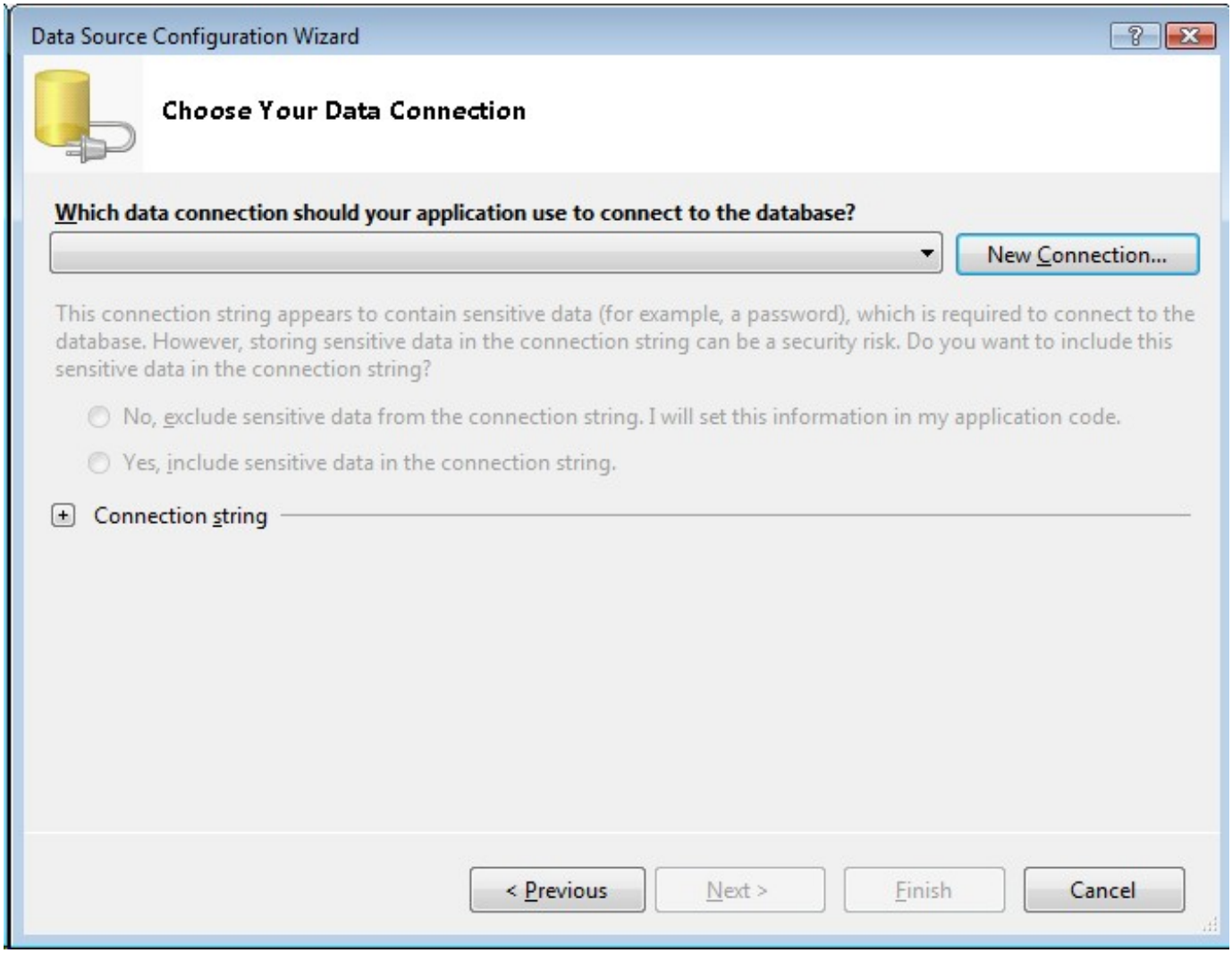
    Choose your Data Connection

    page, select the

    New Connection

    button

Figure 8.314. Data Source Configuration Wizard



12. In the

    Choose Data Source

    dialog, select the OpenLink

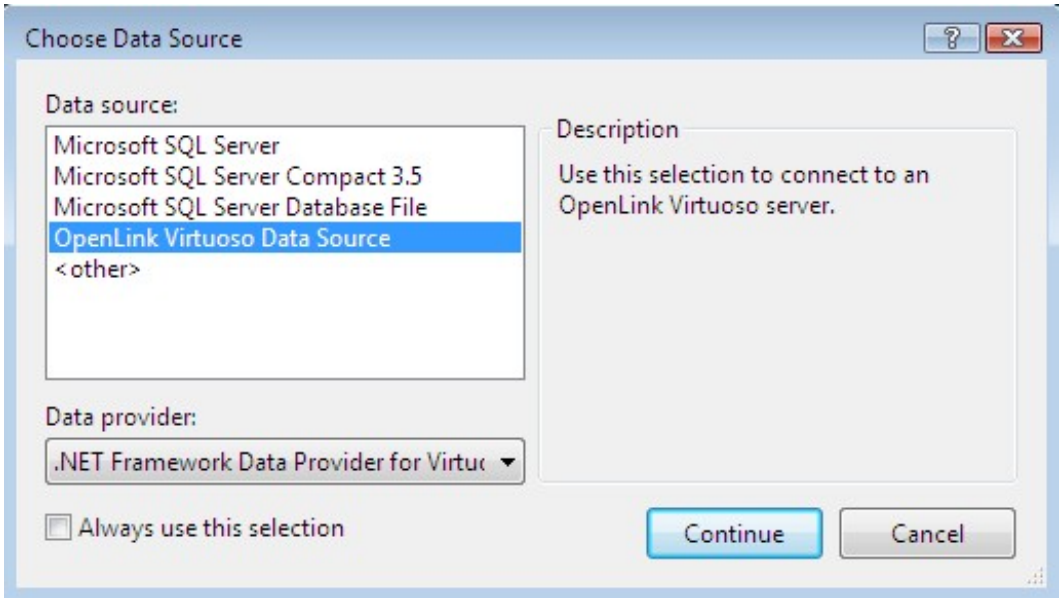
    Virtuoso Data Source

    from the list and click

    Continue

    .

Figure 8.315. Data Source



13. In the

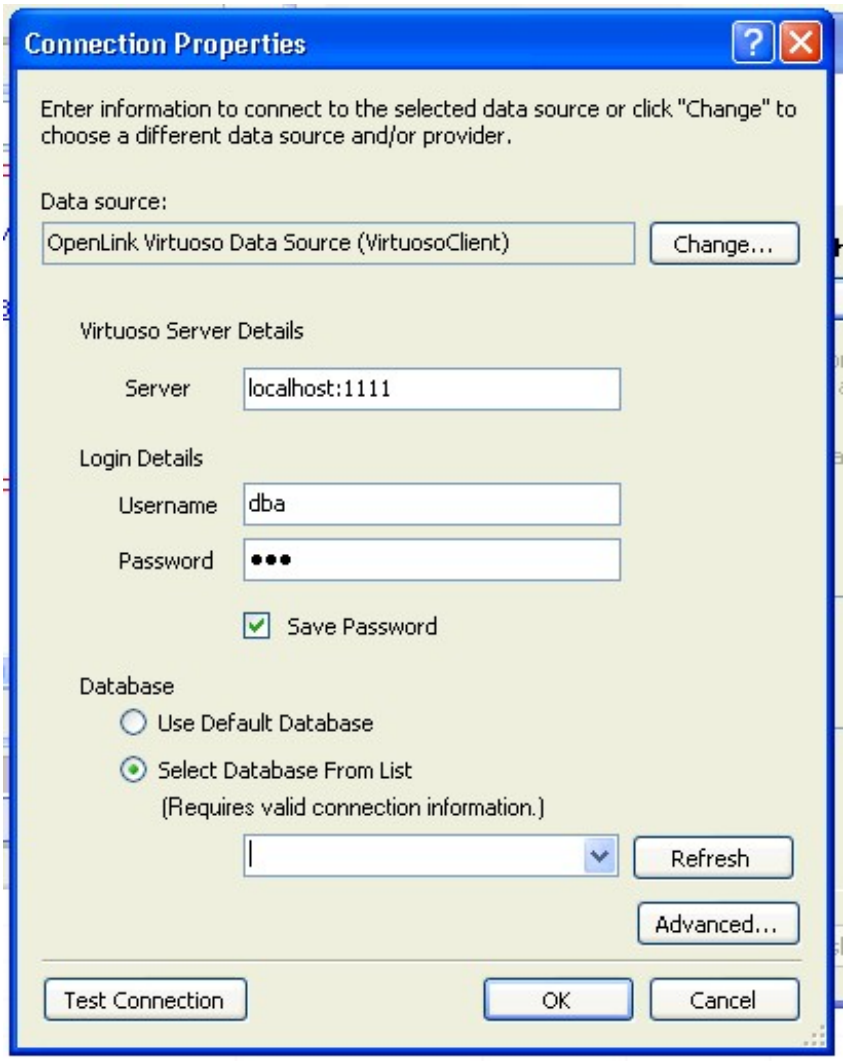
    Add Connection

    dialog, specify the

    hostname, portno, username and password

    for the target Virtuoso Server and check the Save Password check box.

    Figure 8.316. Connection Properties



14. Select the

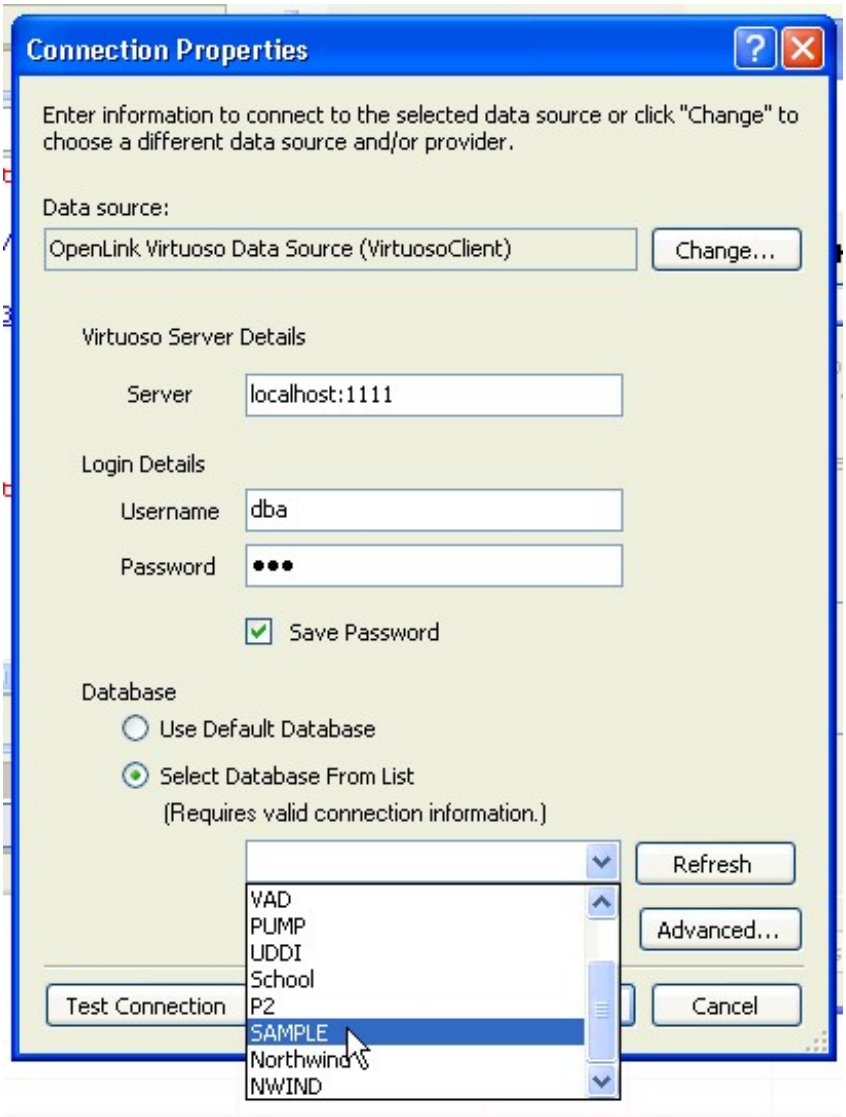
    Select Database From List

    radio button and choose the

    SAMPLE

    database from the drop down list.

Figure 8.317. Add connection

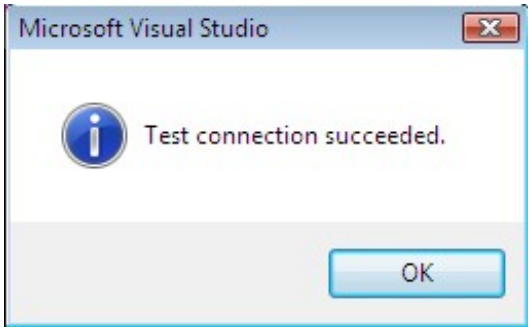


15. Press the

    Test Connection

    dialog to verify that the database is accessible.

    Figure 8.318. Test Connection



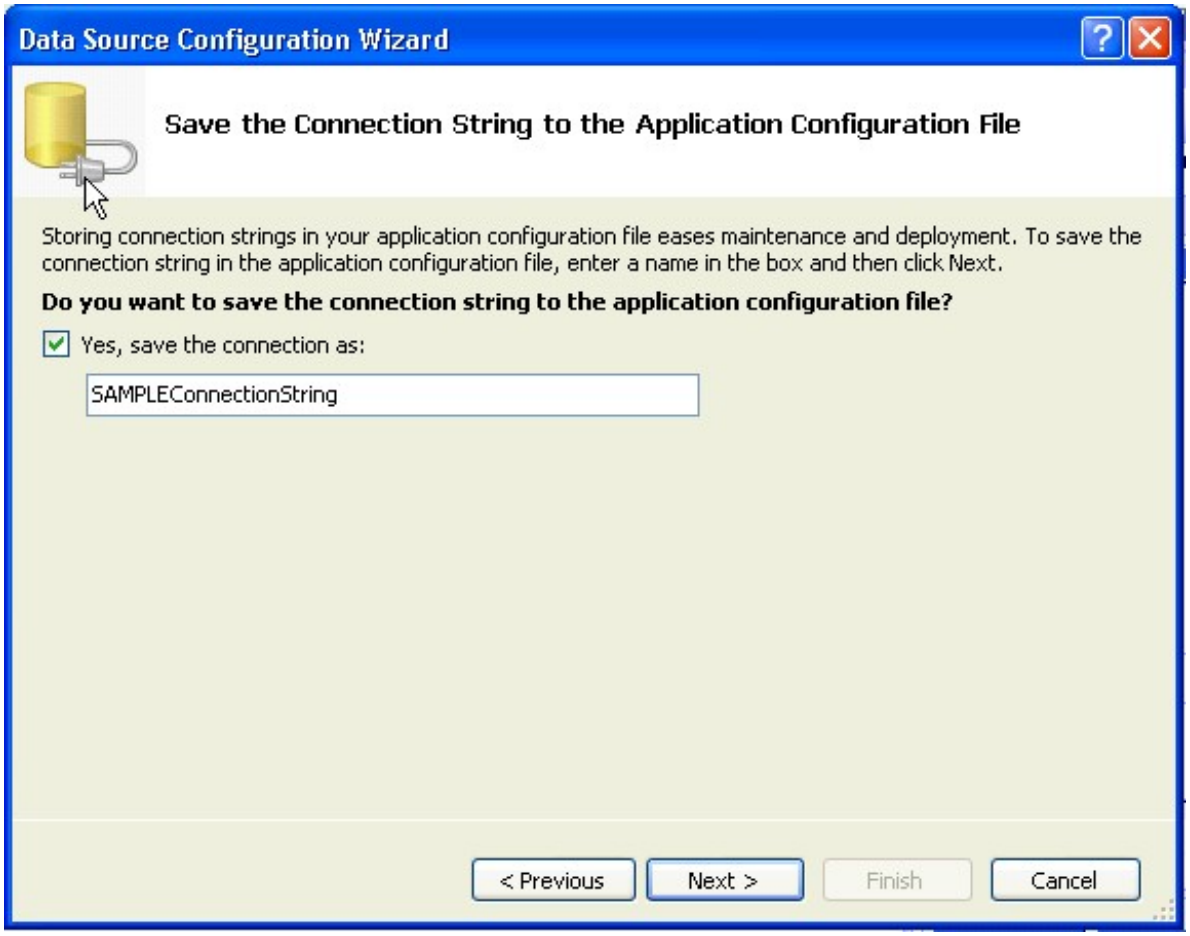
16. Leave the default connect string

    SAMPLEConnectionString

    and click

Next

Figure 8.319. SAMPLEConnectionString



17. From the list of available tables returned for the SAMPLE database select the

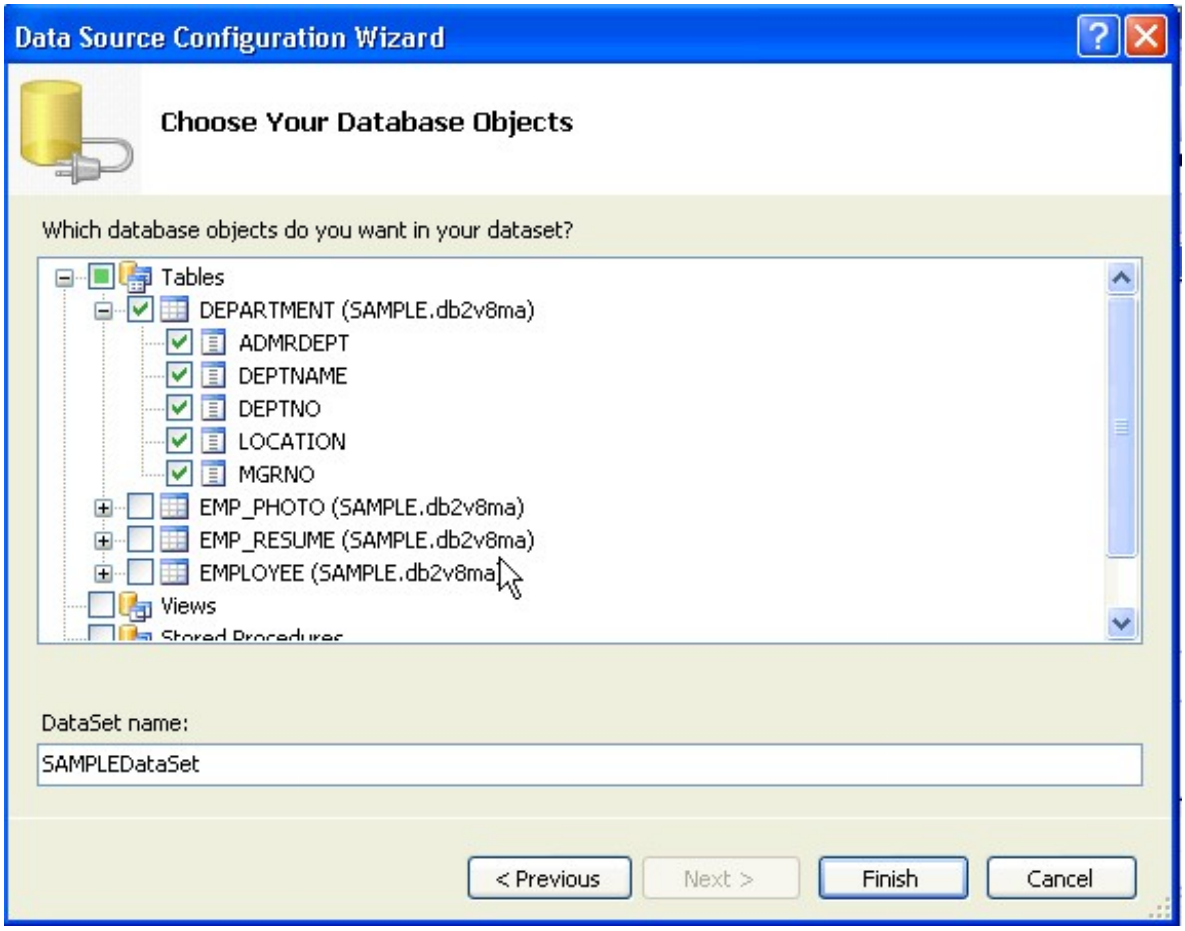
    DEPARTMENT

    table to be associated with the

    DataGridView

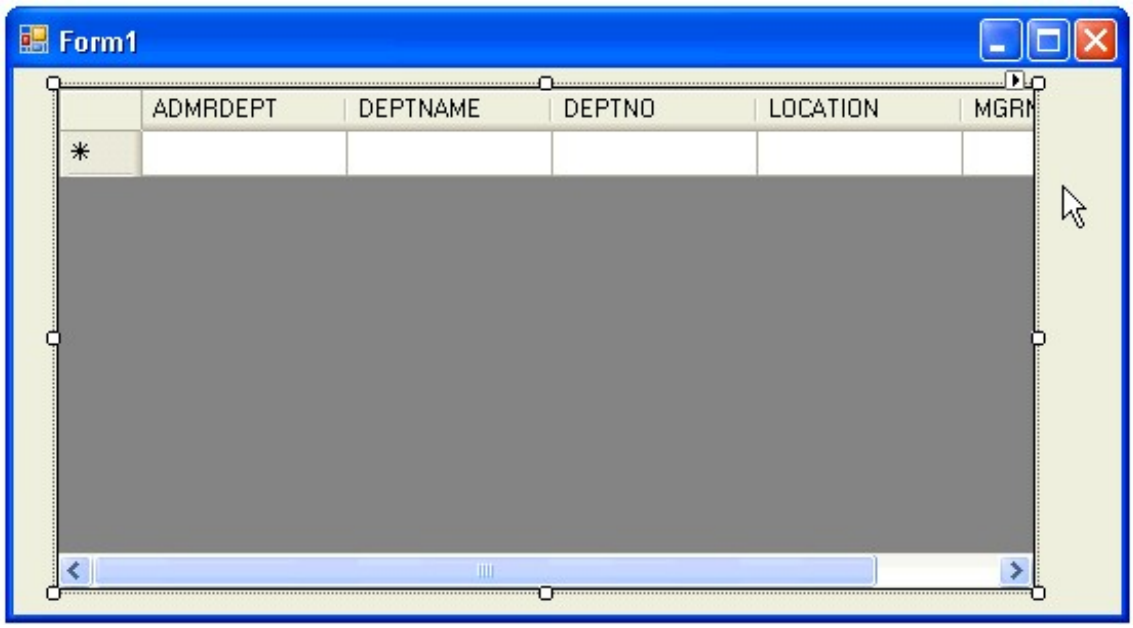
    control

    Figure 8.320. SAMPLE database



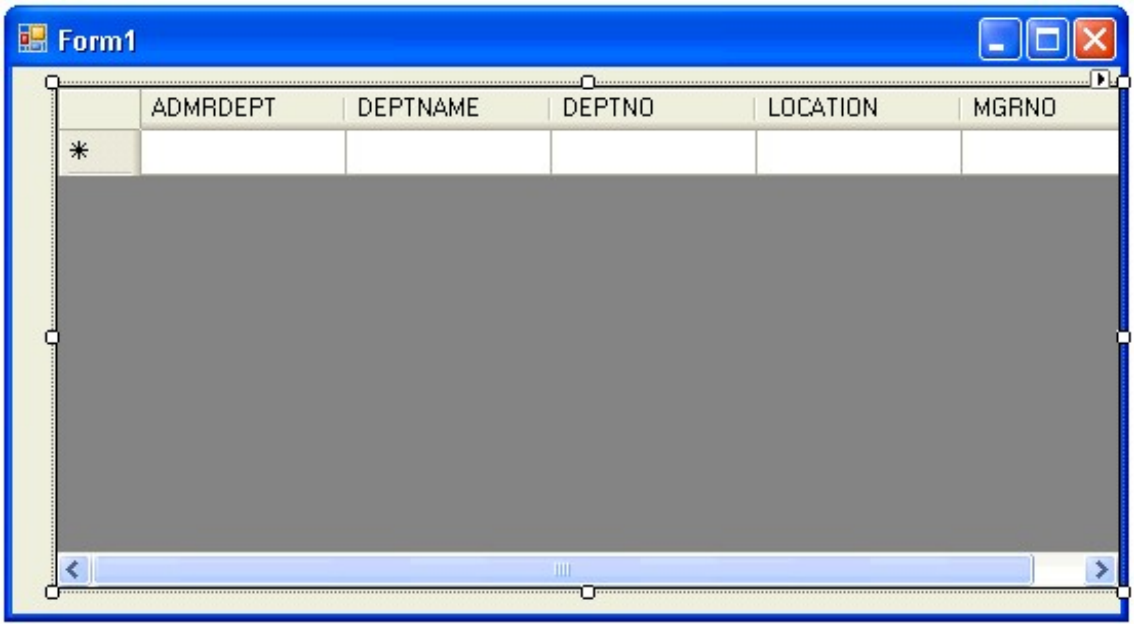
18. The columns names of the select table will be displayed in the DataGridView

Figure 8.321. DataGridView



19. Resize the Form and DataGridView to allow all columns to be visible, if possible.

Figure 8.322. Resize the Form and DataGridView



20. To test the application, simply hit

    Ctrl+F5

    within Visual Studio or select

    Start Debugging

    from the

    Debug

    menu.

    Figure 8.323. Start Debugging

21. The data from the

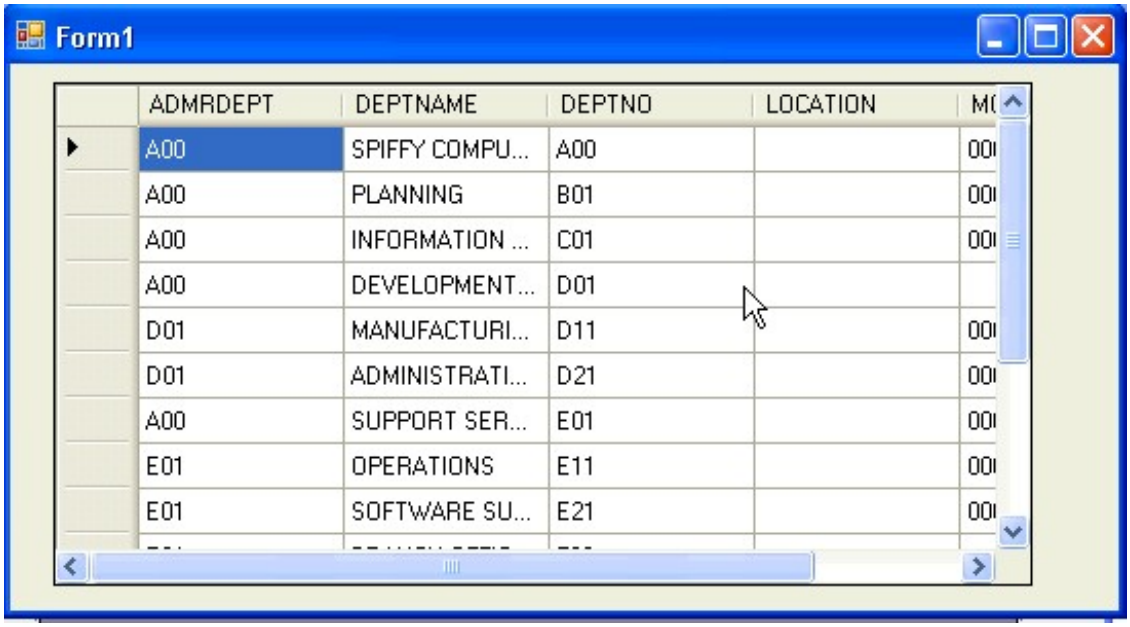
    DEPARTMENT

    table will be displayed in the

    DataGrid

    .

    Figure 8.324. DataGrid



The task is now complete.

8.7. Using Microsoft Entity Frameworks to Access Sybase Schema Objects with Virtuoso

Abstract

This section details the steps required to provide Microsoft Entity Framework access to Sybase Schema Objects using the OpenLink Virtuoso Universal Server. This is achieved by linking the required Sybase Schema objects into Virtuoso using its built in Virtual Database engine, and then using the Virtuoso ADO.NET Entity Framework provider to query the remote Sybase Schema objects linked into the Virtuoso Server.

Prerequisites. The following prerequisites must be in place for this solution to be possible.

Sybase DBMS. An Sybase DBMS hosting the required Schema Objects needs to be available. In this section the *Sybase pubs2 sample* database will be used to demonstrate the process.

An Sybase DBMS hosting the required Schema Objects needs to be available. In this section the *Sybase pubs2 sample* database will be used to demonstrate the process.

ODBC Driver for Sybase. An Sybase ODBC Driver is required for Linking the Sybase Schema Objects into the Virtuoso Server. The OpenLink *ODBC Driver for Sybase* will be used in this section, for which a functional ODBC Data source name of "syb15ma" will be assumed to exist on the machine hosting the Virtuoso Server.

An Sybase ODBC Driver is required for Linking the Sybase Schema Objects into the Virtuoso Server. The OpenLink *ODBC Driver for Sybase* will be used in this section, for which a functional ODBC Data source name of "syb15ma" will be assumed to exist on the machine hosting the Virtuoso Server.

Virtuoso Universal Server. An Virtuoso installation including the Virtuoso Universal Server and ADO.NET Entity Framework Provider is required. The Virtuoso components used must be Release 5.10.x or above, this being the minimum version containing support for Microsoft Entity Frameworks

An Virtuoso installation including the Virtuoso Universal Server and ADO.NET Entity Framework Provider is required. The Virtuoso components used must be Release 5.10.x or above, this being the minimum version containing support for Microsoft Entity Frameworks

Microsoft Visual Studio 2008 SP1. Microsoft Visual Studio 2008 Service Pack 1 is required, this being the only version containing the necessary Entity Framework support available at the time of writing.

Microsoft Visual Studio 2008 Service Pack 1 is required, this being the only version containing the necessary Entity Framework support available at the time of writing.

Tasks

Ensure Sybase Primary Keys (PKs) are not nullable. The Visual Studio 2008 Entity Data Model (EDM) requires that all primary keys are *NOT* Nullable, and will fail to generate an EDM if any are. Thus ensure any tables to be used are defined as not nullable in the Sybase database schema before attempting to generate an EDM. In the case of the Sybase pubs2 database all tables are not nullable, thus this should not be an issue in this case.

The Visual Studio 2008 Entity Data Model (EDM) requires that all primary keys are *NOT* Nullable, and will fail to generate an EDM if any are. Thus ensure any tables to be used are defined as not nullable in the Sybase database schema before attempting to generate an EDM. In the case of the Sybase pubs2 database all tables are not nullable, thus this should not be an issue in this case.

8.7.1. Install and configure OpenLink ODBC Driver for Sybase

The Virtuoso Virtual Database engine uses ODBC as the connectivity mechanism for linking remote database objects into its local schema. Thus an Sybase ODBC Driver must be available with a suitably configured DSN for connecting to the target database. The OpenLink ODBC Drivers for Sybase have been used in this section, although in theory any Sybase ODBC Driver can be used.

Installation instructions for the OpenLink ODBC Driver for Sybase are available from:

- Product Installation & Basic Configuration (ODBC)

8.7.2. Install and configure OpenLink Virtuoso Universal Server

- Install and configure OpenLink Virtuoso Universal Server

8.7.3. Linking Sybase tables into OpenLink Virtuoso

1. Start the Virtuoso Web User Interface

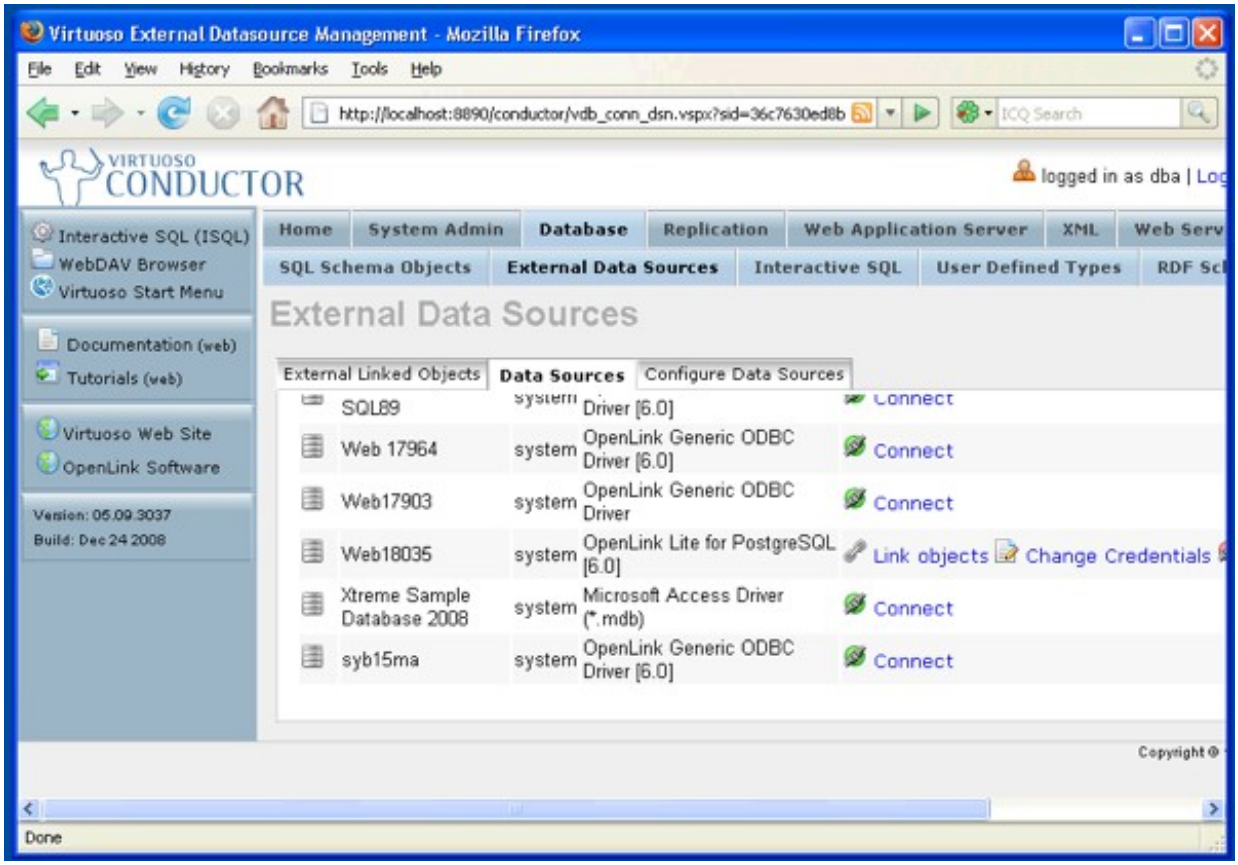
   Figure 8.325. Start

2. Select the "Conductor" link in the Left Frame and Login as the "dba" user.

Figure 8.326. Conductor

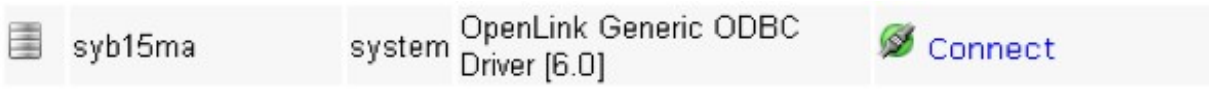
3. Select the "Databases" -> "External Data Source" -> "Data Sources" tab menu items

Figure 8.327. Databases



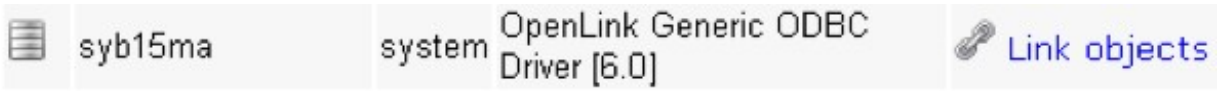
4. Select the "Connect" button for the "syb15ma" Sybase DSN.

Figure 8.328. Connect



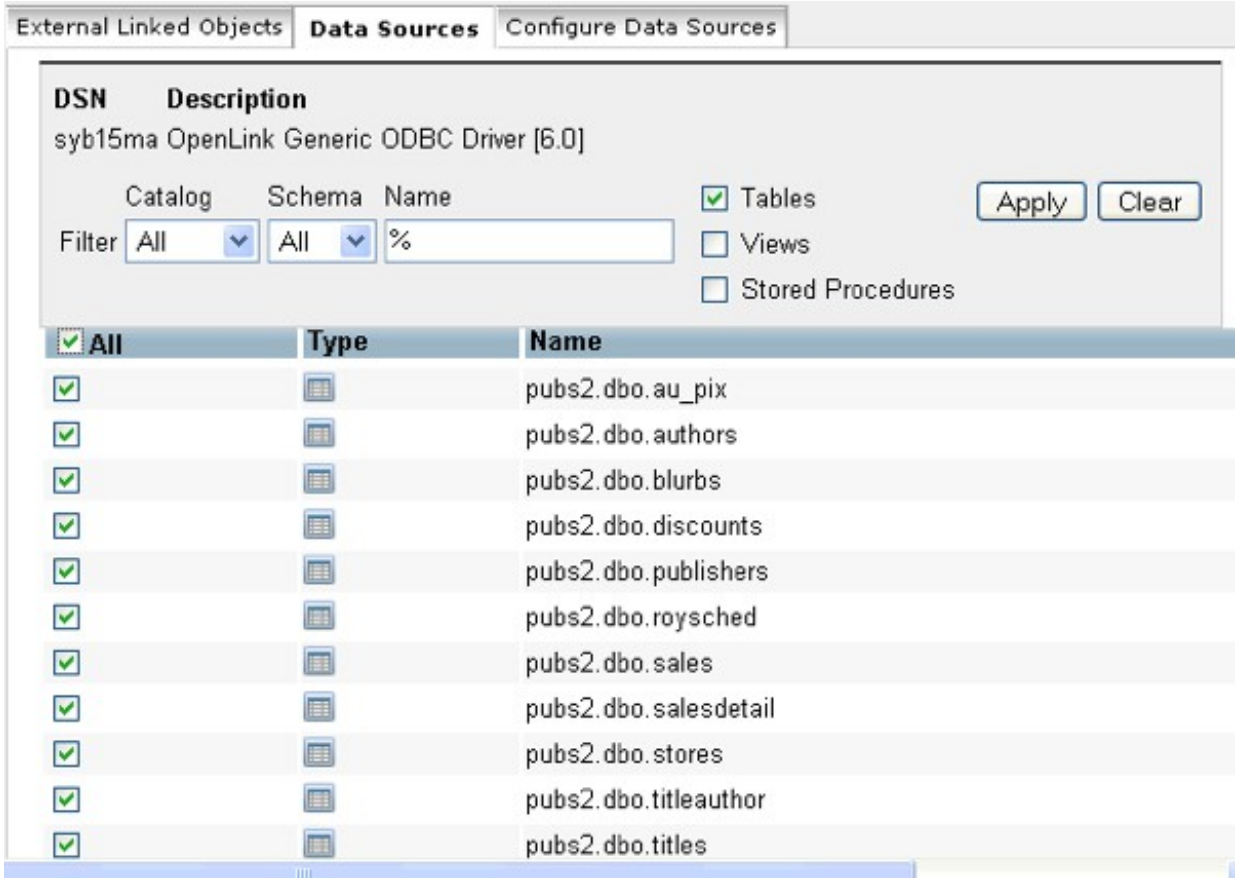
5. On successful connection Select the "Link Objects" button to obtain a list of available tables

Figure 8.329. Link Objects



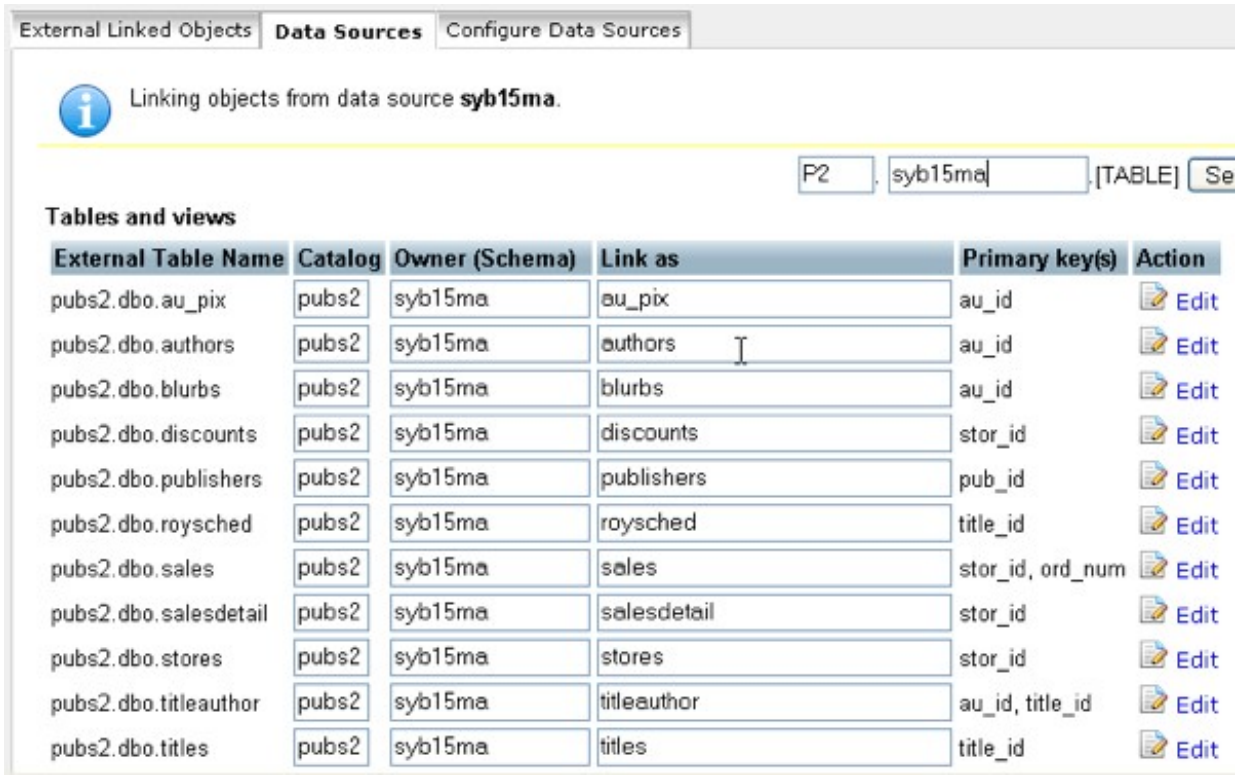
6. Select all the tables that are part of the "pubs2" catalog.

Figure 8.330. Select tables



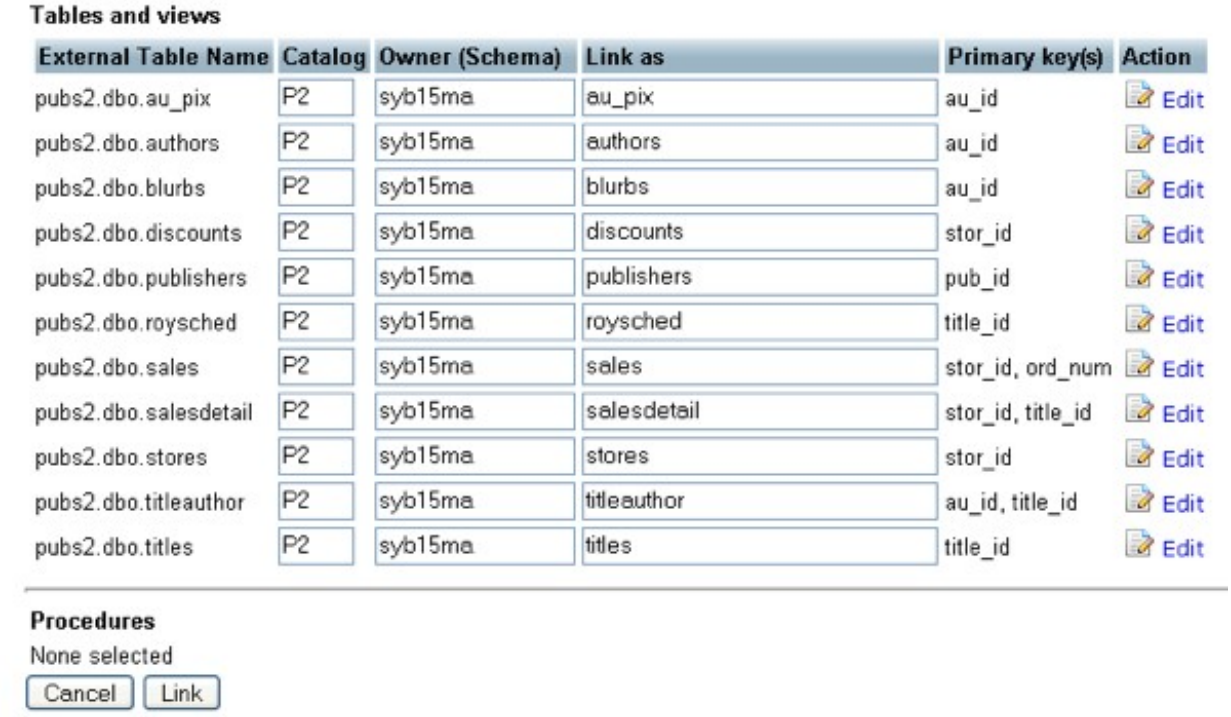
7. Change the Catalog for each table to be "P2" using the "Set All" button.

Figure 8.331. Catalog



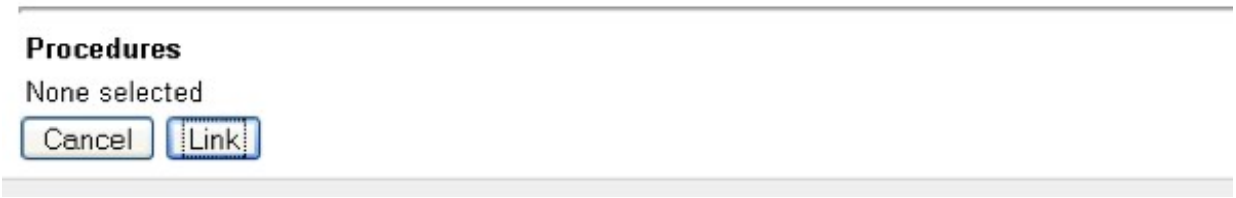
8. All the catalog names are changed to be "P2".

Figure 8.332. catalog names



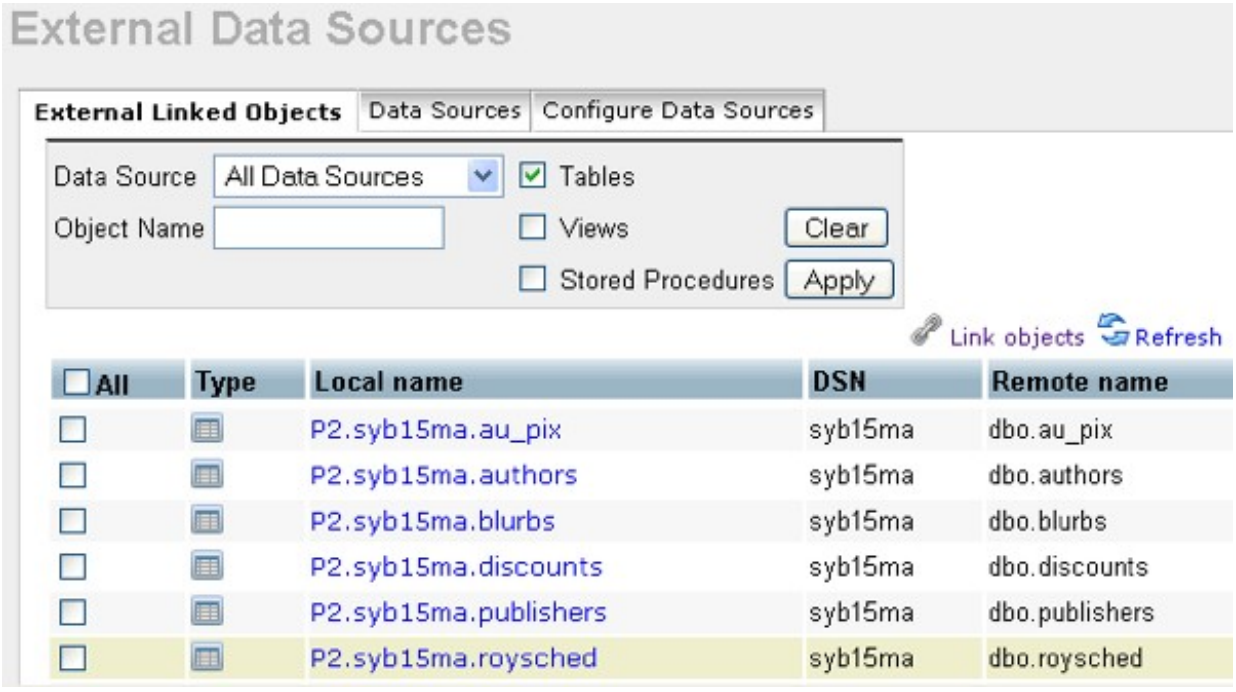
9. Select the "Link" button to link the selected tables into Virtuoso

Figure 8.333. "Link" button



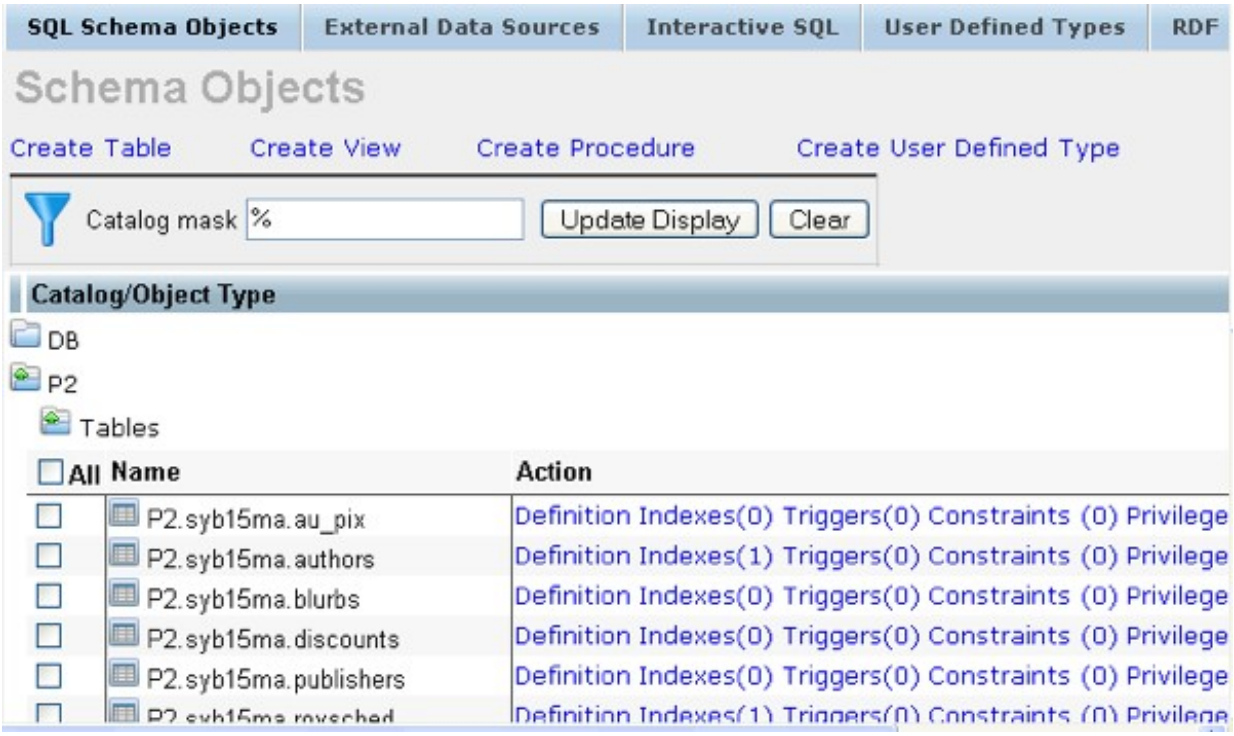
10. On completion of the Link process the tables will be displayed in the "External Linked Objects" tab.

Figure 8.334. Completion



11. The tables can also be viewed as part of the Virtuoso "SQL Schema Objects" under the "P2" catalog name.

Figure 8.335. view tables



The Link process is now complete enabling the tables to be queried as if part of the Virtuoso Schema.

8.7.4. Creating EDM in Visual Studio 2008

The following steps can be used to create an Entity Data Model (EDM) for the Sybase pubs2 database:

1. Launch the Visual Studio 2008 SP1 IDE.

Figure 8.336. Visual Studio 2008 SP1 IDE

2. Create a

   Web Application

   project by going to the

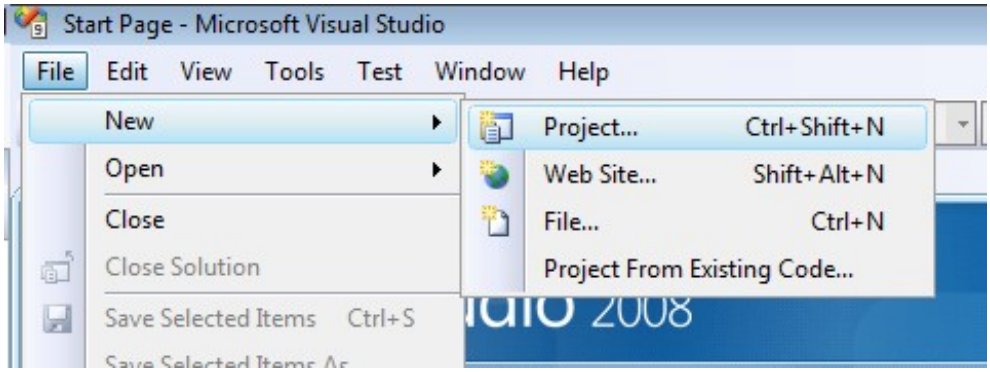
   File

menu in Visual Studio and choosing

New Project

.

Figure 8.337. Web Application



3. When the New Project window appears, choose either

   Visual Basic

   or

   Visual C#

   as your programming language.
4. Within the language category, click on

   Web

   and select

   ASP.NET Web Application

   from the right-hand panel.
5. Choose a name for the project, for example

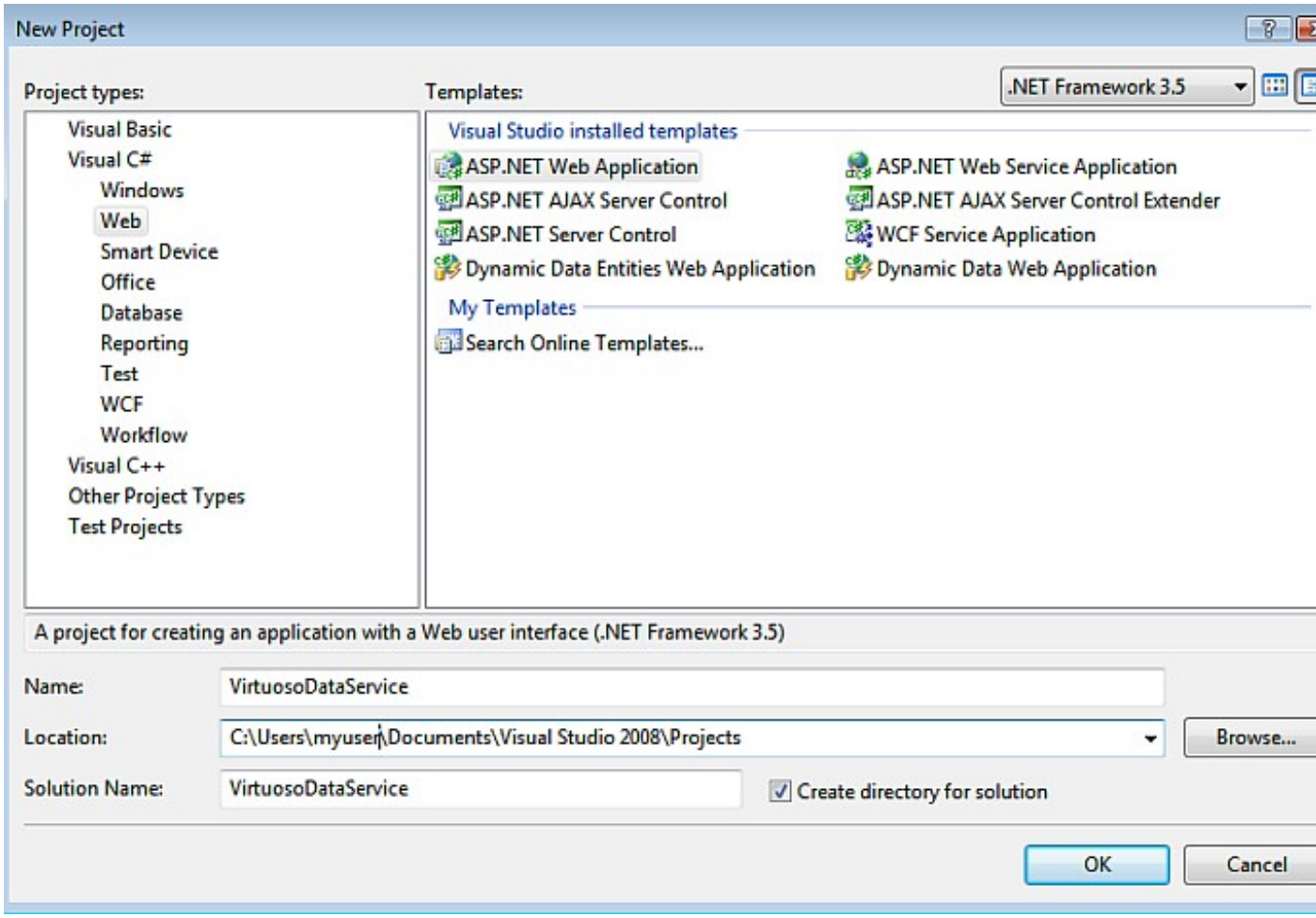
   VirtuosoDataService

   , and click

   OK

   .

Figure 8.338. name for the project

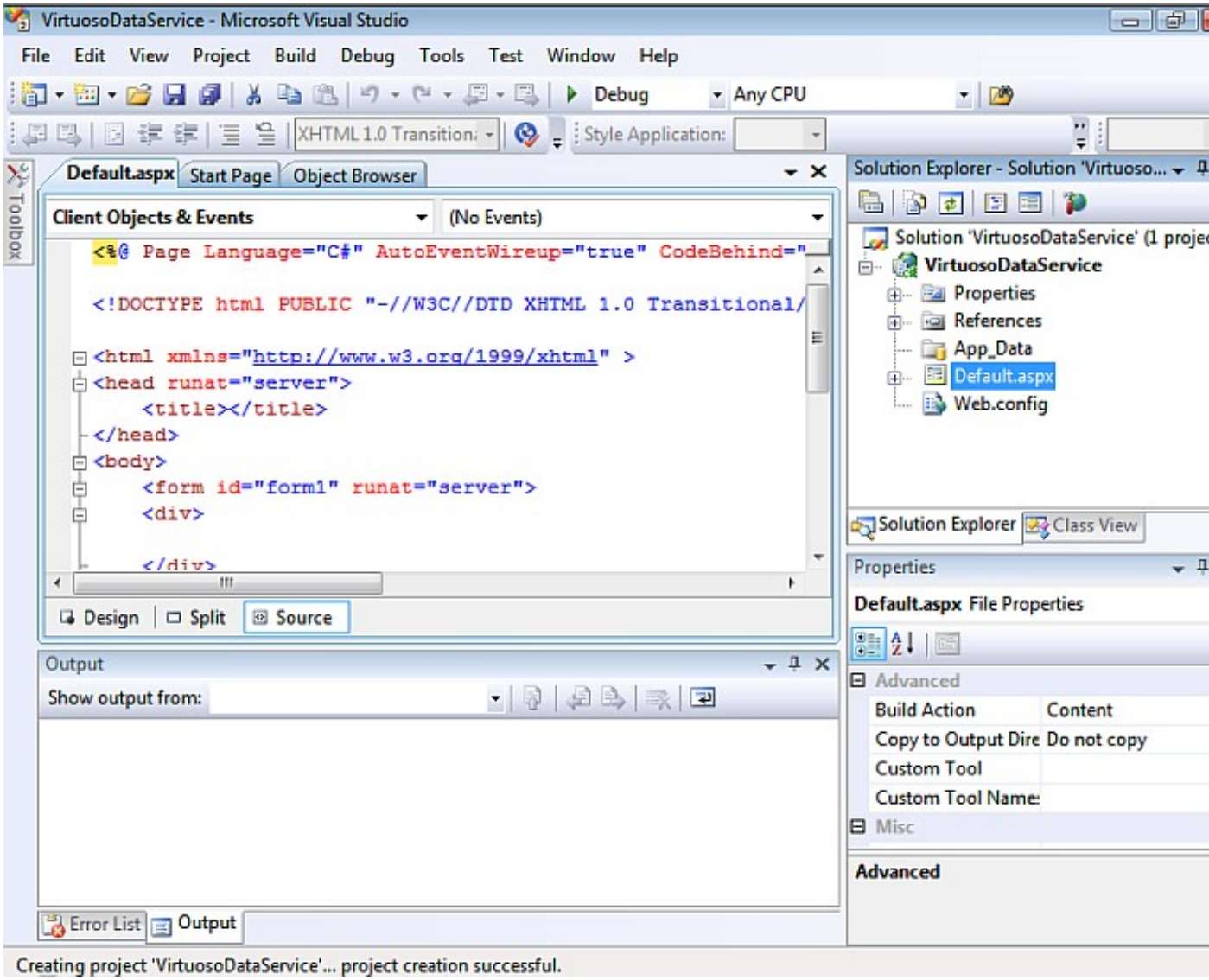


6. This will create a new project called

   VirtuosoDataService

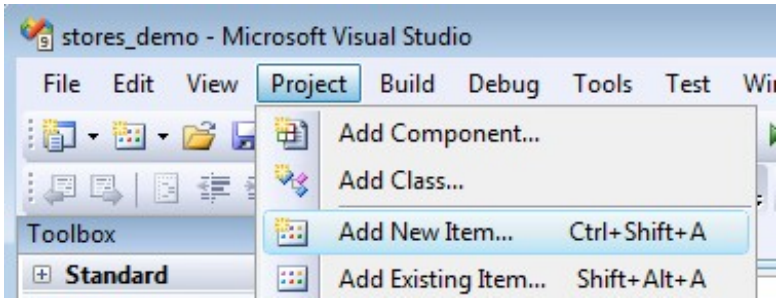
   .

Figure 8.339. create a new project



7. Select the Project -> Add New Item menu option.

Figure 8.340. VirtuosoDataService



8. The

Add New Item

dialog will appear. Choose the

ADO.NET Entity Data Model

template. Give it the name

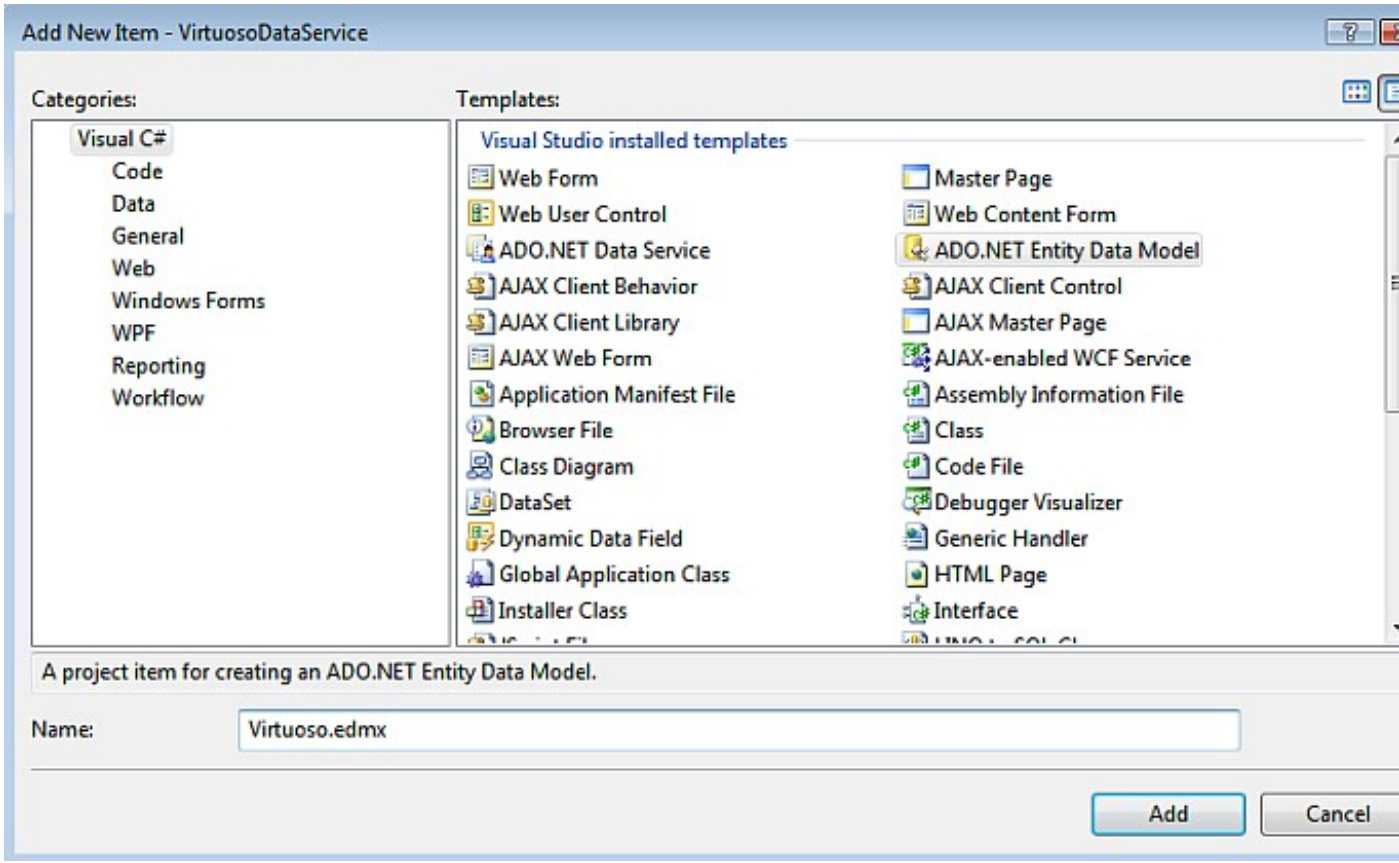
Virtuoso.edmx

and click

Add

to start the creation of the ADO.Net Entity Data Model.

Figure 8.341. Add New Item



9. In the

Entity Data Model Wizard

dialog

Choose Model Contents

page select the

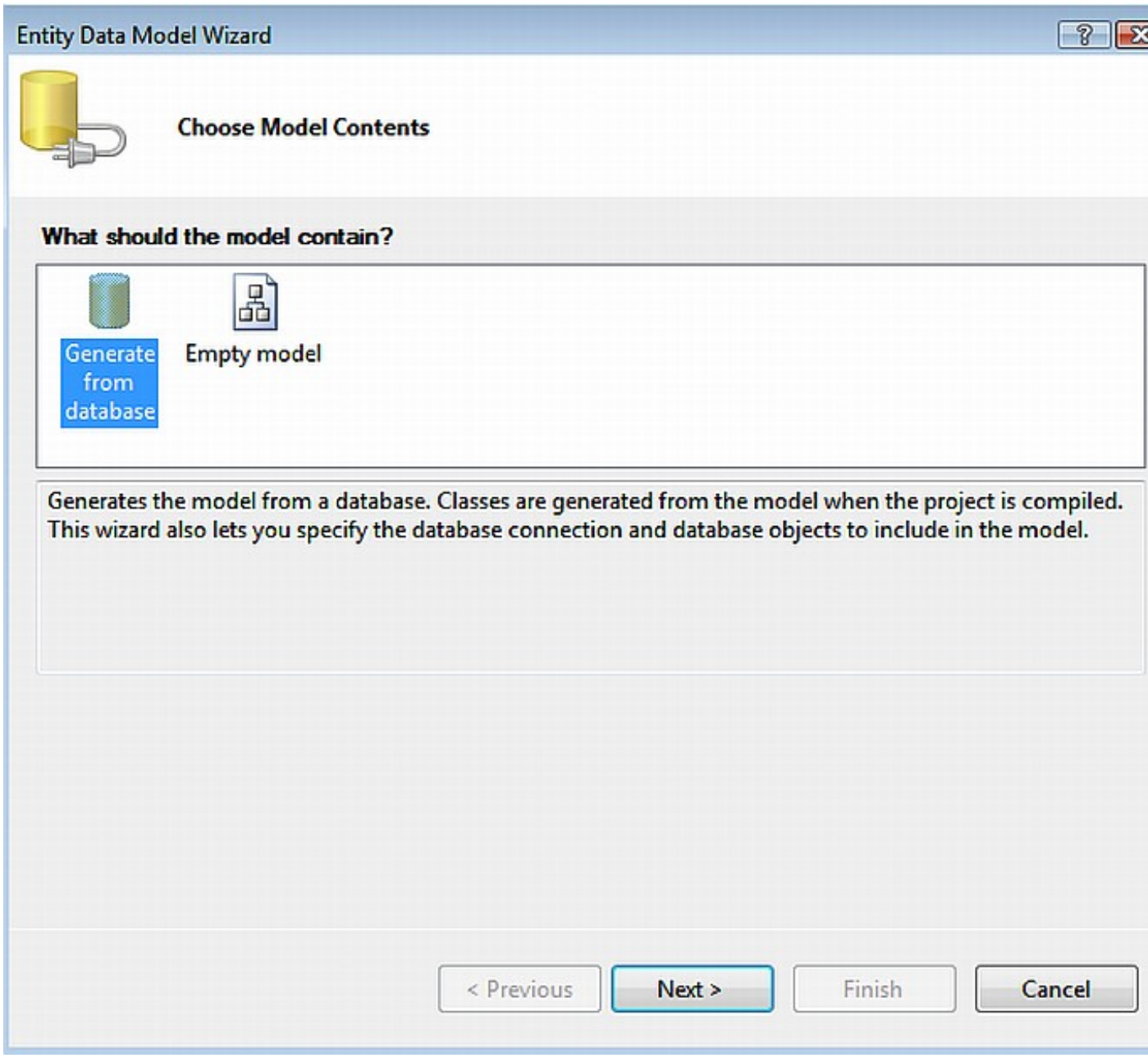
Generate from Database

model type and click

Next

.

Figure 8.342. Choose Model Contents



10. In the

    Entity Data Model Wizard

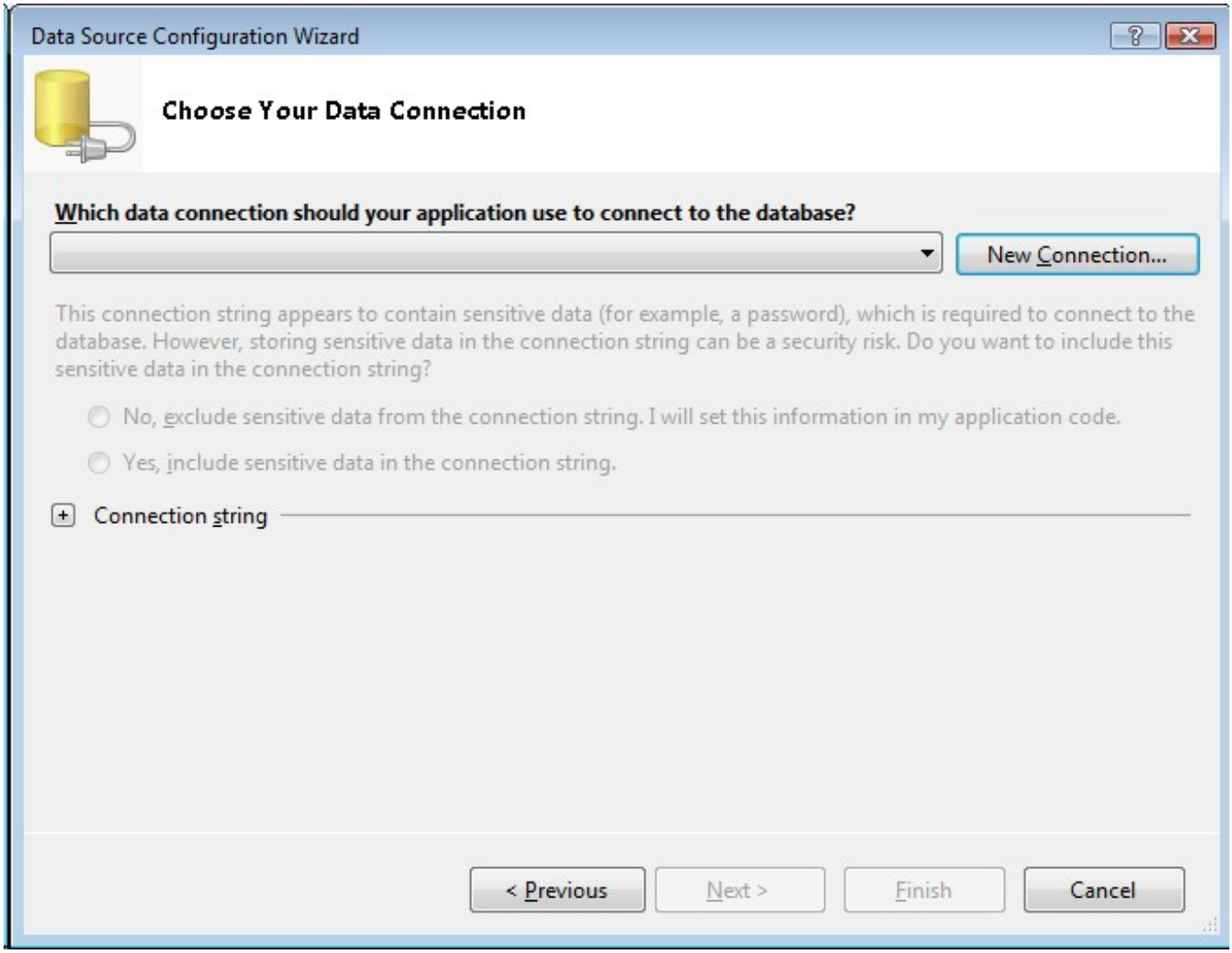
    dialog

    Choose your Data Connection

    page, select the

    New Connection button

Figure 8.343. Entity Data Model Wizard



11. In the

    Choose Data Source

    dialog, select the OpenLink

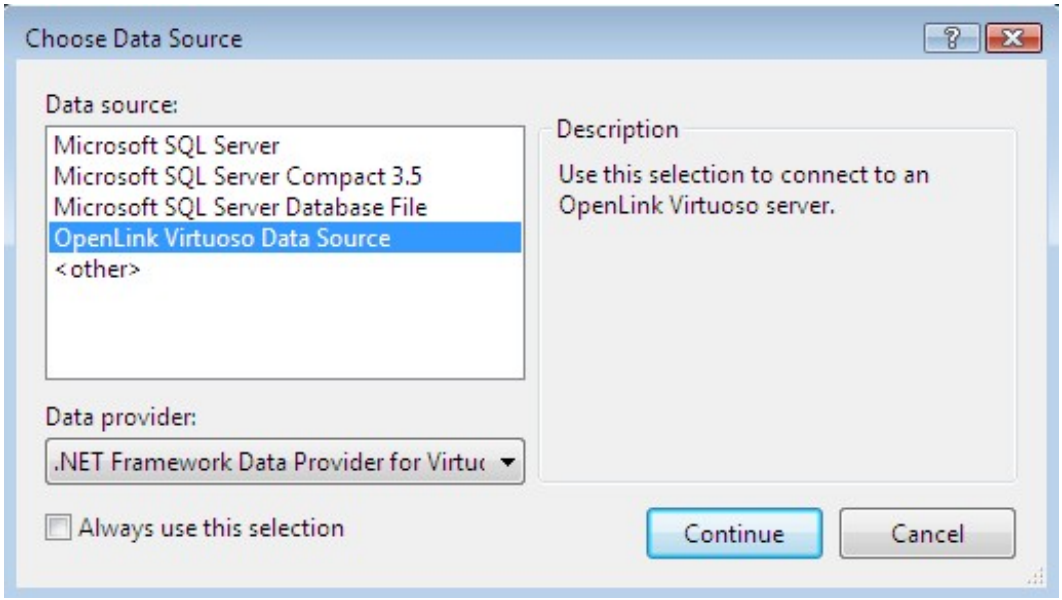
    Virtuoso Data Source

    from the list and click

    Continue

    .

Figure 8.344. Choose Data Source



12. In the

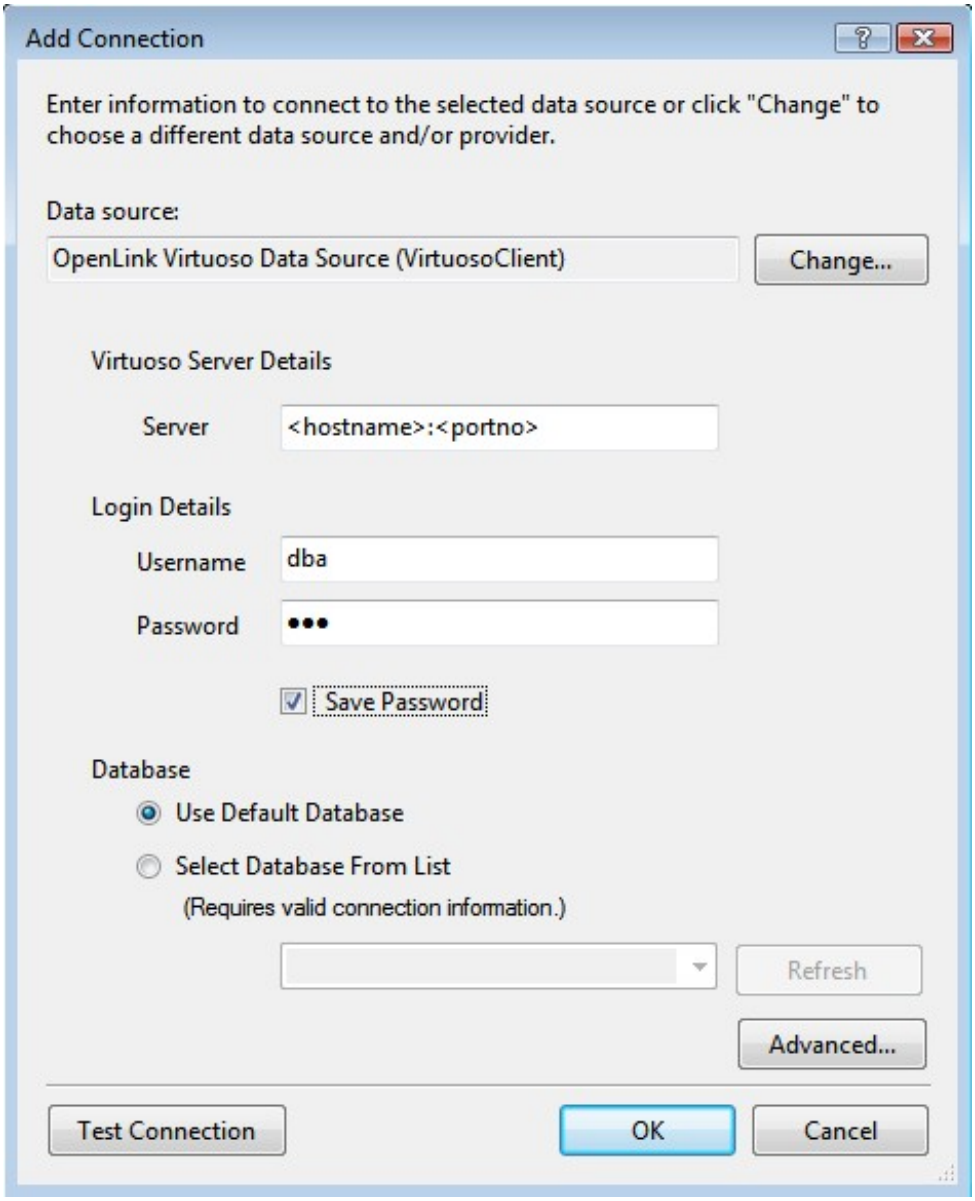
    Add Connection

    dialog, specify the

    hostname, portno, username and password

    for the target Virtuoso Server and check the Save Password check box.

    Figure 8.345. Connection Properties



13. Select the

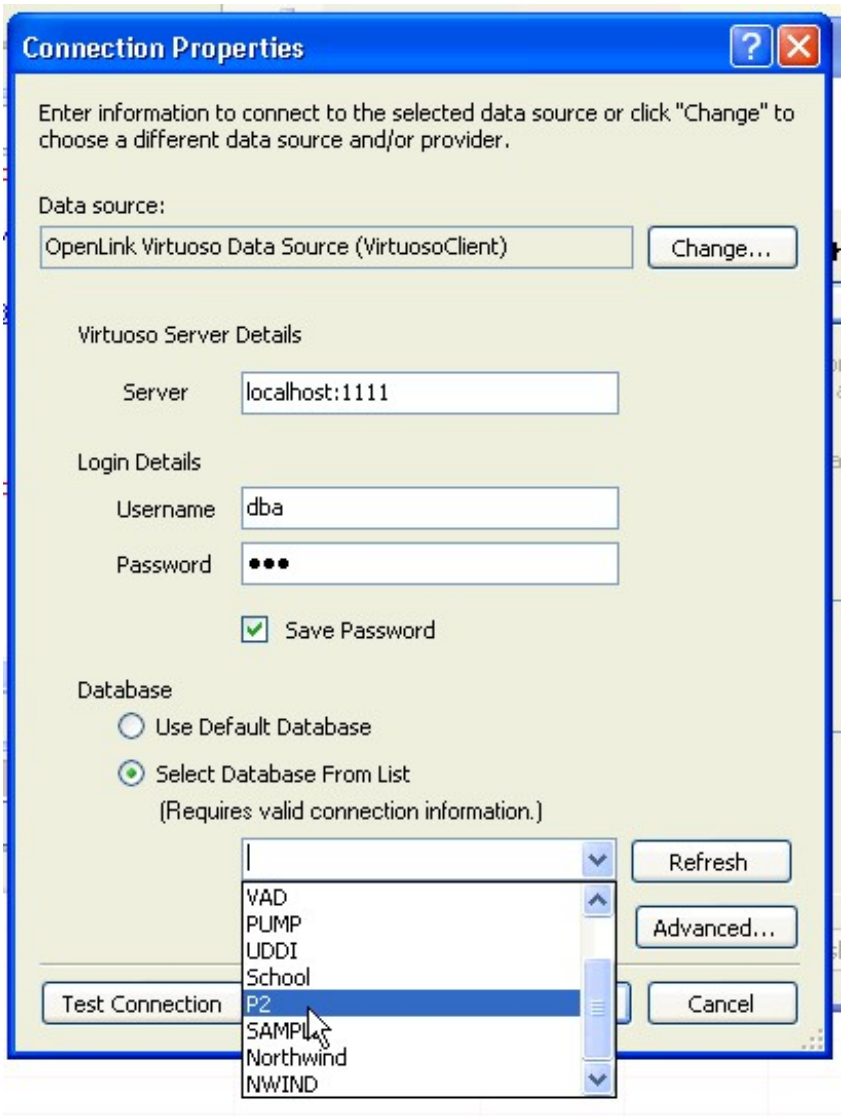
    Select Database From List

    radio button and choose the

    SAMPLE

    database from the drop down list.

Figure 8.346. Add connection

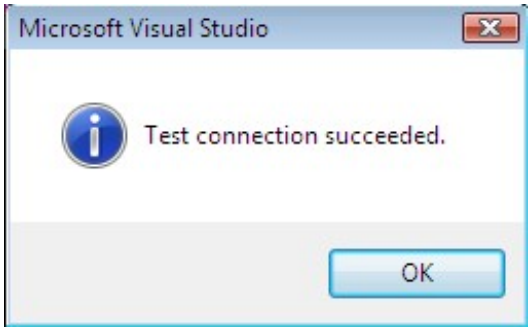


14. Press the

    Test Connection

    dialog to verify that the database is accessible.

    Figure 8.347. Test Connection



15. Set the

    entity connect string

    name to

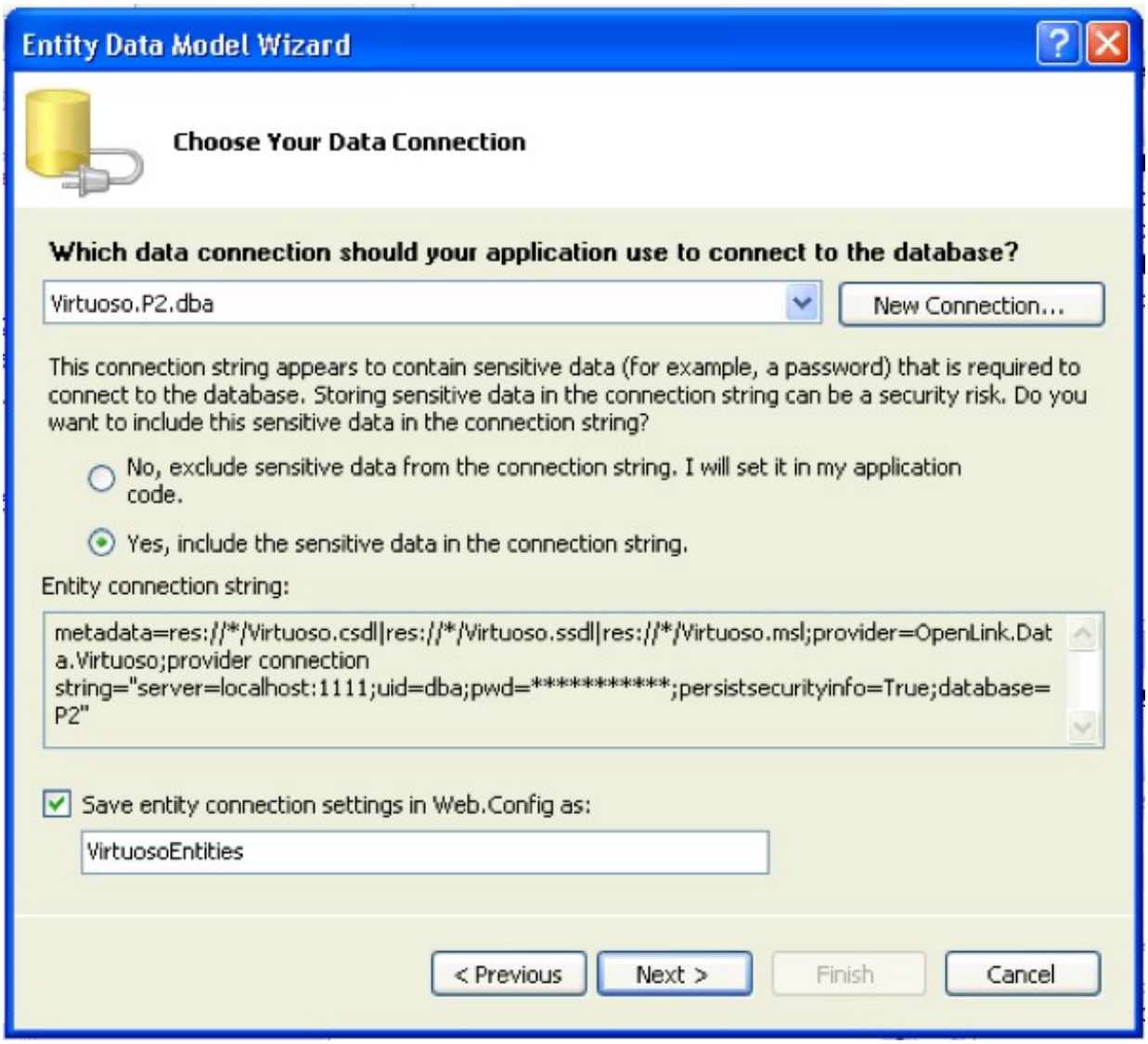
VirtuosoEntities

(note this name as it is required in the section on creating and ADO.Net Data Service) and click

Next

.

Figure 8.348. entity connect string



16. In the

Choose your Database Objects

page select the

Tables

check box to select all tables in the P2 catalog for addition to the Entity Data Model. Set the

Model Namespace

to

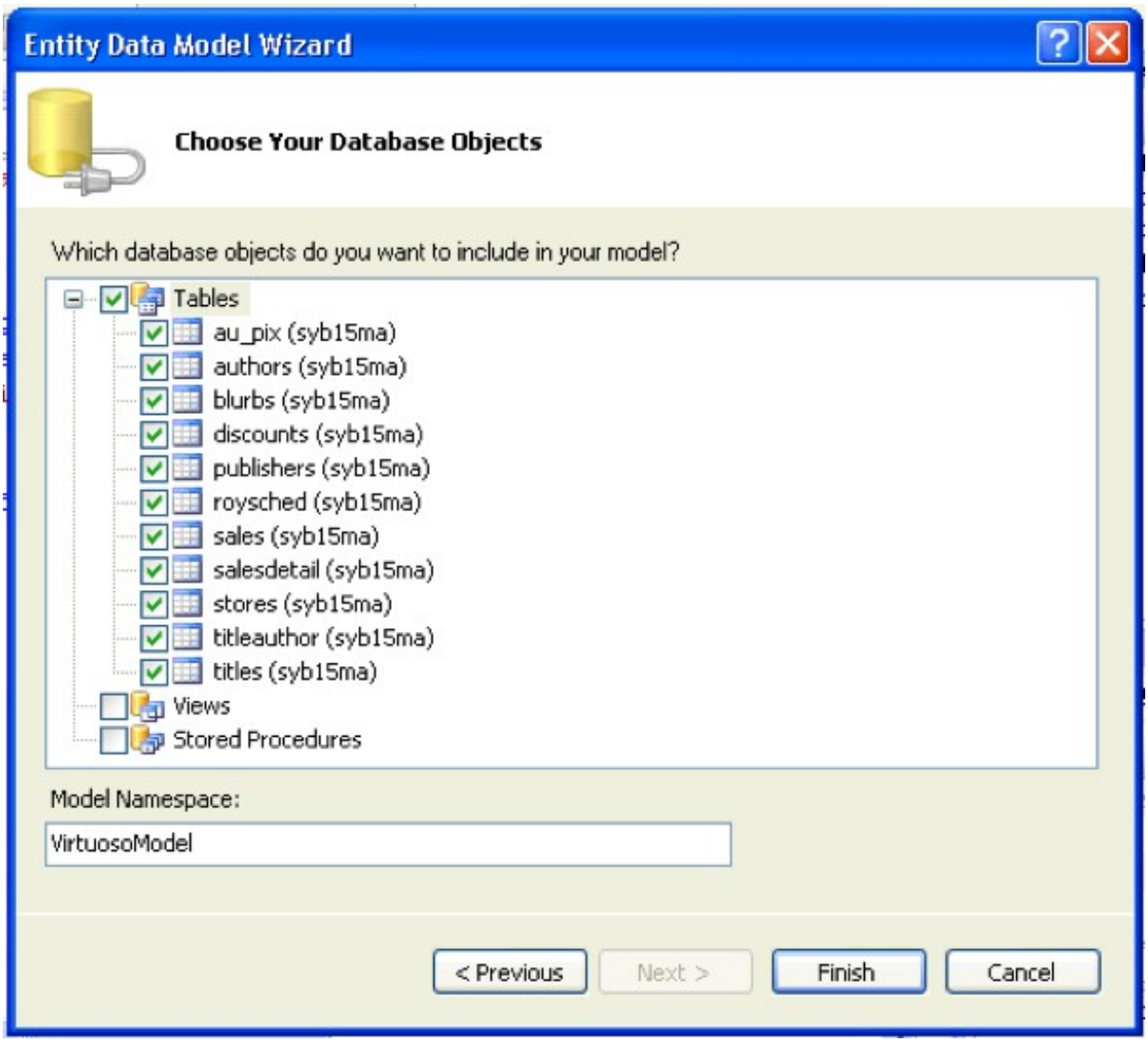
VirtuosoModel

and click

Finish

.

Figure 8.349. Database Objects

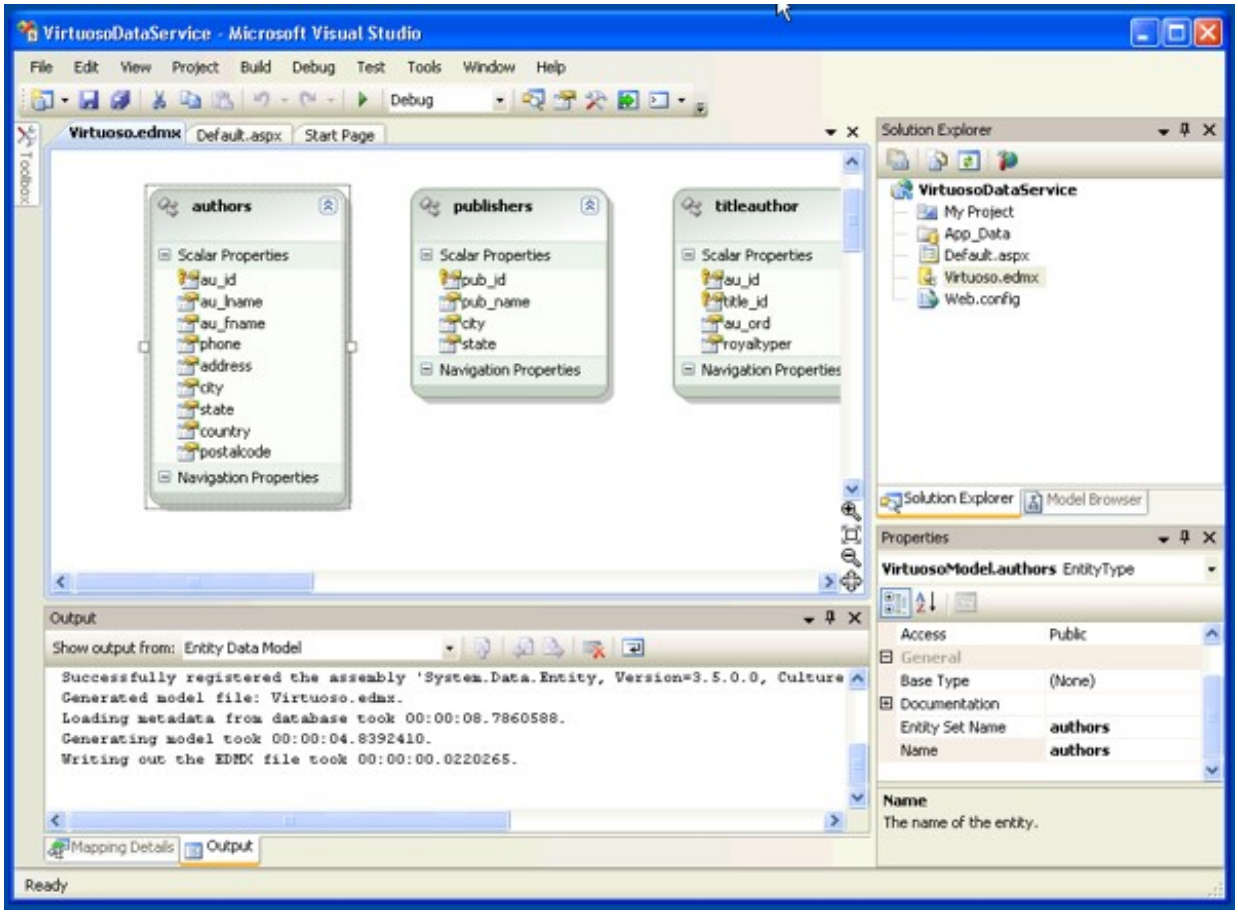


17. The

Virtuoso.edmx

EDM will be created with the tables and relationships displayed in the Visual Studio IDE

Figure 8.350. Virtuoso.edmx



Creation for the Entity Data Model for the Sybase pubs2 database is now complete.

8.7.5. Manually creating EDM Associations (FKs) for the Sybase pubs2 database

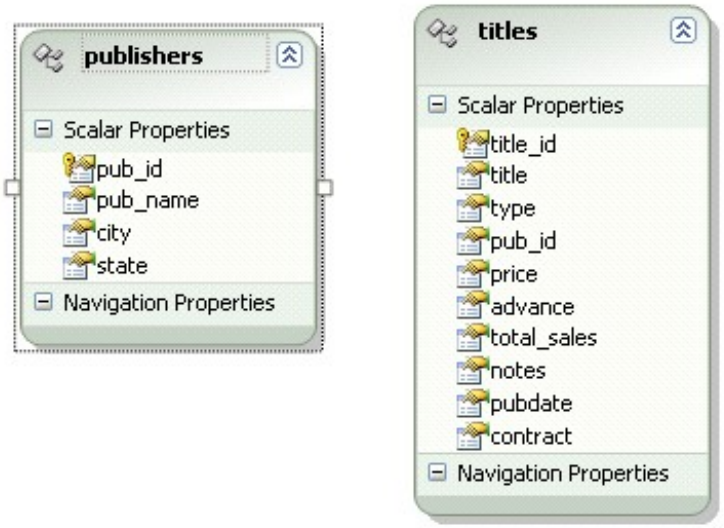
The following steps will detail what is required to manually create "Associations" in your Entity Data Model.

You will need to determine where these associations exist and their multiplicity (one-to-one, one-to-many, etc.) before commencing with the following steps.

Note: These steps will need to be repeated for each association.

1. The first Association will deal with is the relationship between publishers and titles, identified by the presence of the scalar property pub_id in both entities. This is a one-to-many relationship, as a Customer may have any number of Invoices.

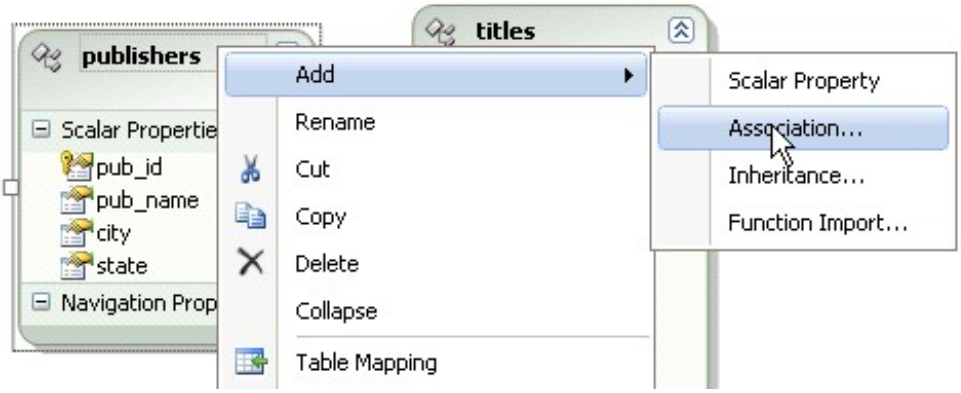
   Figure 8.351. Association



This is a one to many relationship insomuch that publishers may have zero or many titles.

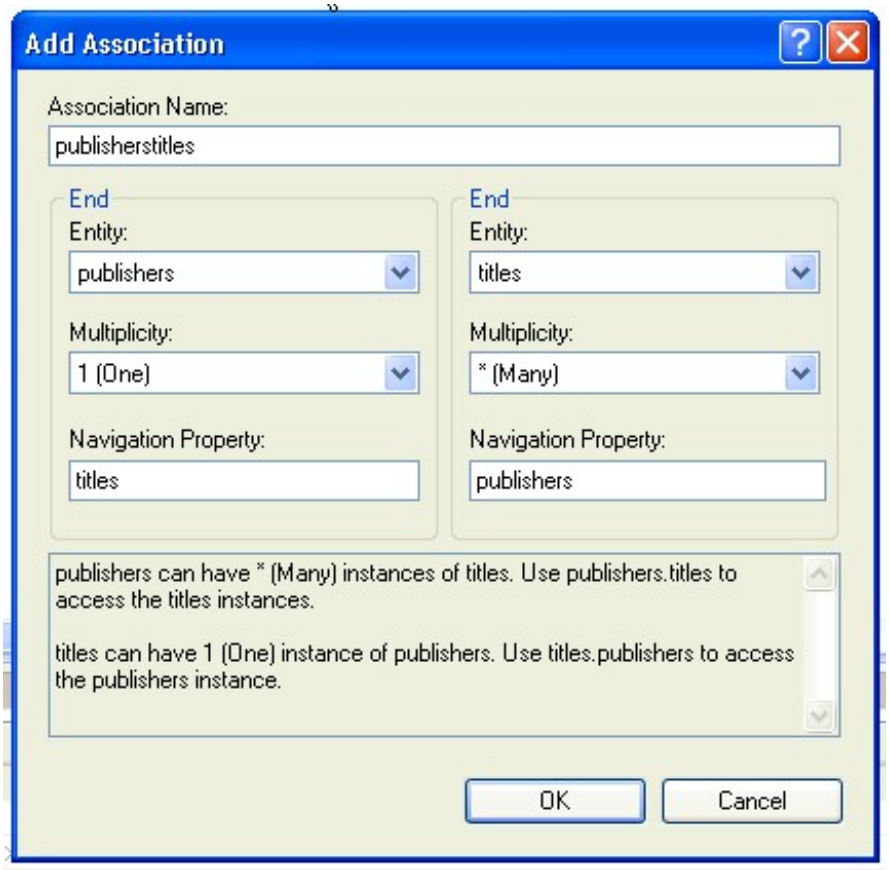
2. To add the Association, right click on the publishers entity then Add -> Association.

Figure 8.352. add the Association



3. You will now see the Add Association dialog.

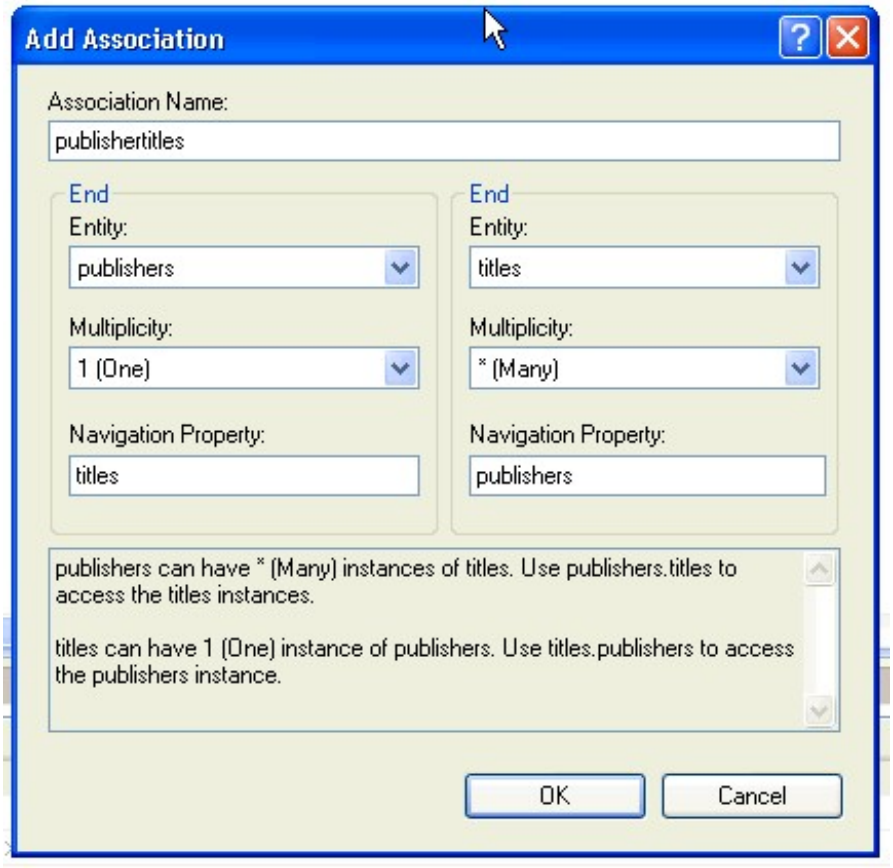
Figure 8.353. Add Association dialog



4. For this association the only thing that needs changing is the name of the Navigation Property from publishers to publisher on the publishers end of the association.

   This more correctly reflects the multiplicity of the association such that a publisher is associated with zero or many titles (plural).

   Figure 8.354. Navigation Property name

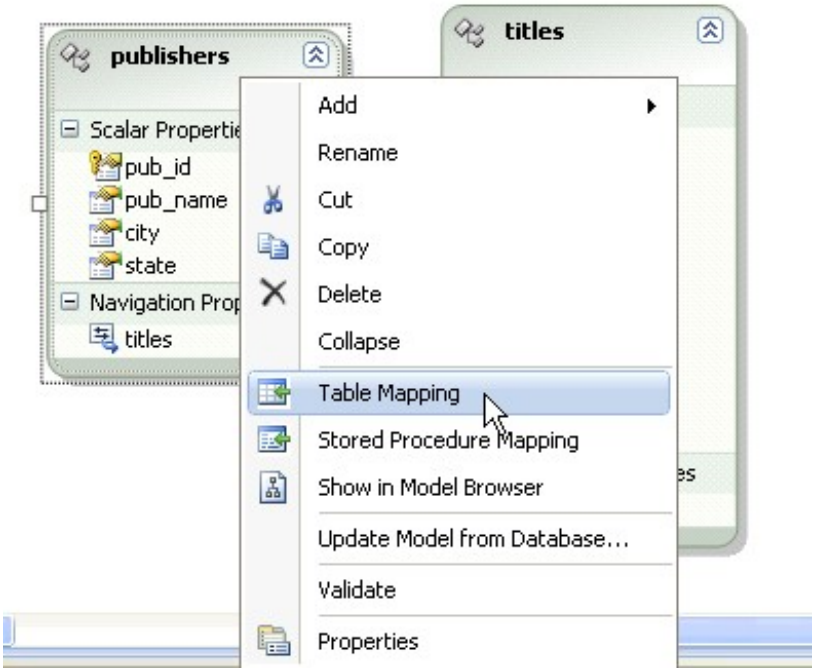


5. Once you then hit OK the diagram is refreshed to include the newly created association.

Figure 8.355. diagram

6. You now need to edit the mappings associated with the newly created association, so right-click the association on the diagram, and select Table Mapping to display the Mapping Details pane.

Figure 8.356. Table Mapping

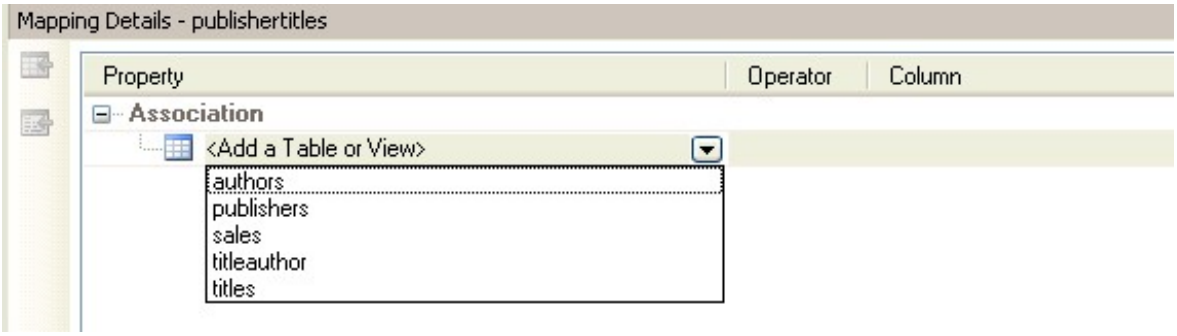


7. Click that line reading <Add a Table or View> to reveal a drop down list of all entities.

Figure 8.357. Add a Table or View

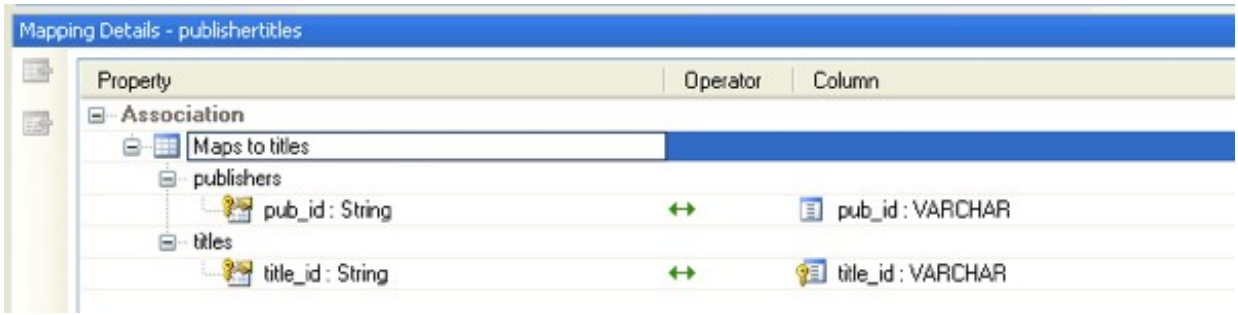
8. Here you need to select the entity on the right/far side of the association (the entity where the foreign key exists). In this example, it is the titles entity.

Figure 8.358. Entity



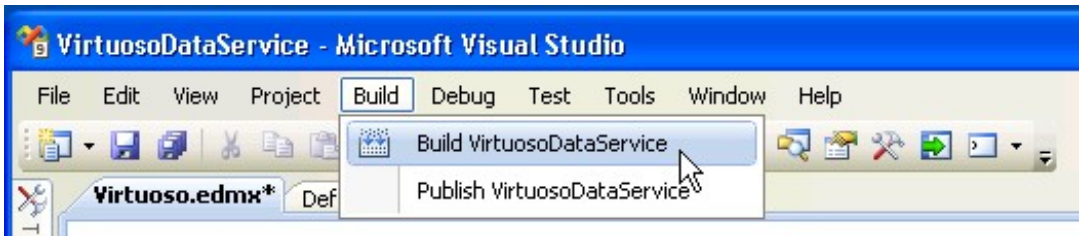
9. The Mapping Details pane now refreshes to display both ends of the association, requiring that you provide relevant target store data types in the Column column for the key fields, as depicted here.

Figure 8.359. Mapping Details



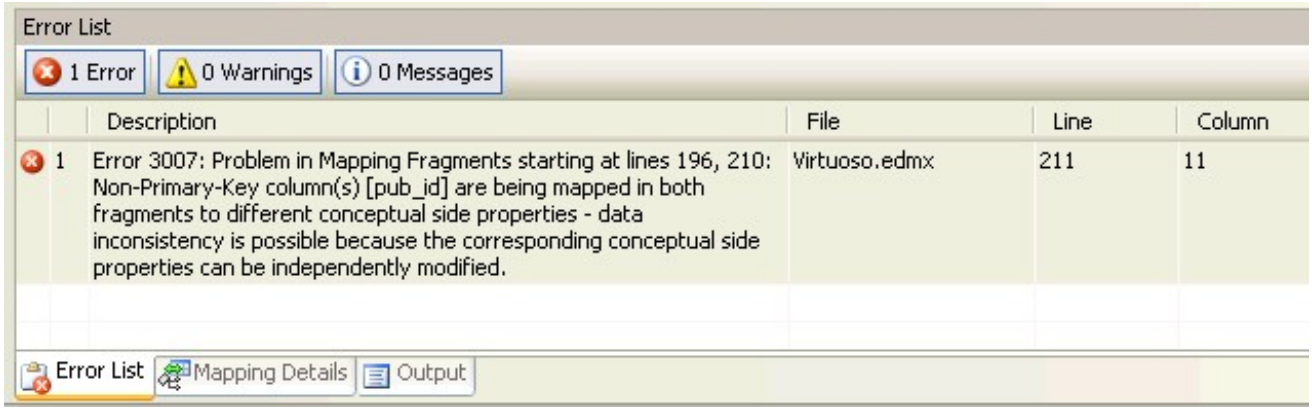
10. Once the mapping is complete, you can build the project using Build -> Build Solution. NOTE: It is worthwhile building as each association is made, since the error messages can be a little confusing.

Figure 8.360. Build the project



11. This should result in the following error:

Figure 8.361. Error

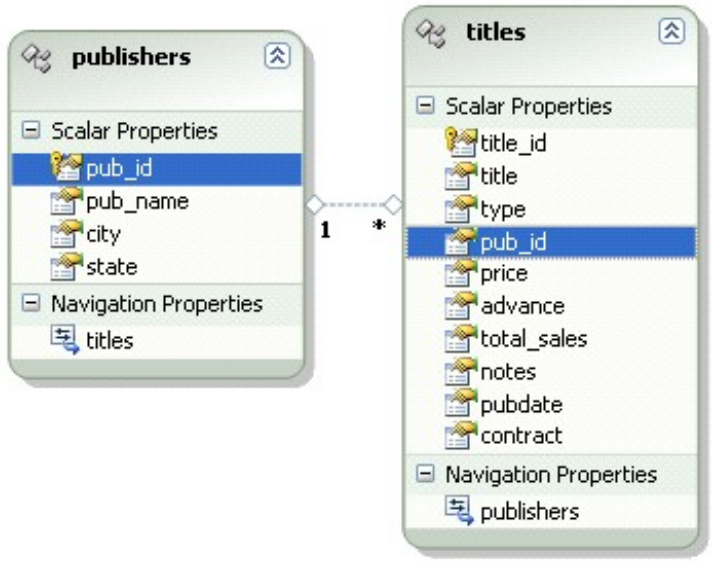


It seems that this error is attempting to say that there are two mappings which map to the same target source column.

In this case it is the target column titles.pub_id.

That is, the scalar property pub_id and the Navigation Property publishers in the Invoice entity, both map to the Sybase table column titles.pub_id - which is not supported

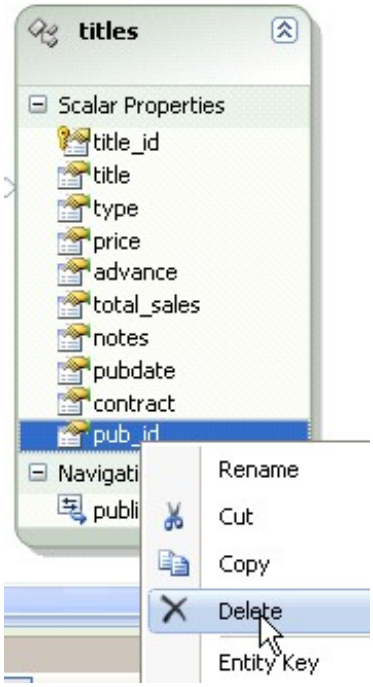
Figure 8.362. Error



12. The solution is simple! Simply delete the scalar property titles.pub_id since its purpose is only to hold data representing a relationship/association (it is a Foreign Key) which has already been represented by the newly created association and resulting Navigation Property publishers.

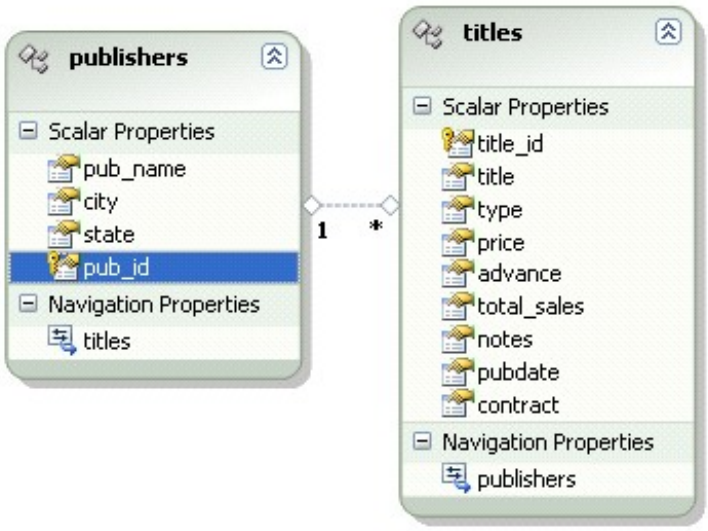
    Right click on titles.pub_id then Delete.

    Figure 8.363. delete the mapping



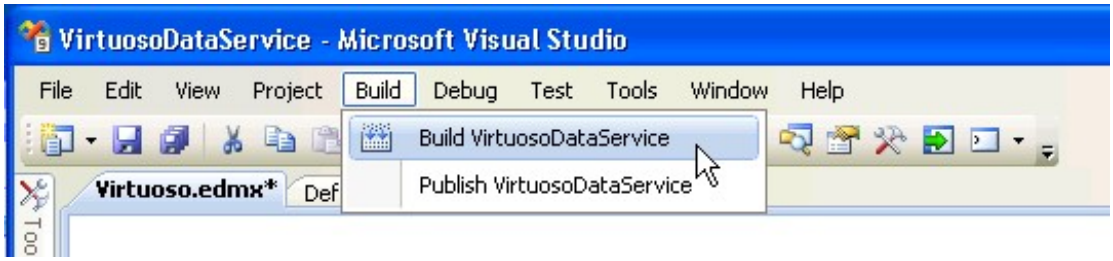
13. The model diagram will refresh to reflect this change.

    Figure 8.364. Model Diagram



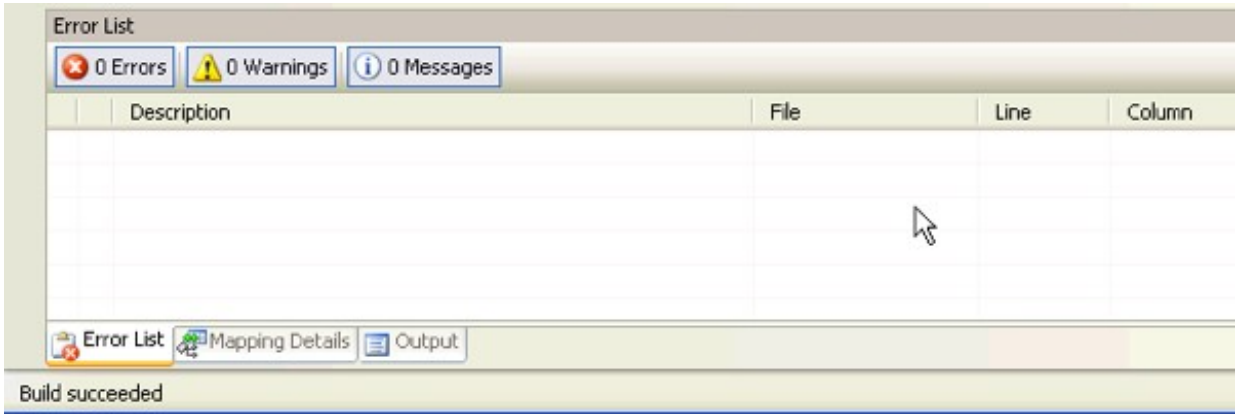
14. Build the project, again, using Build -> Build Solution.

Figure 8.365. Build project



15. The project should now be fine.

Figure 8.366. Build Project



You will need to repeat these steps for each association until you have a completed Entity Data Model.

Figure 8.367. Entity Data Model

8.7.6. Using EDM to create Entity Framework based applications

Now that a Microsoft Entity Data Model has been created for the Sybase pubs2 database, Entity Framework applications can be created to make use of it.

Entity Frameworks based ADO.NET Data Service

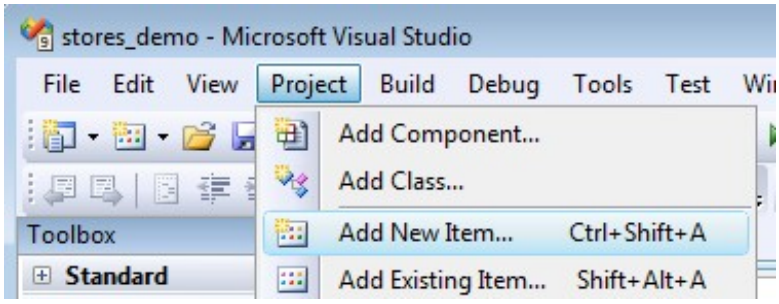
An ADO.Net Data Service for the Sybase tables can be created using the Entity Data Model created in the Creating EDM in Visual Studio 2008 section .

1. Open the

   VirtuosoDataService

   project created in the Creating EDM in Visual Studio 2008 section .
2. Select the Project -> Add New Item menu option.

   Figure 8.368. VirtuosoDataService

3. The

   Add New Item

   dialog will appear. Choose the

   ADO.NET Data Service template

   . Give it the name

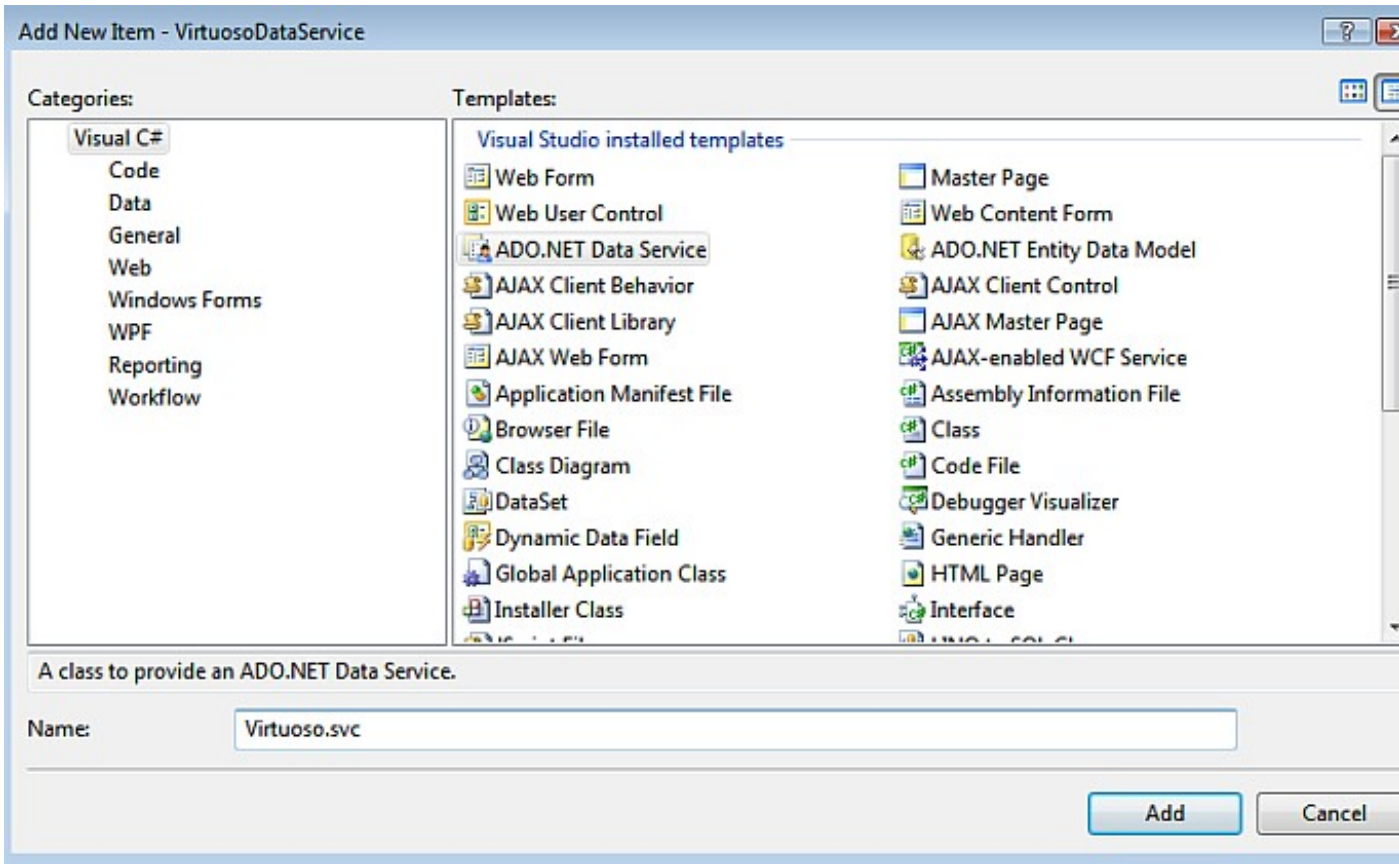
   Virtuoso.svc

   , and click

Add

to create the ADO.Net Data Service.

Figure 8.369. Add New Item



4. In the newly created

   Virtuoso.svc.cs

   Data Service file, add the data source class name of

   VirtuosoEntities

   (note this is the name set in the Creating EDM in Visual Studio 2008 section) as the

   DataService

   name. Enable the access to the Data Service by adding the entry

   config.SetEntitySetAccessRule("", EntitySetRights.All);*

   in the

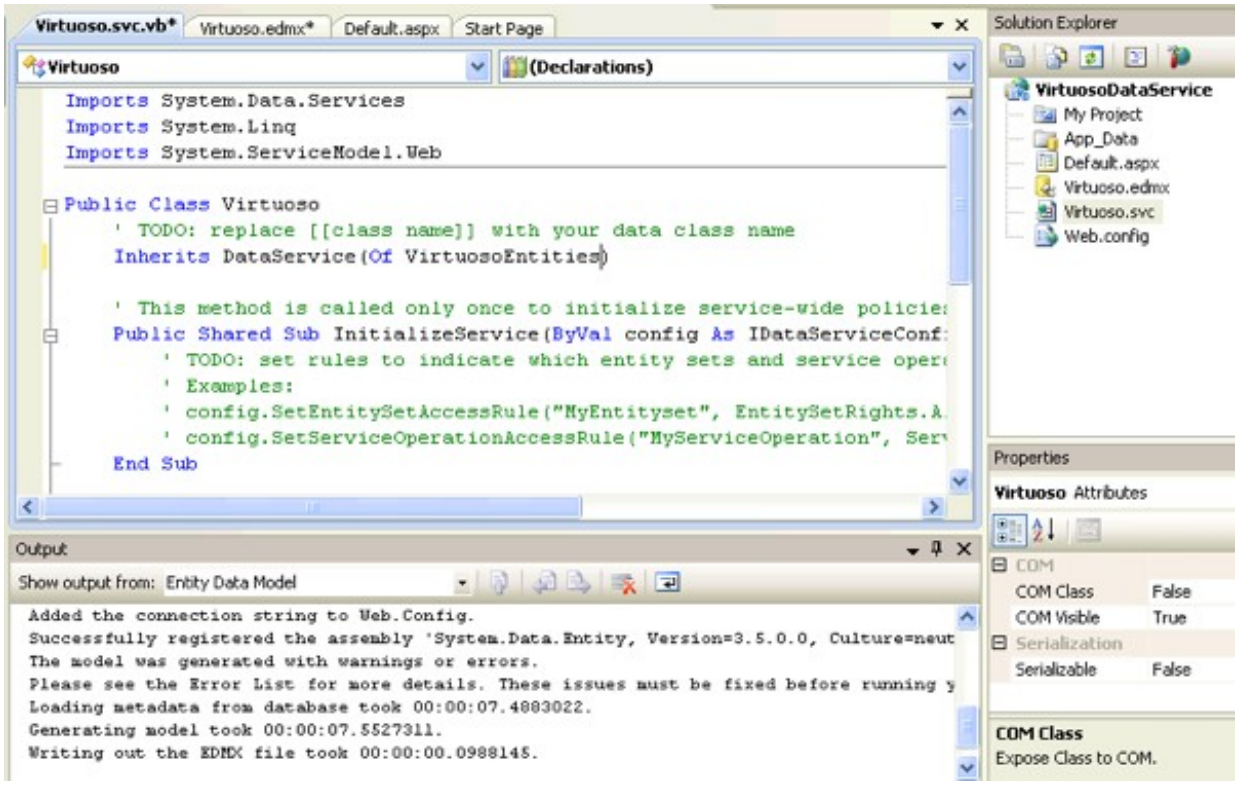
   InitializeService

   method.

```
// C#

using System;
using System.Web;
using System.Collections.Generic;
using System.ServiceModel.Web;
using System.Linq;
```

```
using System.Data.Services;

namespace SimpleDataService
{
    public class Northwind : DataService<VirtuosoDemoEntities>
    {
        public static void InitializeService(IDataServiceConfiguration  config)
        {
            config.SetEntitySetAccessRule("*", EntitySetRights.All);
        }
    }
}
```

Figure 8.370. Virtuoso.svc.cs

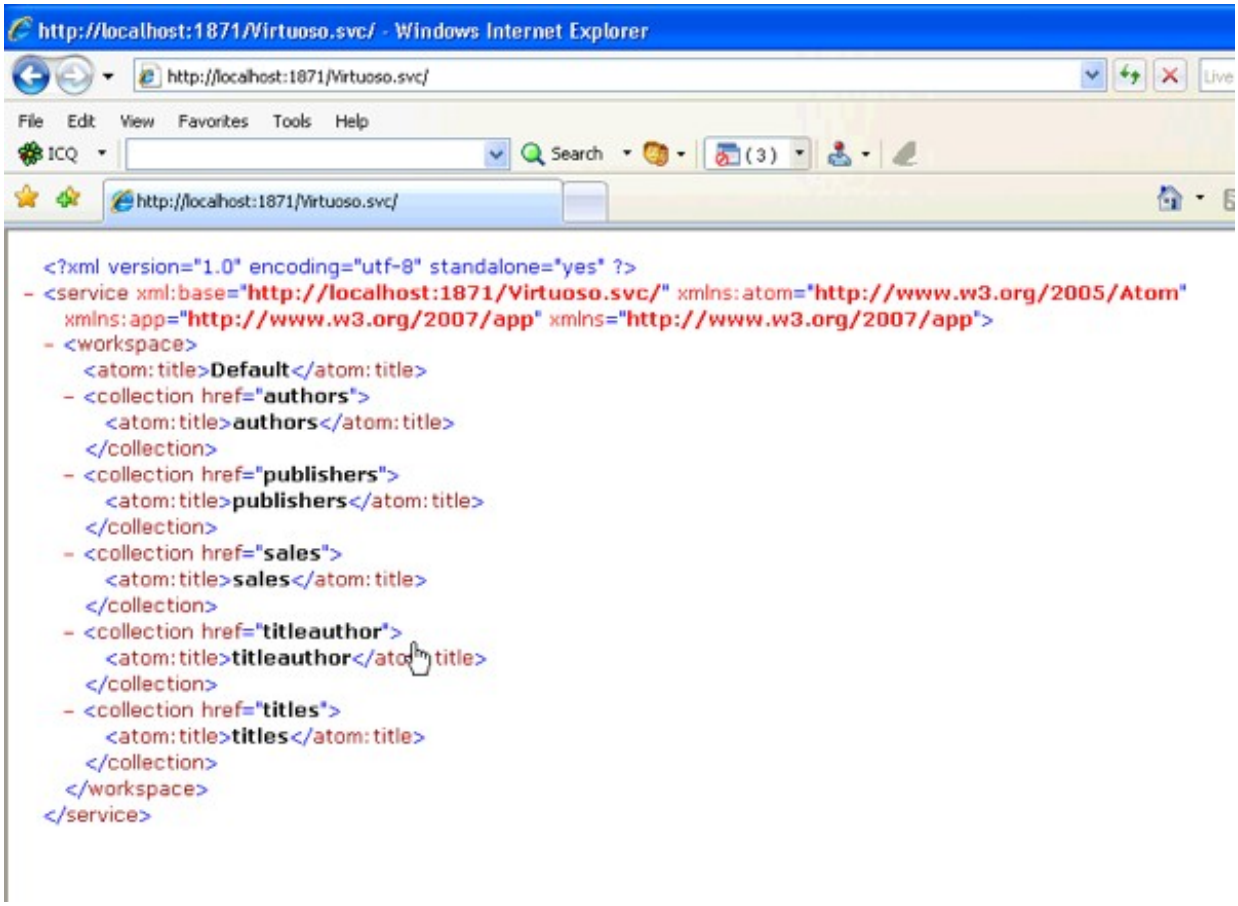


5. To test the Data Service, simply hit

   Ctrl+F5

   within Visual Studio. This will start the development web server, run the Data Services server inside and load a Web browser page displaying the list of available tables/entities for the P2 database catalog.

Figure 8.371. Data Service test



6. To access a specific entity instance like the

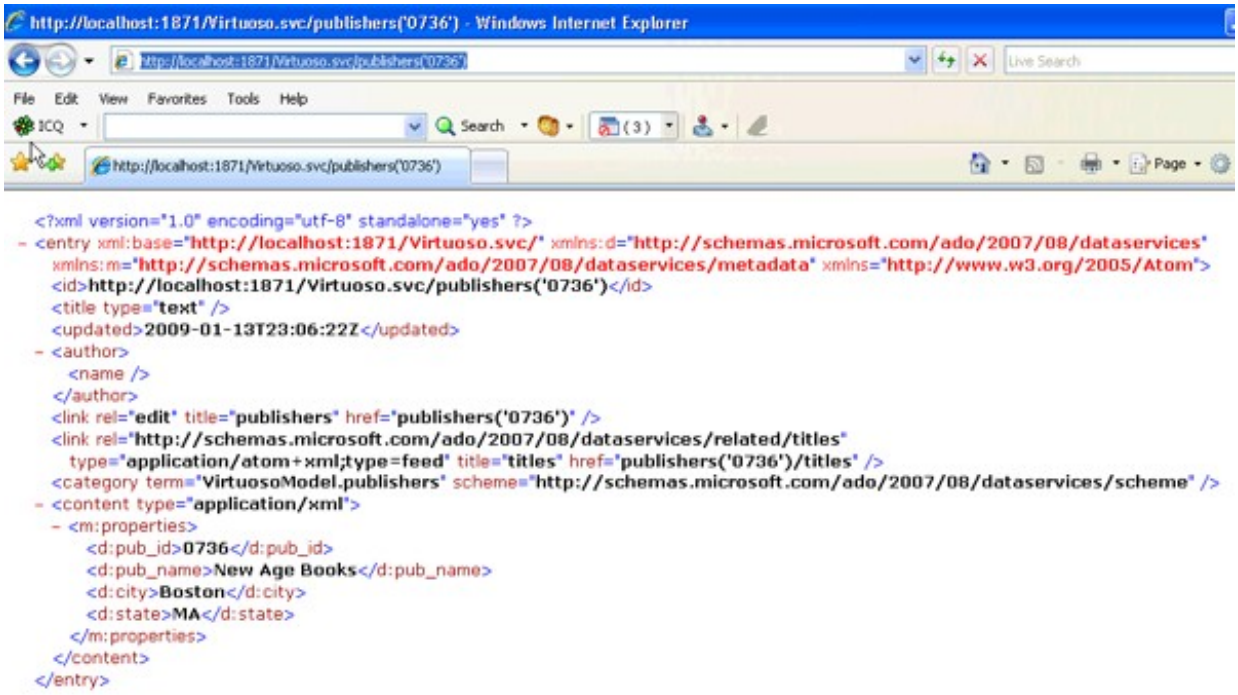
   publisher

   table publisher number

   0736

   record, use this convention http://host/vdir/Virtuoso.svc/publishers("0736") .

   Figure 8.372. publisher



Notes:

1. *Important*

   - To view

   Atom

   (the default format returned by an ADO.NET Data Service) in Internet Explorer, you must first ensure that

   Feed Reading View

   is turned

   off

   . This can be done on the

   Content tab

   of

   Tools in Internet Options

   .
2. If a Data Services entity instance URI page fails to load you can turn

   Verbose

   errors on by adding

   config.UseVerboseErrors = true;

in the

virtuoso.svc.cs InitializeService

method to obtain more detailed information from the server as to why the page failed to load:

```
public static void InitializeService(IDataServiceConfiguration config)
{
config.UseVerboseErrors = true;
config.SetEntitySetAccessRule("*", EntitySetRights.All);
}
```

Visual Studio Windows DataGrid Form Application

This section details the steps required to create a simple Visual Studio 2008 Windows Form application, with associated DataGridView control for displaying data in selected tables from the target database.

1. Launch the Visual Studio 2008 SP1 IDE.

   Figure 8.373. Visual Studio 2008 SP1 IDE

2. Create a

   Web Application

   project by going to the

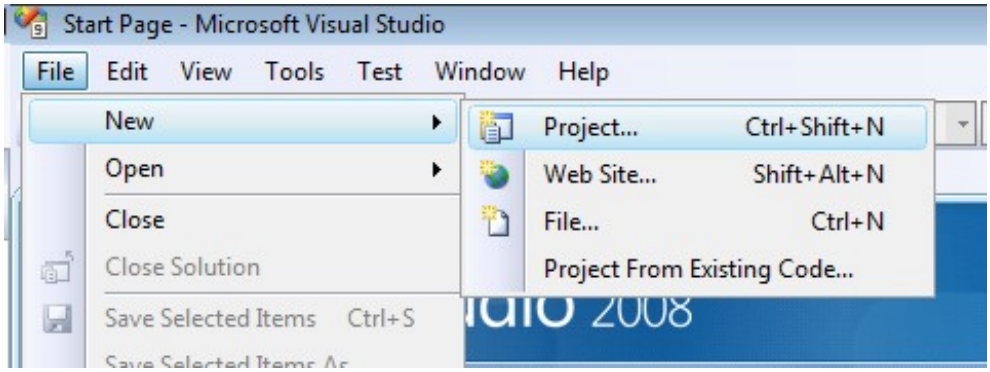
   File

   menu in Visual Studio and choosing

   New Project

   .

   Figure 8.374. Web Application



3. When the New Project window appears, choose either

   Visual Basic

   or

   Visual C#

   as your programming language.
4. Within the language category, click on

   Windows

   and select

   Windows Form Application

   from the right-hand panel.
5. Choose a name for the project, for example

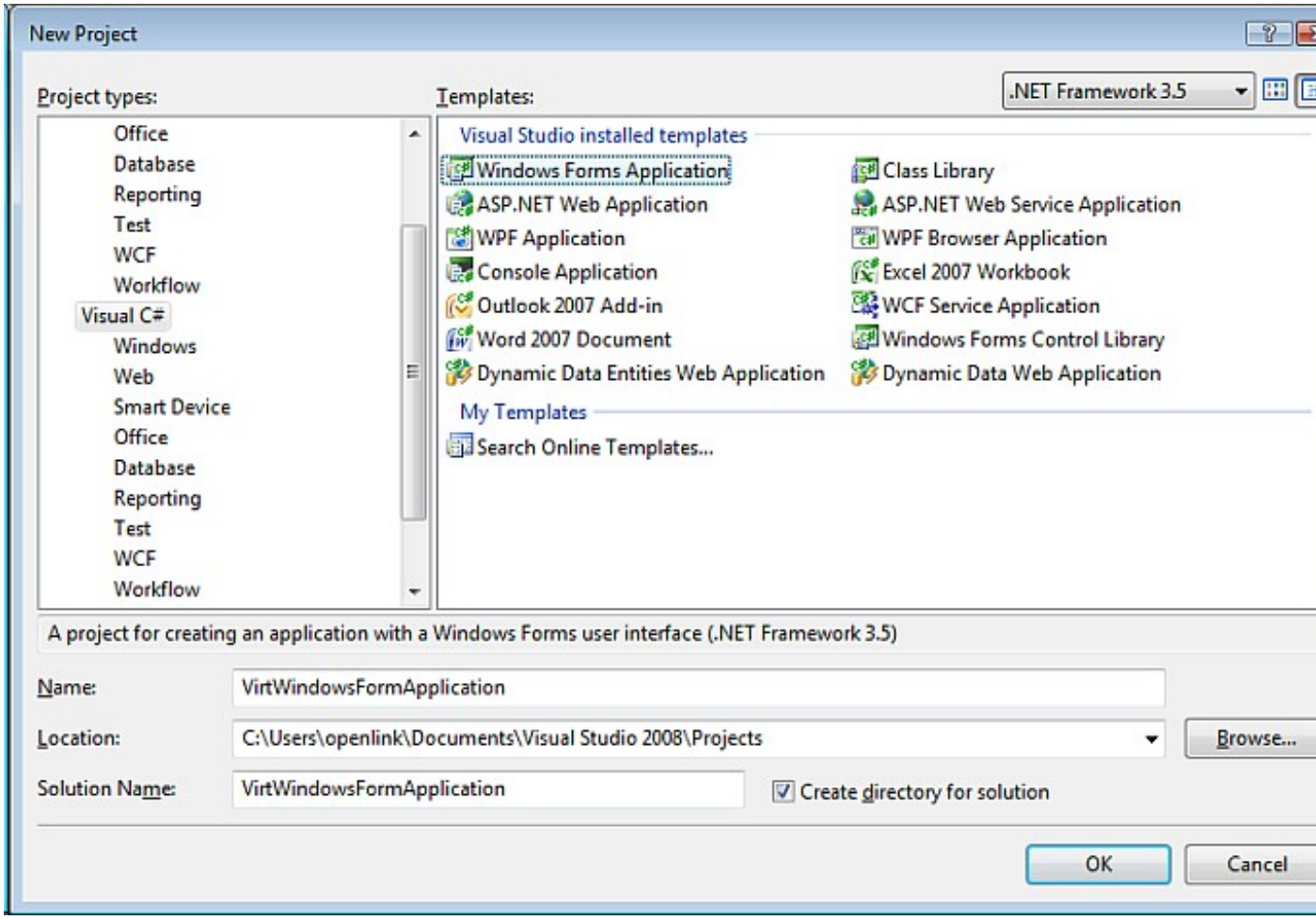
   VirtWindowsFormApplication

   , and click

   OK

   .

   Figure 8.375. Web Application



6. In the

   Toolbox

   , expand

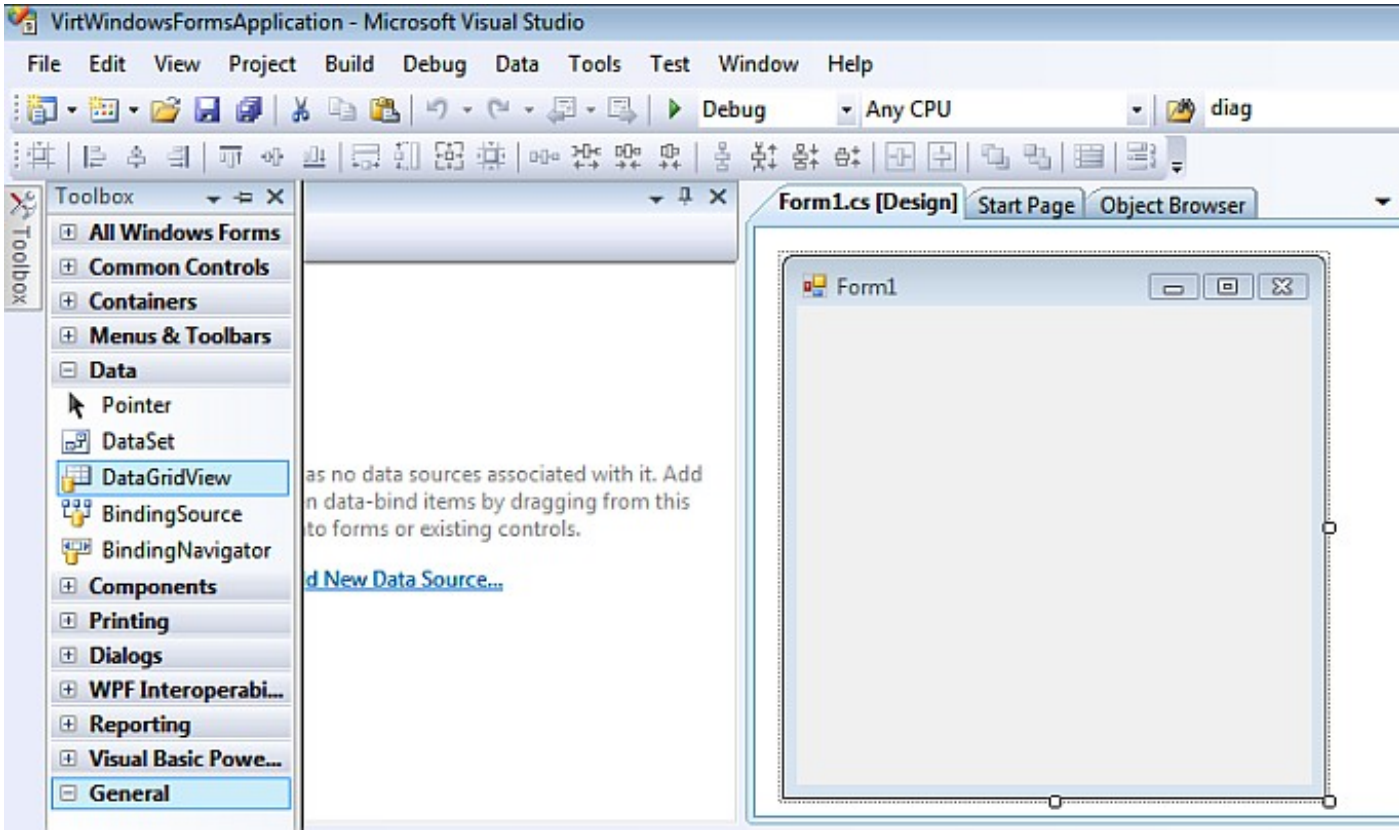
   Data Controls

   , and drag the

   DataGridView

   control onto the form.

   Figure 8.376. Toolbox



7. Click on the little

   arrow

   in the top right of the

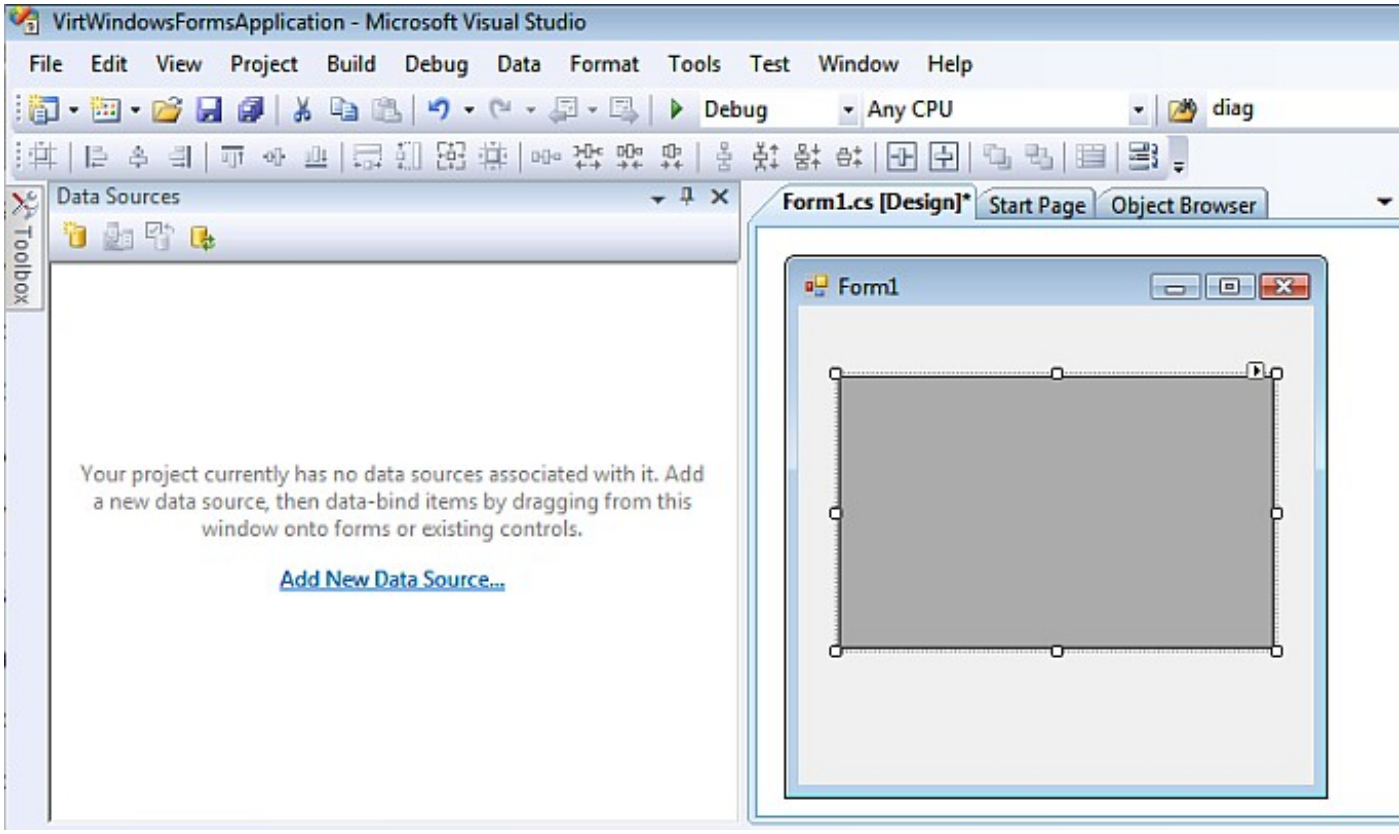
   DataGridView

   control. This loads the

   DataGridView Task

   menu.

Figure 8.377. DataGridView Task

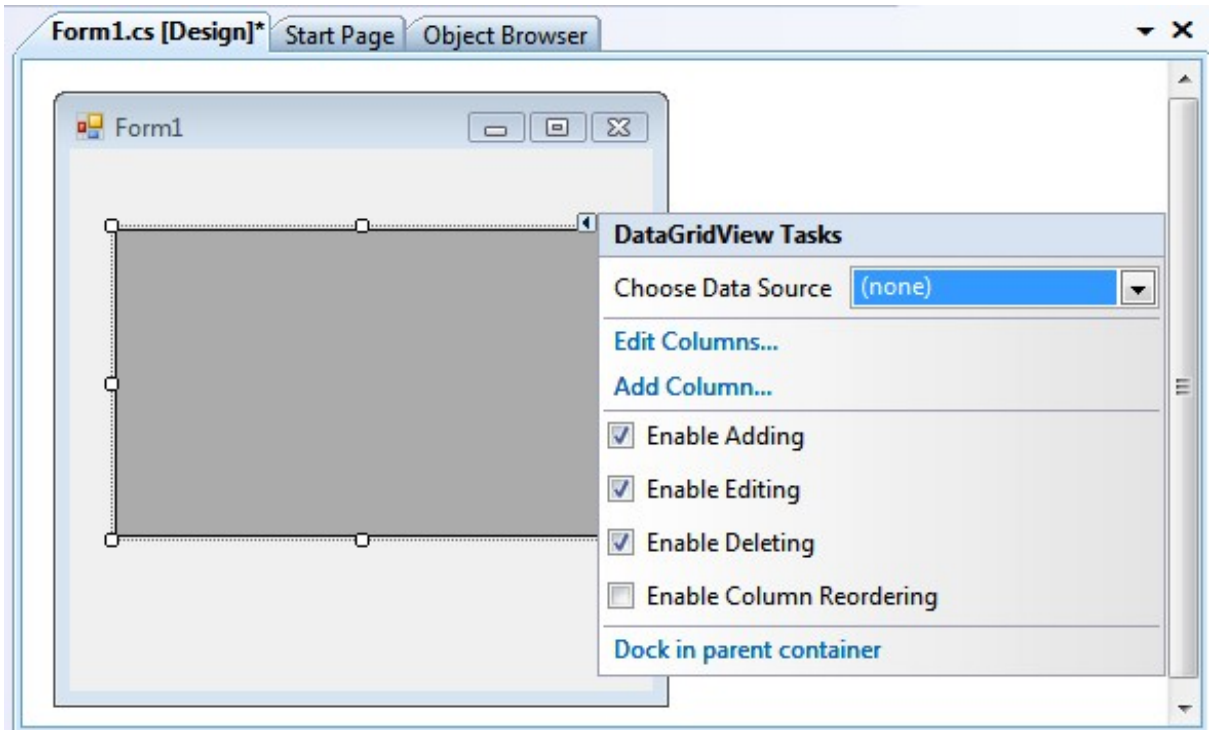


8. Click on the

   Choose Data Source

   list box.

   Figure 8.378. Choose Data Source

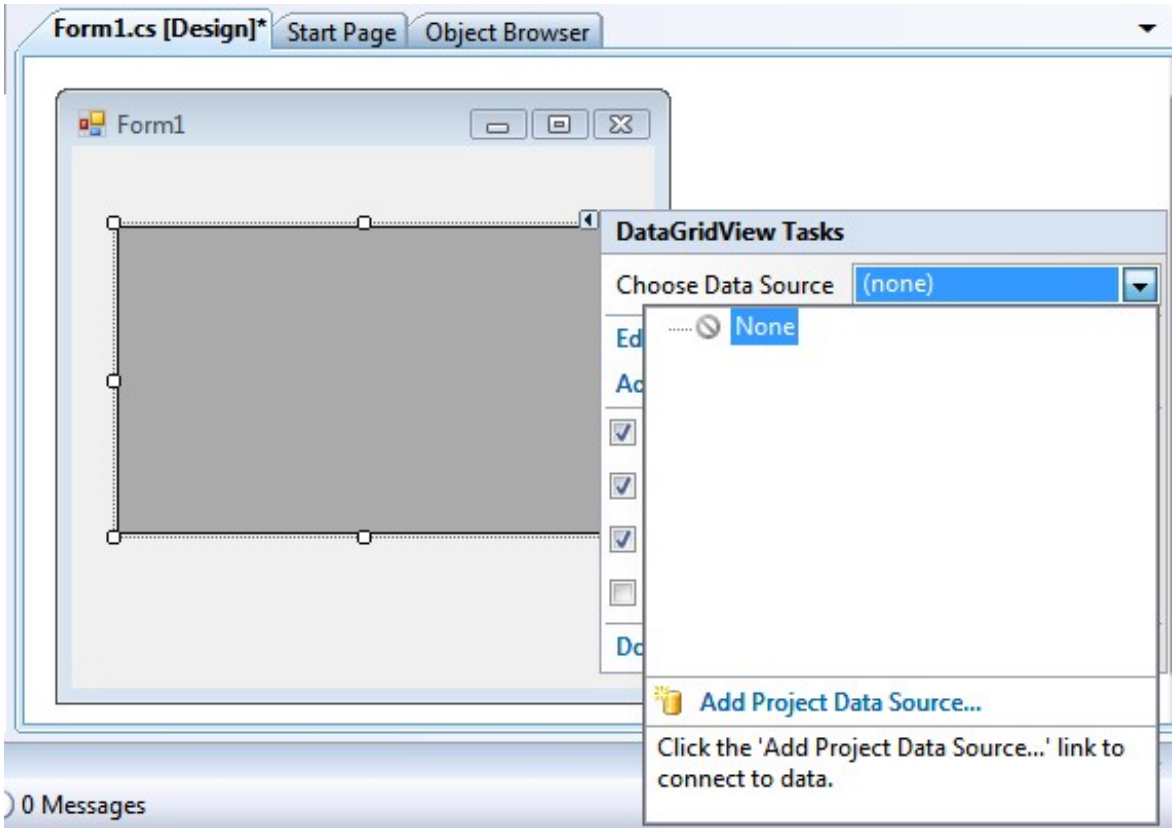


9. Click on the

   Add Project Data Source

link to connect to a data source.

Figure 8.379. Add Project Data Source



10. In the

    Data Source Configuration Wizard

    dialog

    Choose Data Source Type

    page select the

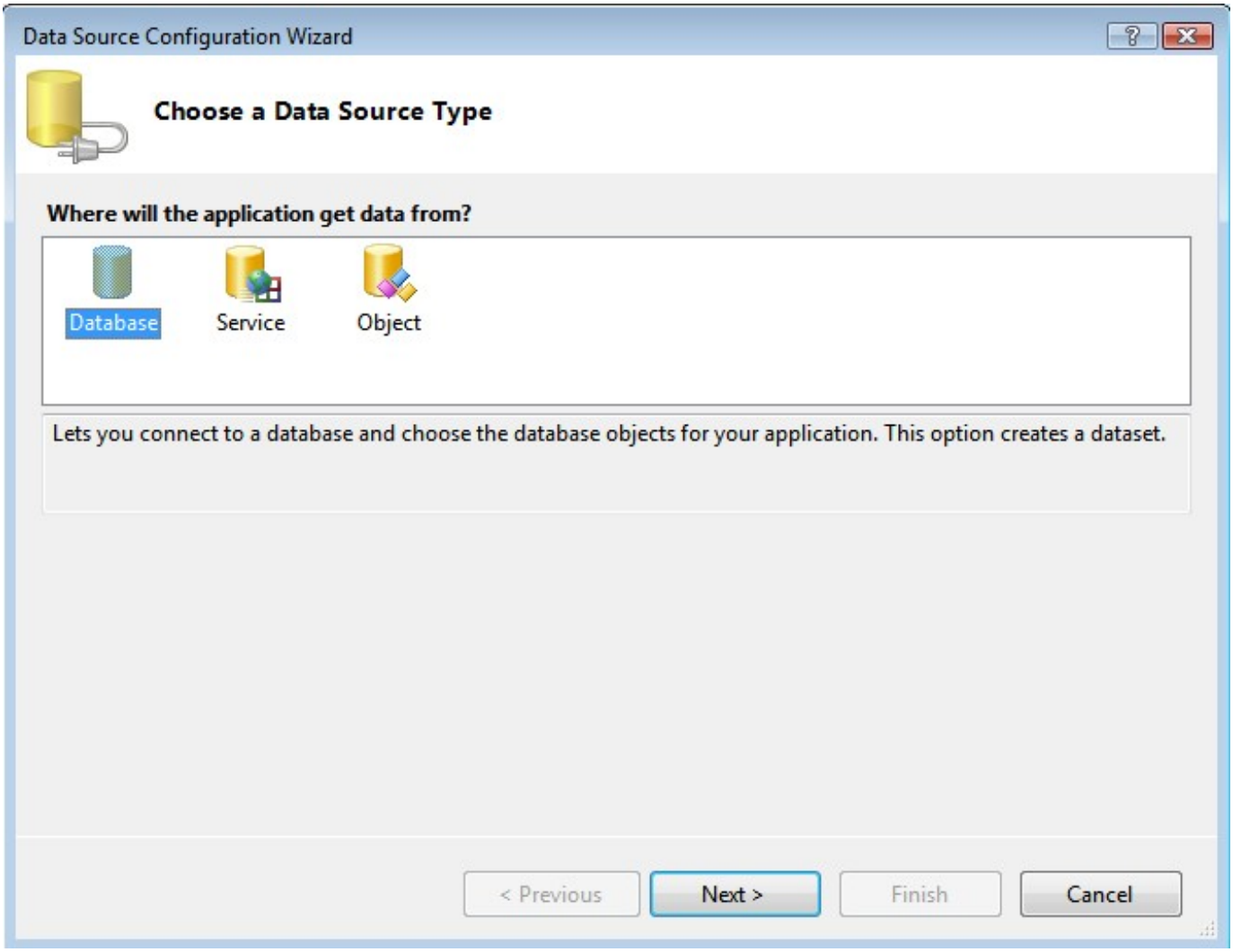
    Database

    data source type and click

    Next

    .

Figure 8.380. Data Source Type



11. In the

    Data Source Configuration Wizard

    dialog

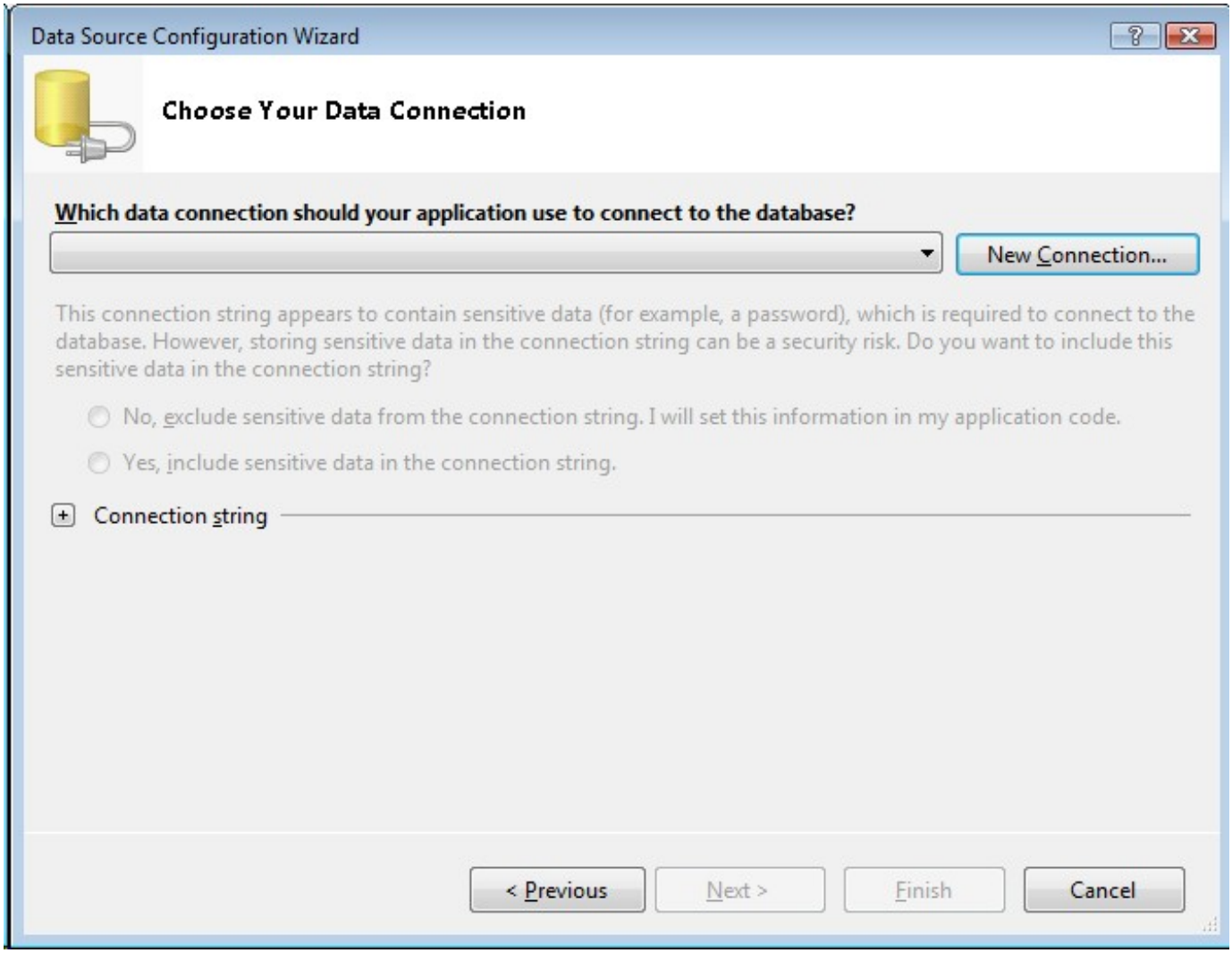
    Choose your Data Connection

    page, select the

    New Connection

    button

Figure 8.381. Data Source Configuration Wizard



12. In the

    Choose Data Source

    dialog, select the OpenLink

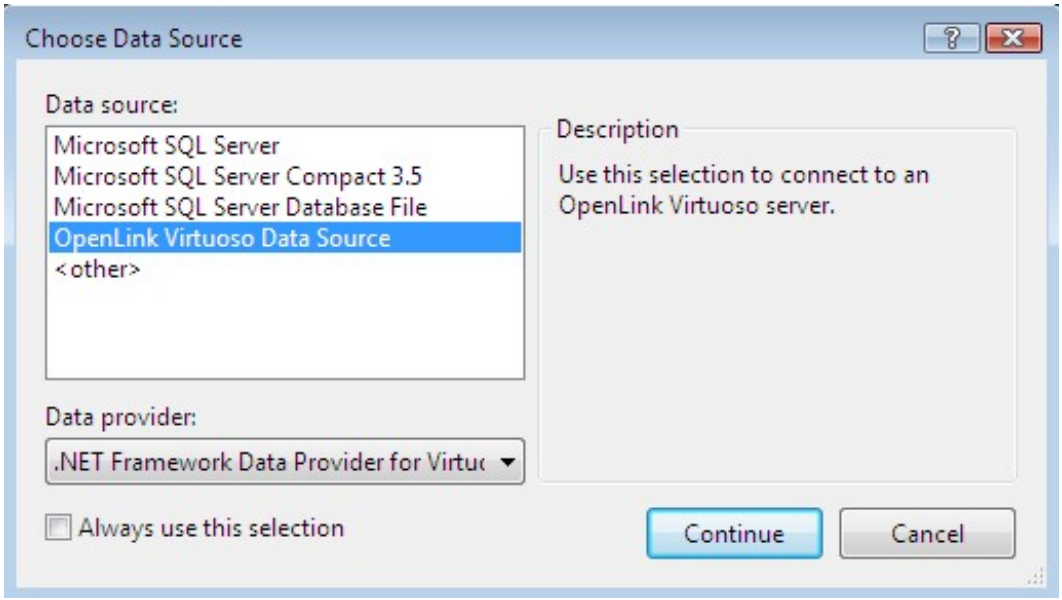
    Virtuoso Data Source

    from the list and click

    Continue

    .

Figure 8.382. Data Source



13. In the

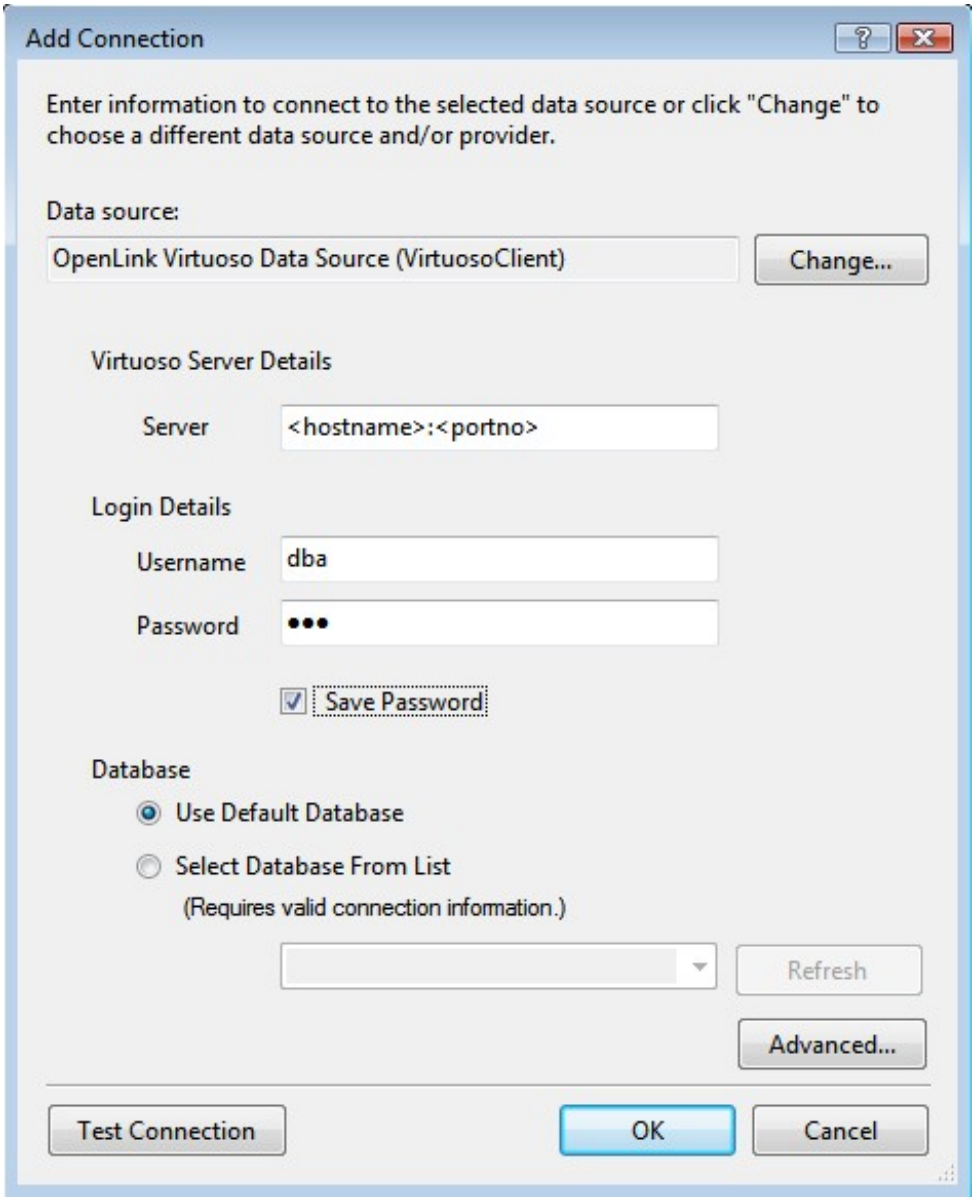
    Add Connection

    dialog, specify the

    hostname, portno, username and password

    for the target Virtuoso Server and check the Save Password check box.

    Figure 8.383. Connection Properties



14. Select the

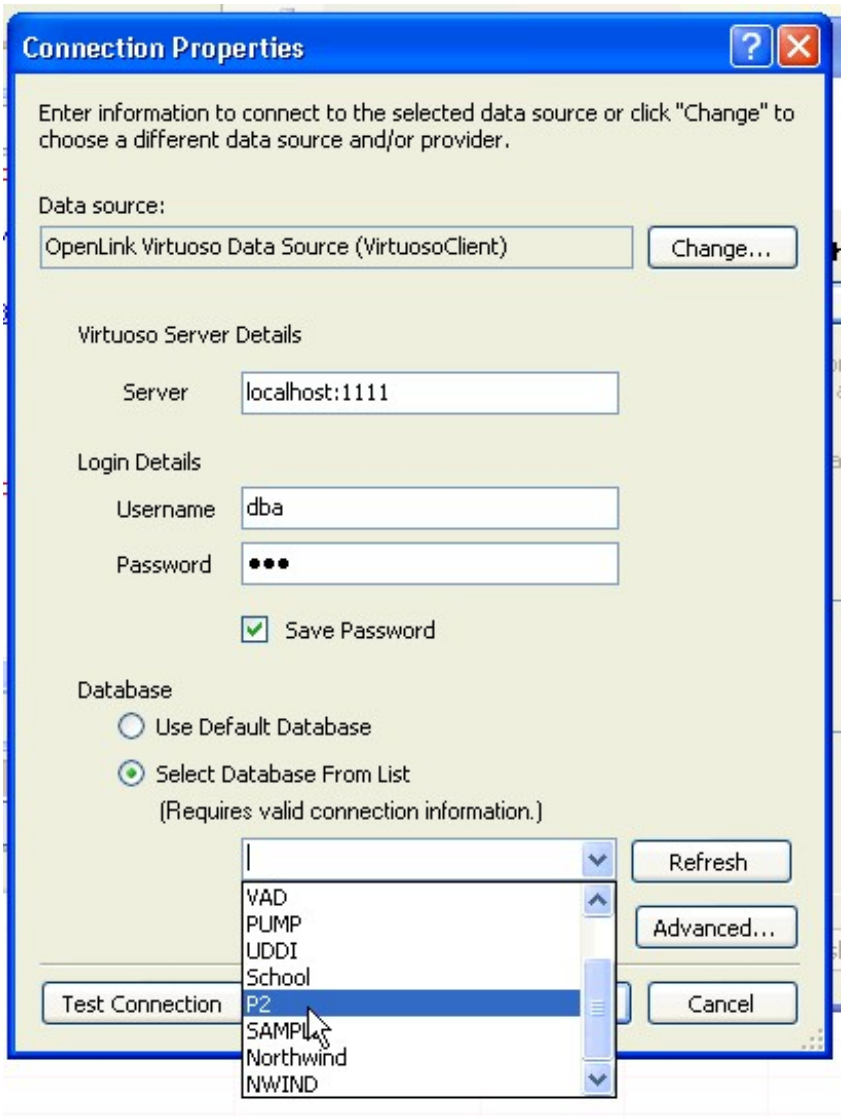
    Select Database From List

    radio button and choose the

    P2

    database from the drop down list.

Figure 8.384. Add connection

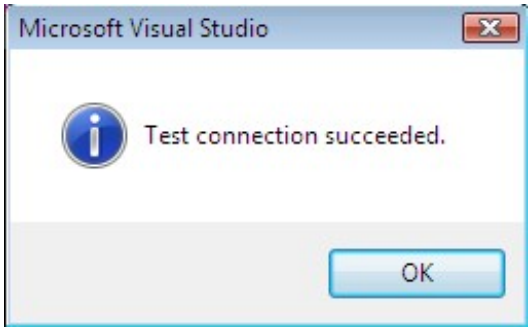


15. Press the

    Test Connection

    dialog to verify that the database is accessible.

    Figure 8.385. Test Connection



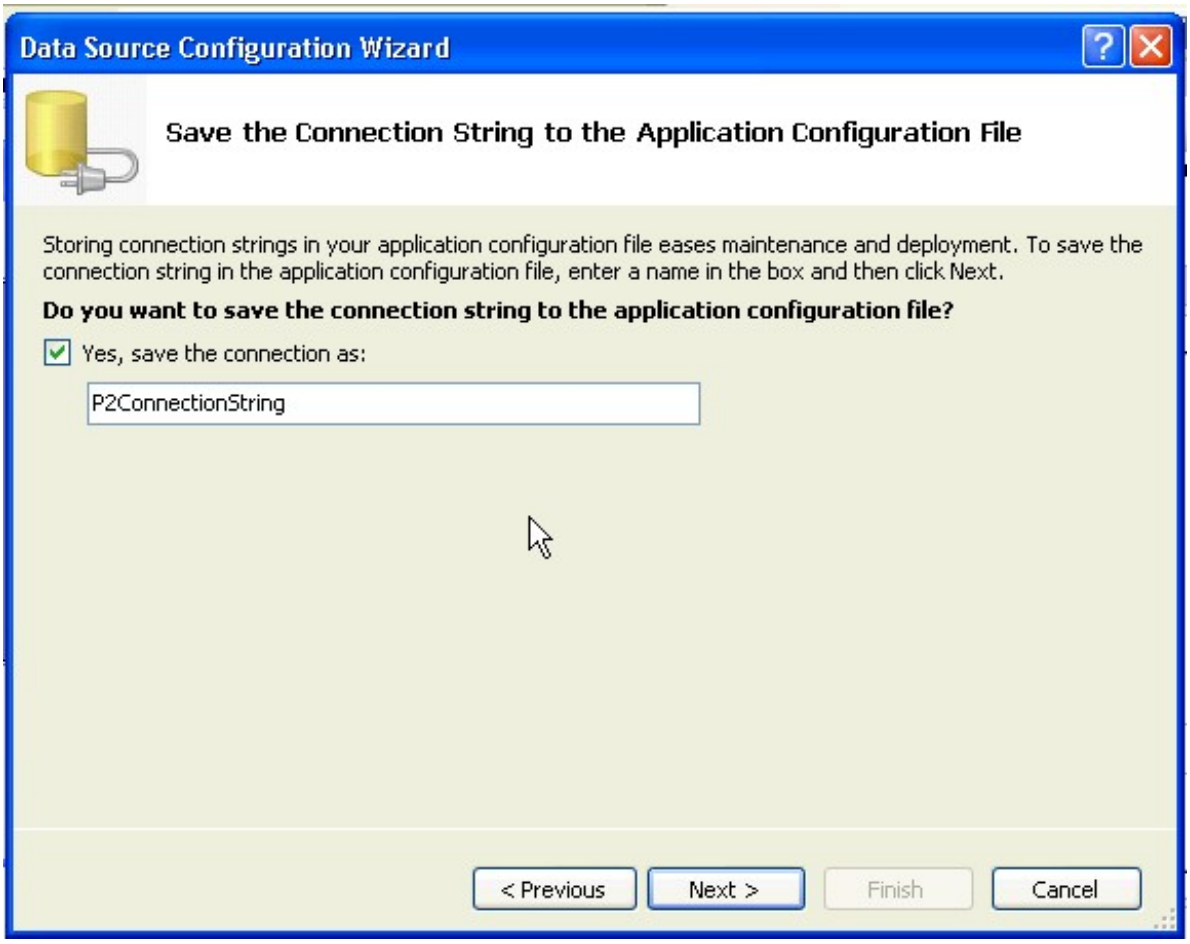
16. Leave the default connect string

    P2ConnectionString

    and click

Next

Figure 8.386. P2ConnectionString



17. From the list of available tables returned for the P2 database, select the

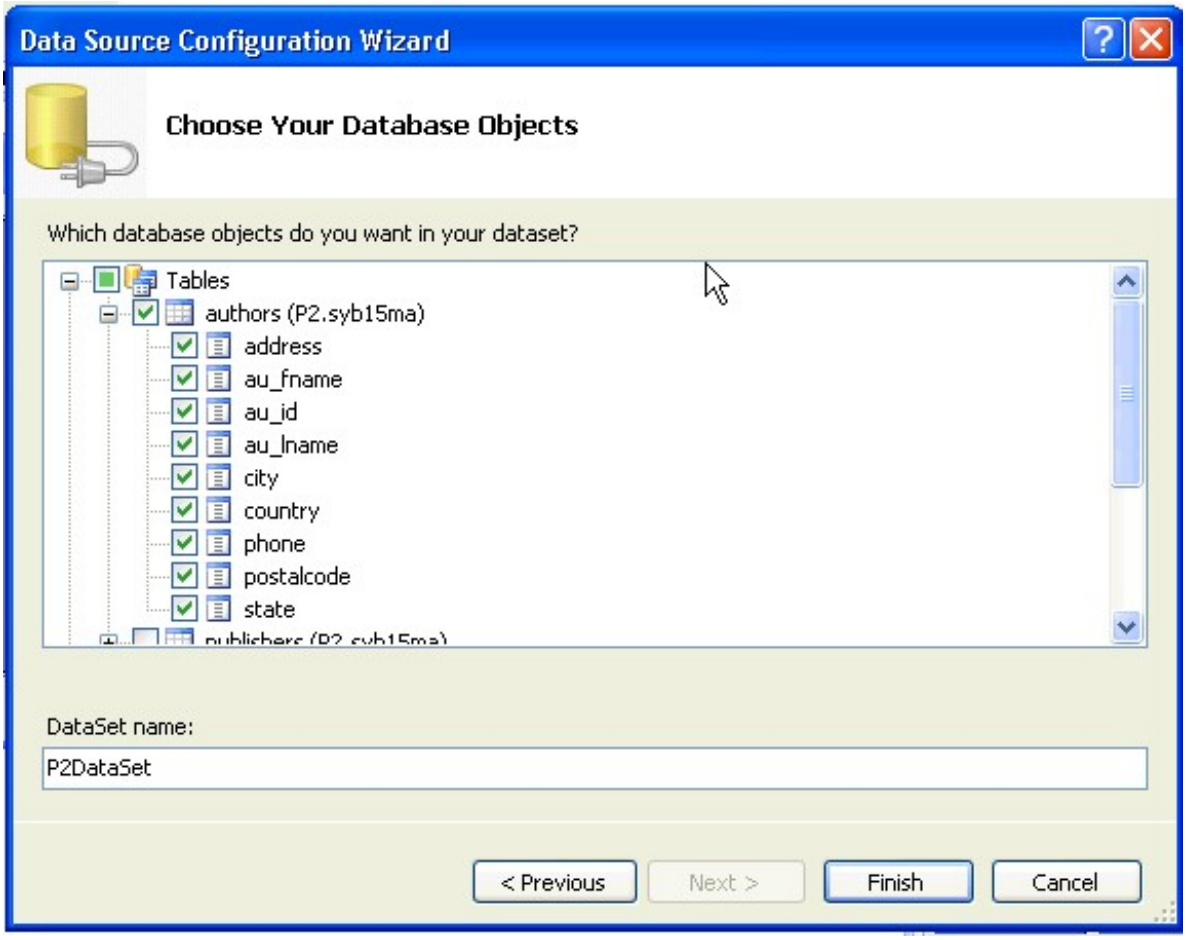
    authors

    table to be associated with the

    DataGridView

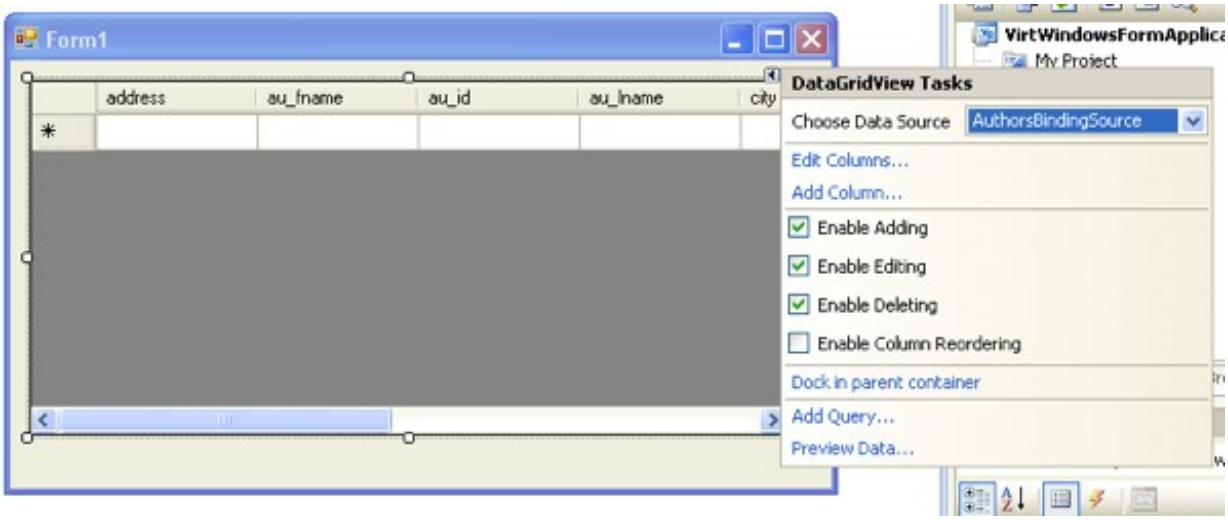
    control.

Figure 8.387. P2 database



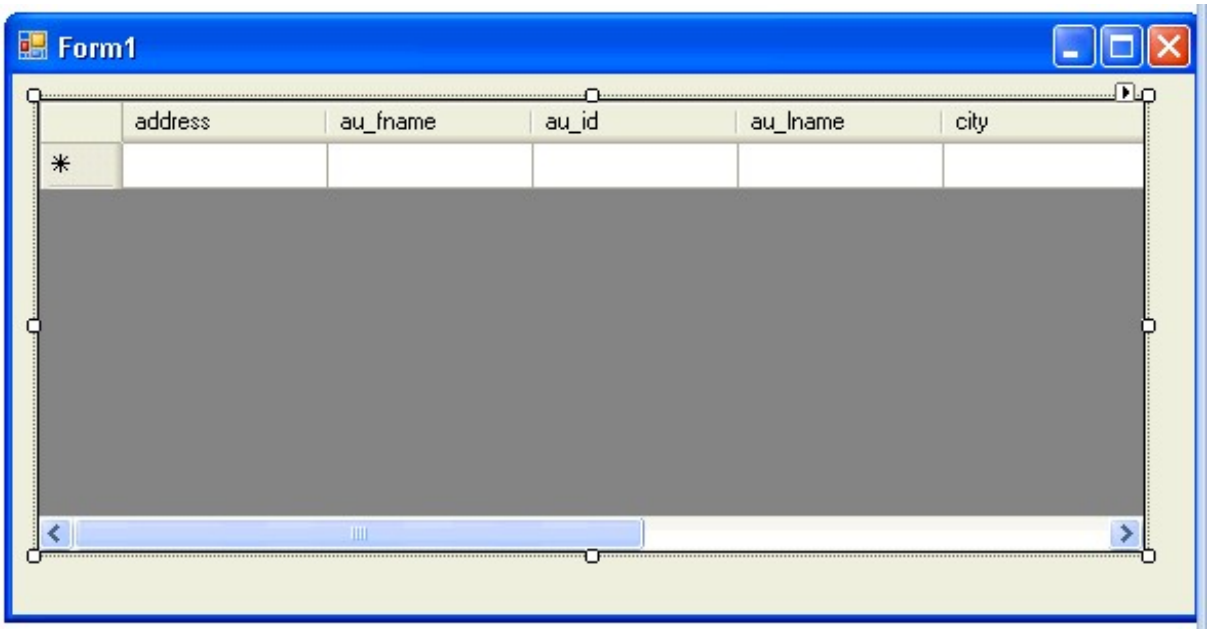
18. The columns names of the select table will be displayed in the DataGridView.

Figure 8.388. DataGridView



19. Resize the Form and DataGridView to allow all columns to be visible, if possible.

Figure 8.389. Resize the Form and DataGridView



20. To test the application, simply hit

    Ctrl+F5

    within Visual Studio or select

    Start Debugging

    from the

    Debug

    menu.

    Figure 8.390. Start Debugging

21. The data from the

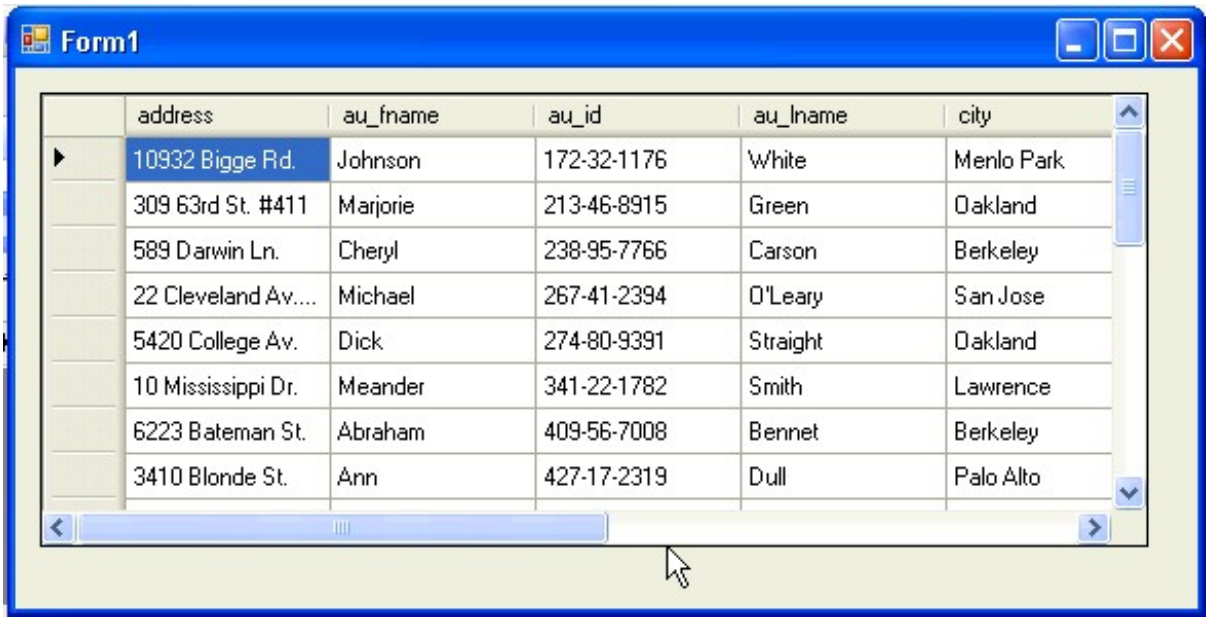
    authors

    table will be displayed in the

    DataGrid

    .

    Figure 8.391. DataGrid



The task is now complete.

8.8. Using Microsoft Entity Frameworks to Access MySQL Schema Objects with Virtuoso

Abstract

This section details the steps required to provide Microsoft Entity Framework access to MySQL Schema Objects using the OpenLink Virtuoso Universal Server. This is achieved by linking the required MySQL Schema objects into Virtuoso using its built in Virtual Database engine, and then using the Virtuoso ADO.NET Entity Framework provider to query the remote MySQL Schema objects linked into the Virtuoso Server.

Prerequisites. The following prerequisites must be in place for this solution to be possible.

MySQL DBMS. An MySQL DBMS hosting the required Schema Objects needs to be available. In this section the *Northwind sample* database will be used to demonstrate the process.

An MySQL DBMS hosting the required Schema Objects needs to be available. In this section the *Northwind sample* database will be used to demonstrate the process.

ODBC Driver for MySQL. An MySQL ODBC Driver is required for Linking the MySQL Schema Objects into the Virtuoso Server. The OpenLink *ODBC Driver for MySQL* will be used in this section, for which a functional ODBC Data source name of "mysqlma" will be assumed to exist on the machine hosting the Virtuoso Server.

An MySQL ODBC Driver is required for Linking the MySQL Schema Objects into the Virtuoso Server. The OpenLink *ODBC Driver for MySQL* will be used in this section, for which a functional ODBC Data source name of "mysqlma" will be assumed to exist on the machine hosting the Virtuoso Server.

Virtuoso Universal Server. An Virtuoso installation including the Virtuoso Universal Server and ADO.NET Entity Framework Provider is required. The Virtuoso components used must be Release 5.10.x or above, this being the minimum version containing support for Microsoft Entity Frameworks

An Virtuoso installation including the Virtuoso Universal Server and ADO.NET Entity Framework Provider is required. The Virtuoso components used must be Release 5.10.x or above, this being the minimum version containing support for Microsoft Entity Frameworks

Microsoft Visual Studio 2008 SP1. Microsoft Visual Studio 2008 Service Pack 1 is required, this being the only version containing the necessary Entity Framework support available at the time of writing.

Microsoft Visual Studio 2008 Service Pack 1 is required, this being the only version containing the necessary Entity Framework support available at the time of writing.

Tasks

Ensure MySQL Primary Keys (PKs) are not nullable. The Visual Studio 2008 Entity Data Model (EDM) requires that all primary keys are *NOT* Nullable, and will fail to generate an EDM if any are. Thus ensure any tables to be used are defined as not nullable in the MySQL database schema before attempting to generate an EDM. In the case of the Northwind database all tables are not nullable, thus this should not be an issue in this case.

The Visual Studio 2008 Entity Data Model (EDM) requires that all primary keys are *NOT* Nullable, and will fail to generate an EDM if any are. Thus ensure any tables to be used are defined as not nullable in the MySQL database schema before attempting to generate an EDM. In the case of the Northwind database all tables are not nullable, thus this should not be an issue in this case.

8.8.1. Install and configure OpenLink ODBC Driver for MySQL

The Virtuoso Virtual Database engine uses ODBC as the connectivity mechanism for linking remote database objects into its local schema. Thus an MySQL ODBC Driver must be available with a suitably configured DSN for connecting to the target database. The OpenLink ODBC Drivers for MySQL have been used in this section, although in theory any MySQL ODBC Driver can be used.

Installation instructions for the OpenLink ODBC Driver for MySQL are available from:

- Product Installation & Basic Configuration (ODBC)

8.8.2. Install and configure OpenLink Virtuoso Universal Server

- Install and configure OpenLink Virtuoso Universal Server

8.8.3. Linking MySQL tables into OpenLink Virtuoso

1. Start the Virtuoso Web User Interface

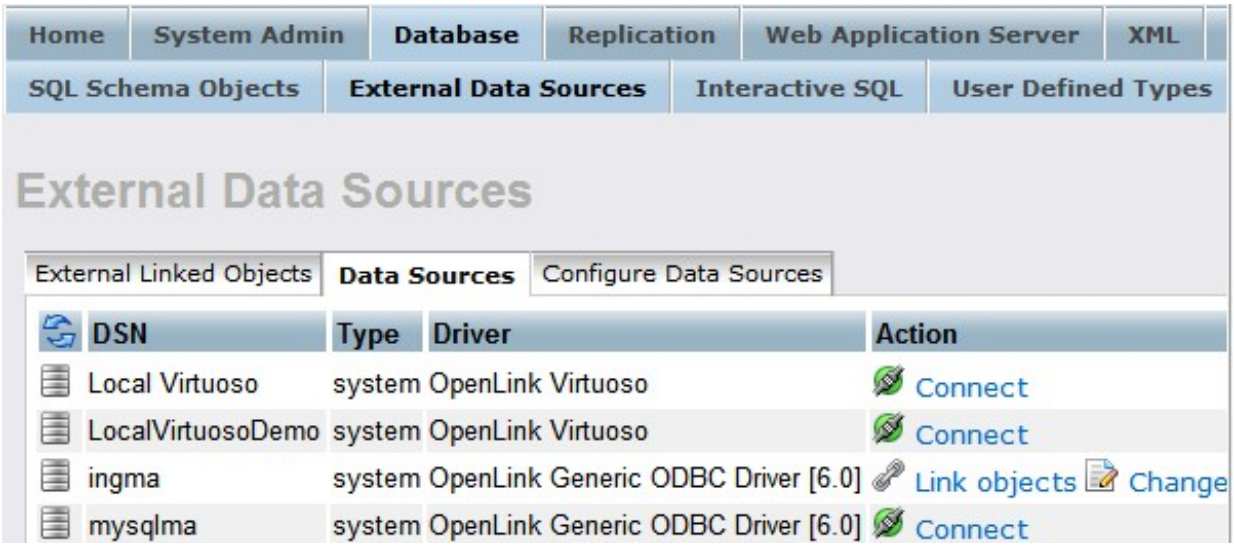
   Figure 8.392. Start

2. Select the "Conductor" link in the Left Frame and Login as the "dba" user.

Figure 8.393. Conductor

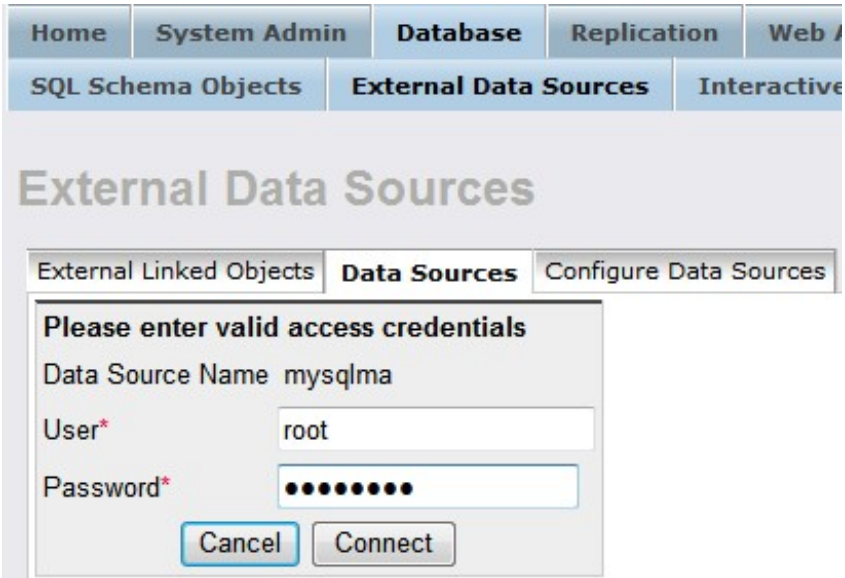
3. Select the "Databases" -> "External Data Source" -> "Data Sources" tab menu items

Figure 8.394. Databases



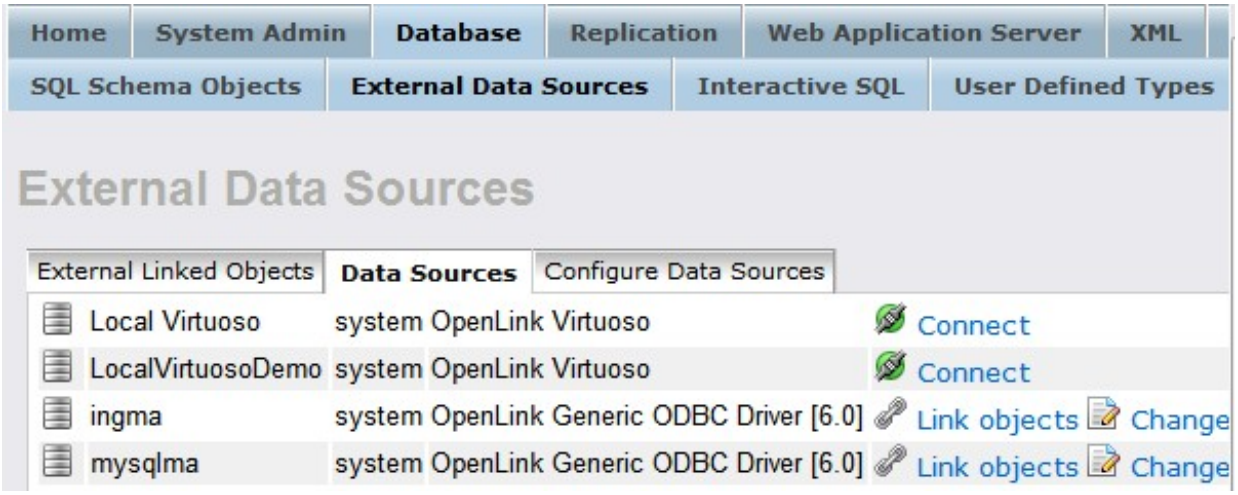
4. Select the "Connect" button for the "mysqlma" MySQL DSN.

Figure 8.395. Connect



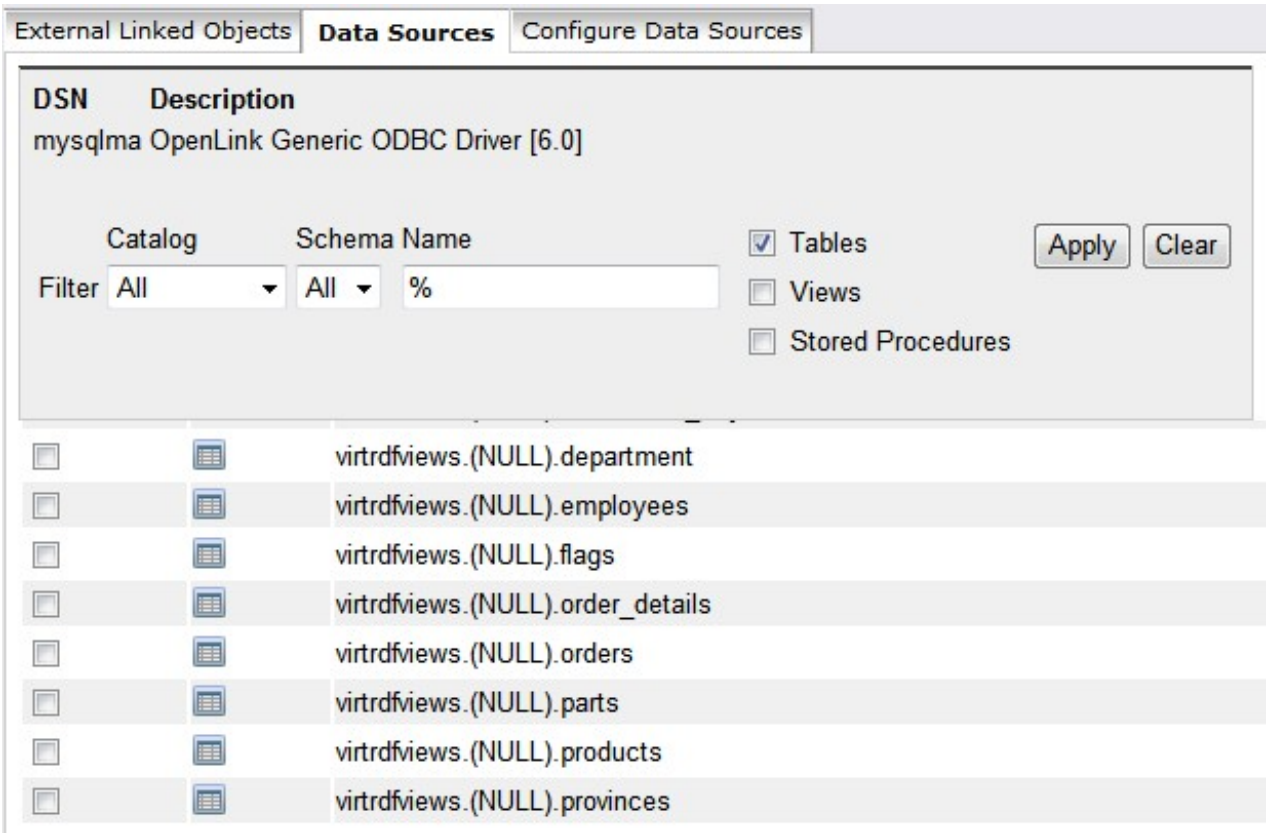
5. On successful connection Select the "Link Objects" button to obtain a list of available tables

Figure 8.396. Link Objects



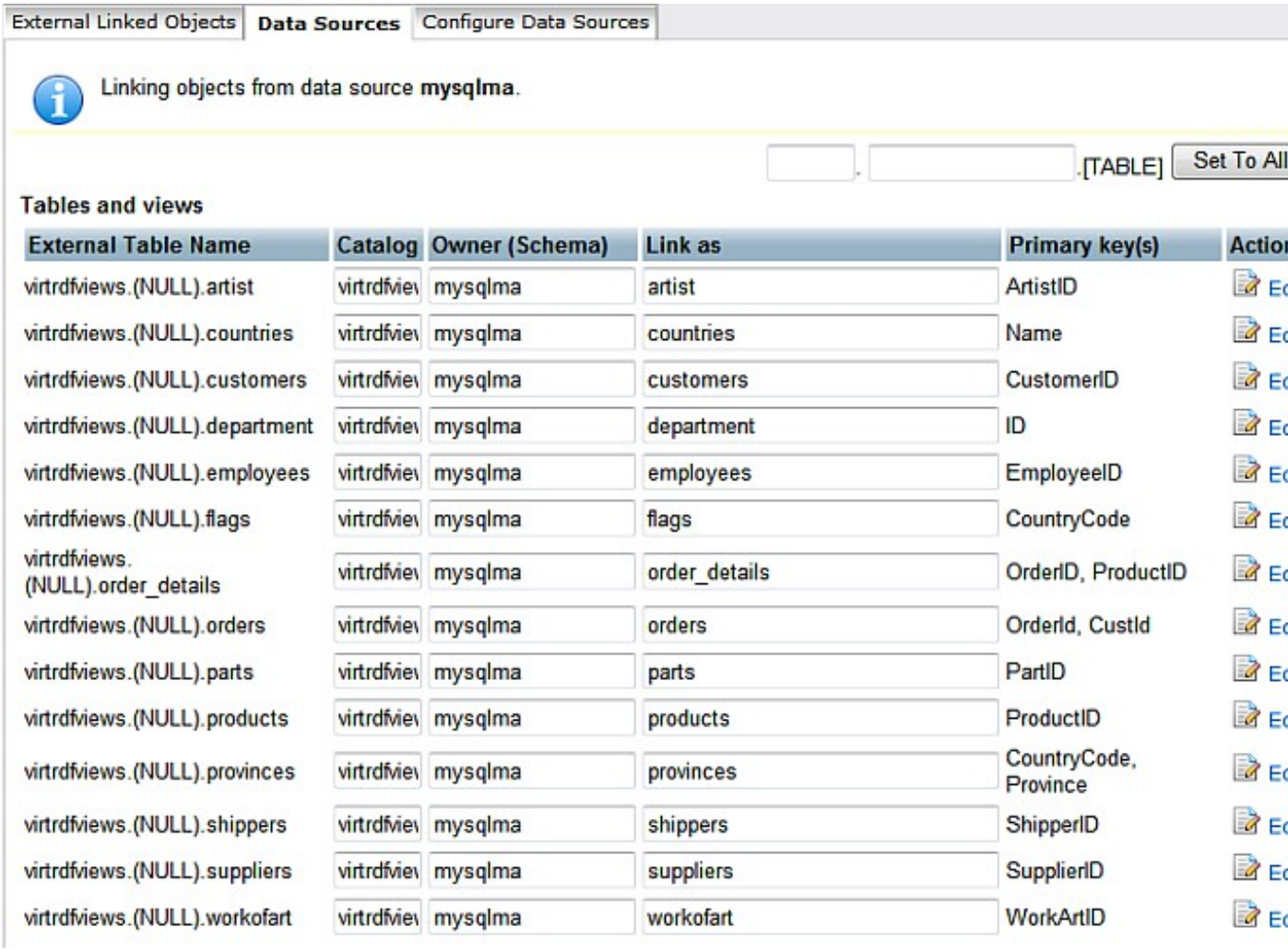
6. Select all the tables that are part of the "mysql" catalog.

Figure 8.397. Select tables



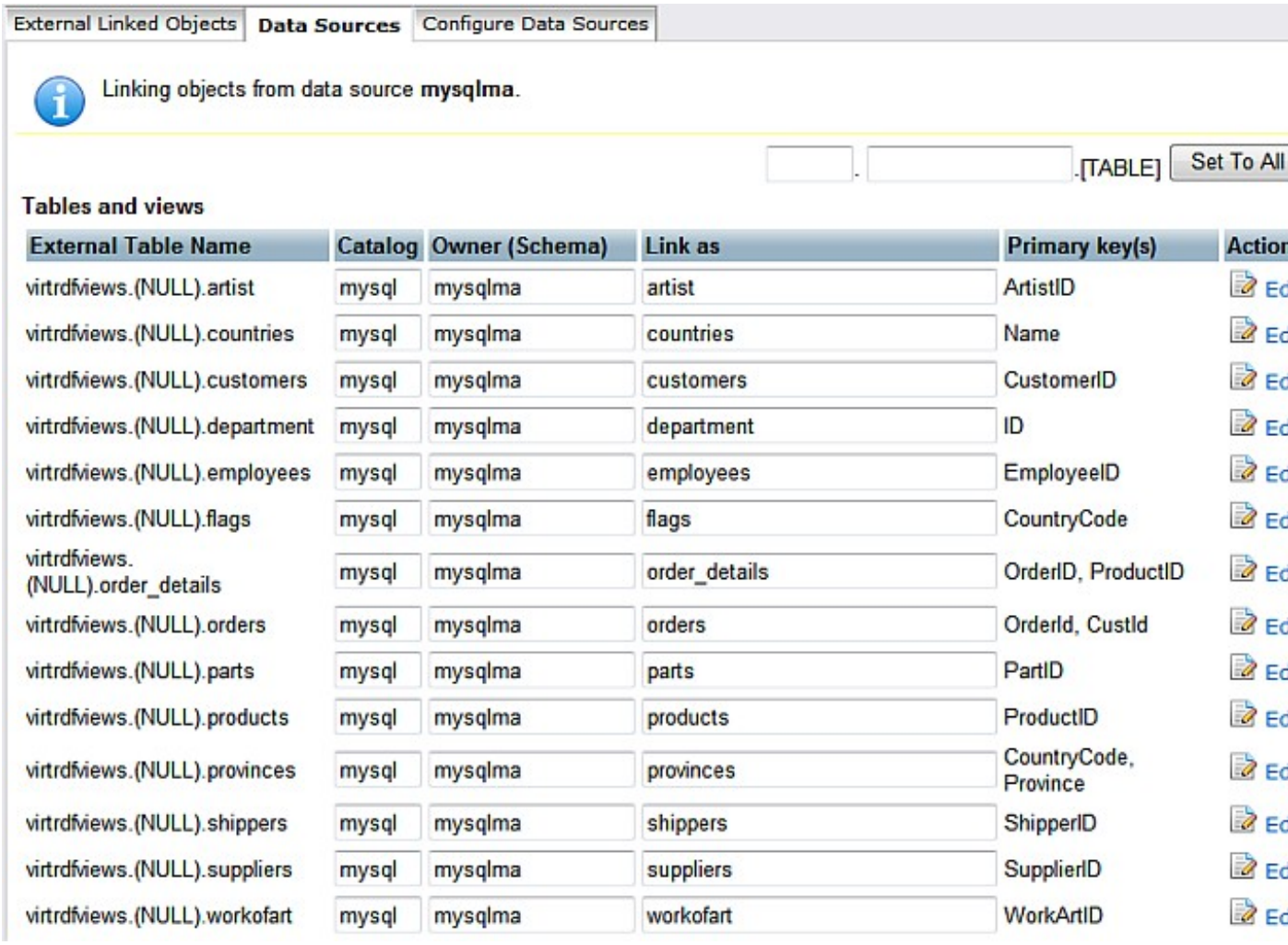
7. Ensure a primary key is assigned to all table that are to be used in the EDM generation phase.

Figure 8.398. Catalog



8. Change the Catalog for each table to be "mysql" using the "Set All" button.
9. Select the "Link" button to link the selected tables into Virtuoso

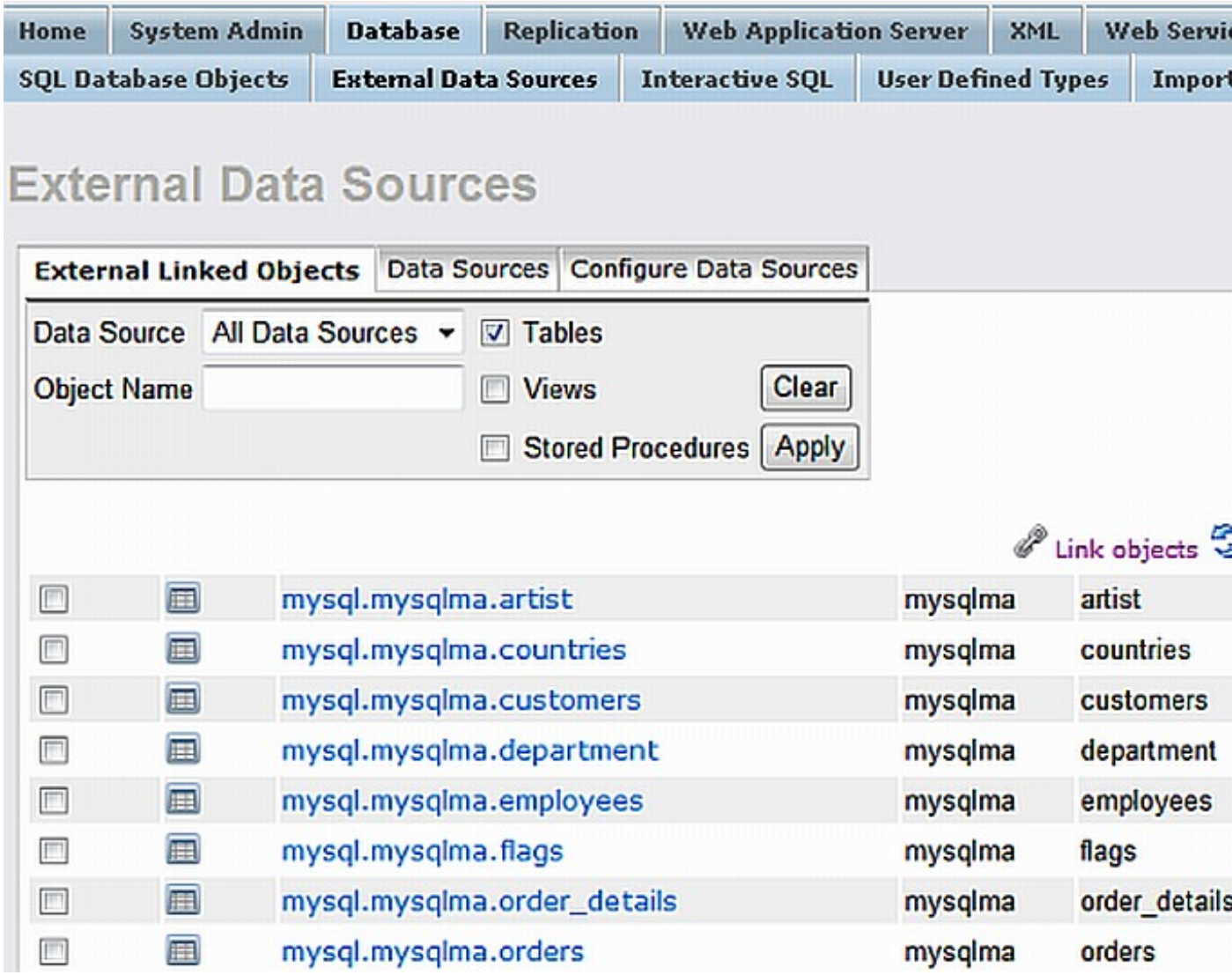
Figure 8.399. "Link" button



External Table Name	Catalog	Owner (Schema)	Link as	Primary key(s)	Action
virtrdfviews.(NULL).artist	mysql	mysqlma	artist	ArtistID	Ed
virtrdfviews.(NULL).countries	mysql	mysqlma	countries	Name	Ed
virtrdfviews.(NULL).customers	mysql	mysqlma	customers	CustomerID	Ed
virtrdfviews.(NULL).department	mysql	mysqlma	department	ID	Ed
virtrdfviews.(NULL).employees	mysql	mysqlma	employees	EmployeeID	Ed
virtrdfviews.(NULL).flags	mysql	mysqlma	flags	CountryCode	Ed
virtrdfviews.(NULL).order_details	mysql	mysqlma	order_details	OrderID, ProductID	Ed
virtrdfviews.(NULL).orders	mysql	mysqlma	orders	OrderId, CustId	Ed
virtrdfviews.(NULL).parts	mysql	mysqlma	parts	PartID	Ed
virtrdfviews.(NULL).products	mysql	mysqlma	products	ProductID	Ed
virtrdfviews.(NULL).provinces	mysql	mysqlma	provinces	CountryCode, Province	Ed
virtrdfviews.(NULL).shippers	mysql	mysqlma	shippers	ShipperID	Ed
virtrdfviews.(NULL).suppliers	mysql	mysqlma	suppliers	SupplierID	Ed
virtrdfviews.(NULL).workofart	mysql	mysqlma	workofart	WorkArtID	Ed

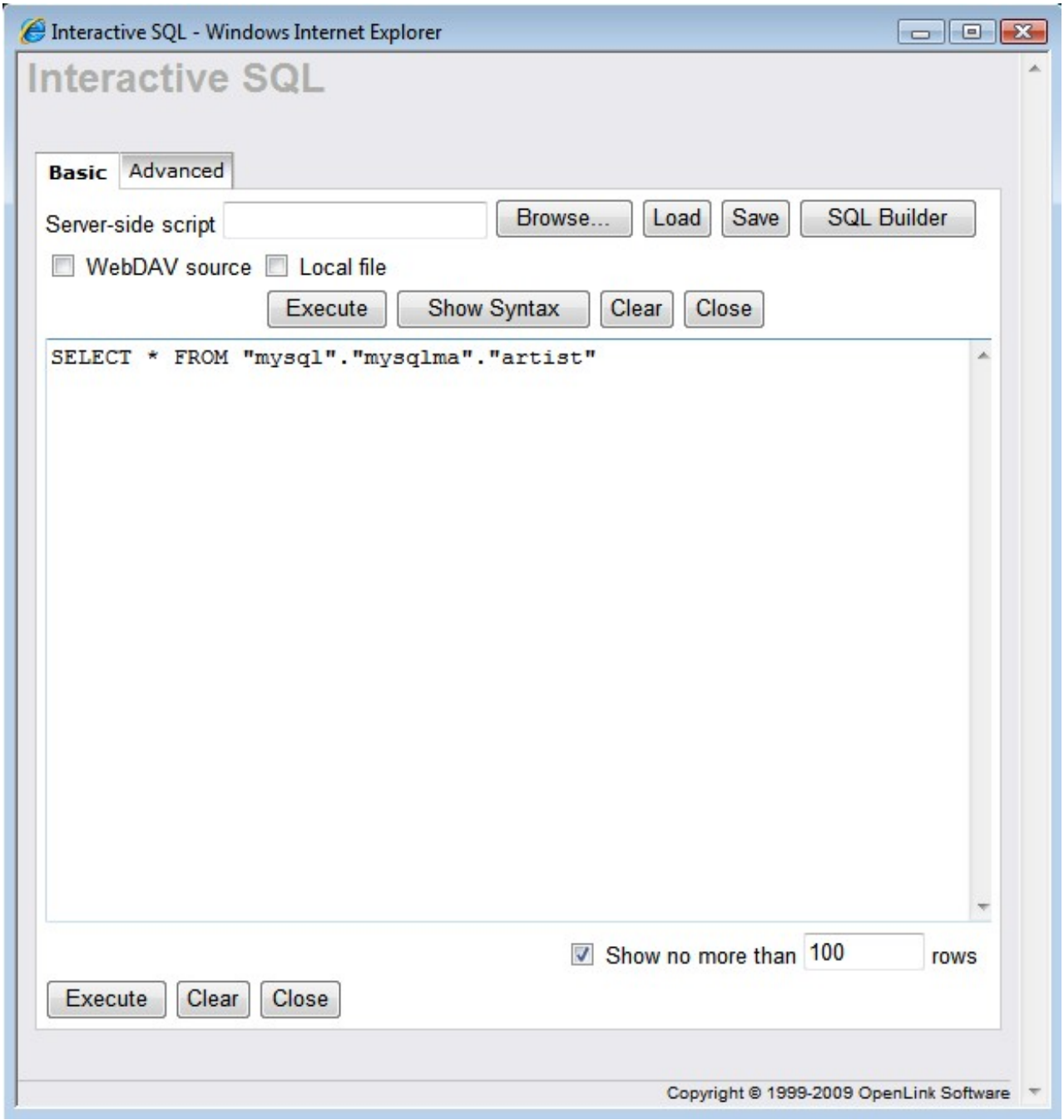
10. On completion of the Link process the tables will be displayed in the "External Linked Objects" tab.

Figure 8.400. Completion



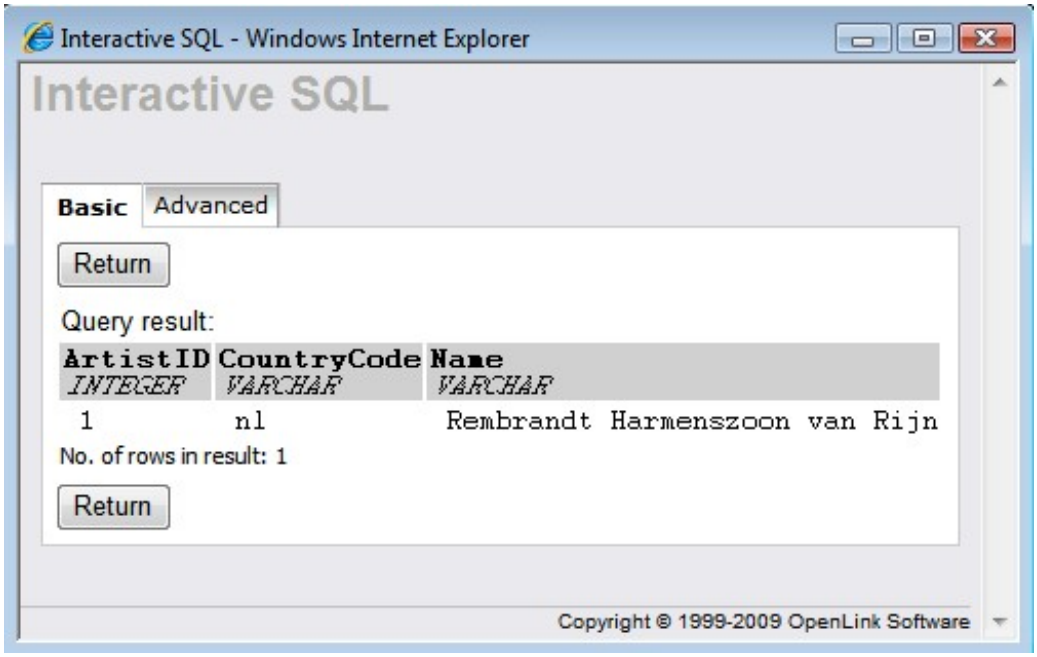
11. The linked tables can be queried by clicking on the hyperlink in the "Local Name" column of the "External Linked Objects" tab above, which loads the Virtuoso "Interactive SQL" interface with the required SQL "Select" for retrieving the remote table data . We shall use the "mysql.mysqlma.artist" table to demonstrate this.

Figure 8.401. Completion



12. Then click the "Execute" button to run the query and retrieve the results from the remote table.

Figure 8.402. Completion



13. The tables can also be viewed as part of the Virtuoso "SQL Schema Objects" under the "mysql" catalog name.

Figure 8.403. view tables

The Link process is now complete enabling the tables to be queried as if part of the Virtuoso Schema.

8.8.4. Creating EDM in Visual Studio 2008

The following steps can be used to create an Entity Data Model (EDM) for the MySQL Northwind database:

1. Launch the Visual Studio 2008 SP1 IDE.

Figure 8.404. Visual Studio 2008 SP1 IDE

2. Create a

   Web Application

   project by going to the

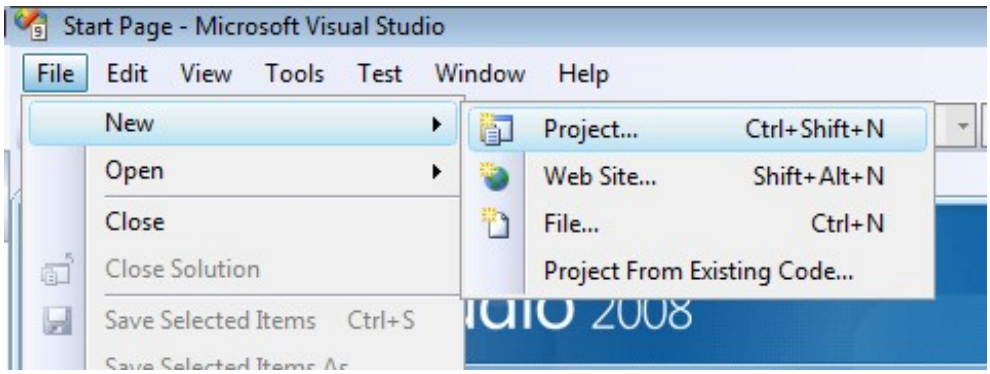
   File

   menu in Visual Studio and choosing

   New Project

   .

   Figure 8.405. Web Application



3. When the New Project window appears, choose either

   Visual Basic

   or

   Visual C#

   as your programming language.
4. Within the language category, click on

   Web

   and select

   ASP.NET Web Application

from the right-hand panel.

5. Choose a name for the project, for example

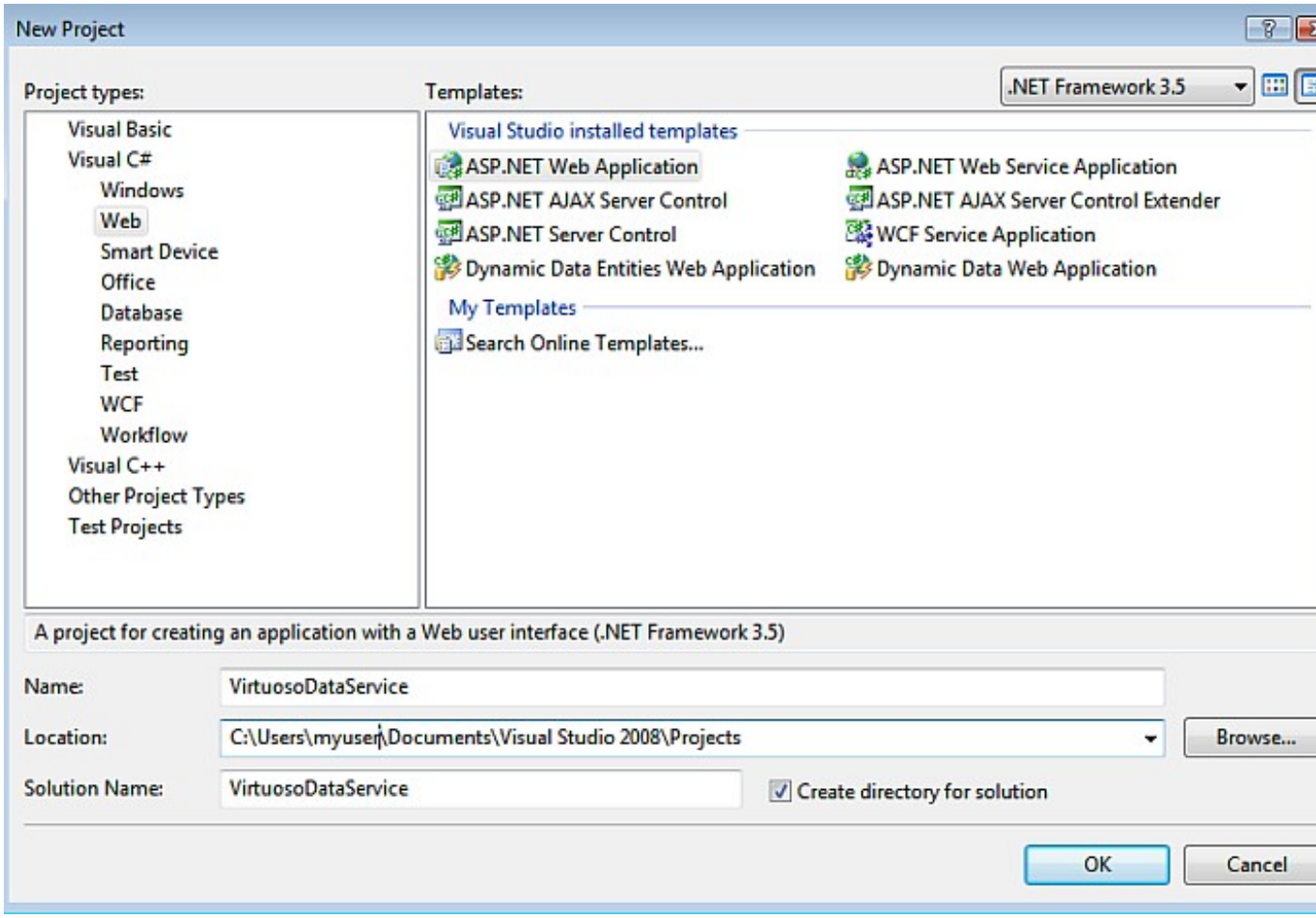
   VirtuosoDataService

   , and click

   OK

   .

   Figure 8.406. name for the project

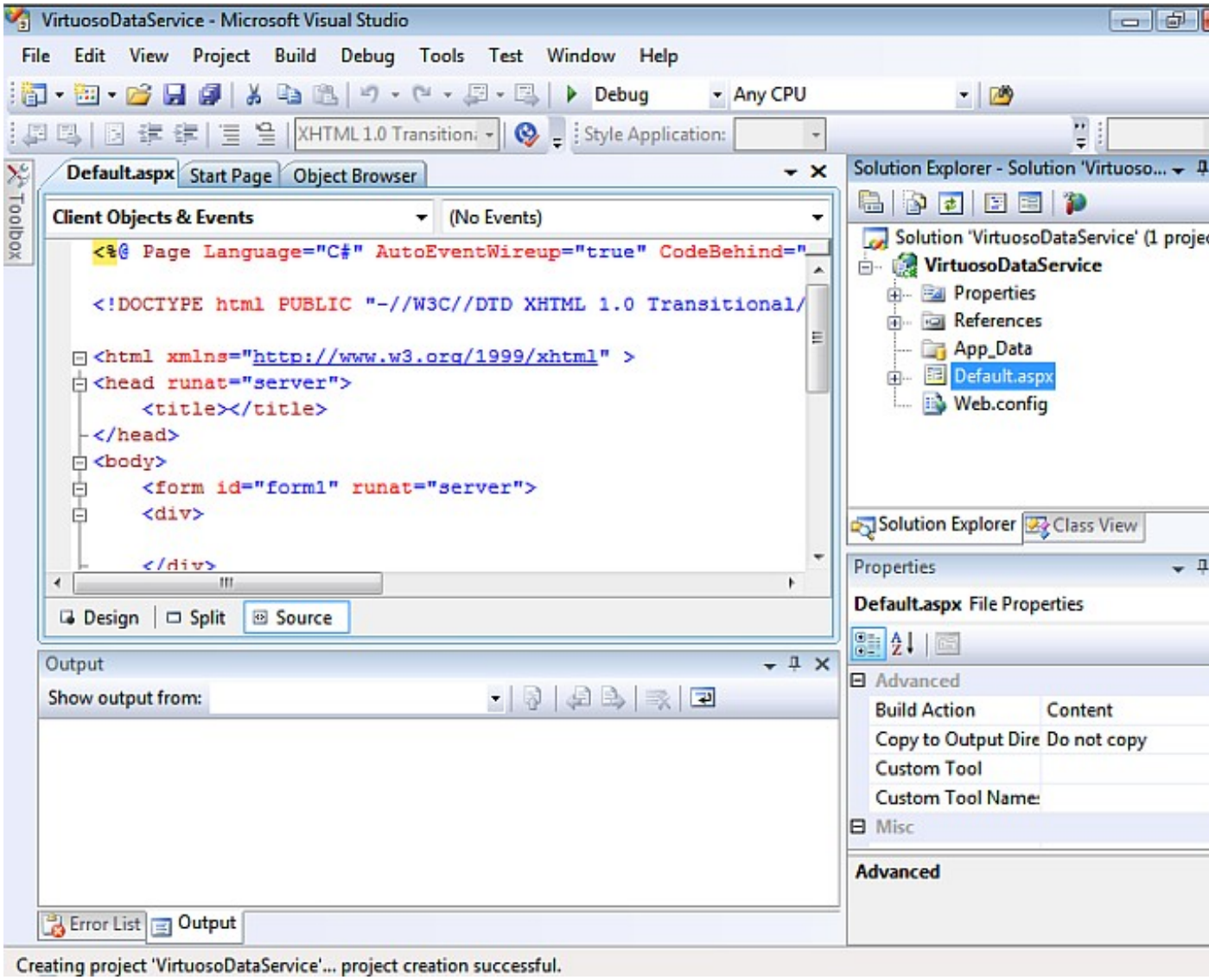


6. This will create a new project called

   VirtuosoDataService

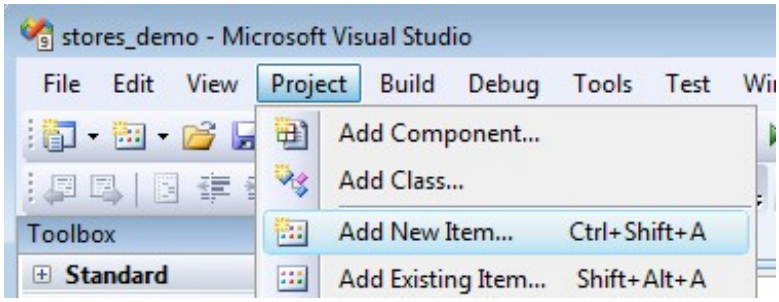
   .

   Figure 8.407. create a new project



7. Select the Project -> Add New Item menu option.

Figure 8.408. VirtuosoDataService



8. The

Add New Item

dialog will appear. Choose the

ADO.NET Entity Data Model

template. Give it the name

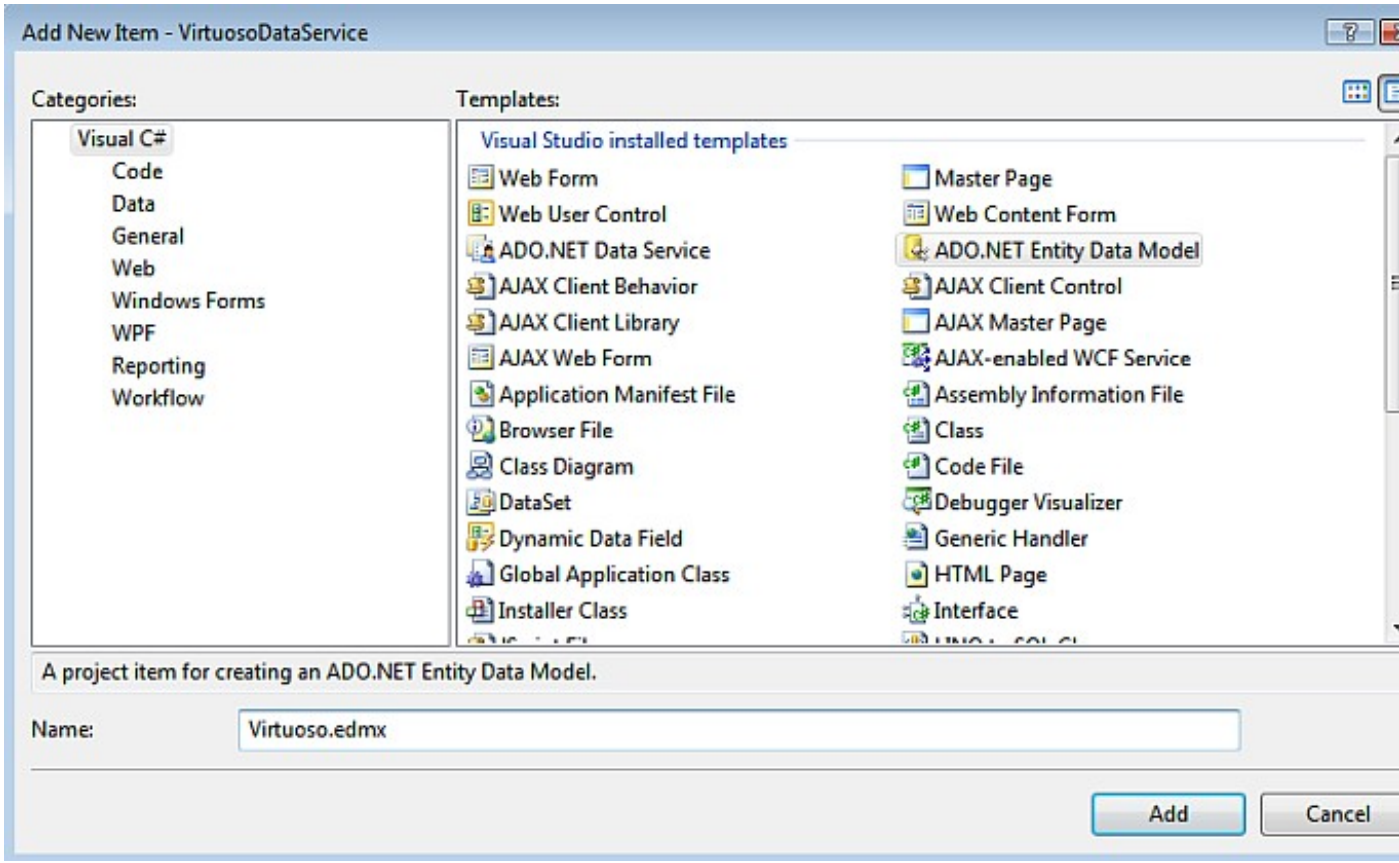
Virtuoso.edmx

and click

Add

to start the creation of the ADO.Net Entity Data Model.

Figure 8.409. Add New Item



9. In the

Entity Data Model Wizard

dialog

Choose Model Contents

page select the

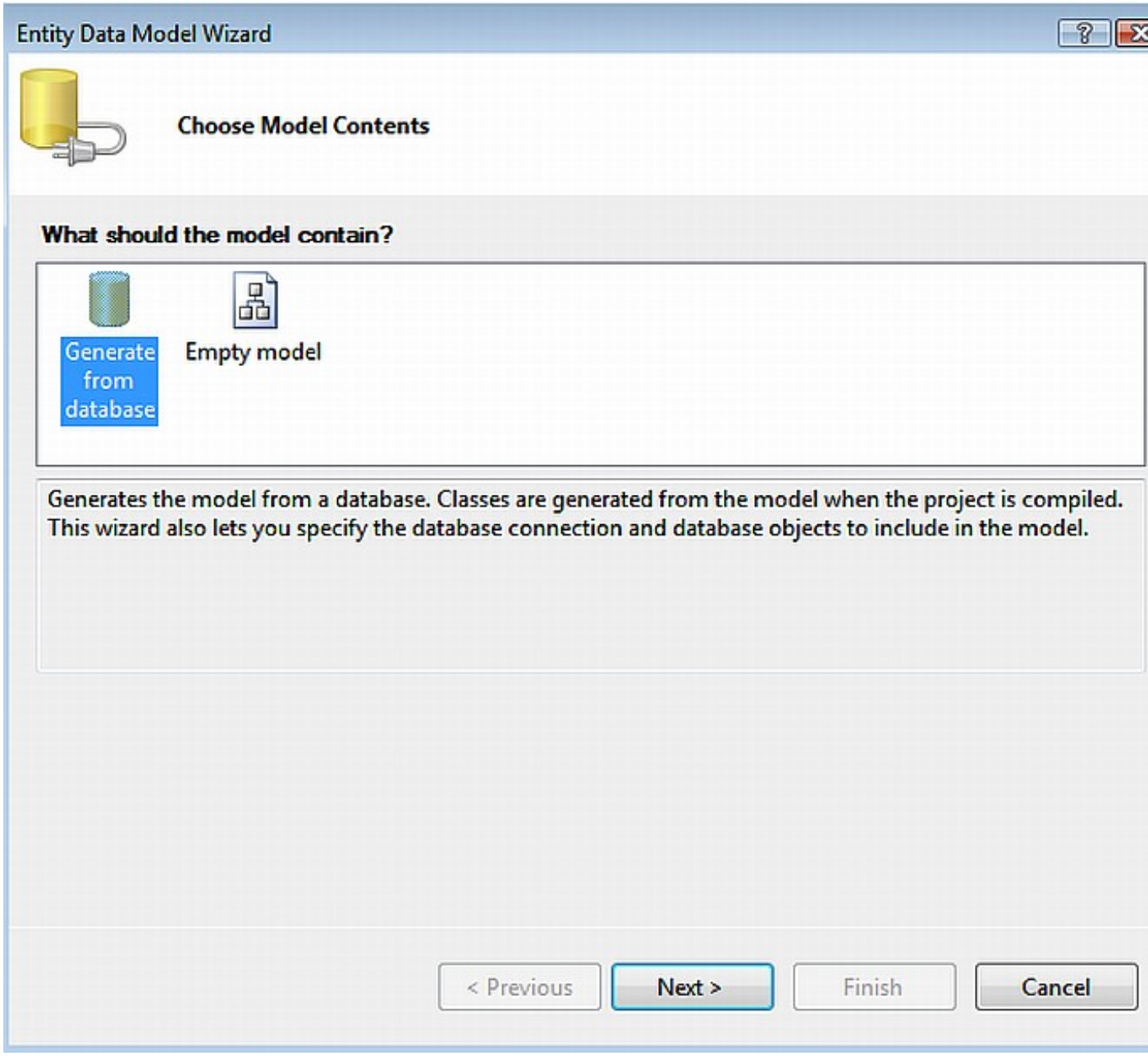
Generate from Database

model type and click

Next

.

Figure 8.410. Choose Model Contents



10. In the

    Entity Data Model Wizard

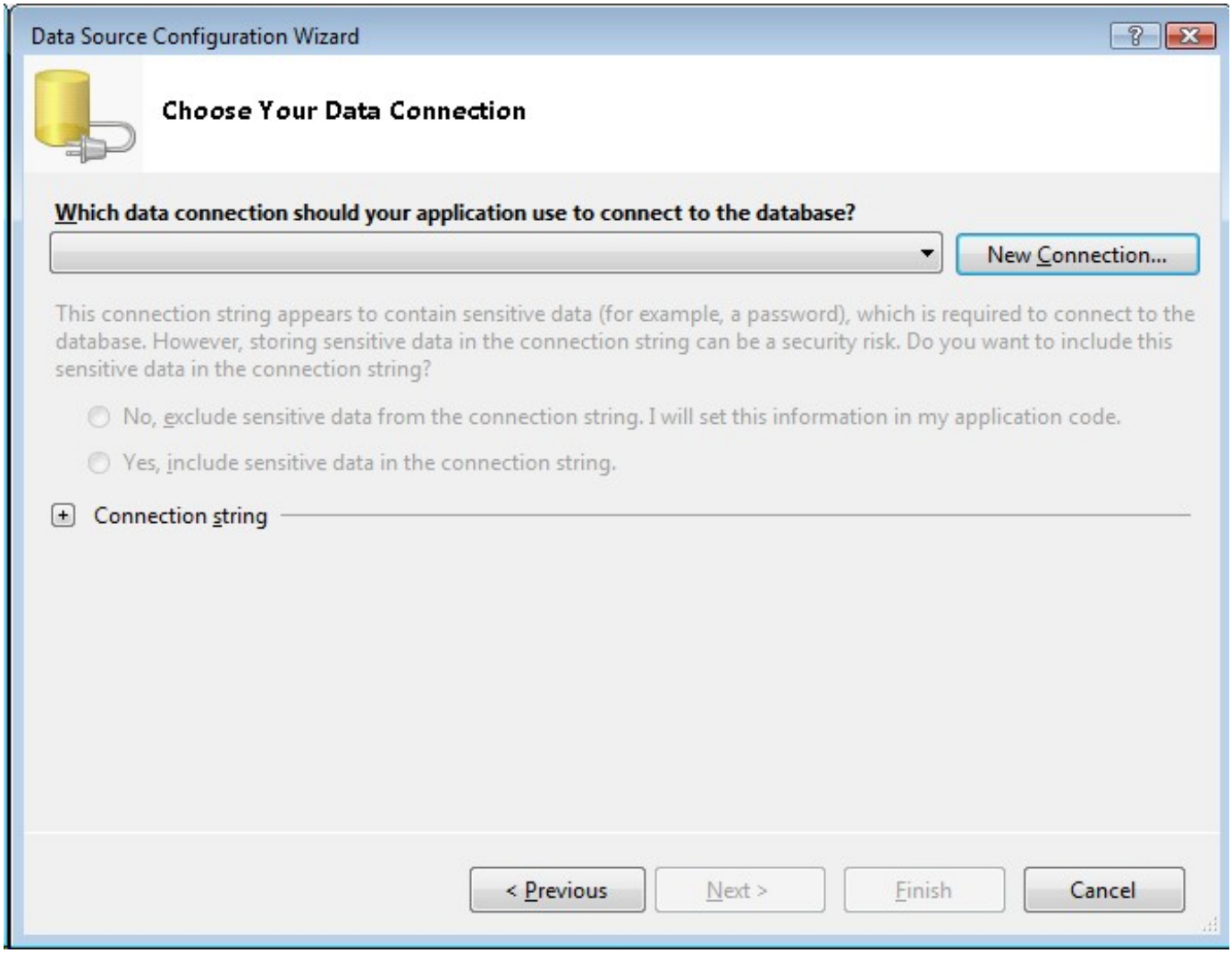
    dialog

    Choose your Data Connection

    page, select the

    New Connection button

Figure 8.411. Entity Data Model Wizard



11. In the

    Choose Data Source

    dialog, select the OpenLink

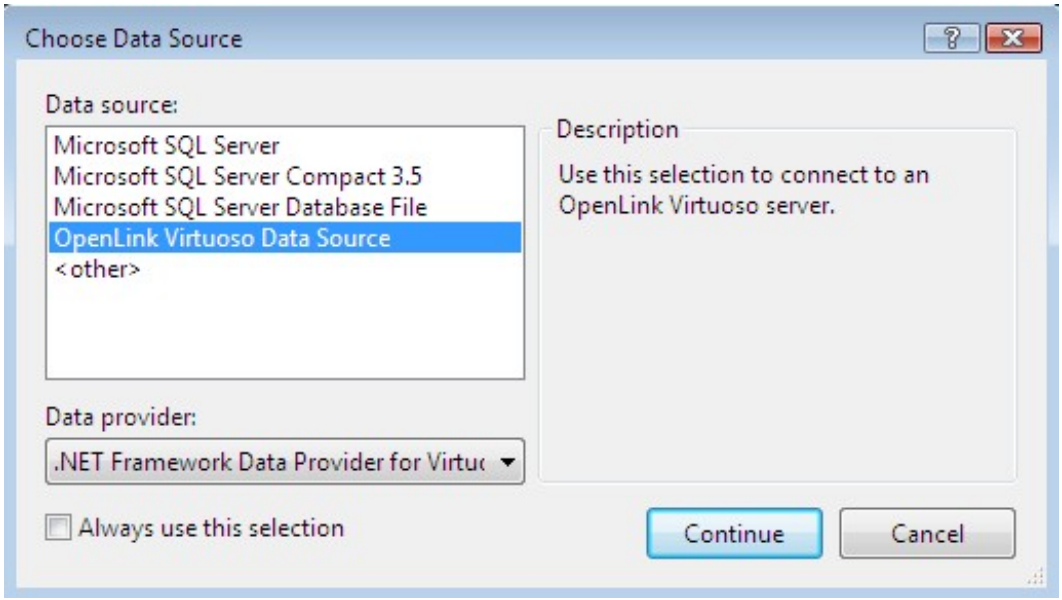
    Virtuoso Data Source

    from the list and click

    Continue

    .

Figure 8.412. Choose Data Source



12. In the

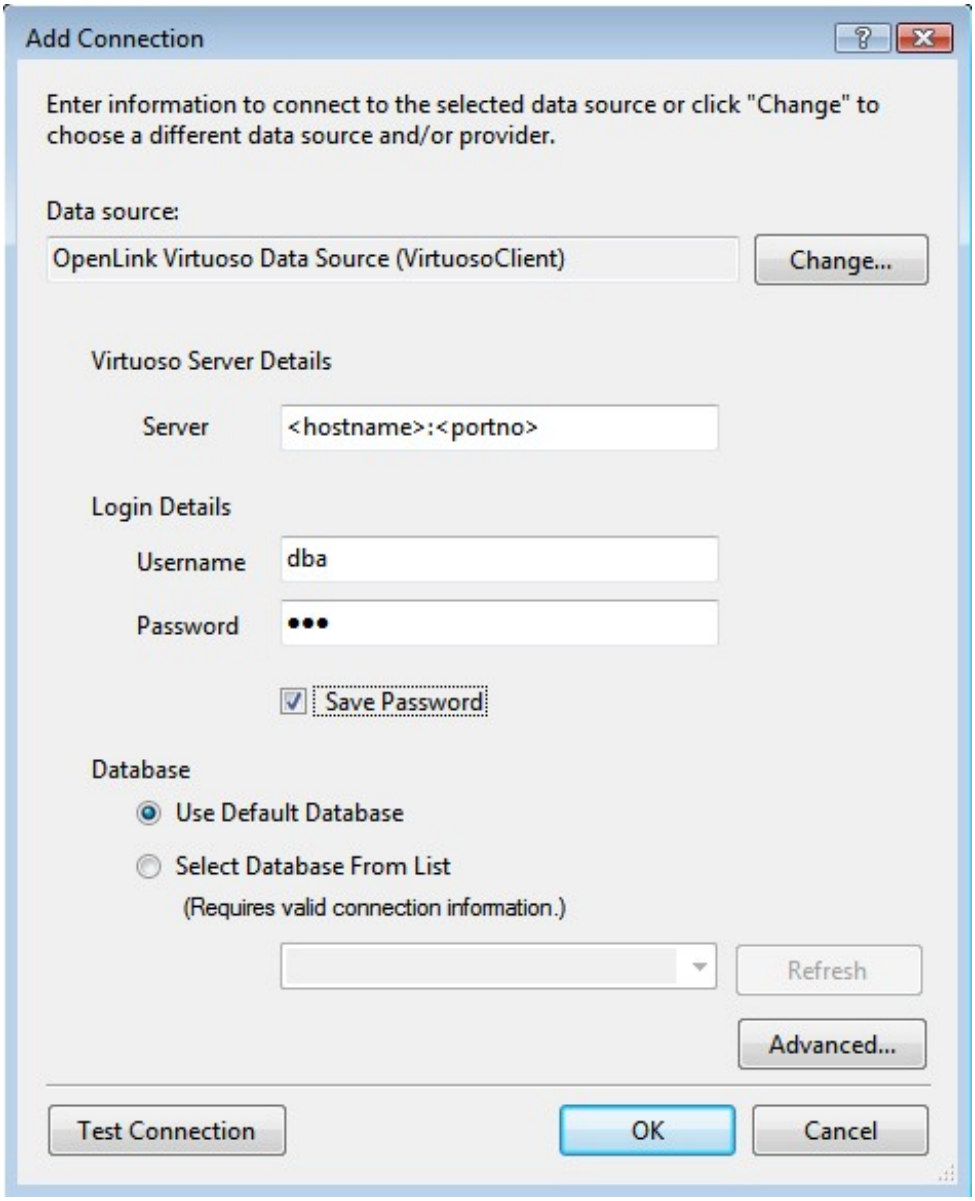
    Add Connection

    dialog, specify the

    hostname, portno, username and password

    for the target Virtuoso Server and check the Save Password check box.

    Figure 8.413. Connection Properties



13. Select the

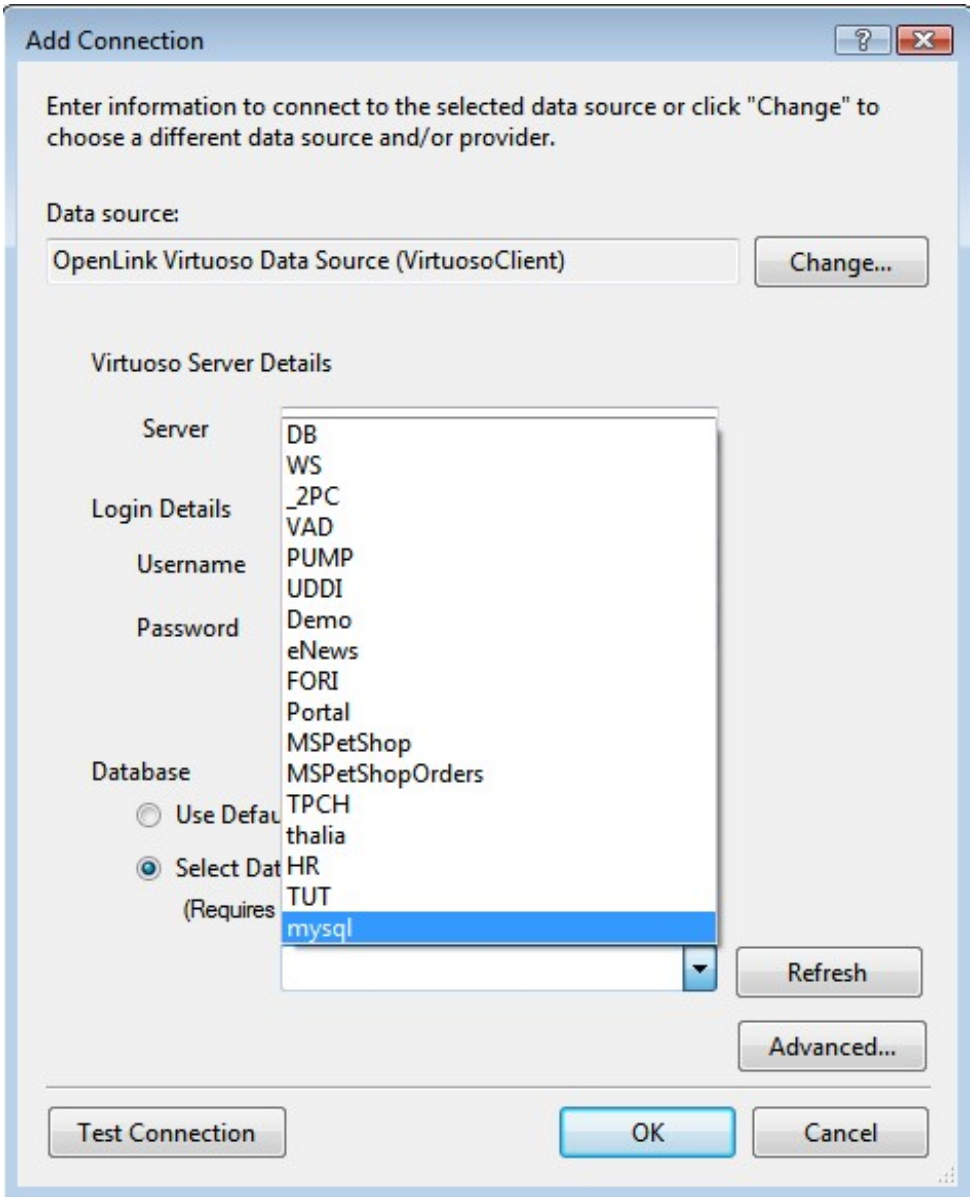
    Select Database From List

    radio button and choose the

    mysql

    database from the drop down list.

Figure 8.414. Add connection

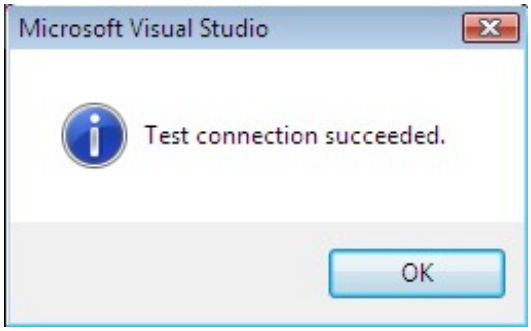


14. Press the

    Test Connection

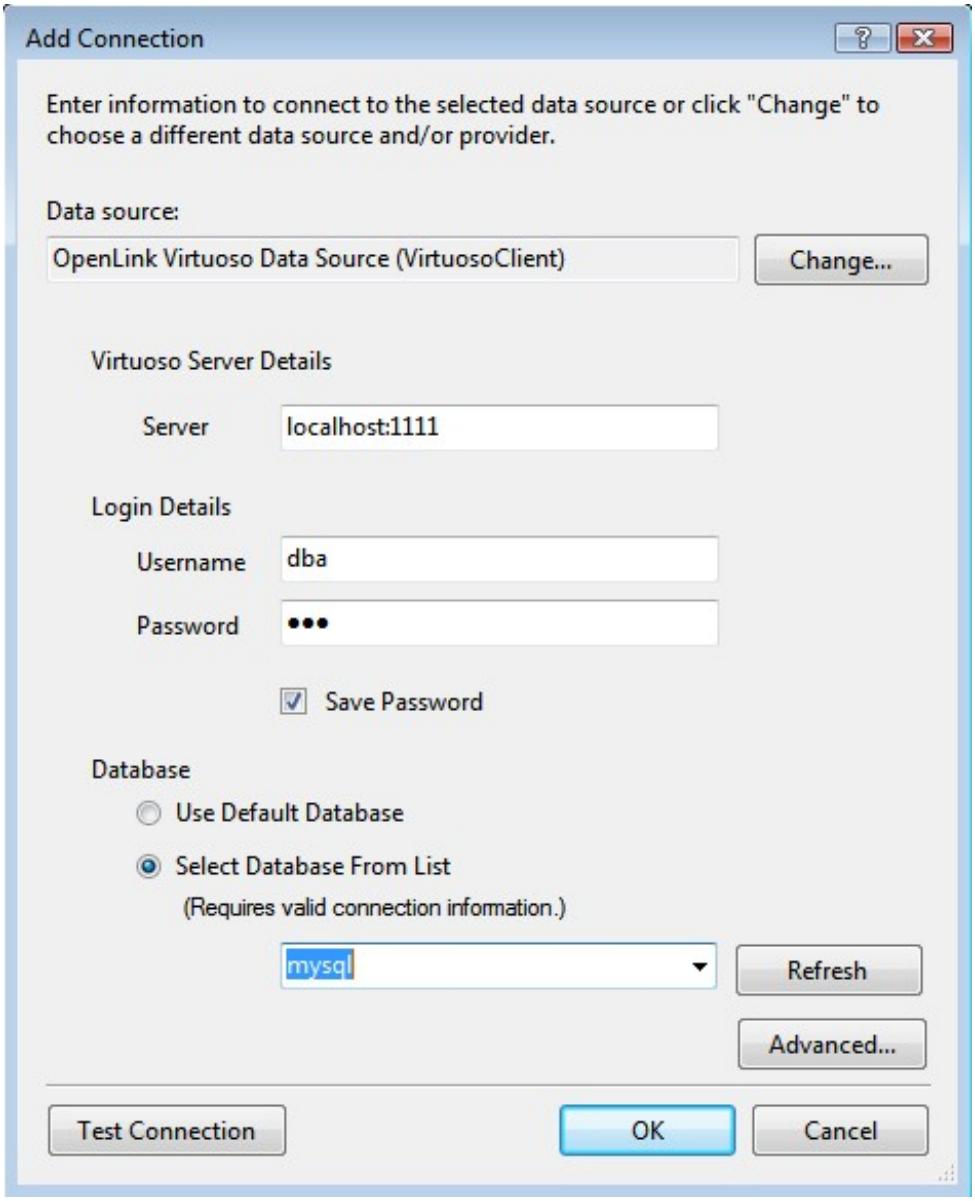
    dialog to verify that the database is accessible.

Figure 8.415. Test Connection



15. Click OK to add the connection.

Figure 8.416. Test Connection



16. Set the

    entity connect string

    name to

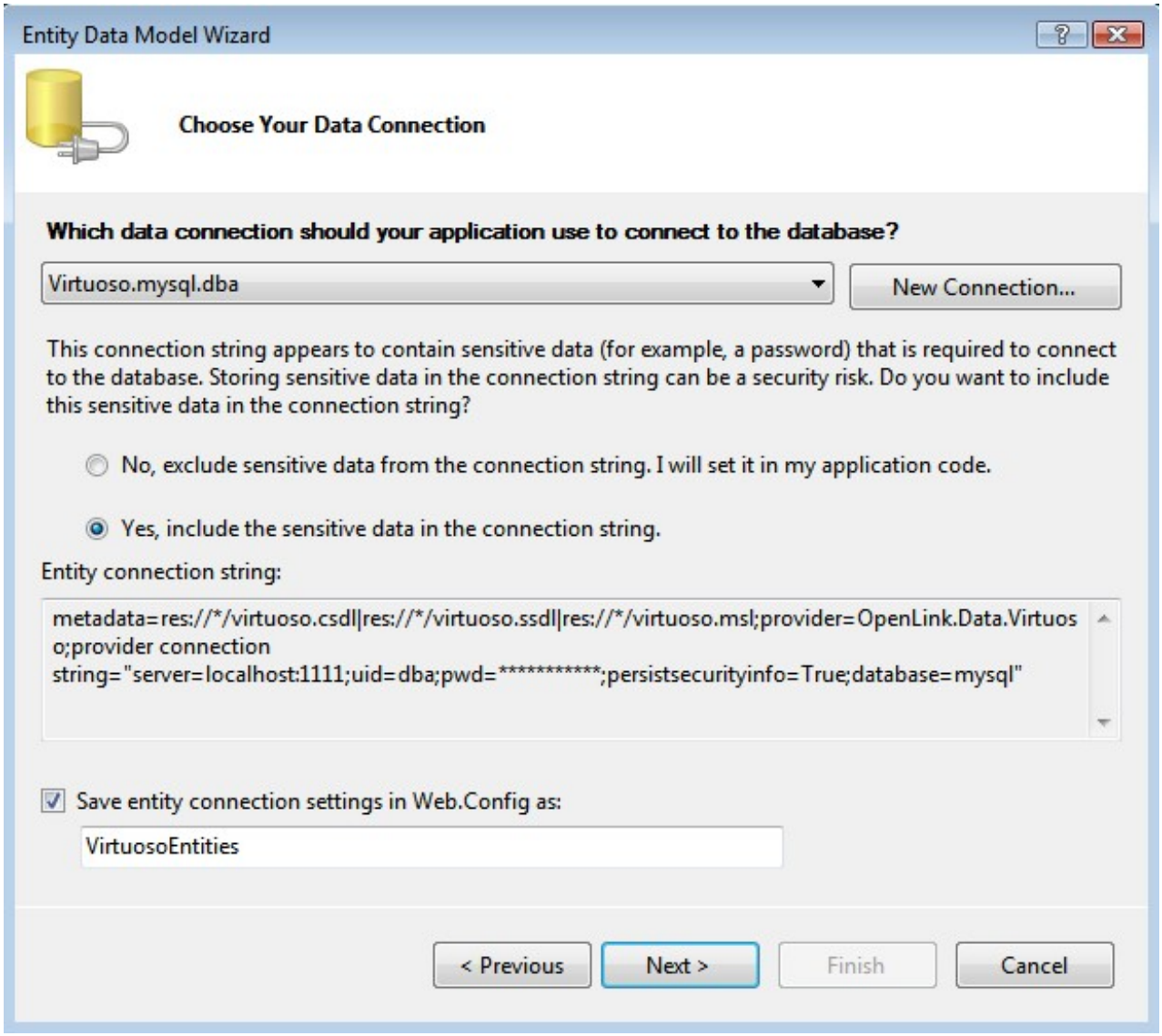
    VirtuosoEntities

    (note this name as it is required in the section on creating and ADO.Net Data Service) and click

    Next

    .

    Figure 8.417. entity connect string



17. In the

    Choose your Database Objects

    page tick the

    Tables

    check box to select all tables in the mysql catalog for addition to the Entity Data Model. Set the

    Model Namespace

    to

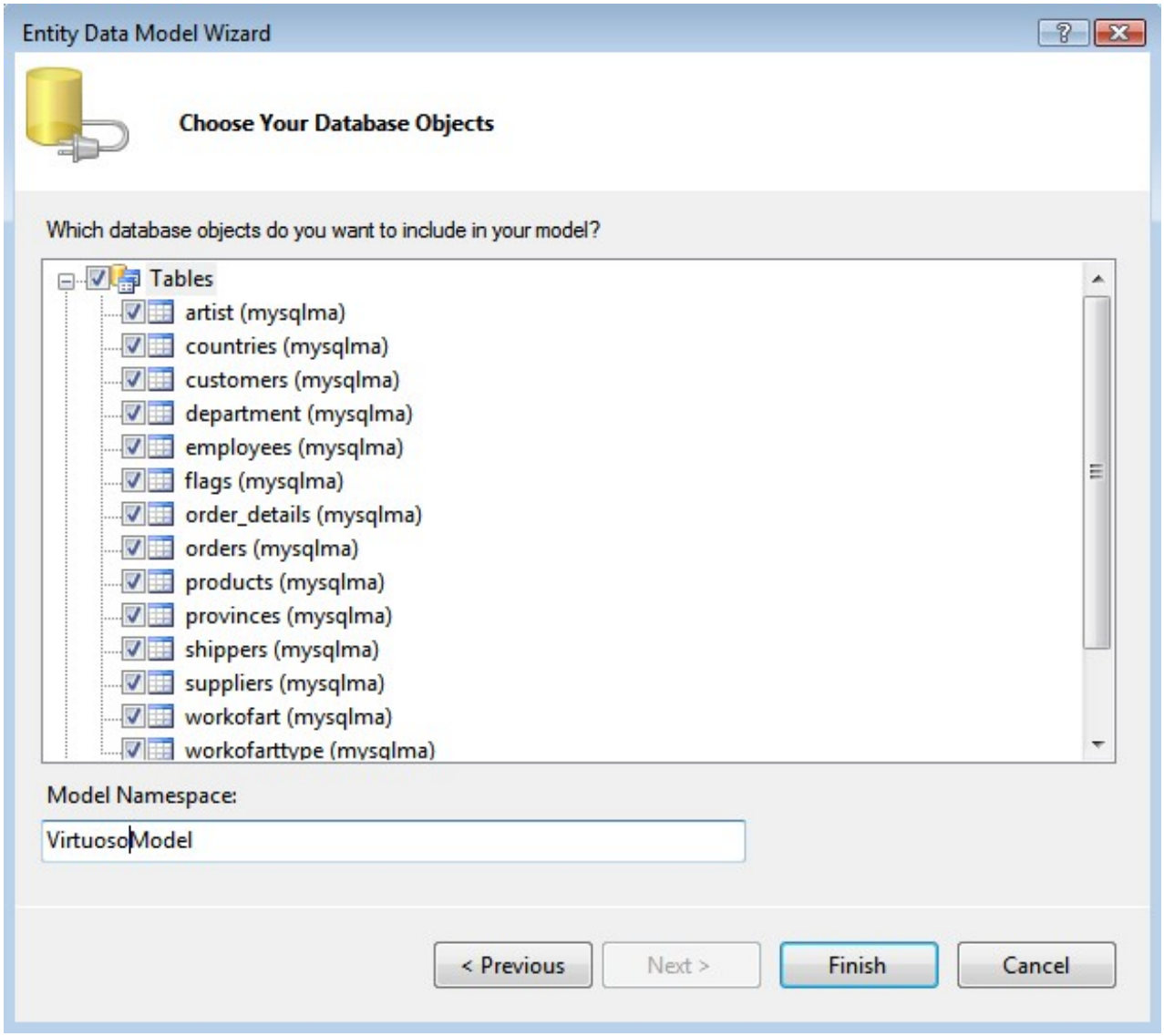
    VirtuosoModel

    and click

    Finish

    .

Figure 8.418. Database Objects

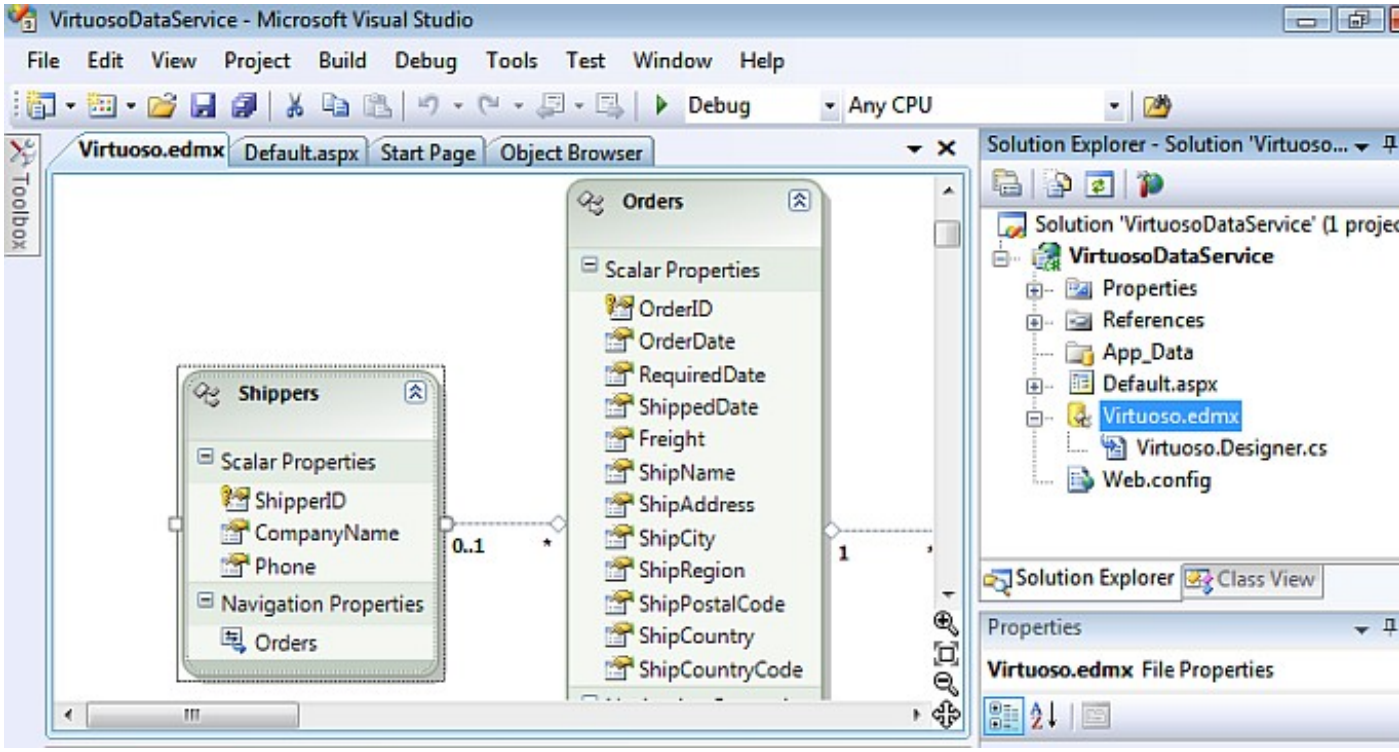


18. The

    Virtuoso.edmx

    EDM will be created with the tables and relationships displayed in the Visual Studio IDE

    Figure 8.419. Virtuoso.edmx



Creation for the Entity Data Model for the MySQL Northwind database is now complete.

8.8.5. Using EDM to create Entity Framework based applications

Now that a Microsoft Entity Data Model has been created for the Northwind database, Entity Framework applications can be created to make use of it.

Entity Frameworks based ADO.NET Data Service

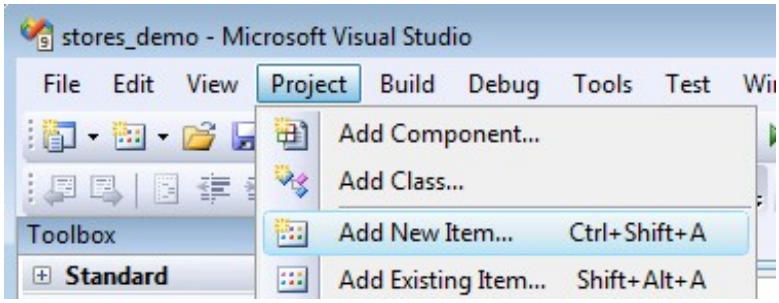
An ADO.Net Data Service for the MySQL tables can be created using the Entity Data Model created in the Creating EDM in Visual Studio 2008 section .

1. Open the

   VirtuosoDataService

   project created in the Creating EDM in Visual Studio 2008 section .
2. Select the Project -> Add New Item menu option.

   Figure 8.420. VirtuosoDataService



3. The

   Add New Item

dialog will appear. Choose the

ADO.NET Data Service template

. Give it the name

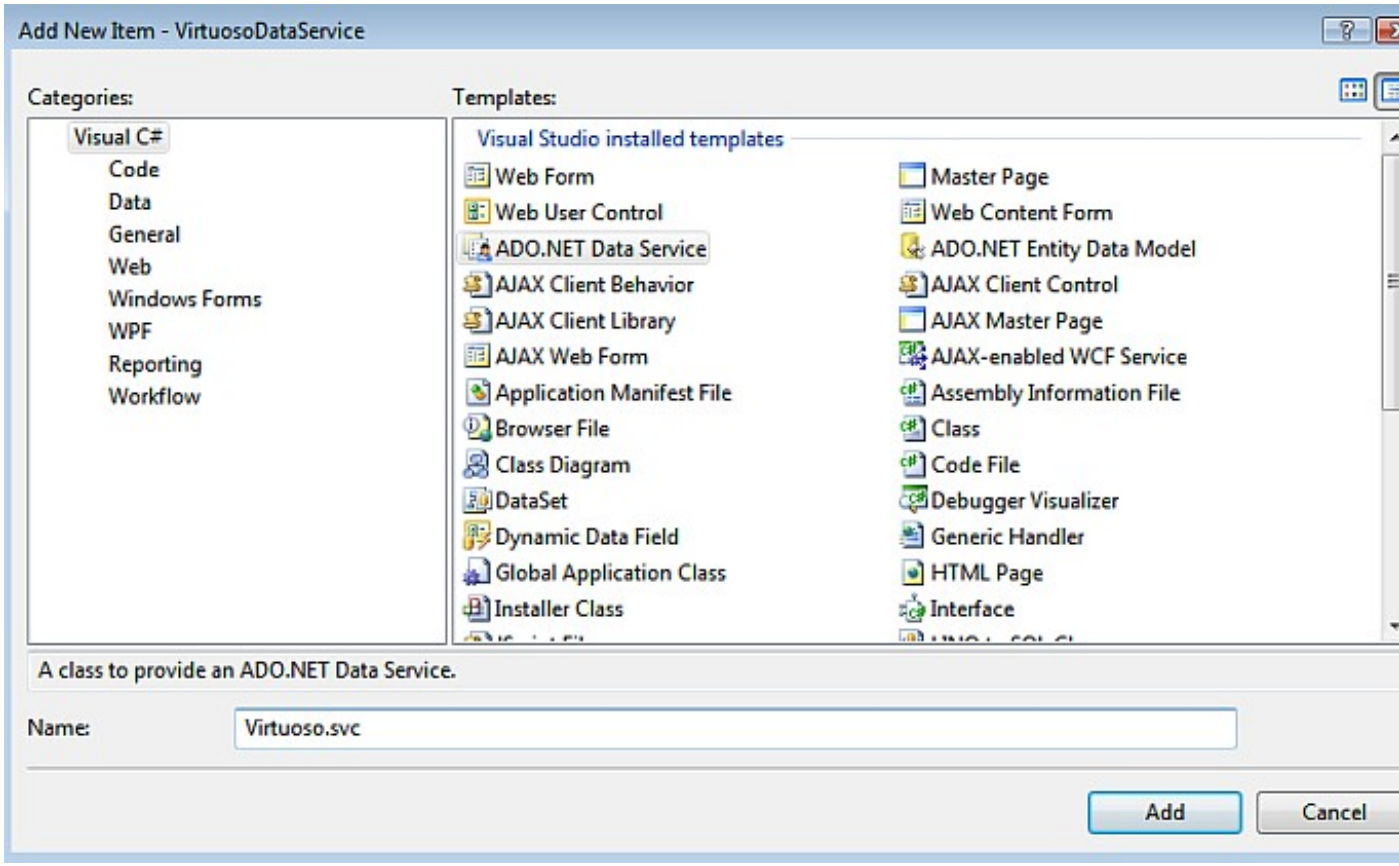
Virtuoso.svc

, and click

Add

to create the ADO.Net Data Service.

Figure 8.421. Add New Item



4. In the newly created

Virtuoso.svc.cs

Data Service file, add the data source class name of

VirtuosoEntities

(note this is the name set in the Creating EDM in Visual Studio 2008 section) as the

DataService

name. Enable the access to the Data Service by adding the entry

config.SetEntitySetAccessRule("", EntitySetRights.All);*

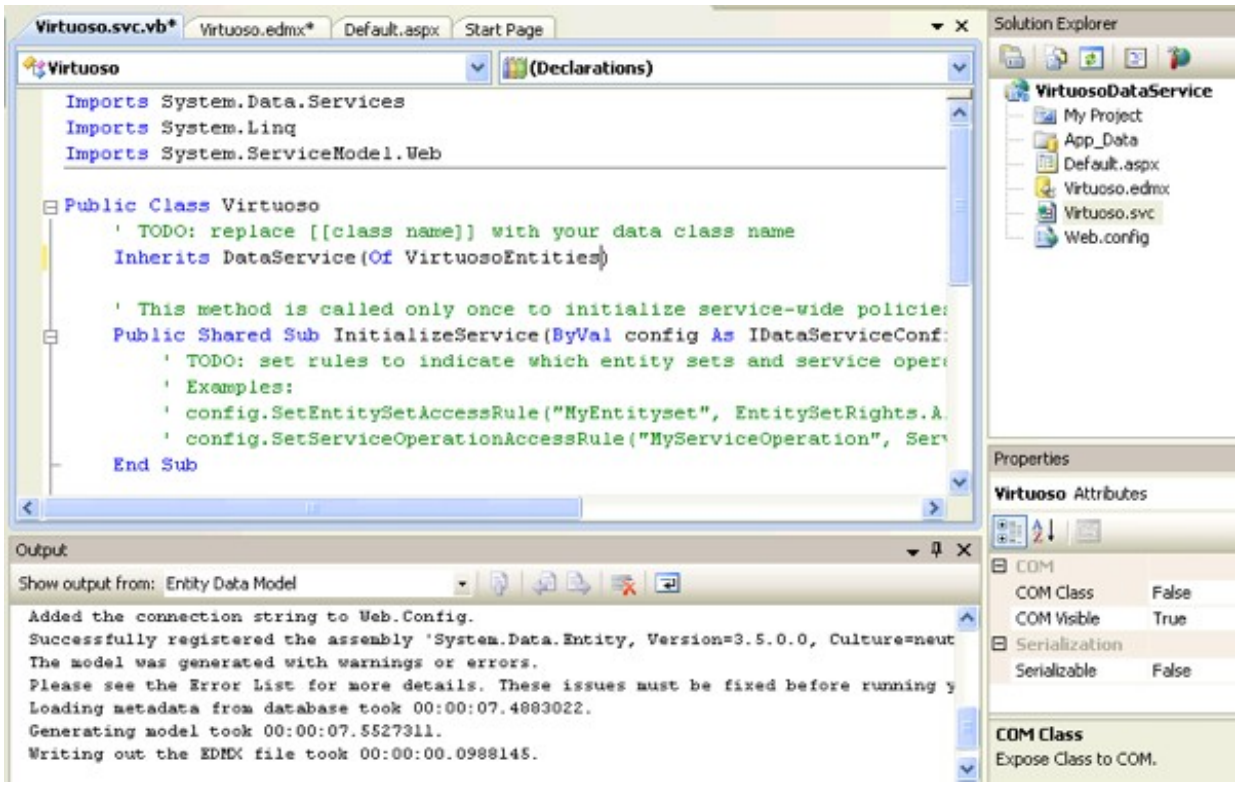
in the

InitializeService

method.

```
// C#

using System;
using System.Web;
using System.Collections.Generic;
using System.ServiceModel.Web;
using System.Linq;
using System.Data.Services;

namespace SimpleDataService
{
    public class Northwind: DataService<VirtuosoDemoEntities>
    {
        public static void InitializeService(IDataServiceConfiguration  config)
        {
            config.SetEntitySetAccessRule("*", EntitySetRights.All);
        }
    }
}
```

Figure 8.422. Virtuoso.svc.cs

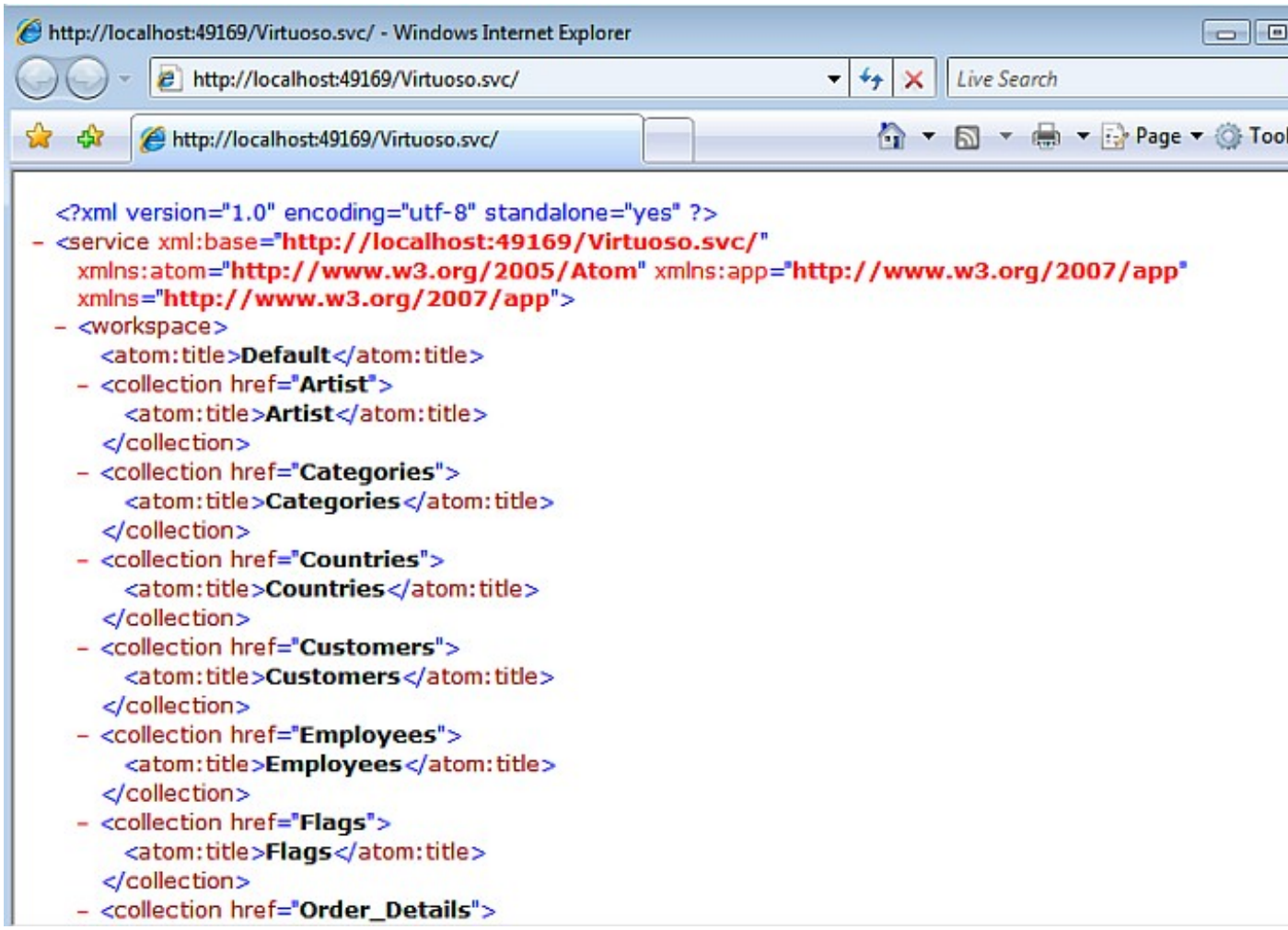


5. To test the Data Service, simply hit

   Ctrl+F5

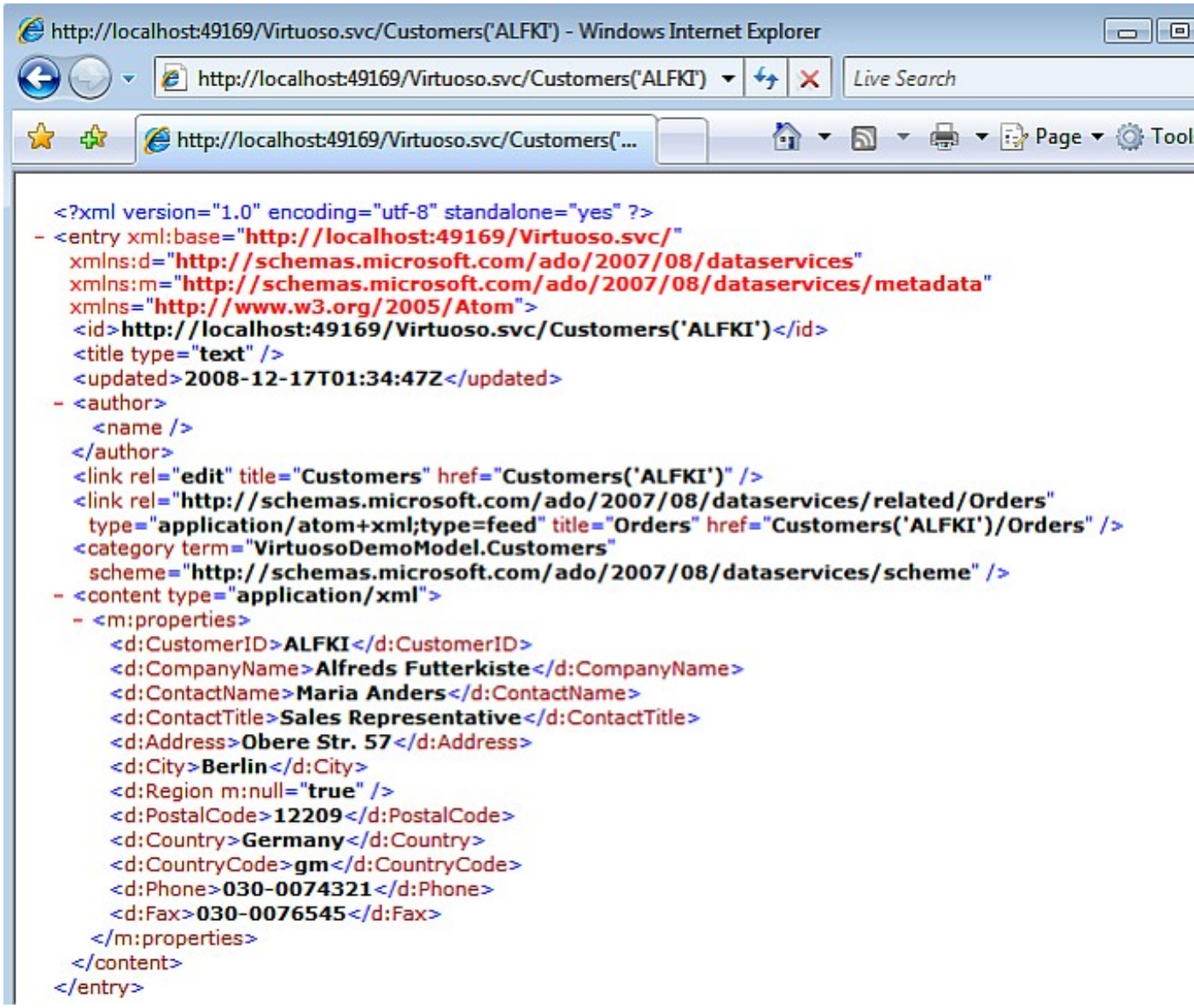
   within Visual Studio. This will start the development web server, run the Data Services server inside and load a Web browser page displaying the list of available tables/entities for the mysql Northwind database catalog.

Figure 8.423. Data Service test



6. To access a specific entity instance like the Customers table ALFKI record, use this convention: http://host/vdir/Virtuoso.svc/Customers('ALFKI') .

Figure 8.424. Customers



Notes:

1. *Important*

   - To view

   Atom

   (the default format returned by an ADO.NET Data Service) in Internet Explorer, you must first ensure that

   Feed Reading View

   is turned

   off

   . This can be done on the

   Content tab

   of

Tools in Internet Options

.

2. If a Data Services entity instance URI page fails to load you can turn

   Verbose

   errors on by adding

   config.UseVerboseErrors = true;

   in the

   virtuoso.svc.cs InitializeService

   method to obtain more detailed information from the server as to why the page failed to load:

```
public static void InitializeService(IDataServiceConfiguration config)
{
config.UseVerboseErrors = true;
config.SetEntitySetAccessRule("*", EntitySetRights.All);
}
```

Visual Studio Windows DataGrid Form Application

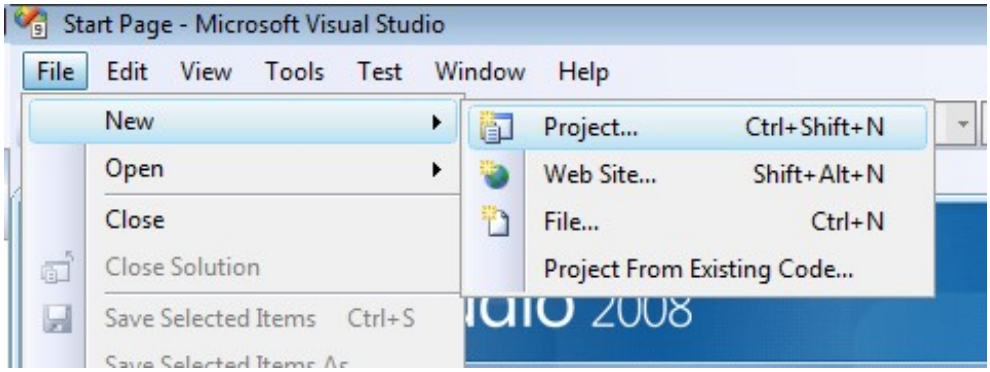
This section details the steps required to create a simple Visual Studio 2008 Windows Form application, with associated DataGridView control for displaying data in selected tables from the target database.

1. Launch the Visual Studio 2008 SP1 IDE.

   Figure 8.425. Visual Studio 2008 SP1 IDE

2. Create a *Web Application* project by going to the *File* menu in Visual Studio and choosing *New Project* .

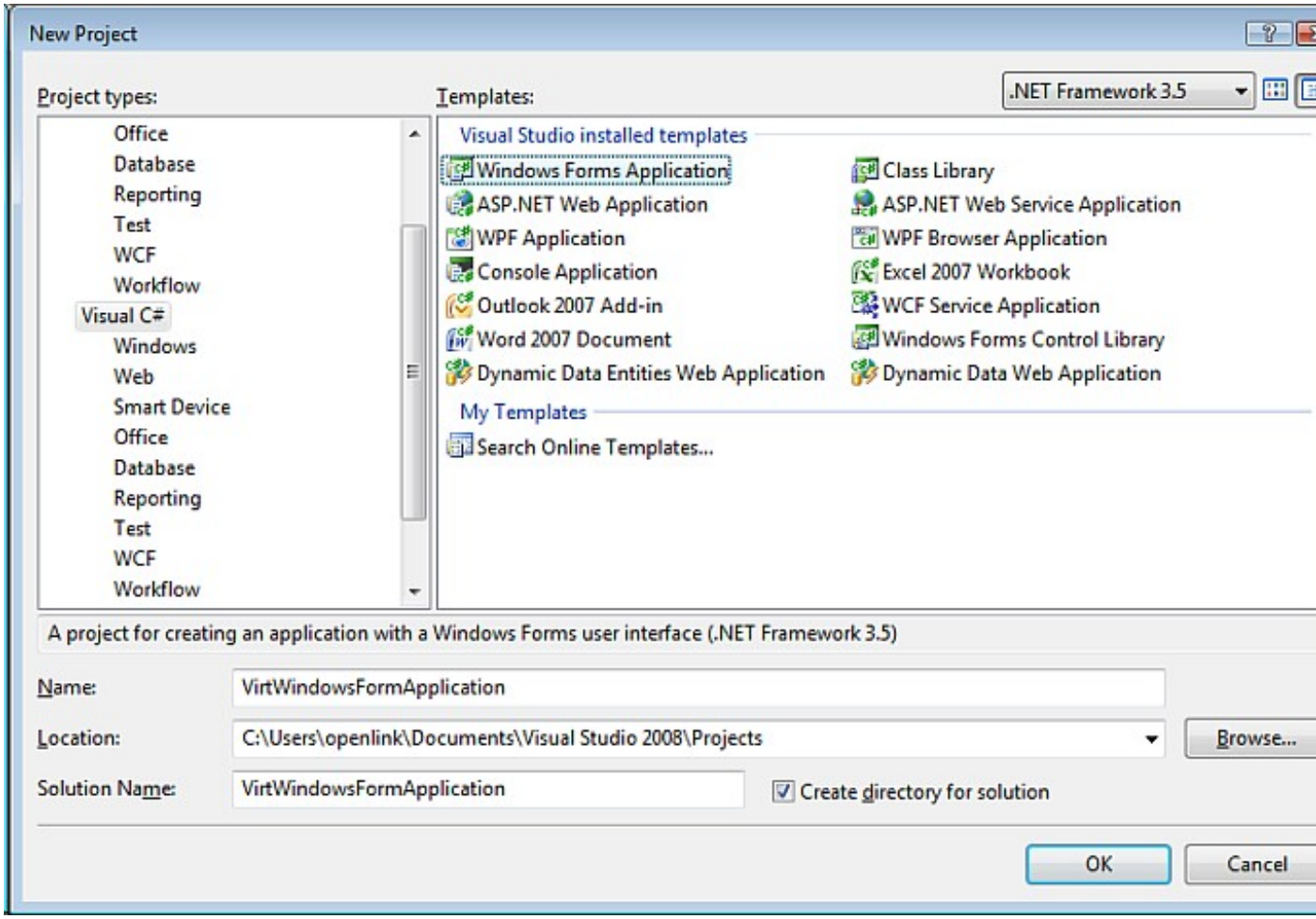
   Figure 8.426. Web Application



3. When the New Project window appears, choose either

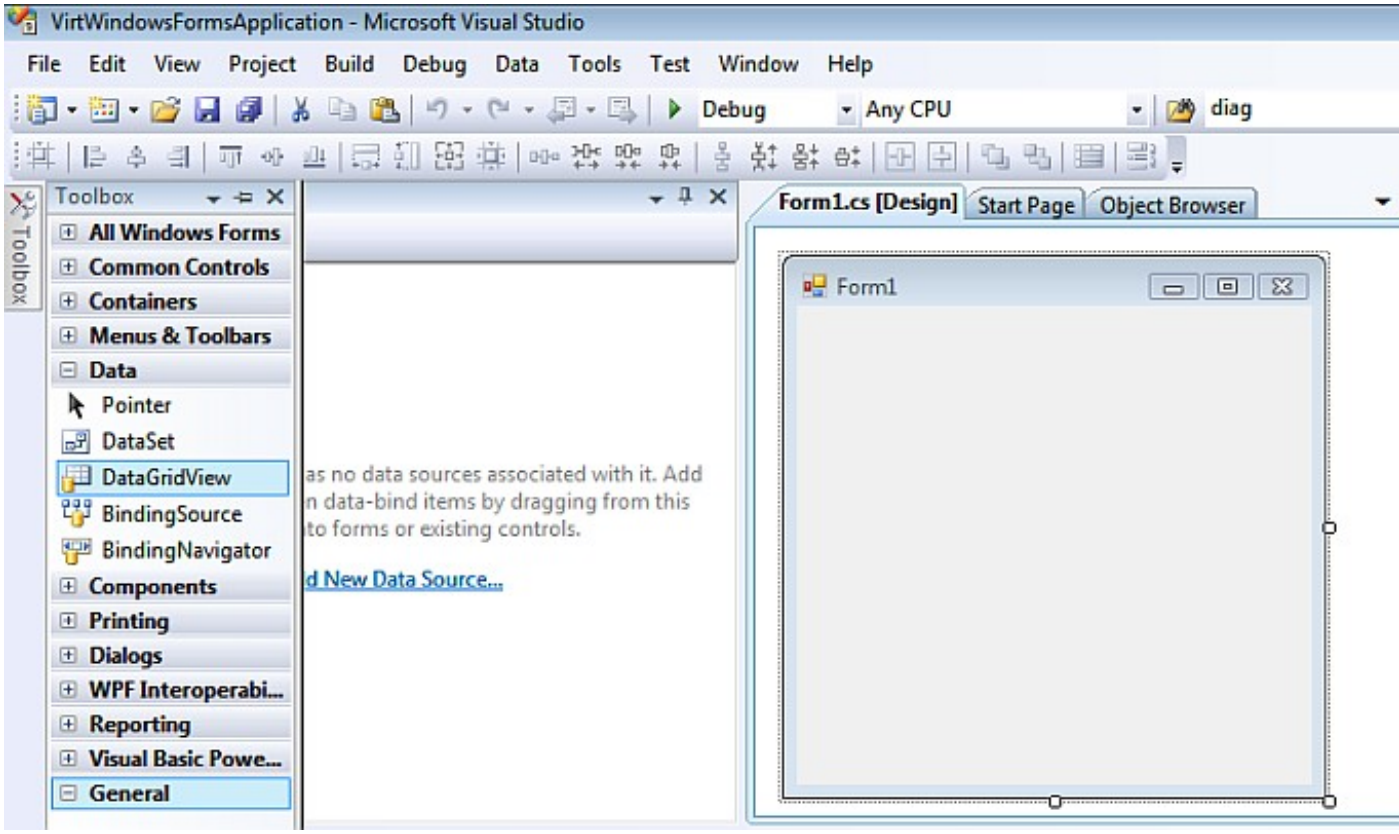
   Visual Basic or *Visual C#* as your programming language.
4. Within the language category, click on *Windows* and select *Windows Form Application* from the right-hand panel.
5. Choose a name for the project, for example *VirtWindowsFormApplication* , and click *OK* .

Figure 8.427. Web Application



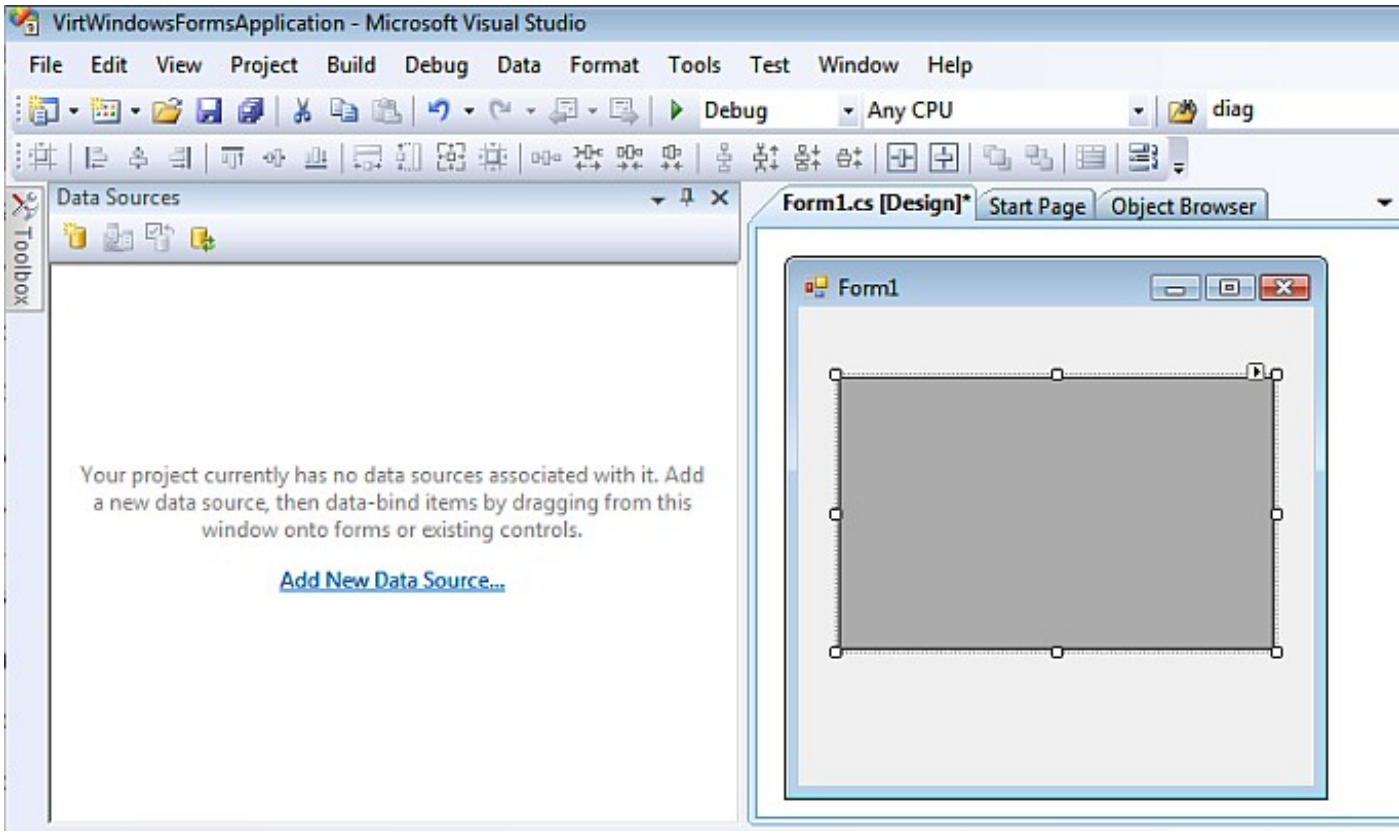
6. In the *Toolbox* , expand *Data Controls* , and drag the *DataGridView* control onto the form.

Figure 8.428. Toolbox



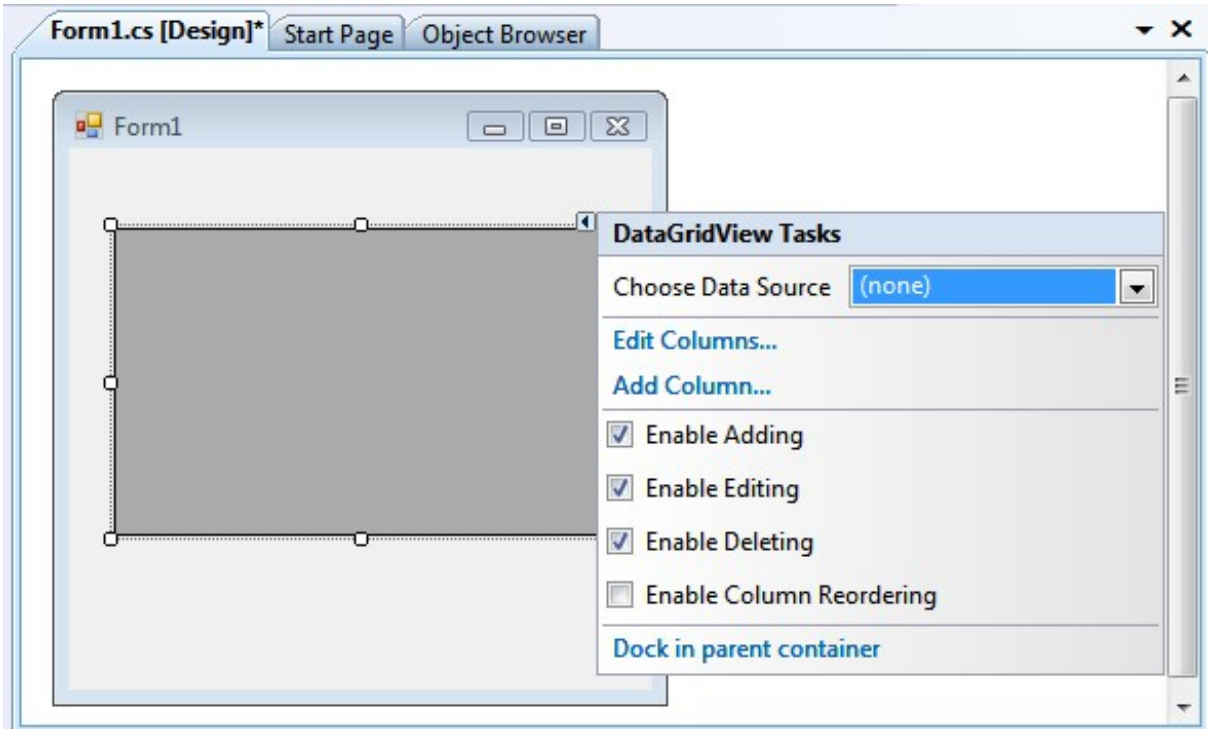
7. Click on the little *arrow* in the top right of the *DataGridView* control. This loads the *DataGridView Task* menu.

Figure 8.429. DataGridView Task



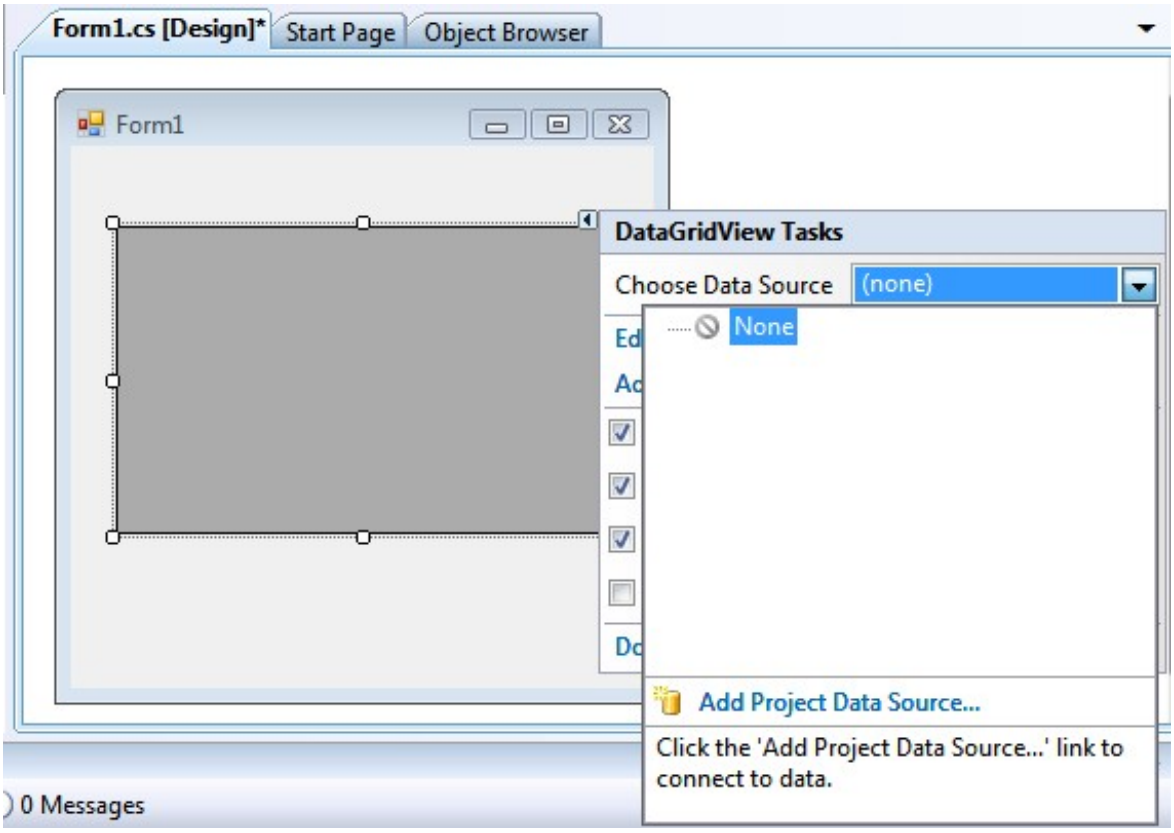
8. Click on the *Choose Data Source* list box.

Figure 8.430. Choose Data Source



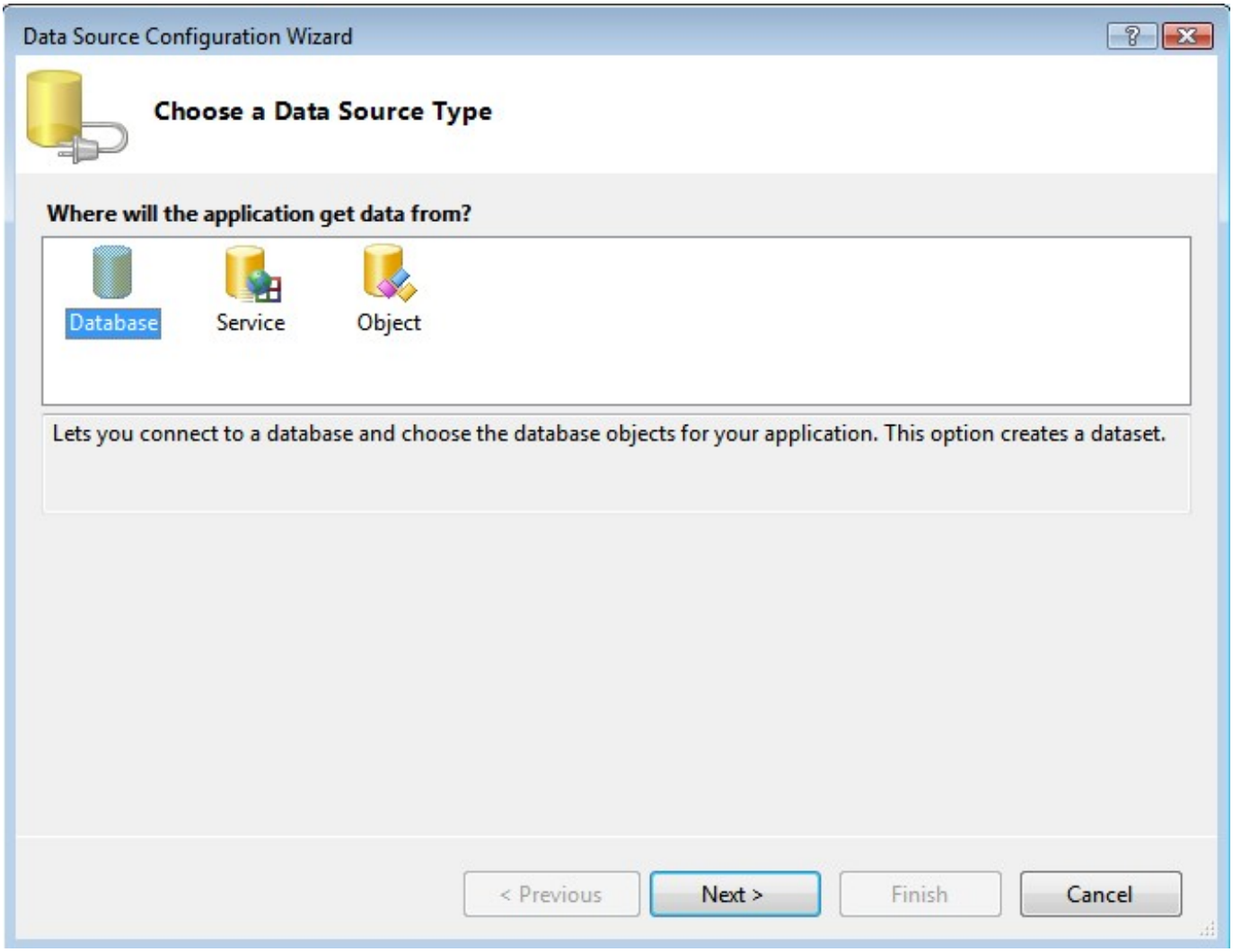
9. Click on the *Add Project Data Source* link to connect to a data source.

Figure 8.431. Add Project Data Source



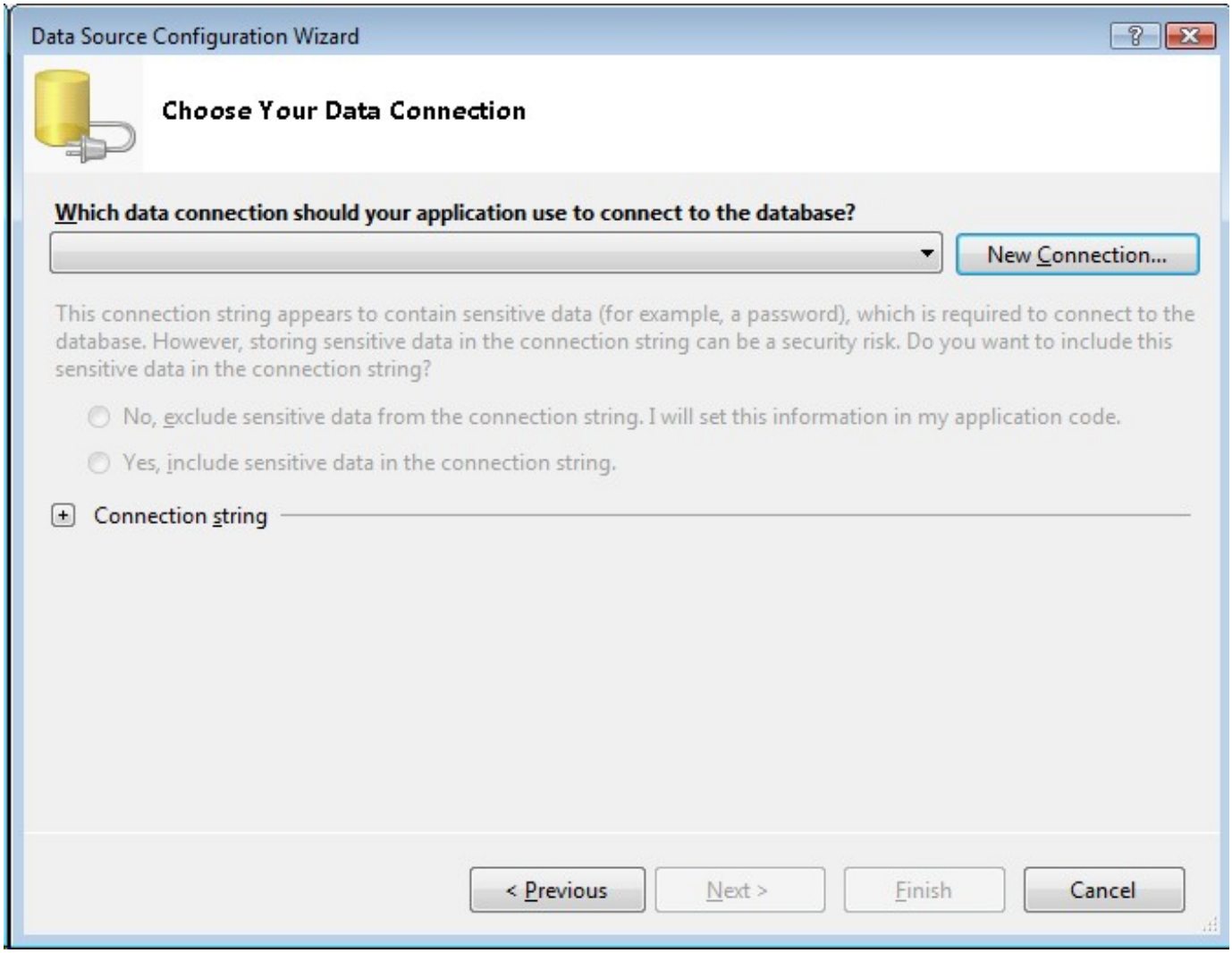
10. In the *Data Source Configuration Wizard* dialog *Choose Data Source Type* page select the *Database* data source type and click *Next* .

Figure 8.432. Data Source Type



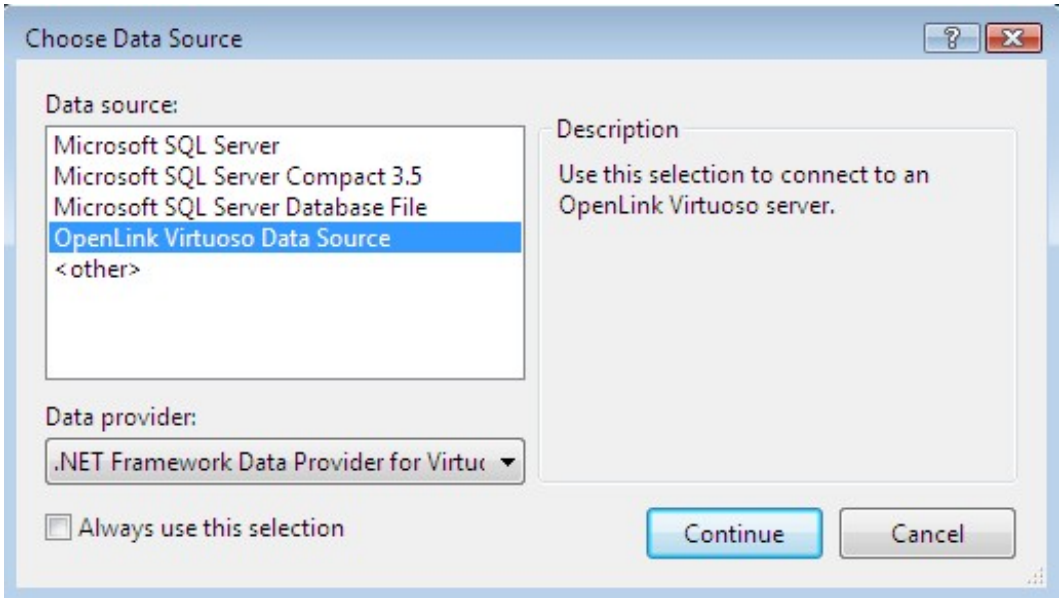
11. In the *Data Source Configuration Wizard* dialog *Choose your Data Connection* page, select the *New Connection* button

Figure 8.433. Data Source Configuration Wizard



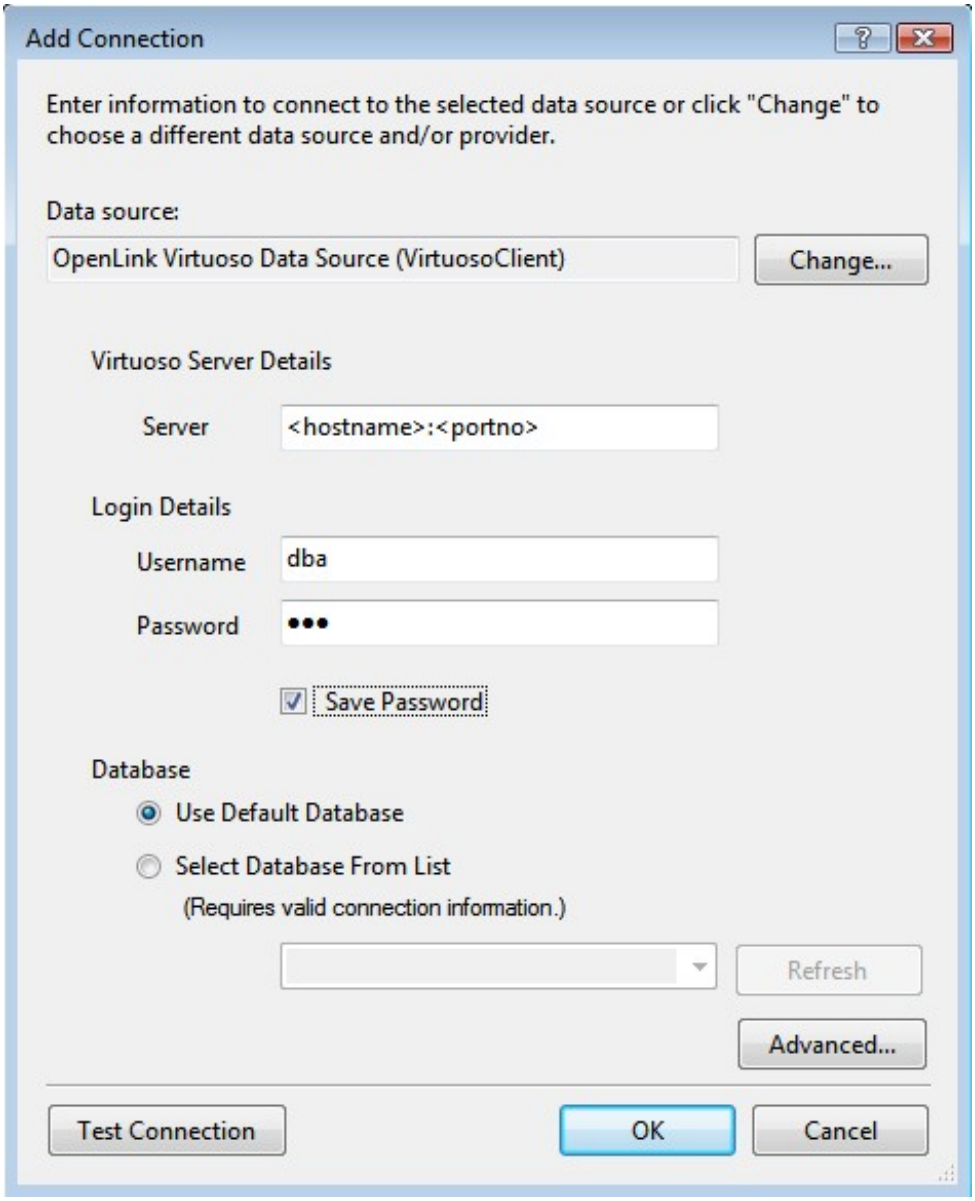
12. In the *Choose Data Source* dialog, select the OpenLink *Virtuoso Data Source* from the list and click *Continue* .

Figure 8.434. Data Source



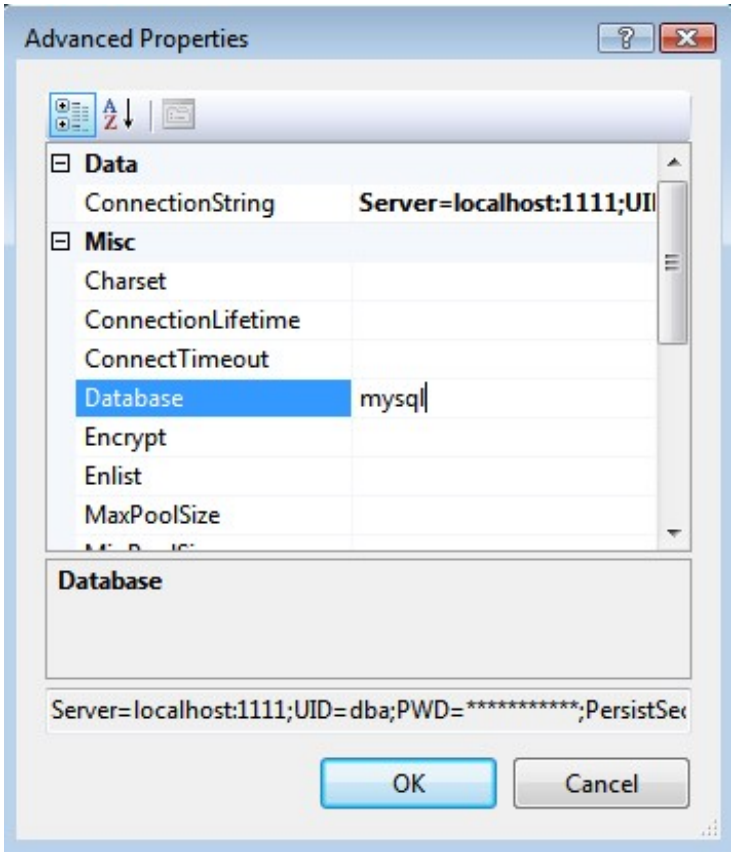
13. In the *Add Connection* dialog, specify the *hostname, portno, username and password* for the target Virtuoso Server and check the Save Password check box.

Figure 8.435. Connection Properties



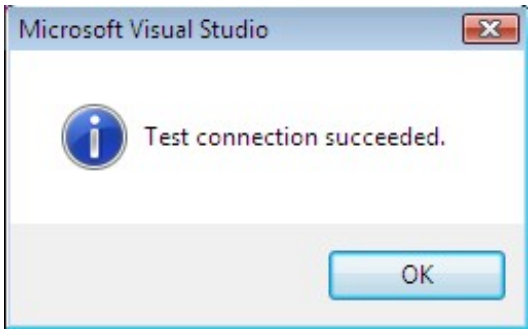
14. Select the *Select Database From List* radio button and choose the *mysql* database from the drop down list.

Figure 8.436. Add connection



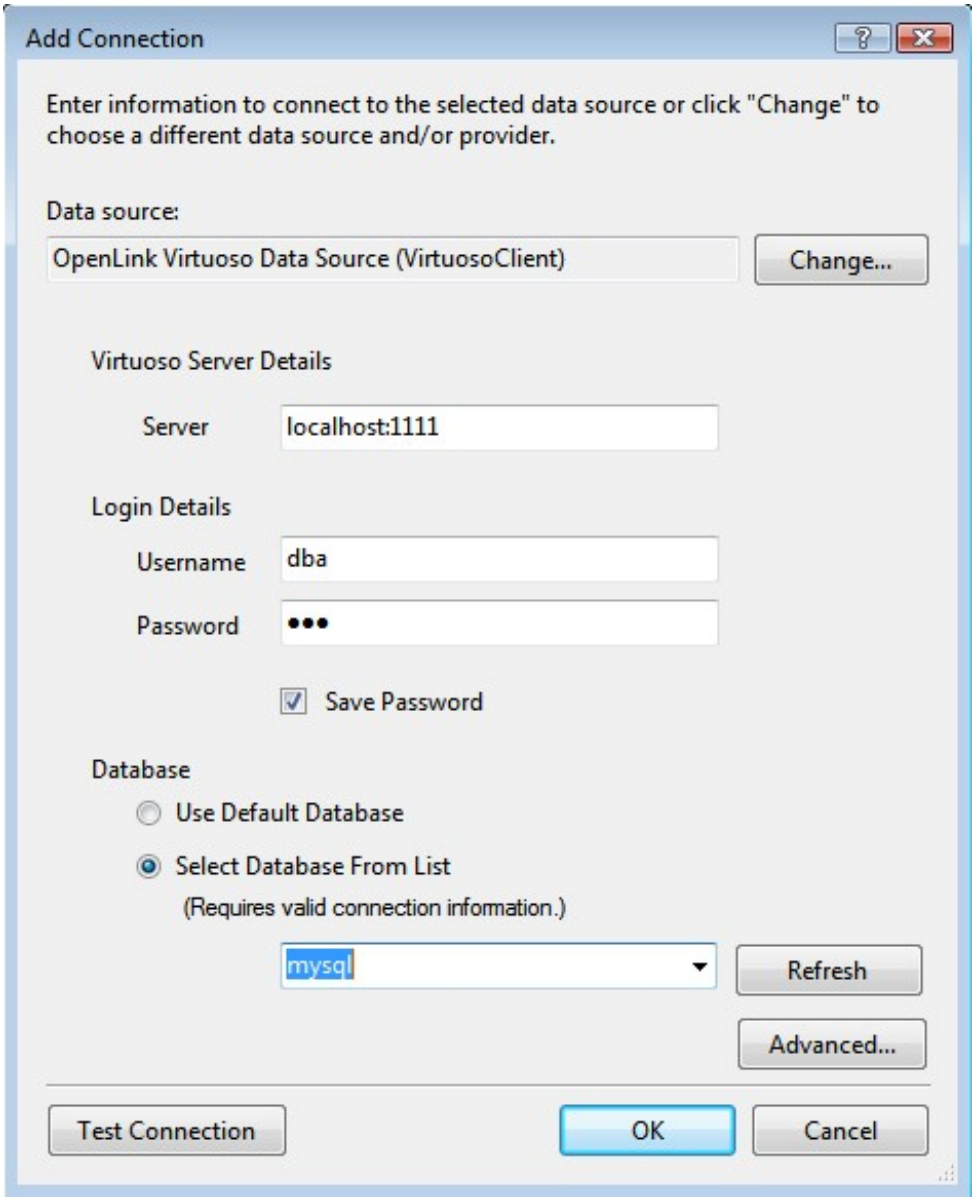
15. Press the *Test Connection* dialog to verify that the database is accessible.

Figure 8.437. Test Connection



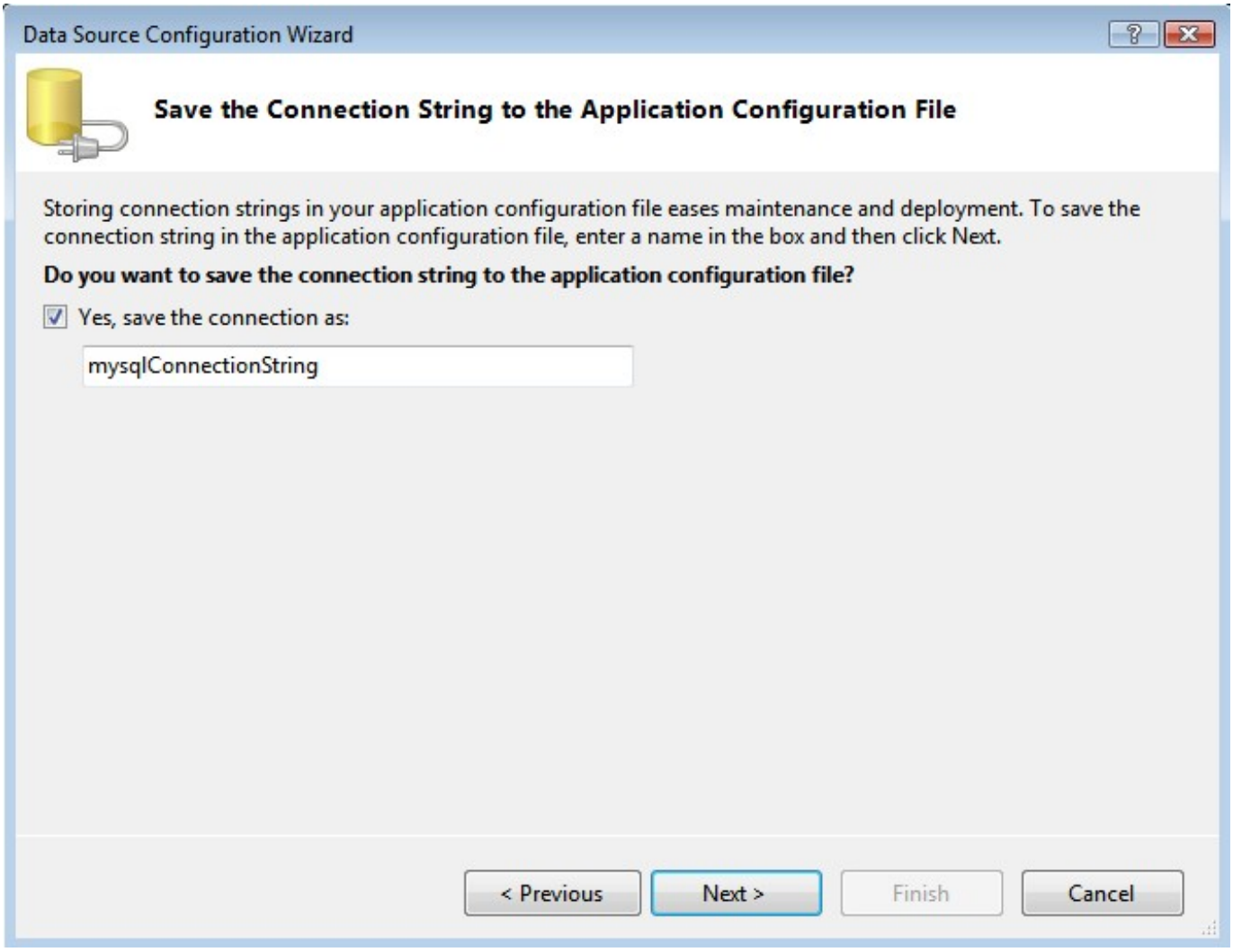
16. Click OK to add the connection.

Figure 8.438. Test Connection



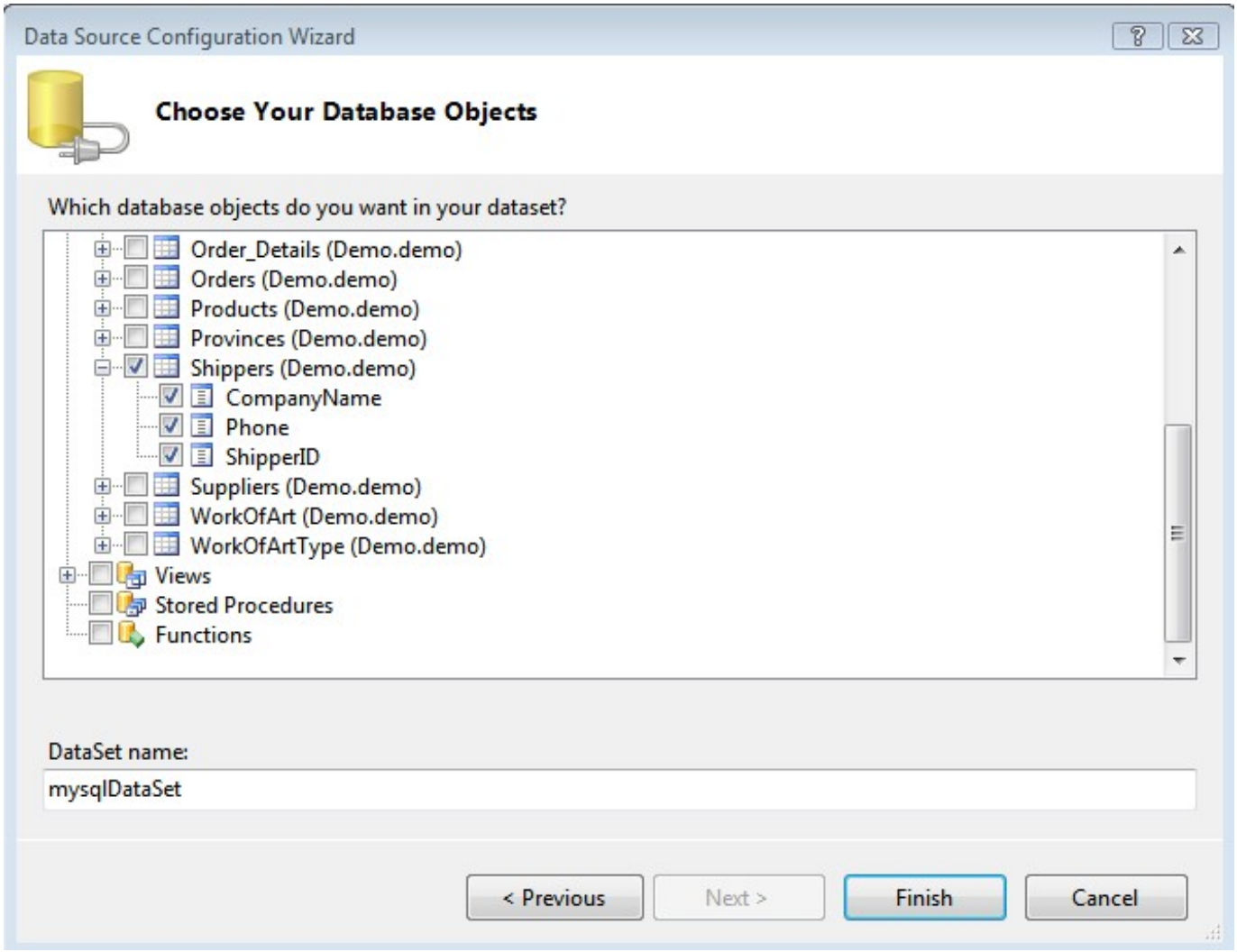
17. Leave the default connect string name*mysqlConnectionString* and click *Next* .

Figure 8.439. mysqlConnectionString



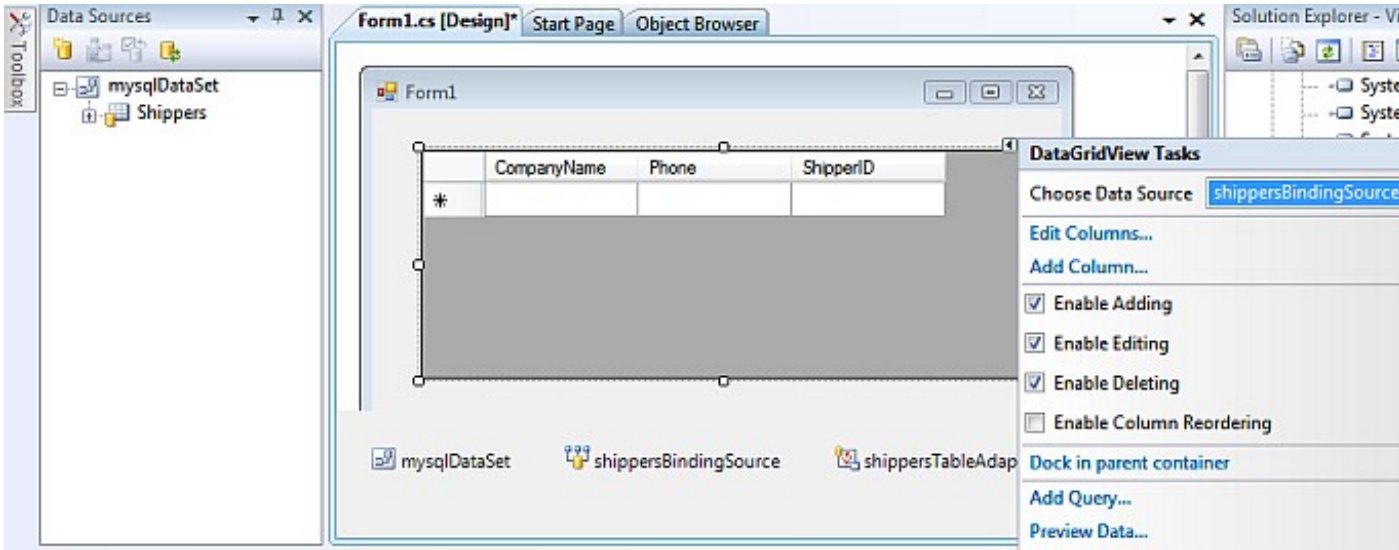
18. From the list of available tables returned for the mysql database, select the *Shippers* table to be associated with the *DataGridView* control.

Figure 8.440. mysql database



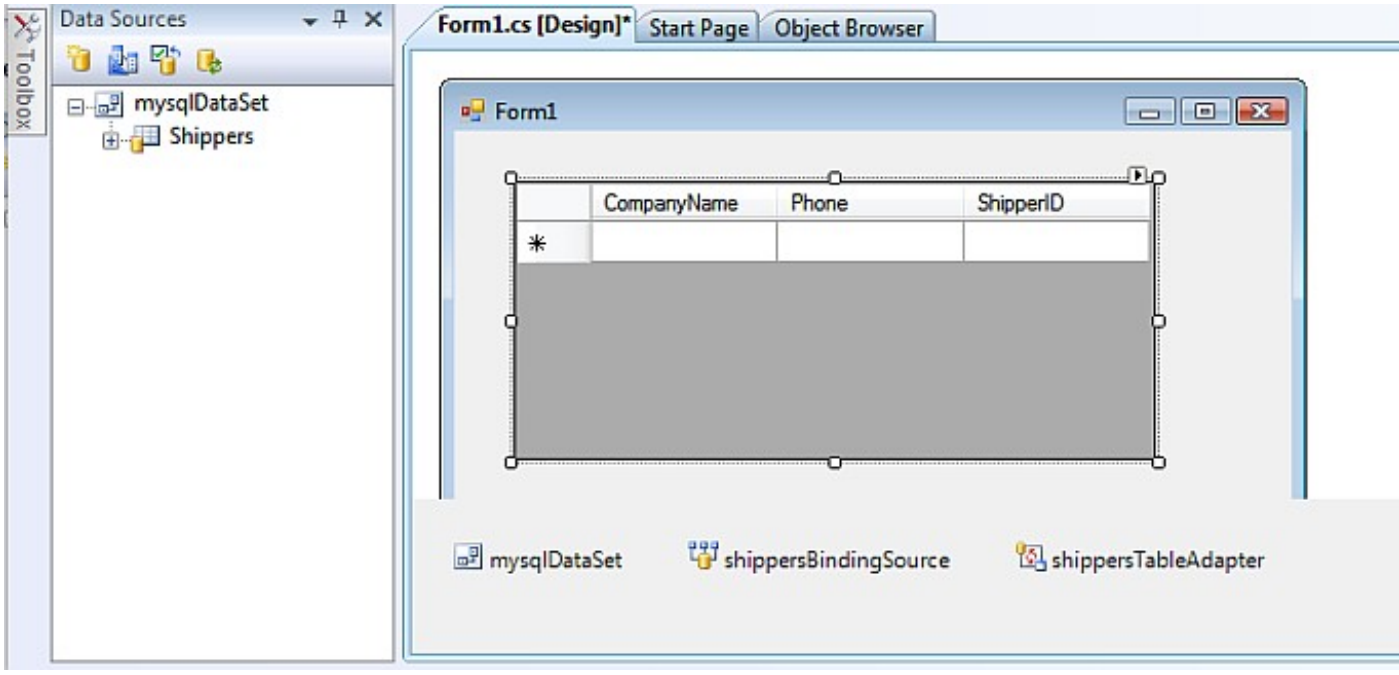
19. The columns names of the select table will be displayed in the DataGridView.

Figure 8.441. DataGridView



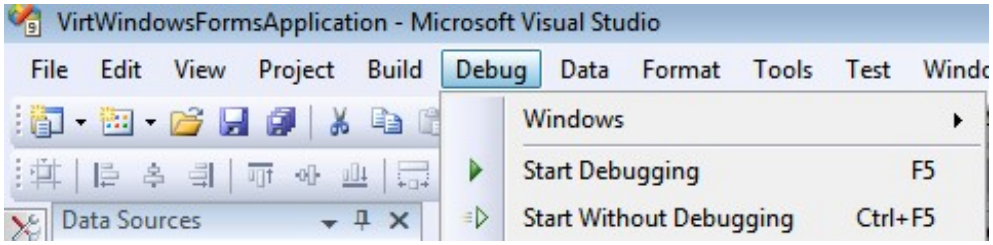
20. Resize the Form and DataGridView to allow all columns to be visible, if possible.

Figure 8.442. Resize the Form and DataGridView



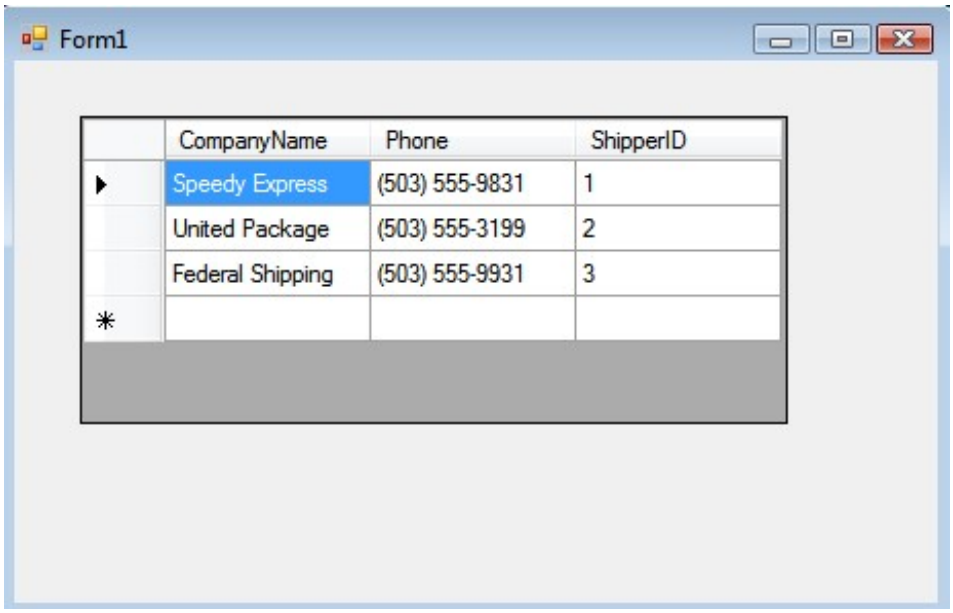
21. To test the application, simply hit *Ctrl+F5* within Visual Studio or select *Start Debugging* from the *Debug* menu.

Figure 8.443. Start Debugging



22. The data from the *Shippers* table will be displayed in the *DataGrid* .

Figure 8.444. DataGrid



The task is now complete.

8.9. Using Microsoft Entity Frameworks to Access PostgreSQL Schema Objects with Virtuoso

Abstract

This document details the steps required to provide Microsoft Entity Framework access to PostgreSQL Schema Objects using the OpenLink Virtuoso Universal Server. This is achieved by linking the required PostgreSQL Schema objects into Virtuoso using its built in Virtual Database engine, and then using the Virtuoso ADO.NET Entity Framework provider to query the remote PostgreSQL Schema objects linked into the Virtuoso Server.

Prerequisites. The following prerequisites must be in place for this solution to be possible.

PostgreSQL DBMS. An PostgreSQL DBMS hosting the required Schema Objects needs to be available. In this section the *PostgreSQL* sample database will be used to demonstrate the process.

An PostgreSQL DBMS hosting the required Schema Objects needs to be available. In this section the *PostgreSQL* sample database will be used to demonstrate the process.

ODBC Driver for PostgreSQL. An PostgreSQL ODBC Driver is required for Linking the PostgreSQL Schema Objects into the Virtuoso Server. The OpenLink *ODBC Driver for PostgreSQL* will be used in this section, for which a functional ODBC Data source name of "ora10ma" will be assumed to exist on the machine hosting the Virtuoso Server.

An PostgreSQL ODBC Driver is required for Linking the PostgreSQL Schema Objects into the Virtuoso Server. The OpenLink *ODBC Driver for PostgreSQL* will be used in this section, for which a functional ODBC Data source name of "ora10ma" will be assumed to exist on the machine hosting the Virtuoso Server.

Virtuoso Universal Server. An Virtuoso installation including the Virtuoso Universal Server and ADO.NET Entity Framework Provider is required. The Virtuoso components used must be Release 5.10.x or above, this being the minimum version containing support for Microsoft Entity Frameworks

An Virtuoso installation including the Virtuoso Universal Server and ADO.NET Entity Framework Provider is required. The Virtuoso components used must be Release 5.10.x or above, this being the minimum version containing support for Microsoft Entity Frameworks

Microsoft Visual Studio 2008 SP1. Microsoft Visual Studio 2008 Service Pack 1 is required, this being the only version containing the necessary Entity Framework support available at the time of writing.

Microsoft Visual Studio 2008 Service Pack 1 is required, this being the only version containing the necessary Entity Framework support available at the time of writing.

Tasks

Ensure PostgreSQL Primary Keys (PKs) are not nullable. The Visual Studio 2008 Entity Data Model (EDM) requires that all primary keys are *NOT* Nullable, and will fail to generate an EDM if any are. Thus ensure any tables to be used are defined as not nullable in the PostgreSQL database schema before attempting to generate an EDM. In the case of the PostgreSQL database all tables are not nullable, thus this should not be an issue in this case.

The Visual Studio 2008 Entity Data Model (EDM) requires that all primary keys are *NOT* Nullable, and will fail to generate an EDM if any are. Thus ensure any tables to be used are defined as not nullable in the PostgreSQL database schema before attempting to generate an EDM. In the case of the PostgreSQL database all tables are not nullable, thus this should not be an issue in this case.

8.9.1. Install and configure OpenLink ODBC Driver for PostgreSQL

The Virtuoso Virtual Database engine uses ODBC as the connectivity mechanism for linking remote database objects into its local schema. Thus an PostgreSQL ODBC Driver must be available with a suitably configured DSN for connecting to the target database. The OpenLink ODBC Drivers for PostgreSQL have been used in this section, although in theory any PostgreSQL ODBC Driver can be used.

Installation instructions for the OpenLink ODBC Driver for PostgreSQL are available from:

- Product Installation & Basic Configuration (ODBC)

8.9.2. Install and configure OpenLink Virtuoso Universal Server

- Install and configure OpenLink Virtuoso Universal Server

8.9.3. Linking PostgreSQL tables into OpenLink Virtuoso

1. Start the Virtuoso Web User Interface

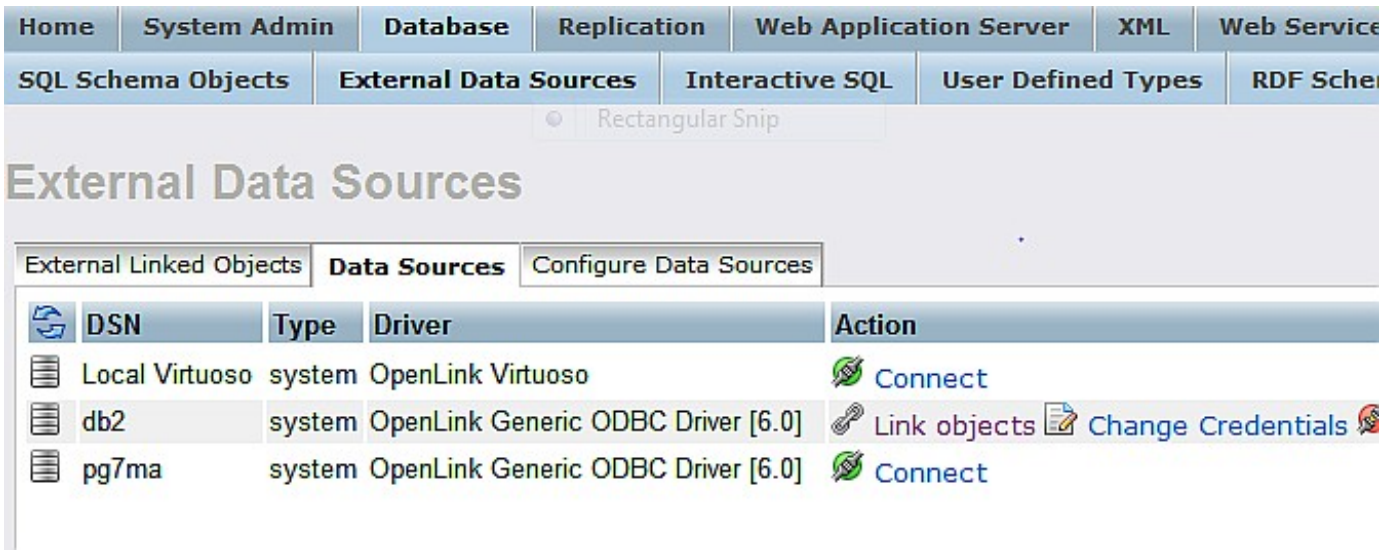
Figure 8.445. Start

2. Select the "Conductor" link in the Left Frame and Login as the "dba" user.

Figure 8.446. Conductor

3. Select the "Databases" -> "External Data Source" -> "Data Sources" tab menu items

Figure 8.447. Databases

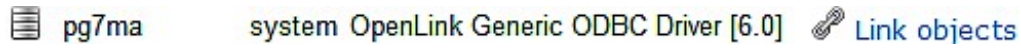


4. Select the "Connect" button for the "pg7ma" PostgreSQL DSN.

Figure 8.448. Connect

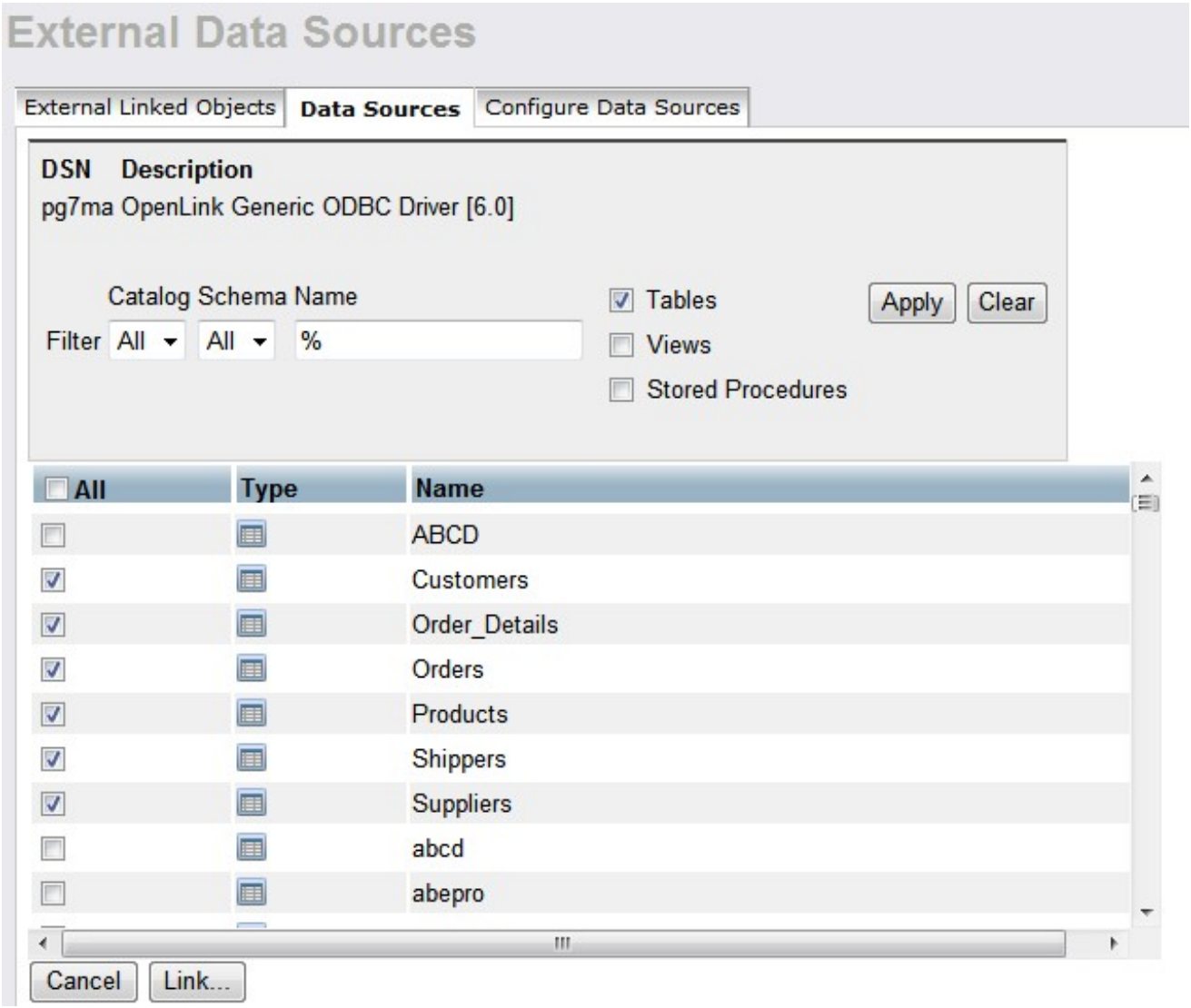
5. On successful connection Select the "Link Objects" button to obtain a list of available tables

Figure 8.449. Link Objects



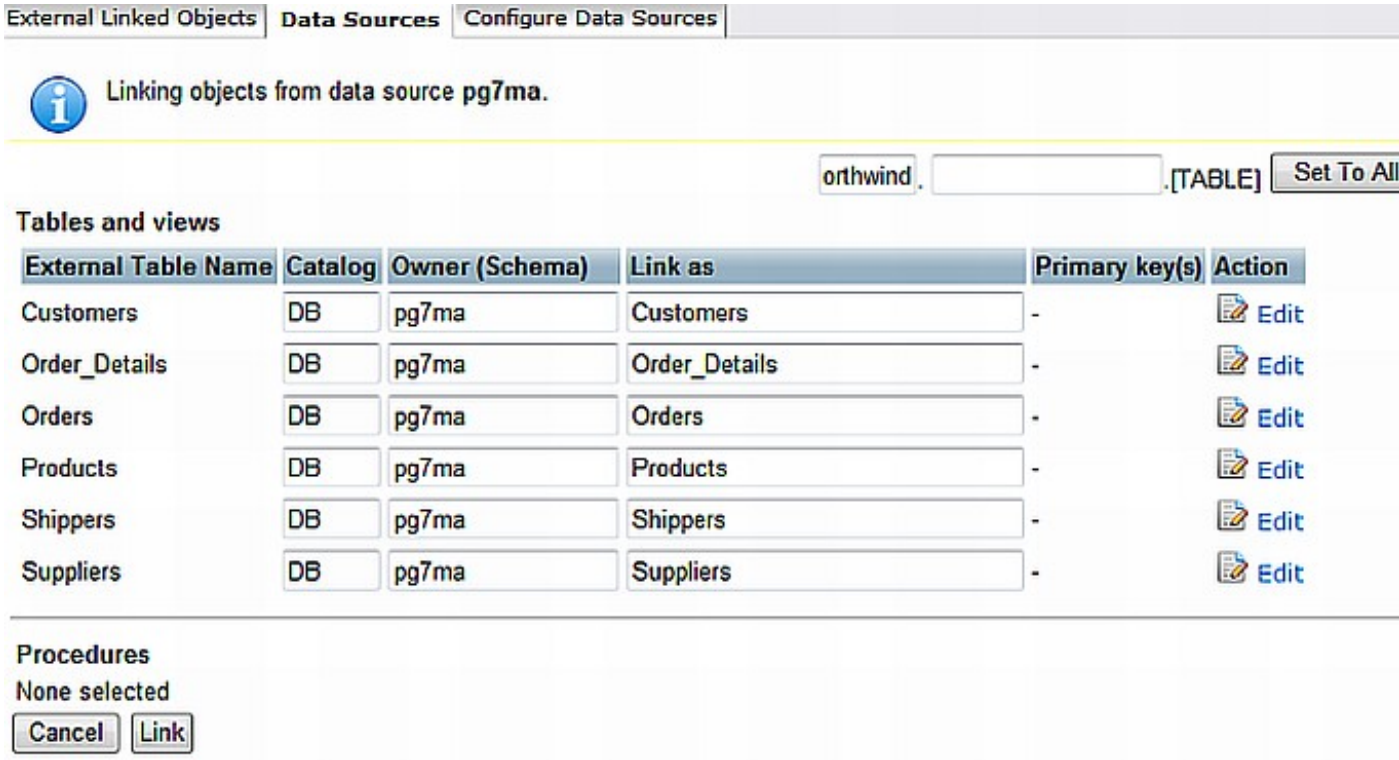
6. Select all the tables that you would like to link.

Figure 8.450. Select all tables



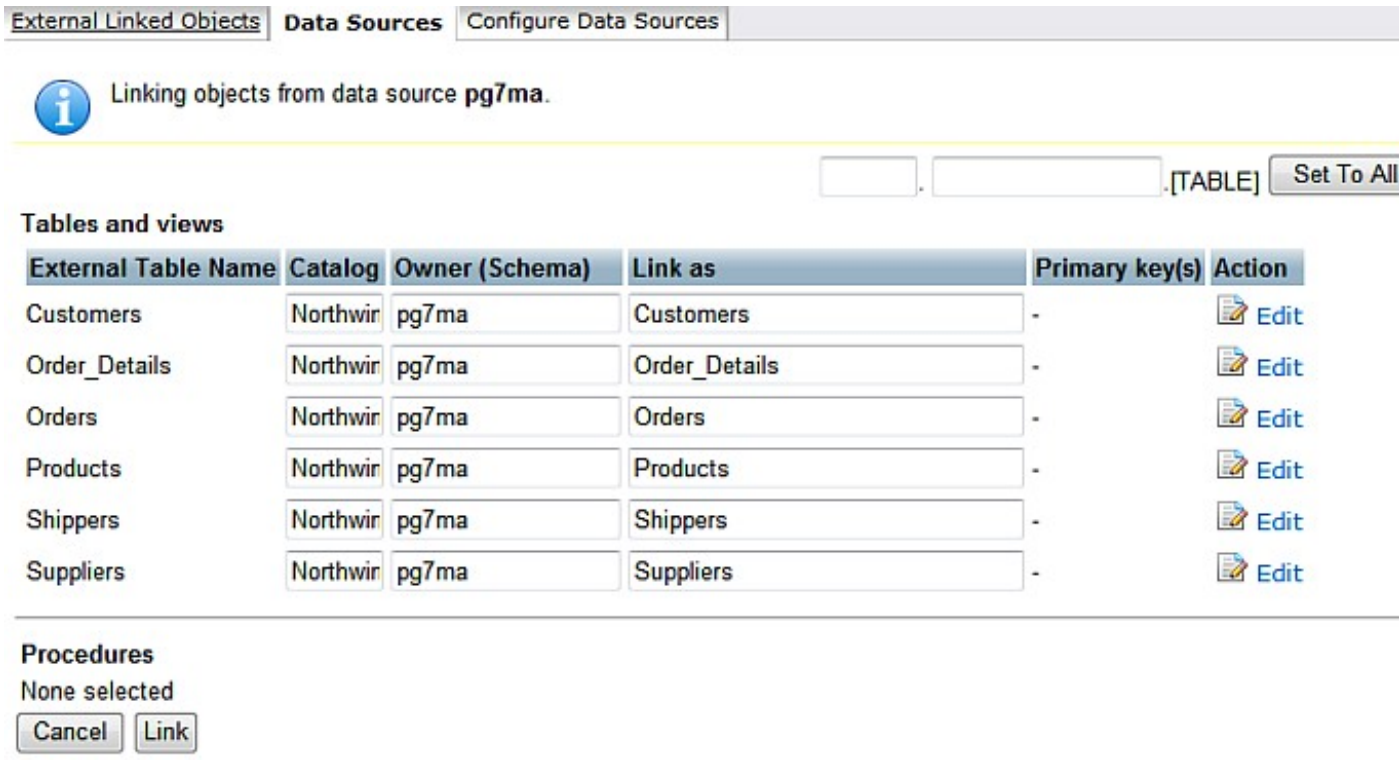
7. Change the Catalog for each table to be "Northwind" using the "Set All" button.

Figure 8.451. Catalog



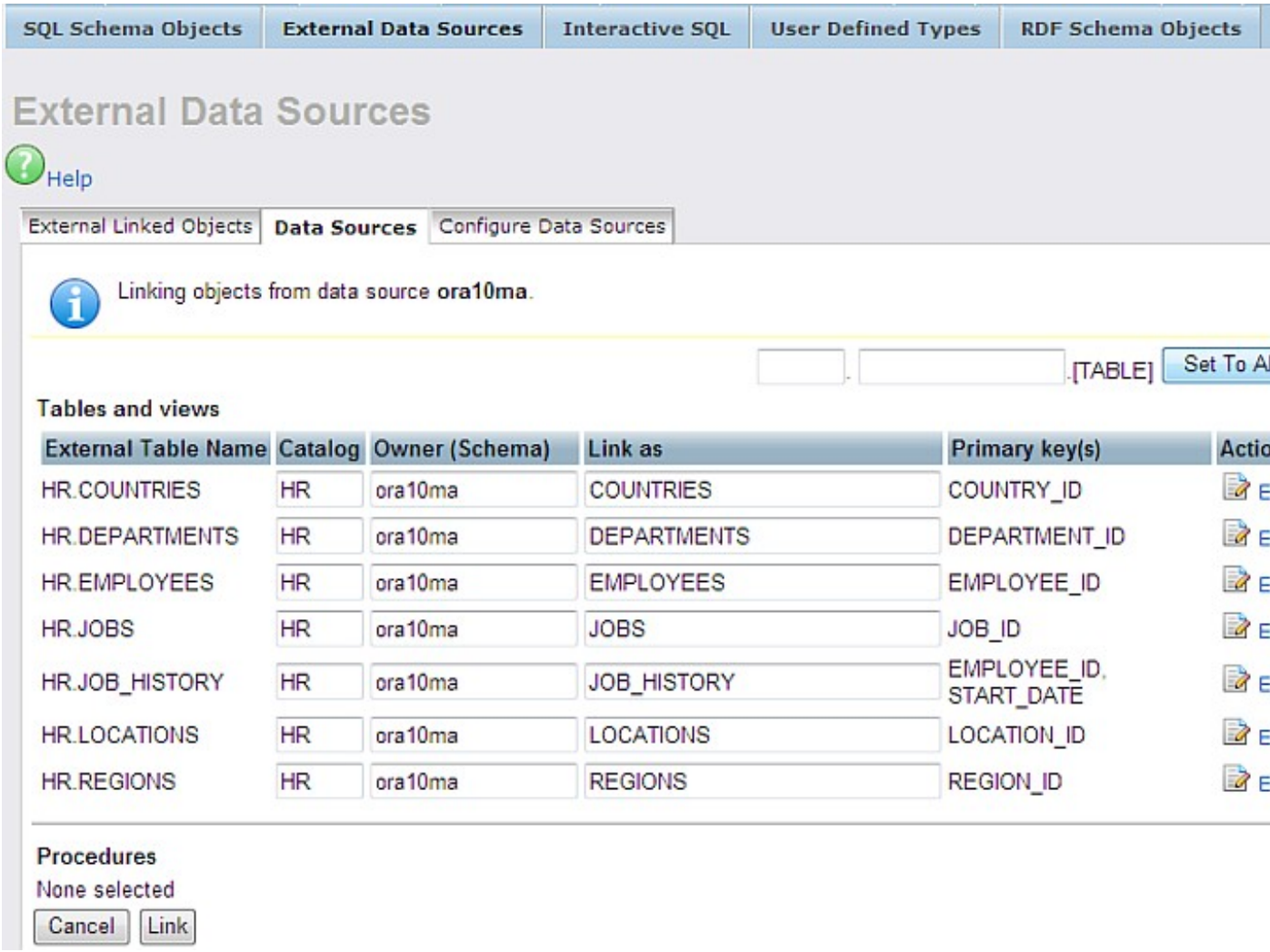
8. All the catalog names are changed to be "Northwind".

Figure 8.452. Catalog



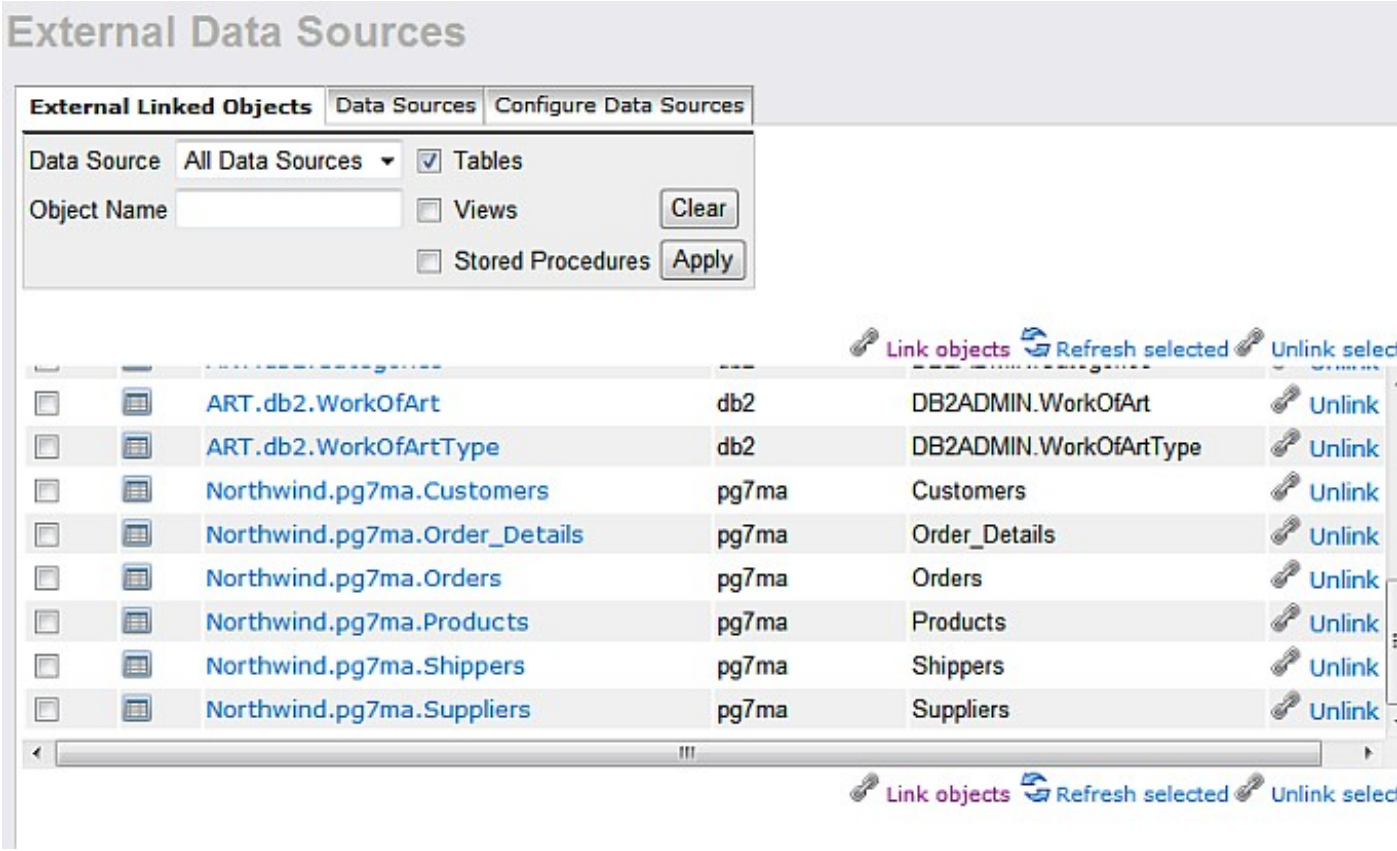
9. Select the "Link" button to link the selected tables into Virtuoso

Figure 8.453. "Link" button



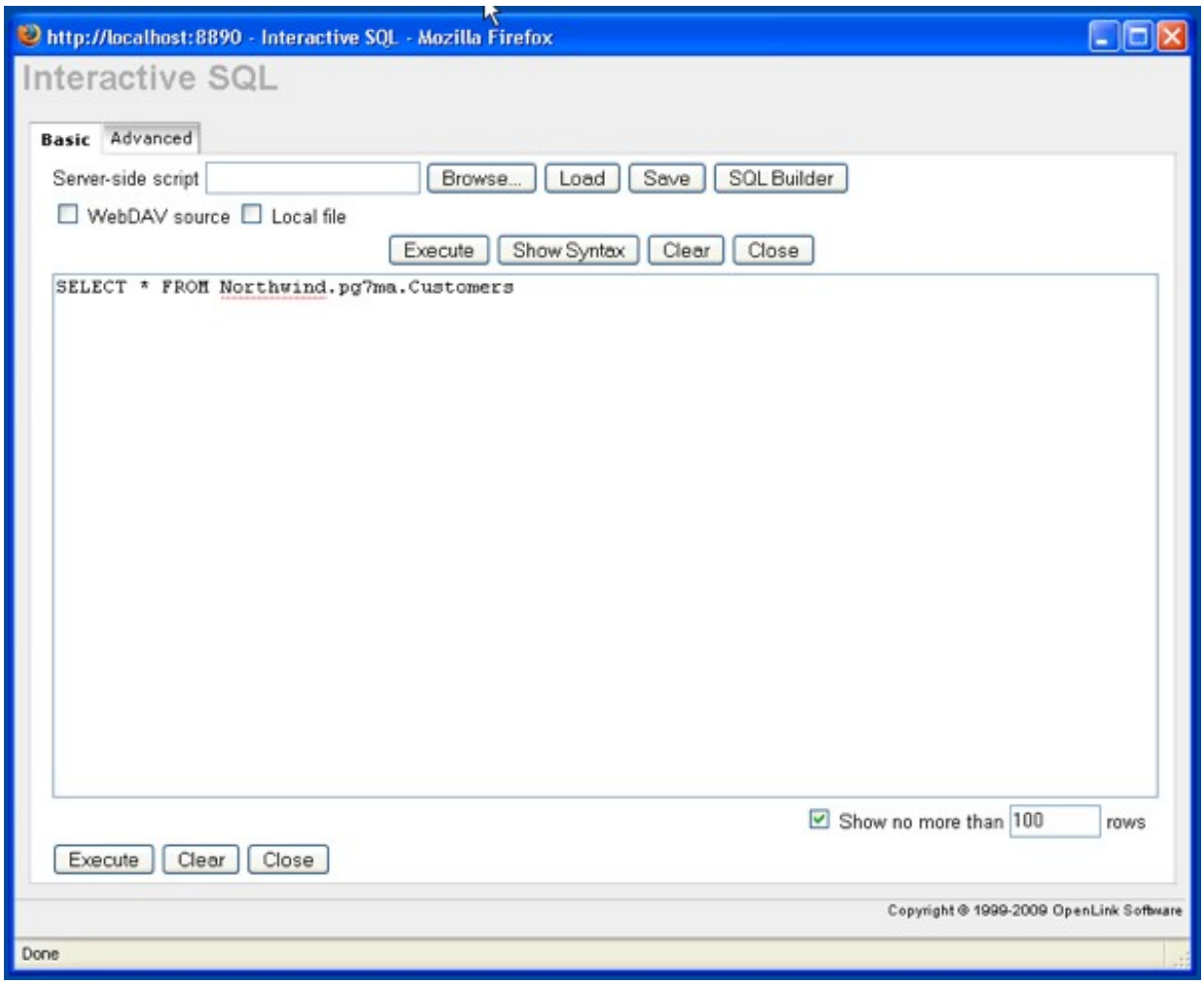
10. On completion of the Link process the tables will be displayed in the "External Linked Objects" tab.

Figure 8.454. Completion



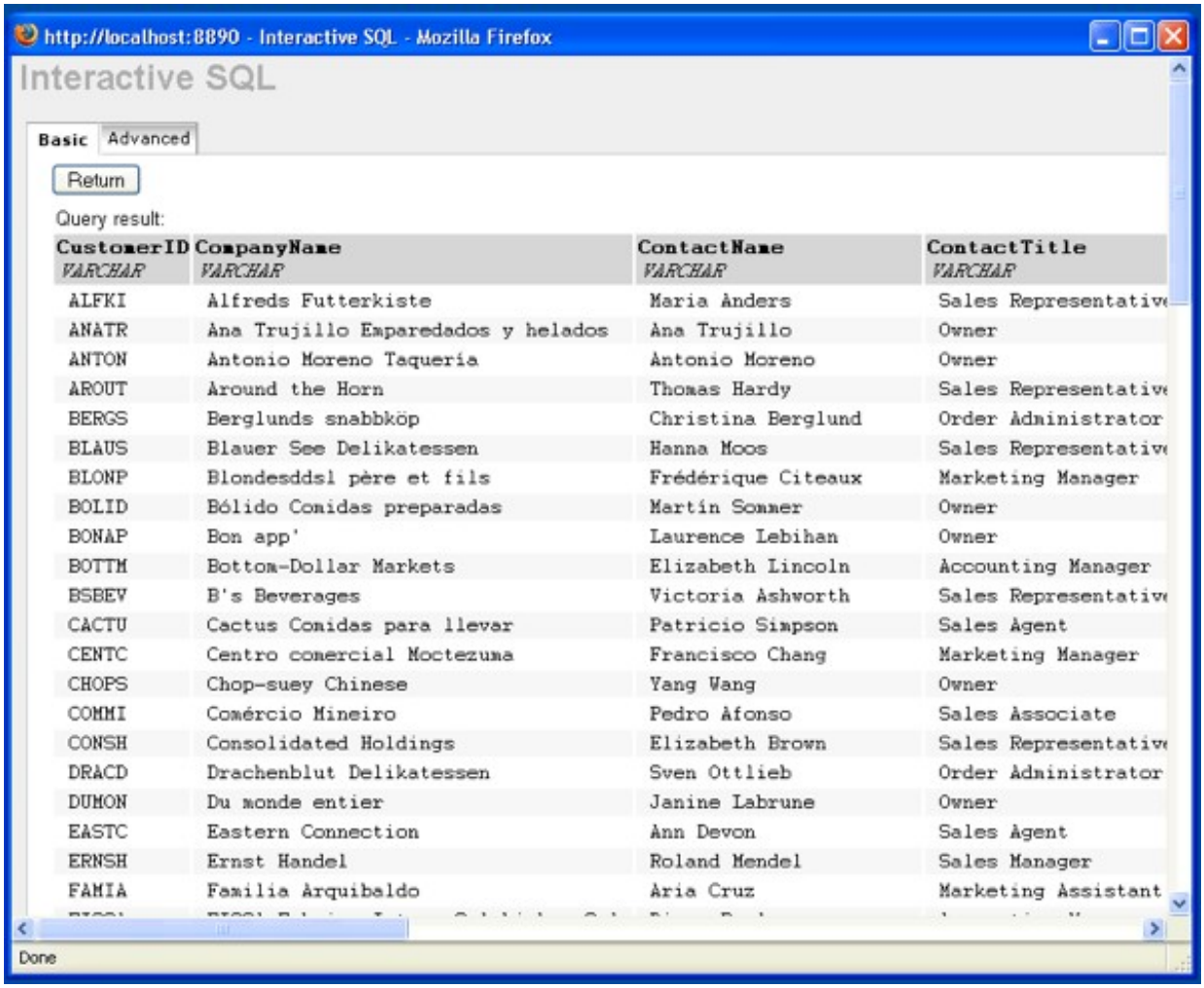
11. The linked tables can be queried by clicking on the hyperlink in the "Local Name" column of the "External Linked Objects" tab above, which loads the Virtuoso "Interactive SQL" interface with the required SQL "Select" for retrieving the remote table data . We shall use the "Northwind.pg7ma.Customers" table to demonstrate this.

Figure 8.455. Querying



12. Then click the "Execute" button to run the query and retrieve the results from the remote table.

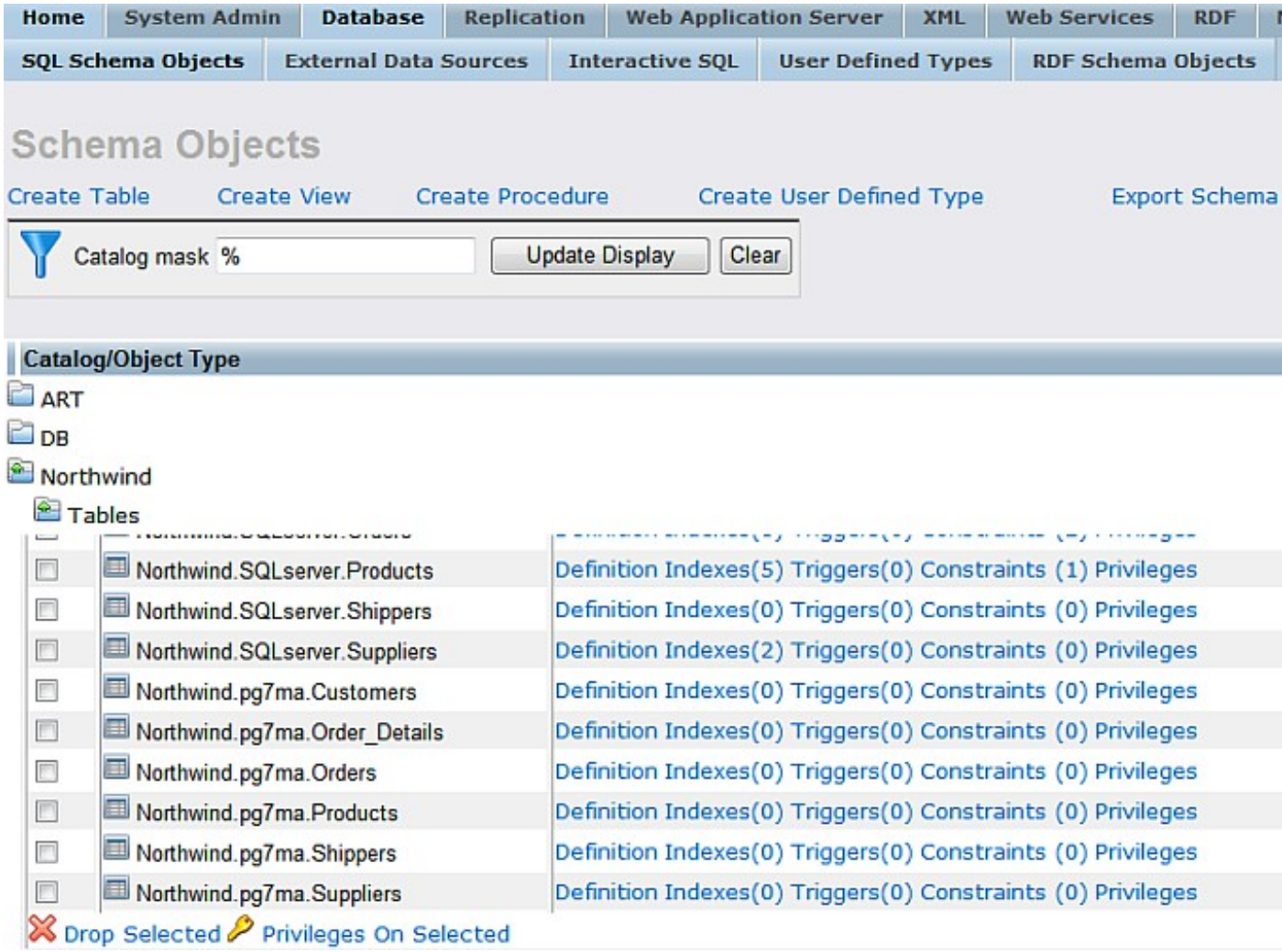
Figure 8.456. Execute



CustomerID VARCHAR	CompanyName VARCHAR	ContactName VARCHAR	ContactTitle VARCHAR
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner
AROUT	Around the Horn	Thomas Hardy	Sales Representative
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative
BLONP	Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager
BOLID	Bólido Comidas preparadas	Martín Sommer	Owner
BONAP	Bon app'	Laurence Lebihan	Owner
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager
BSBEV	B's Beverages	Victoria Ashworth	Sales Representative
CACTU	Cactus Comidas para llevar	Patricio Simpson	Sales Agent
CENTC	Centro comercial Moctezuma	Francisco Chang	Marketing Manager
CHOPS	Chop-suey Chinese	Yang Wang	Owner
COMMI	Comércio Mineiro	Pedro Afonso	Sales Associate
CONSH	Consolidated Holdings	Elizabeth Brown	Sales Representative
DRACD	Drachenblut Delikatessen	Sven Ottlieb	Order Administrator
DUMON	Du monde entier	Janine Labrune	Owner
EASTC	Eastern Connection	Ann Devon	Sales Agent
ERNSH	Ernst Handel	Roland Mendel	Sales Manager
FAMIA	Familia Arquibaldo	Aria Cruz	Marketing Assistant

13. The tables can also be viewed as part of the Virtuoso "SQL Schema Objects" under the "Northwind" catalog name.

Figure 8.457. SQL Schema Objects



The Link process is now complete enabling the tables to be queried as if part of the Virtuoso Schema.

8.9.4. Creating EDM in Visual Studio 2008

The following steps can be used to create an Entity Data Model (EDM) for the PostgreSQL Northwind database:

1. Launch the Visual Studio 2008 SP1 IDE.

Figure 8.458. Visual Studio 2008 SP1 IDE

2. Create a

Web Application

project by going to the

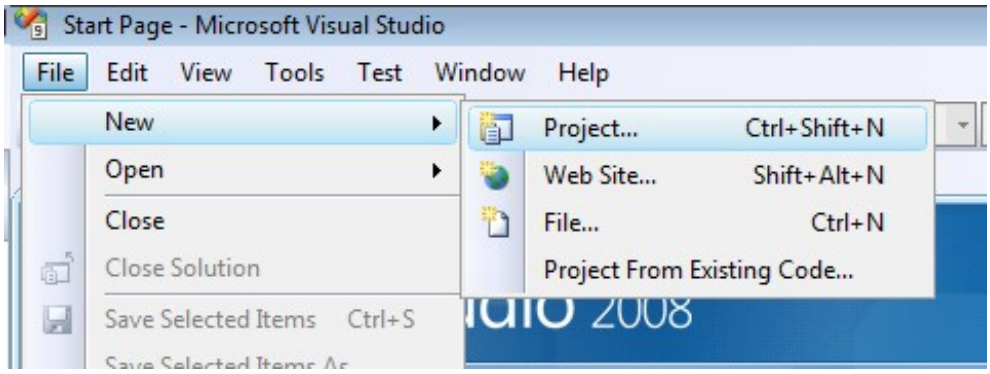
File

menu in Visual Studio and choosing

New Project

.

Figure 8.459. Web Application



3. When the New Project window appears, choose either

   Visual Basic

   or

   Visual C#

   as your programming language.
4. Within the language category, click on

   Web

   , and select

   ASP.NET Web Application

   from the right-hand panel.
5. Choose a name for the project, for example

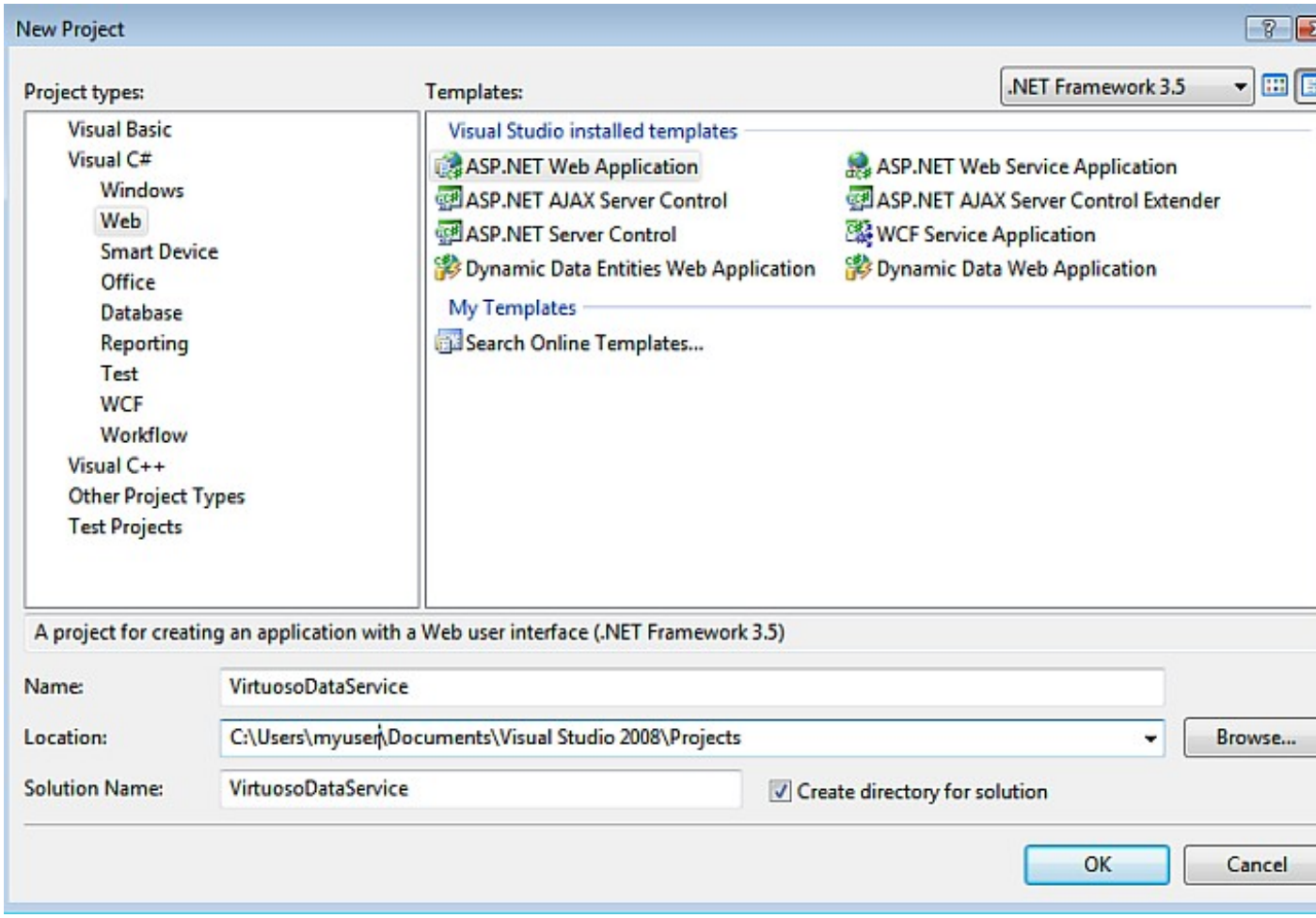
   VirtuosoDataService

   , and click

   OK

   .

Figure 8.460. name for the project

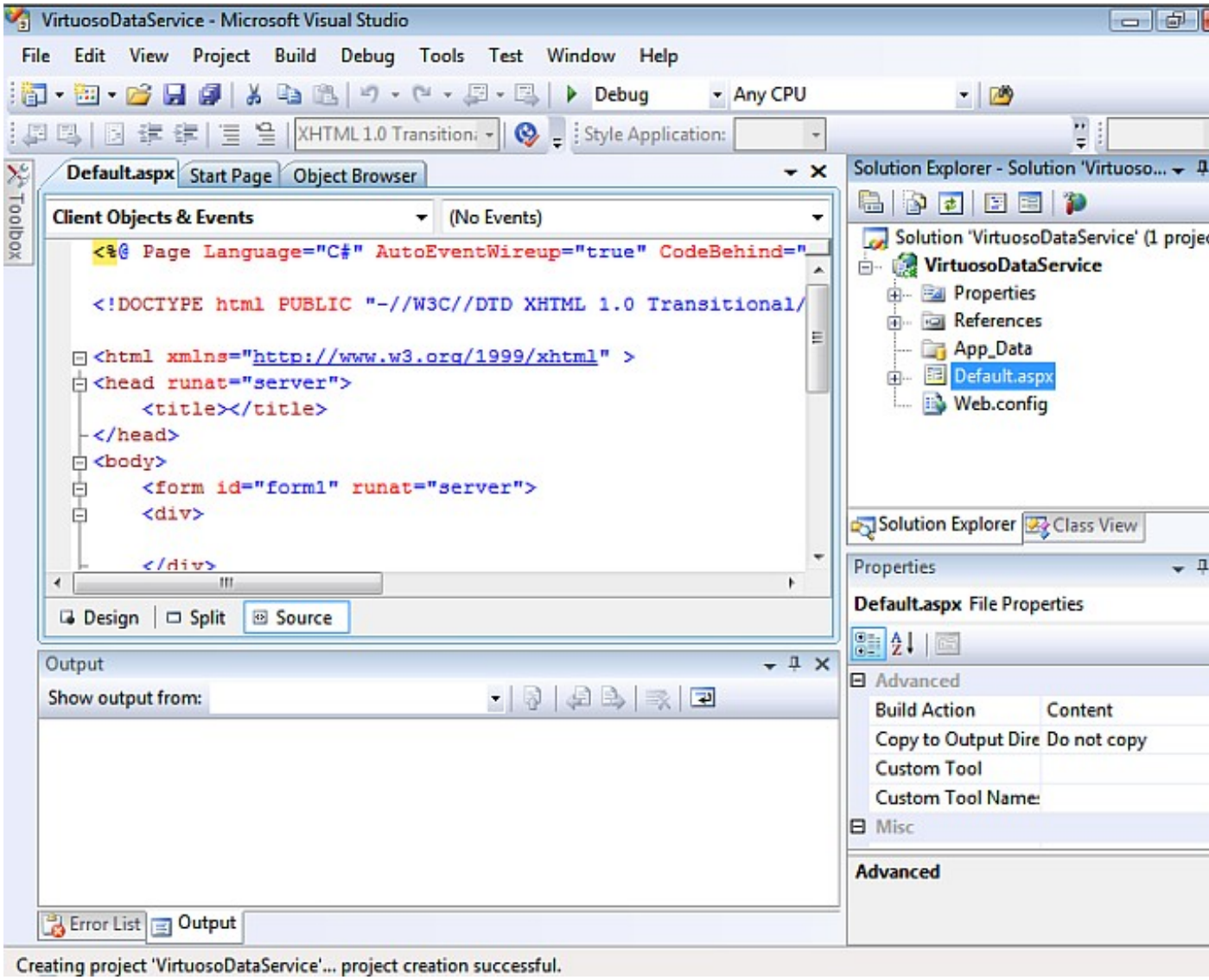


6. This will create a new project called

   VirtuosoDataService

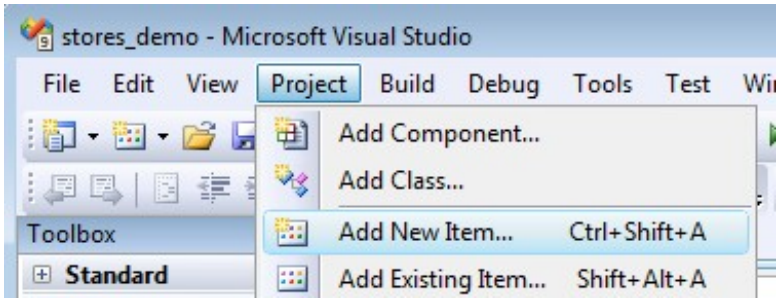
   .

Figure 8.461. create a new project



7. Select the Project -> Add New Item menu option.

Figure 8.462. VirtuosoDataService



8. The

Add New Item

dialog will appear. Choose the

ADO.NET Entity Data Model

template. Give it the name

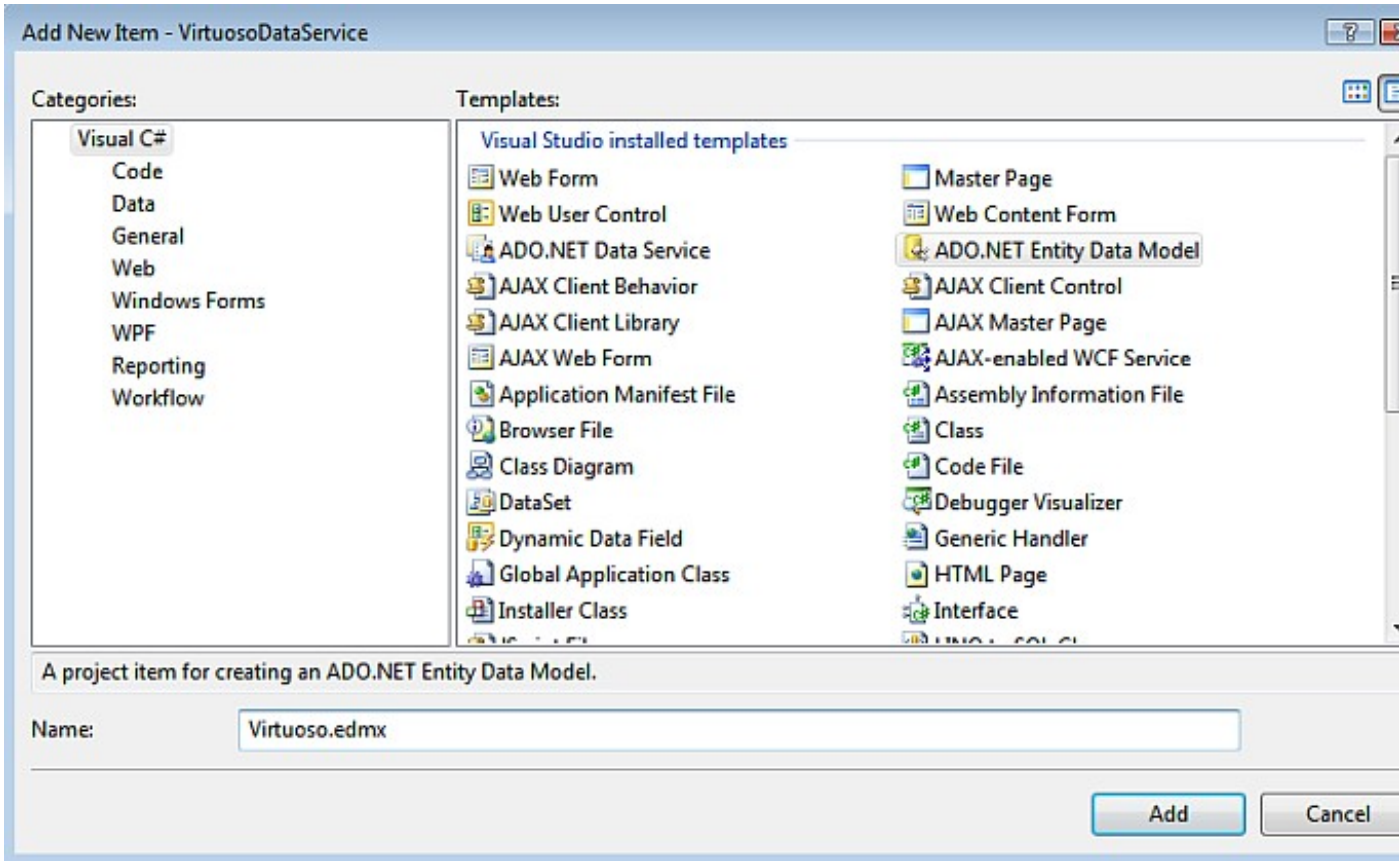
Virtuoso.edmx

and click

Add

to start the creation of the ADO.Net Entity Data Model.

Figure 8.463. Add New Item



9. In the

Entity Data Model Wizard

dialog

Choose Model Contents

page select the

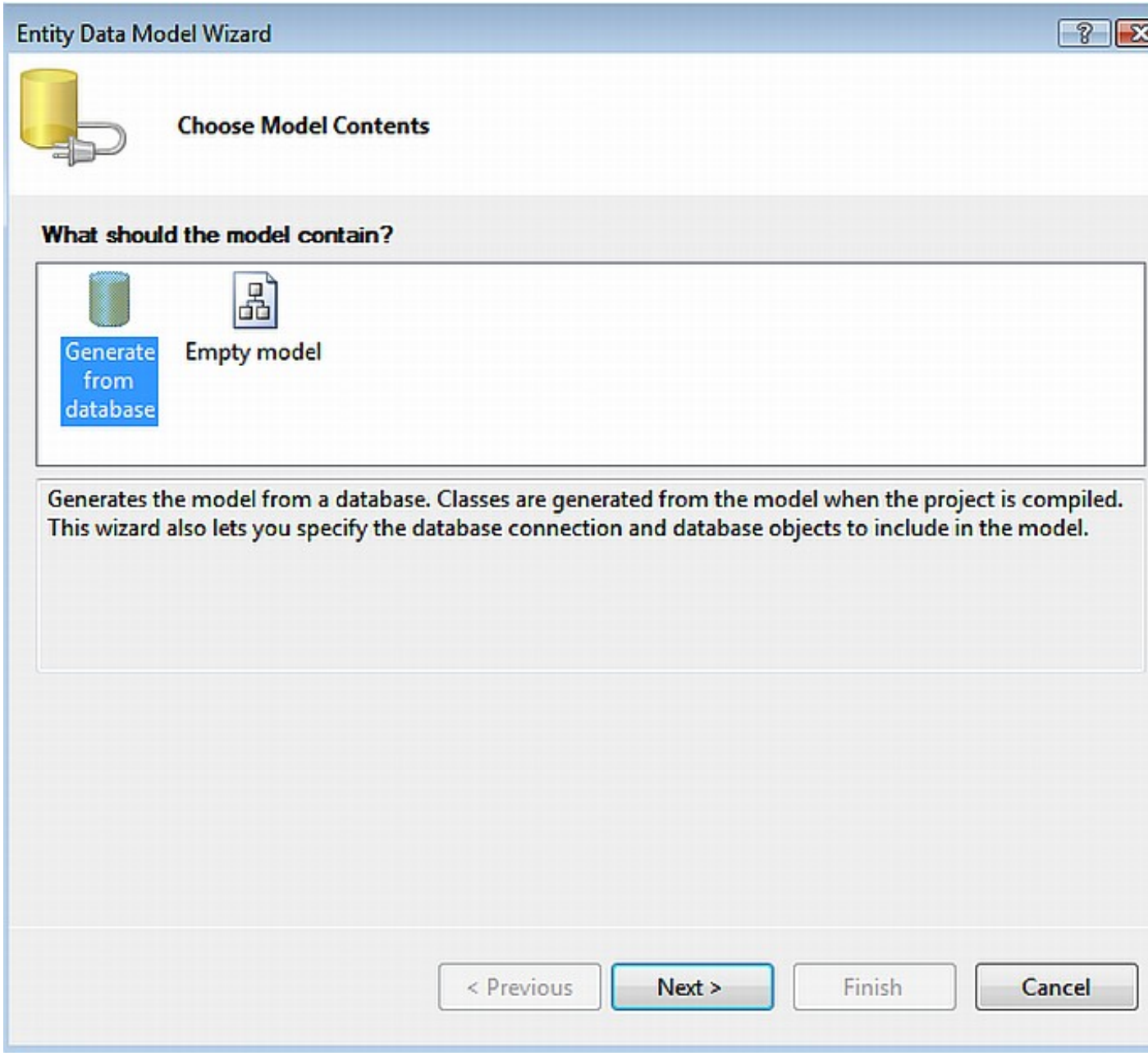
Generate from Database

model type and click

Next

.

Figure 8.464. Choose Model Contents



10. In the

    Entity Data Model Wizard

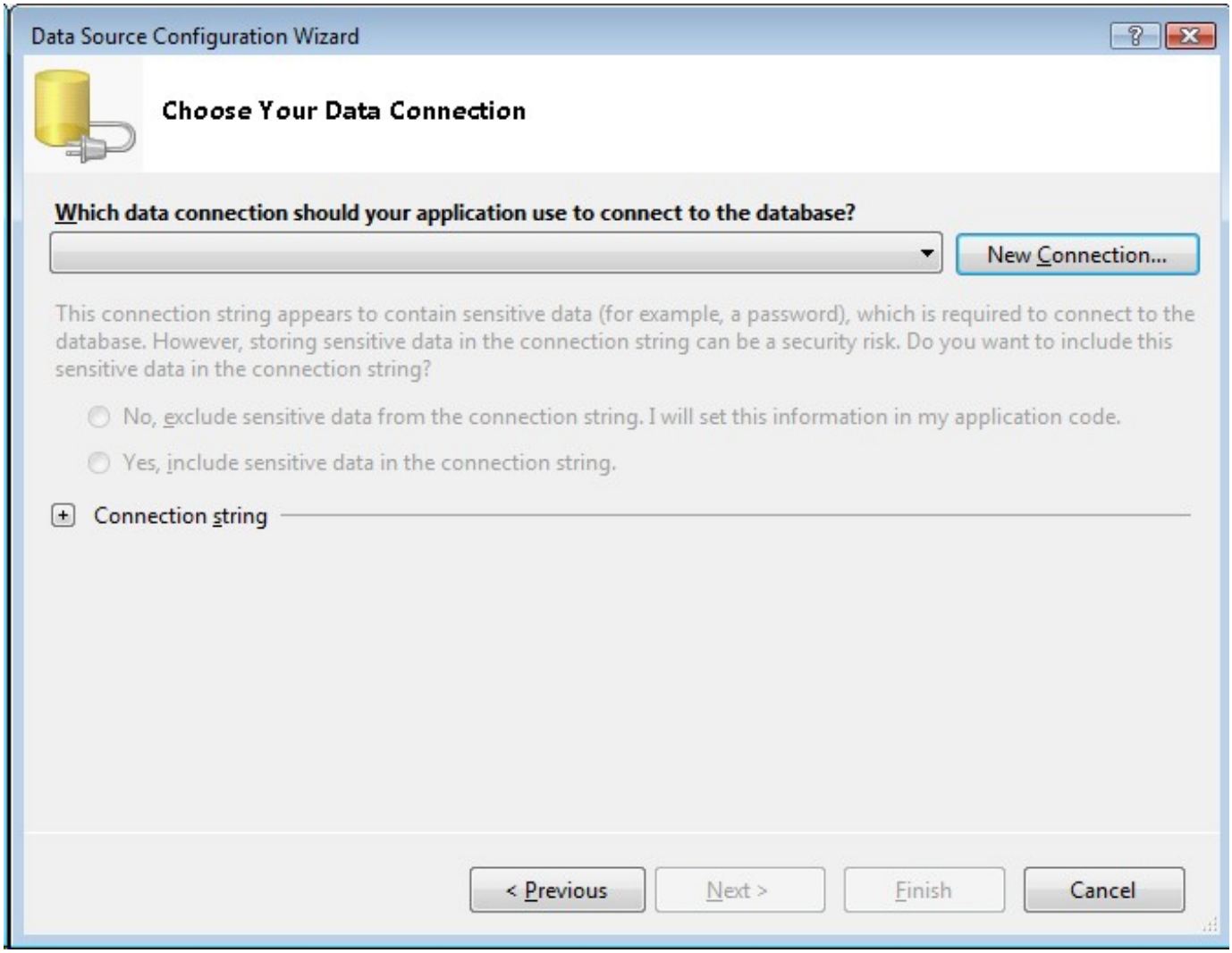
    dialog

    Choose your Data Connection

    page, select the

    New Connection button

Figure 8.465. Entity Data Model Wizard



11. In the

    Choose Data Source

    dialog, select the OpenLink

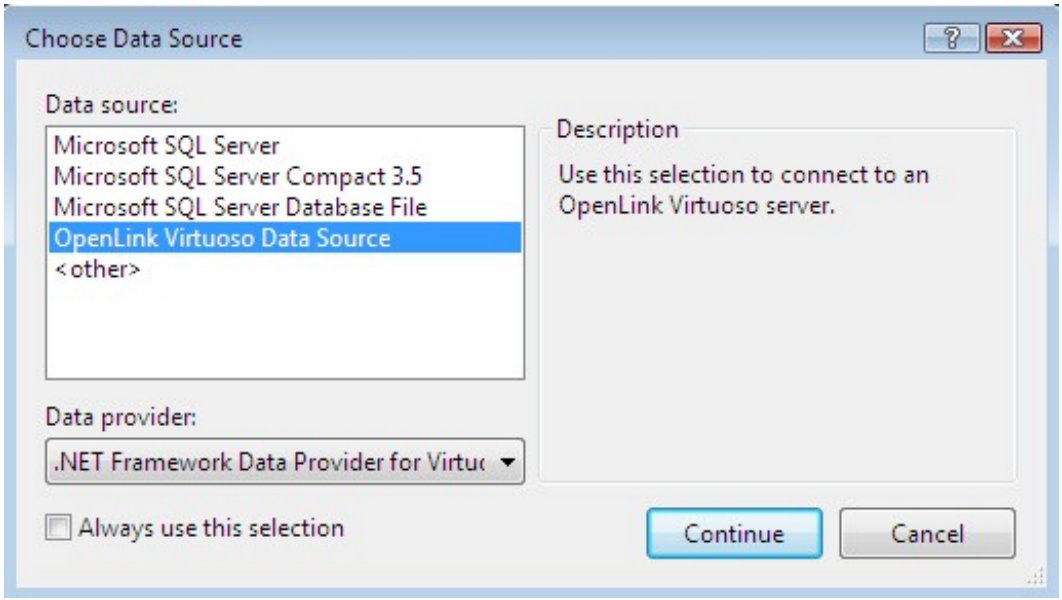
    Virtuoso Data Source

    from the list and click

    Continue

    .

Figure 8.466. Choose Data Source



12. In the

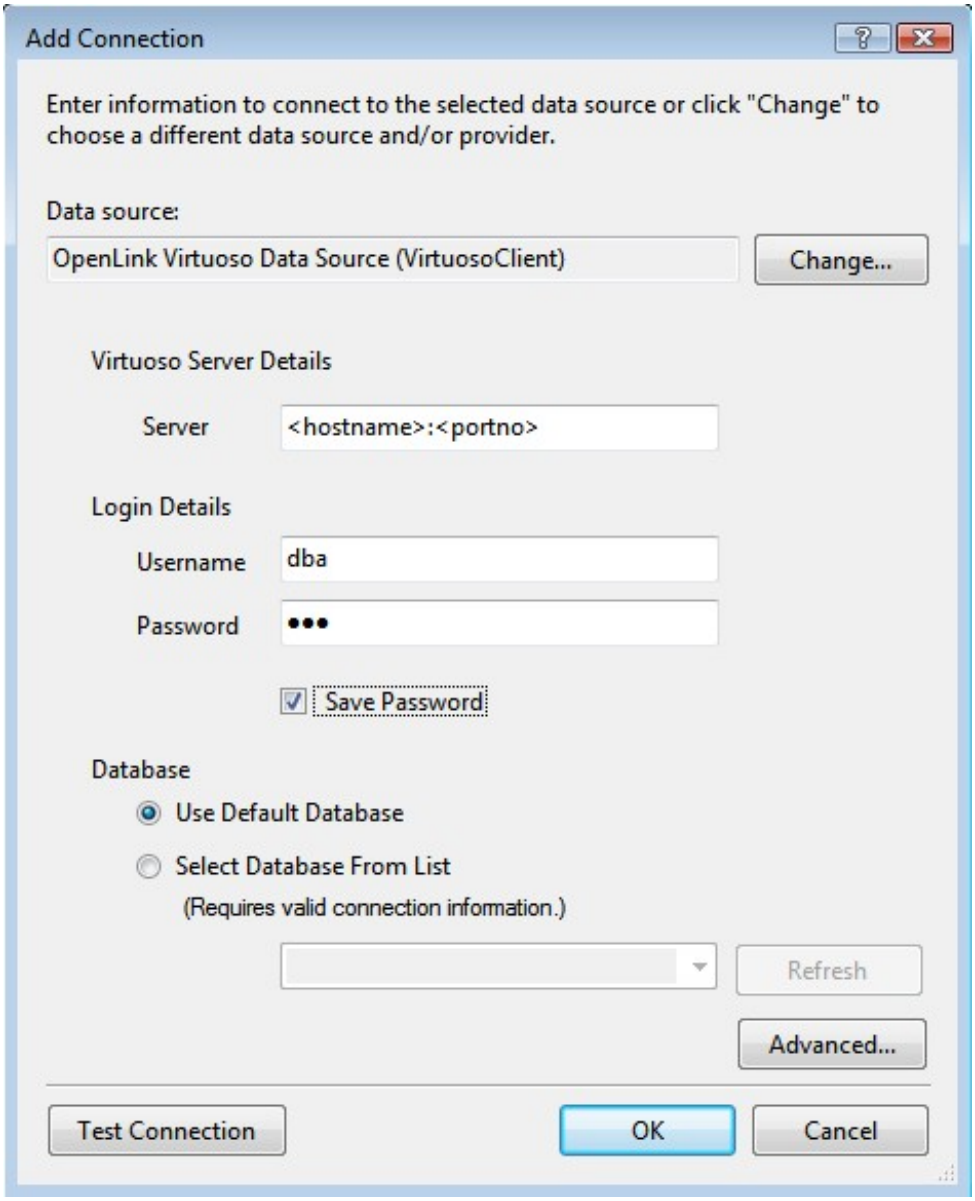
    Add Connection

    dialog, specify the

    hostname, portno, username and password

    for the target Virtuoso Server and check the Save Password check box.

    Figure 8.467. Connection Properties



13. Select the

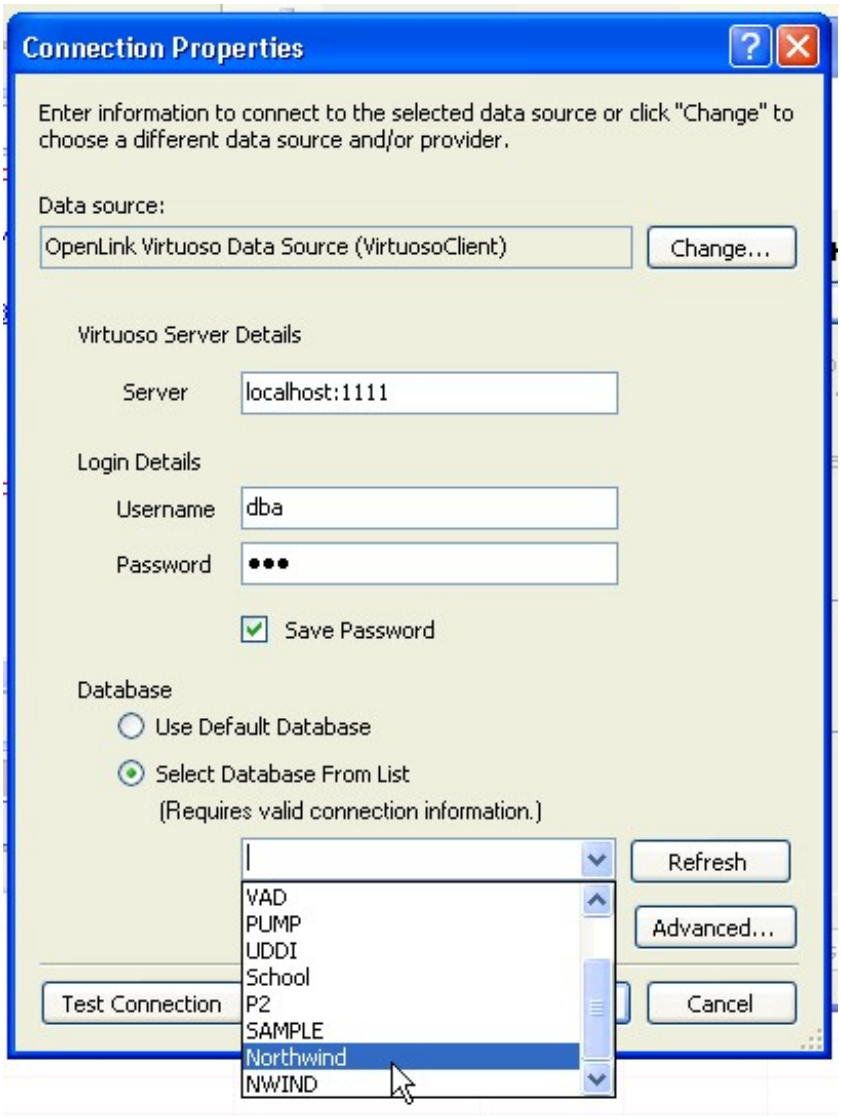
    Select Database From List

    radio button and choose the

    Northwind

    database from the drop down list.

Figure 8.468. Add connection

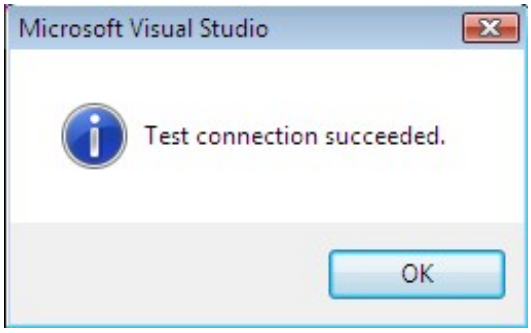


14. Press the

    Test Connection

    dialog to verify that the database is accessible.

    Figure 8.469. Test Connection



15. Set the

    entity connect string

    name to

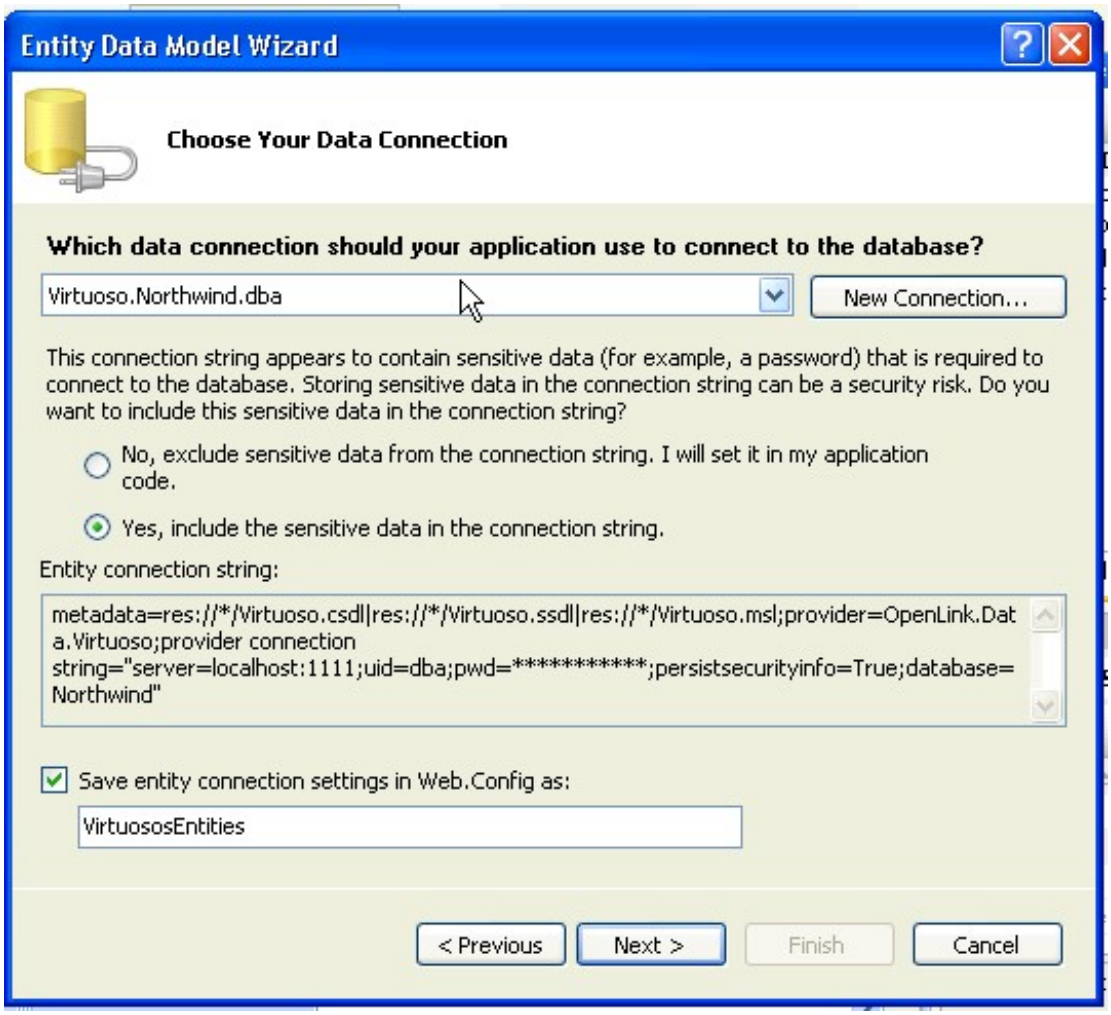
VirtuosoEntities

(note this name as it is required in the section on creating and ADO.Net Data Service) and click

Next

.

Figure 8.470. entity connect string



16. In the

    Choose your Database Objects

    page select the

    Tables

    check box to select all tables in the Northwind catalog for addition to the Entity Data Model. Set the

    Model Namespace

    to

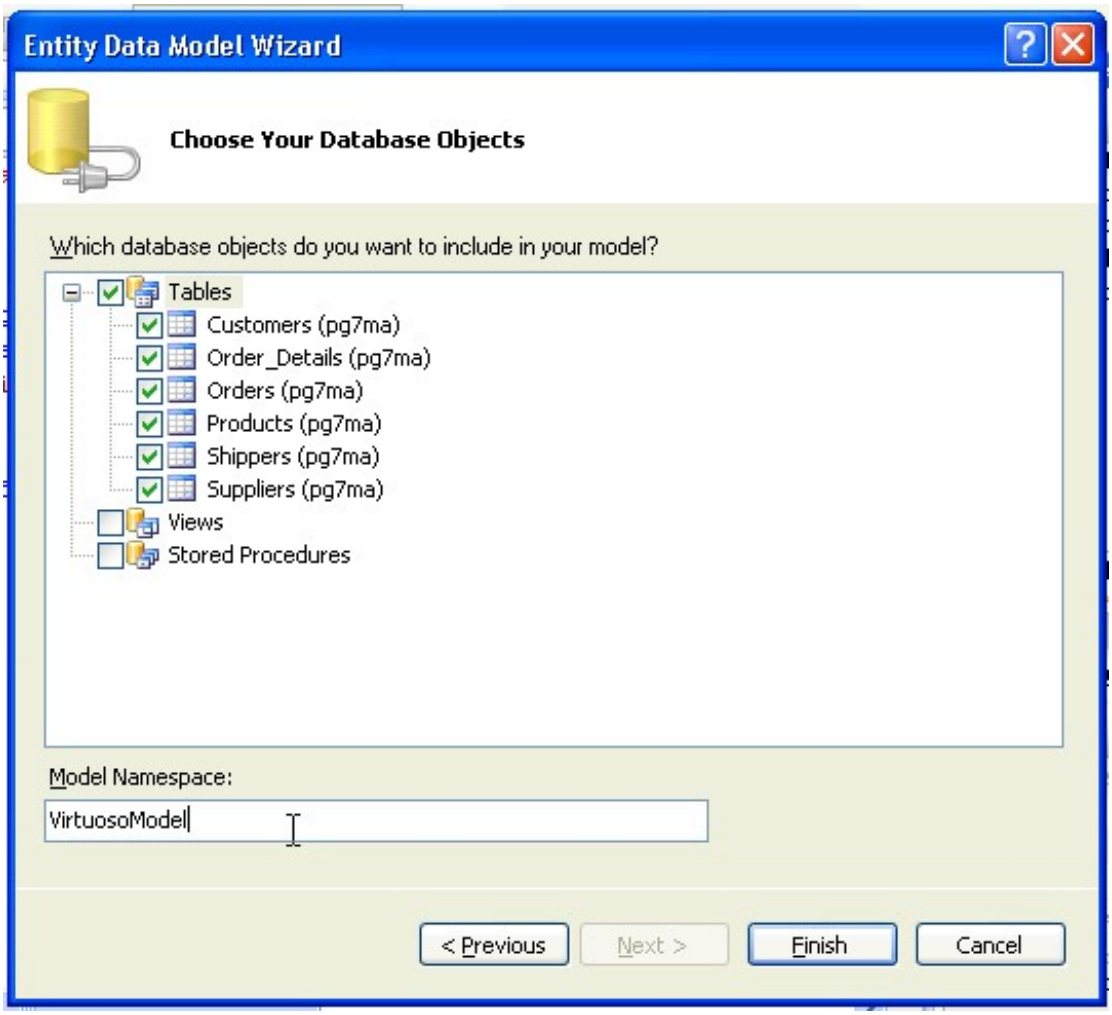
    VirtuosoModel

    and click

    Finish

.

Figure 8.471. Database Objects

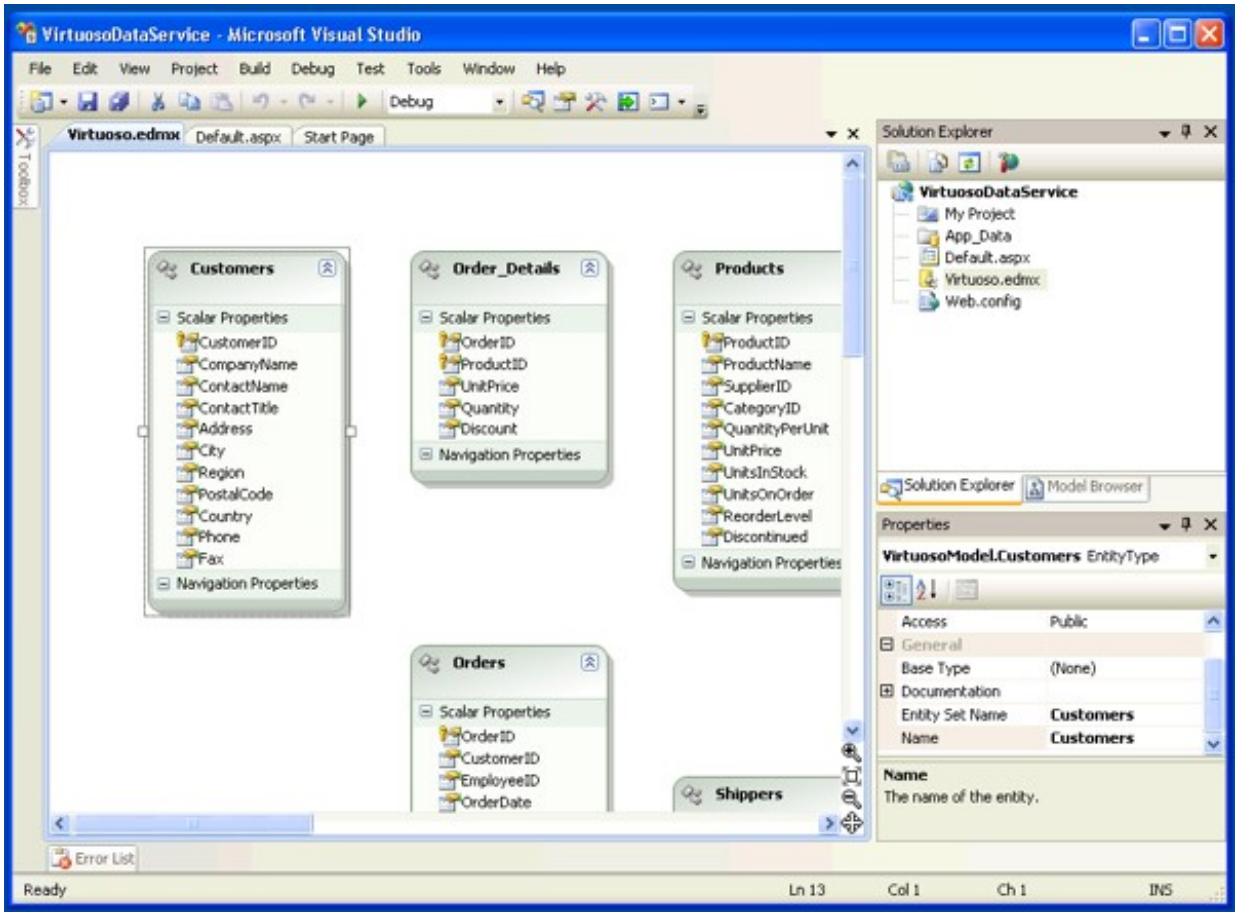


17. The

    Virtuoso.edmx

    EDM will be created with the tables and relationships displayed in the Visual Studio IDE

Figure 8.472. Virtuoso.edmx



Creation for the Entity Data Model for the PostgreSQL Northwind database is now complete.

8.9.5. Using EDM to create Entity Framework based applications

Now that a Microsoft Entity Data Model has been created for the PostgreSQL database, Entity Framework applications can be created to make use of it.

Visual Studio Windows DataGrid Form Application

This section details the steps required to create a simple Visual Studio 2008 Windows Form application, with associated DataGridView control for displaying data in selected tables from the target database.

1. Launch the Visual Studio 2008 SP1 IDE.

Figure 8.473. Visual Studio 2008 SP1 IDE

2. Create a

   Web Application

   project by going to the

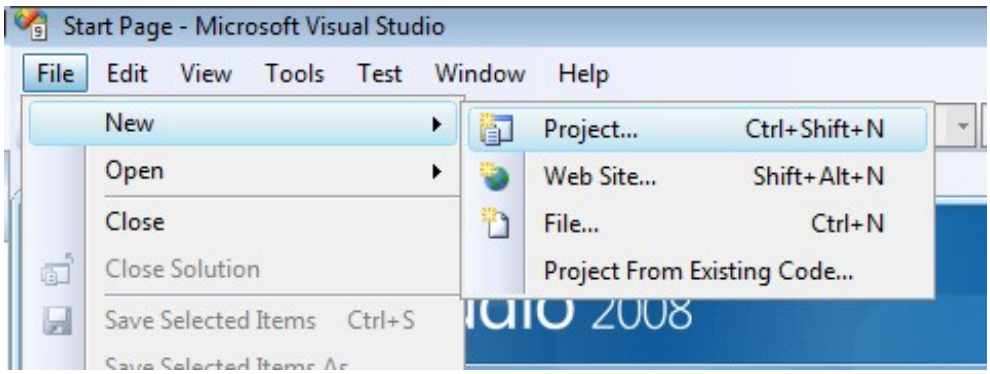
   File

   menu in Visual Studio and choosing

   New Project

   .

   Figure 8.474. Web Application



3. When the New Project window appears, choose either

   Visual Basic

   or

   Visual C#

   as your programming language.
4. Within the language category, click on

   Windows

   and select

   Windows Form Application

from the right-hand panel.

5. Choose a name for the project, for example

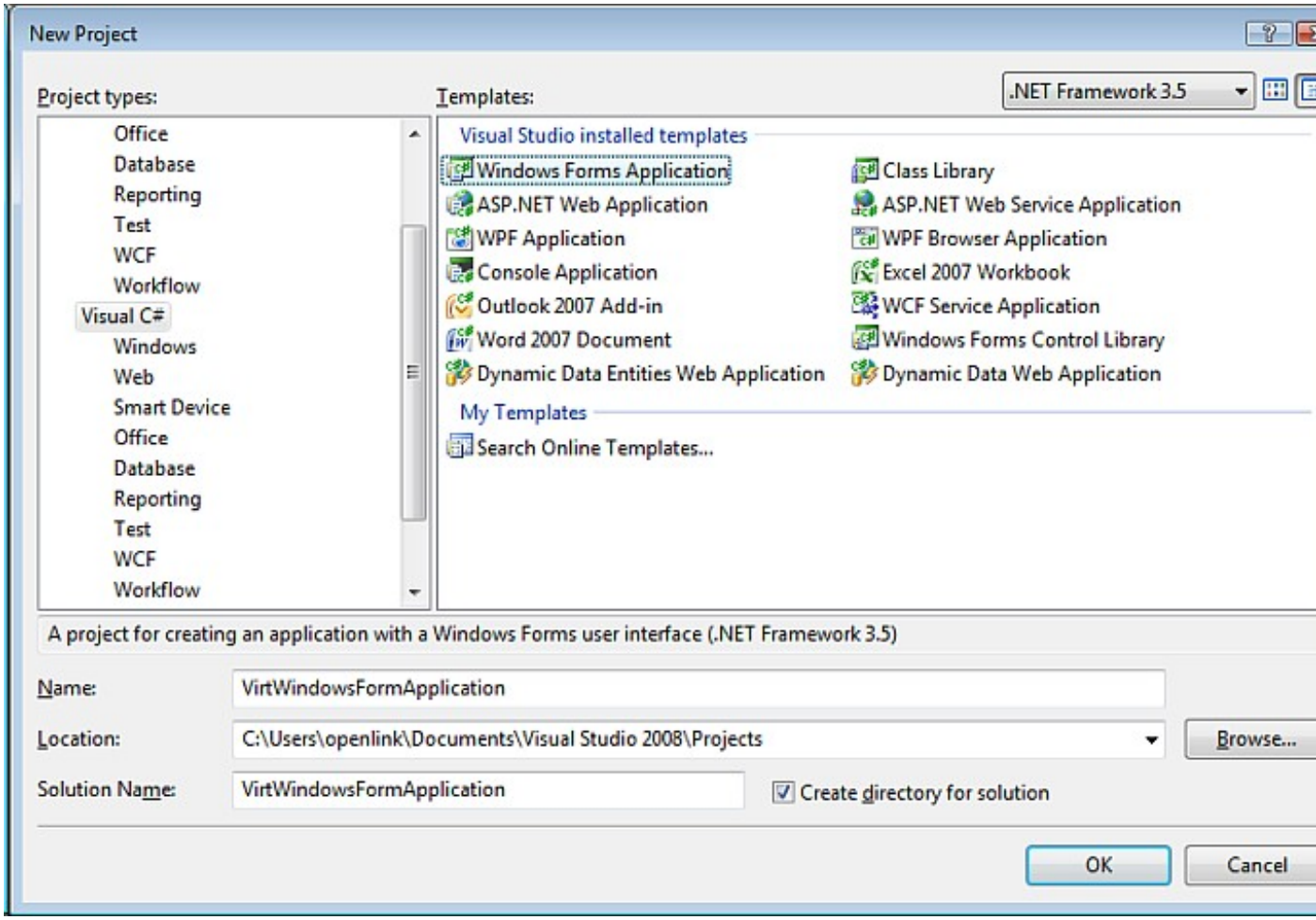
   VirtWindowsFormApplication

   , and click

   OK

   .

   Figure 8.475. Web Application



6. In the

   Toolbox

   , expand

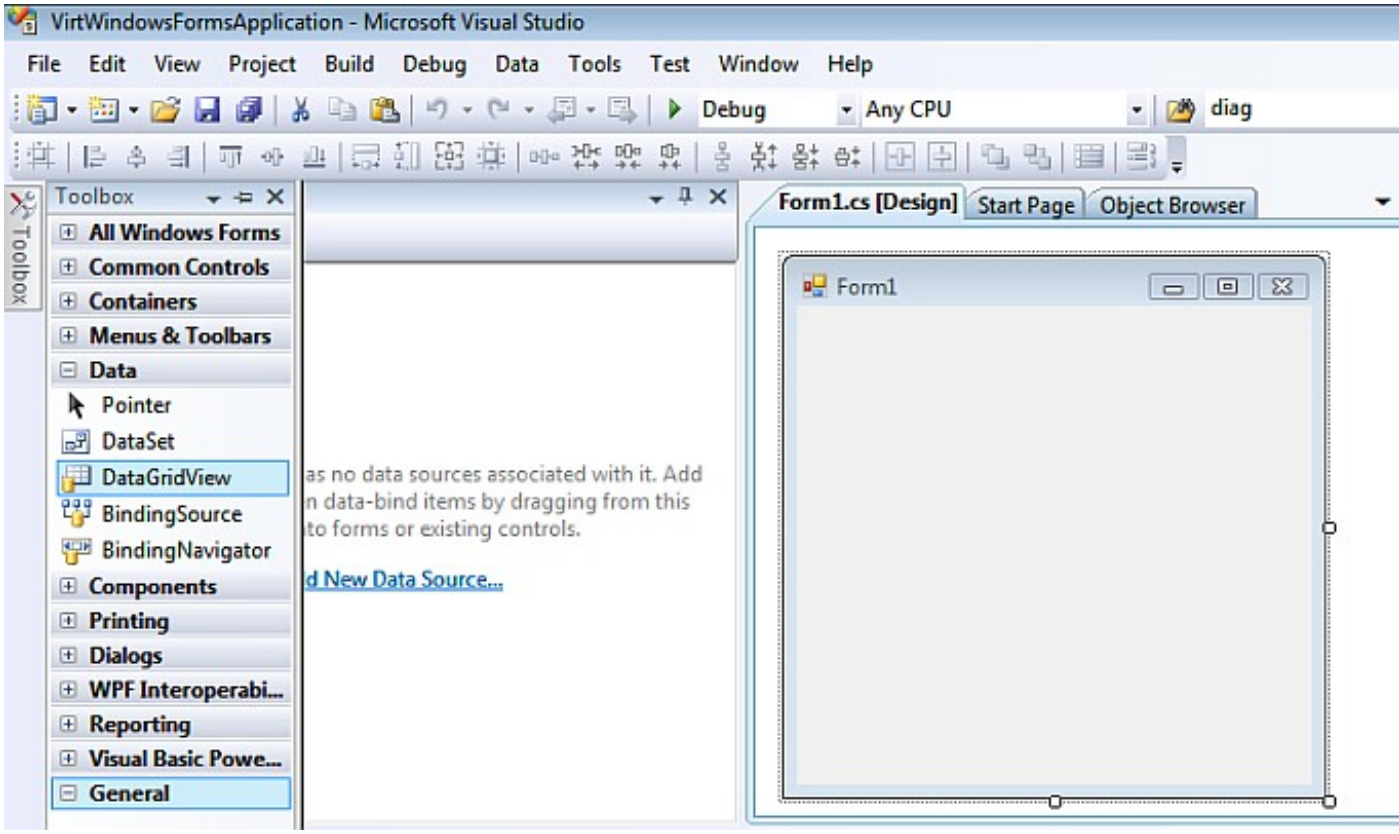
   Data Controls

   , and drag the

   DataGridView

   control onto the form.

   Figure 8.476. Toolbox



7. Click on the little

   arrow

   in the top right of the

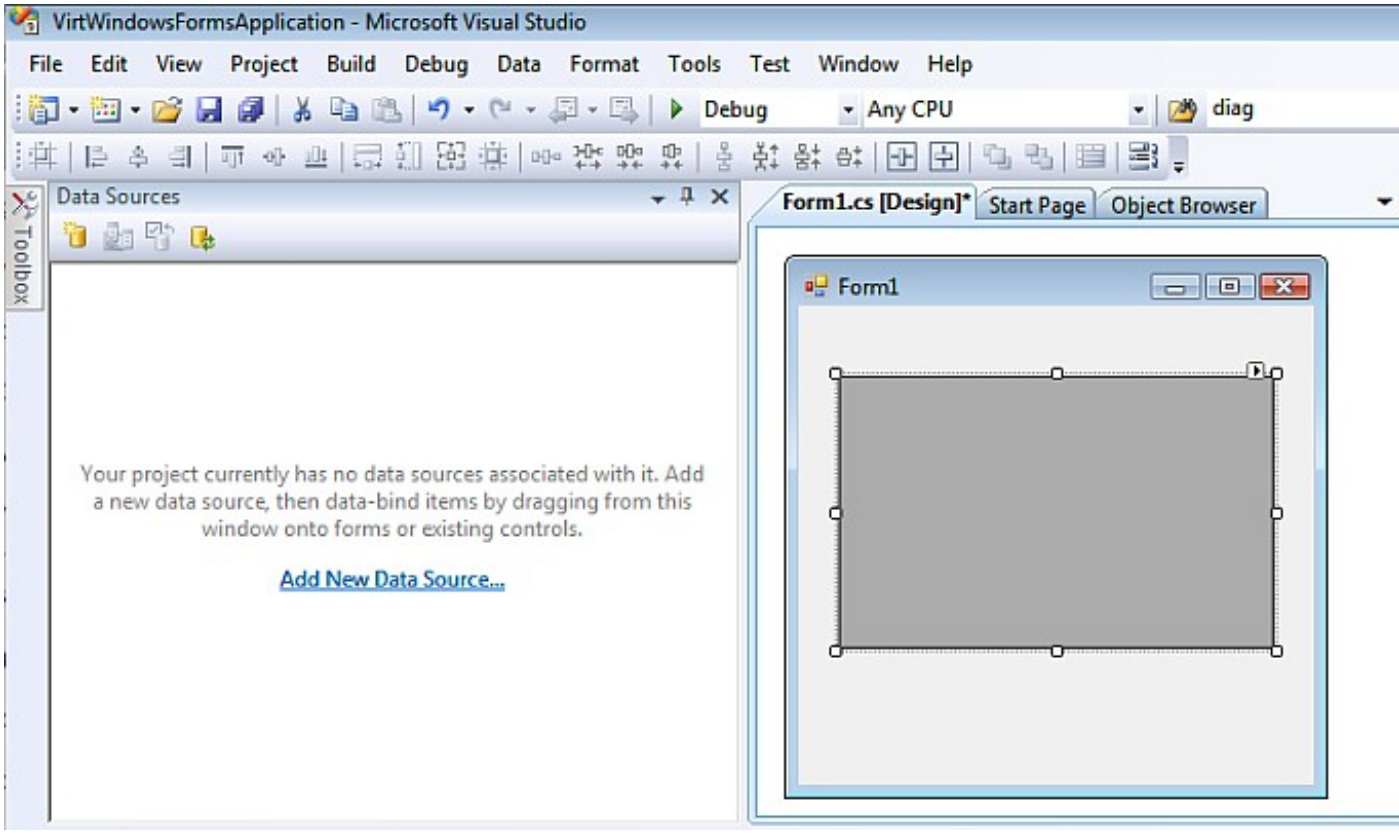
   DataGridView

   control. This loads the

   DataGridView Task

   menu.

Figure 8.477. DataGridView Task

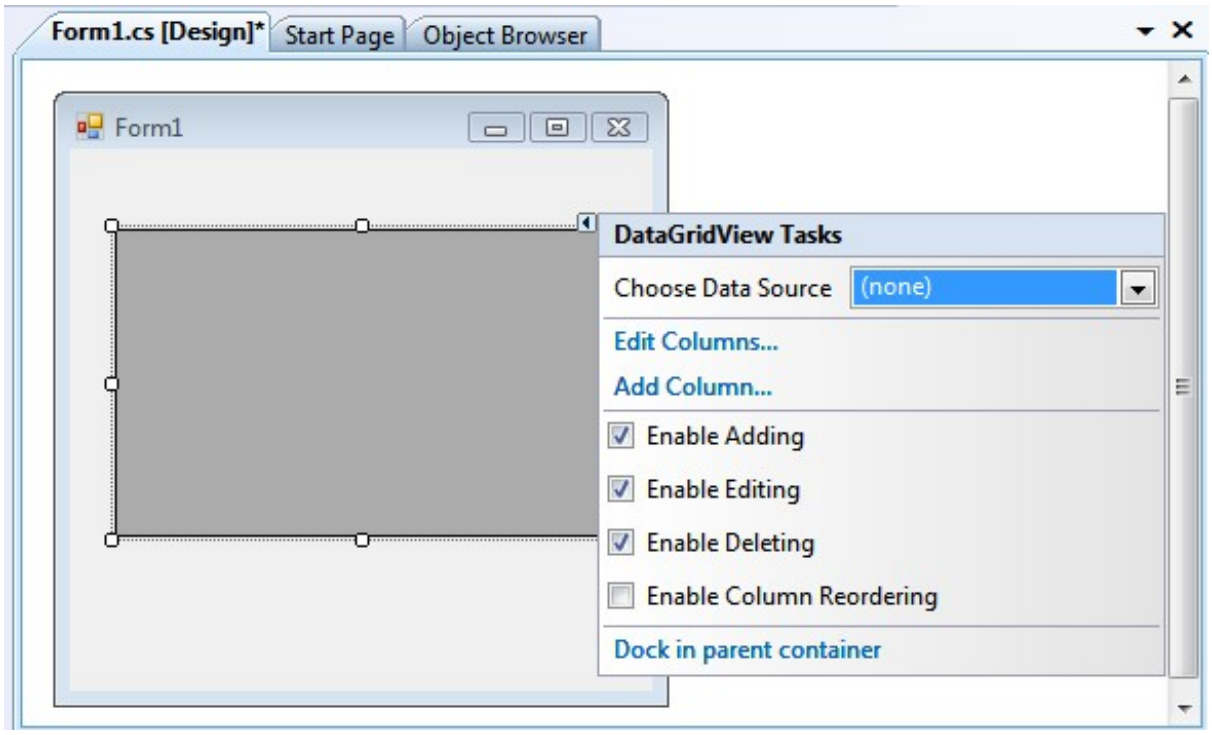


8. Click on the

   Choose Data Source

   list box.

   Figure 8.478. Choose Data Source

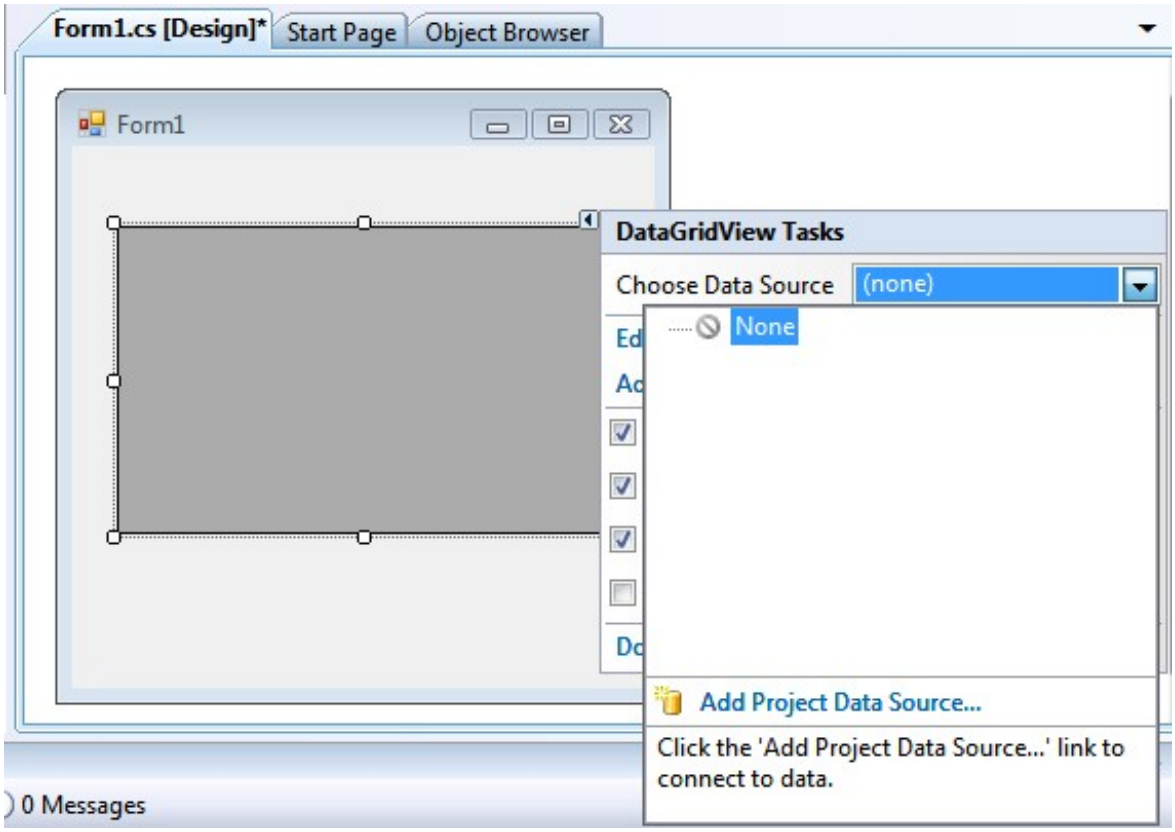


9. Click on the

   Add Project Data Source

link to connect to a data source.

Figure 8.479. Add Project Data Source



10. In the

    Data Source Configuration Wizard

    dialog

    Choose Data Source Type

    page select the

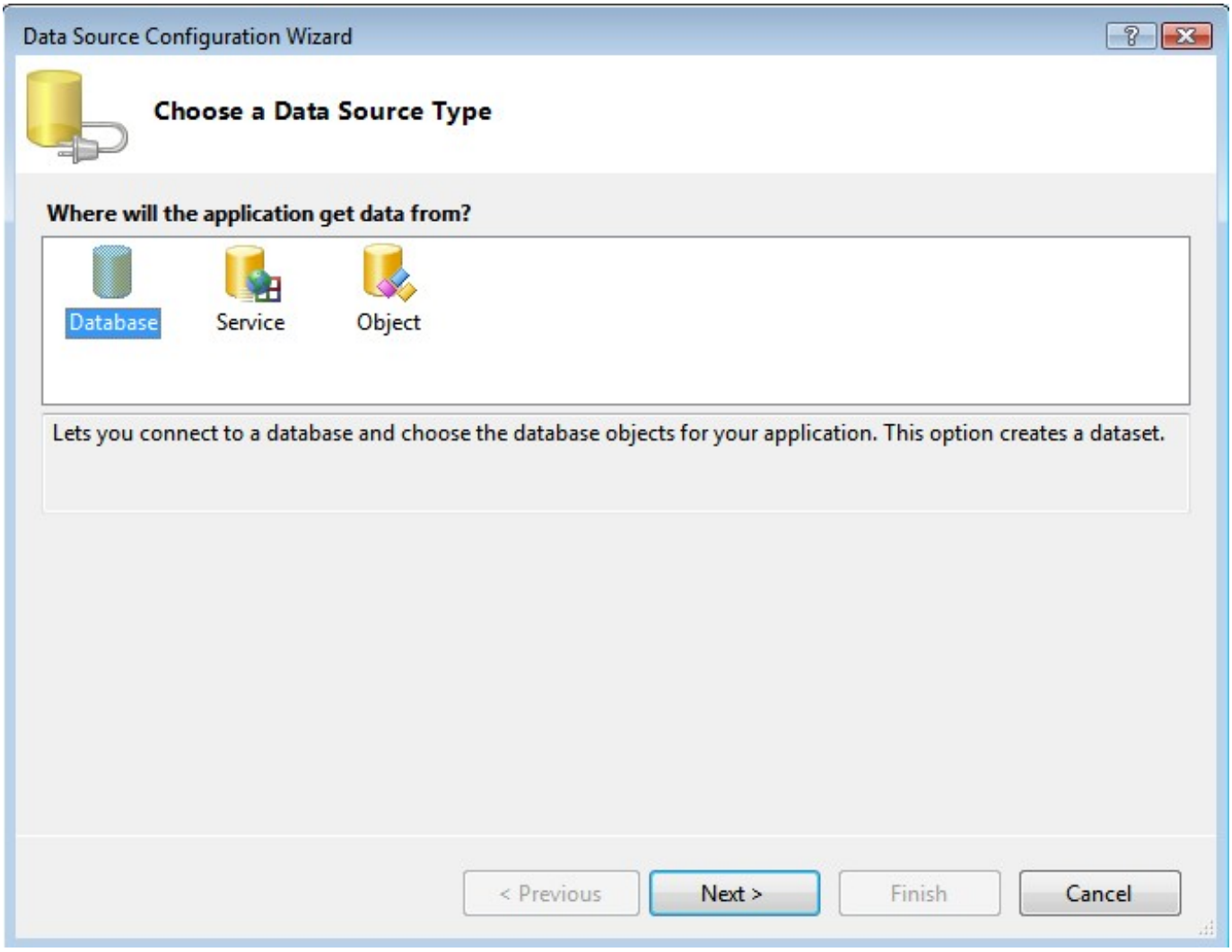
    Database

    data source type and click

    Next

    .

Figure 8.480. Data Source Type



11. In the

    Data Source Configuration Wizard

    dialog

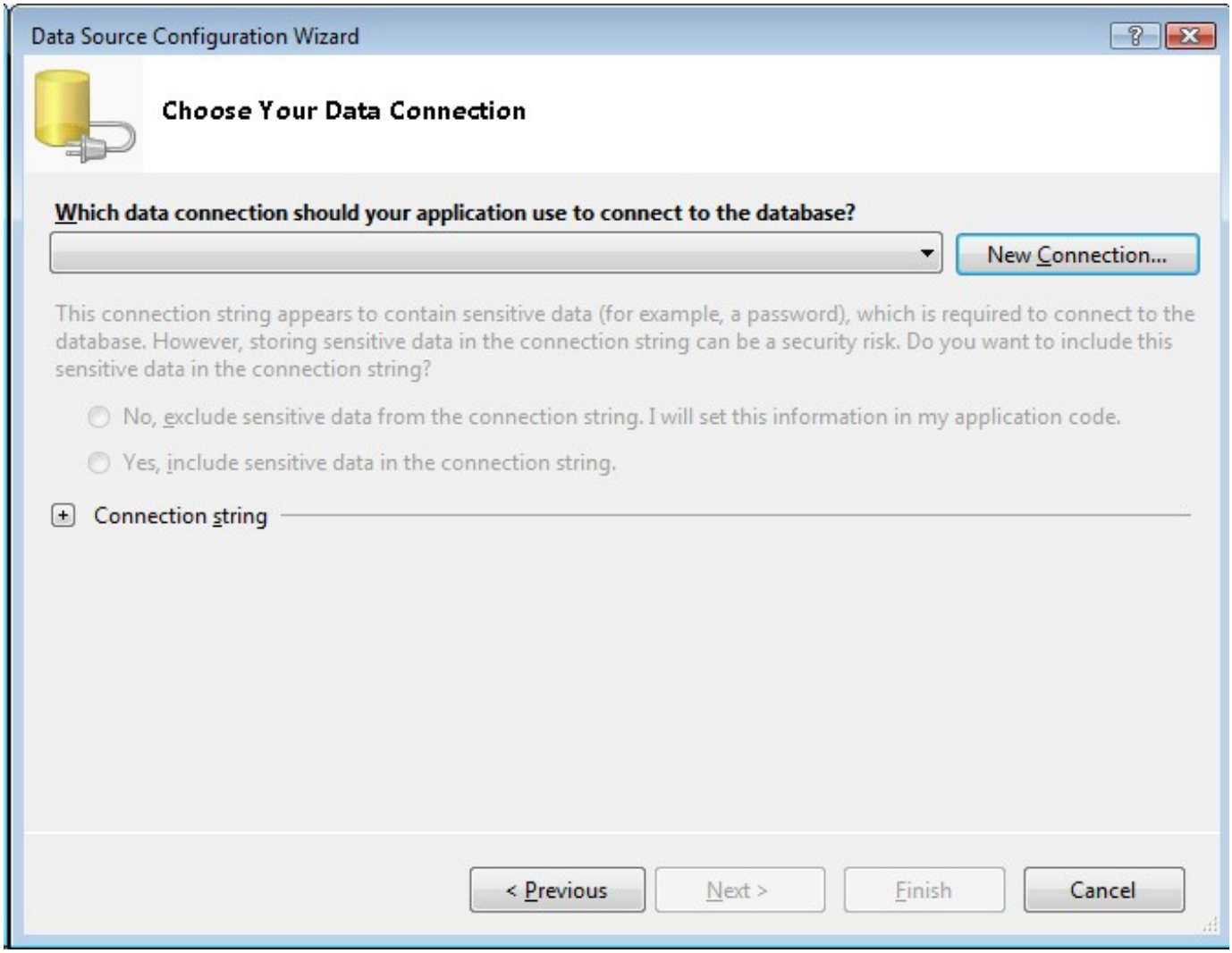
    Choose your Data Connection

    page, select the

    New Connection

    button

Figure 8.481. Data Source Configuration Wizard



12. In the

    Choose Data Source

    dialog, select the OpenLink

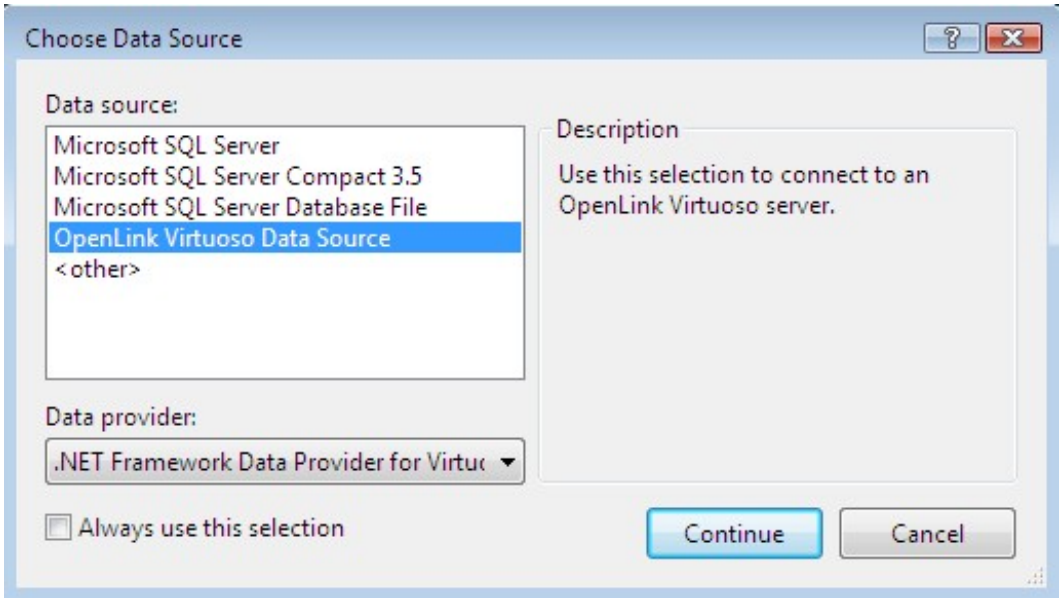
    Virtuoso Data Source

    from the list and click

    Continue

    .

Figure 8.482. Data Source



13. In the

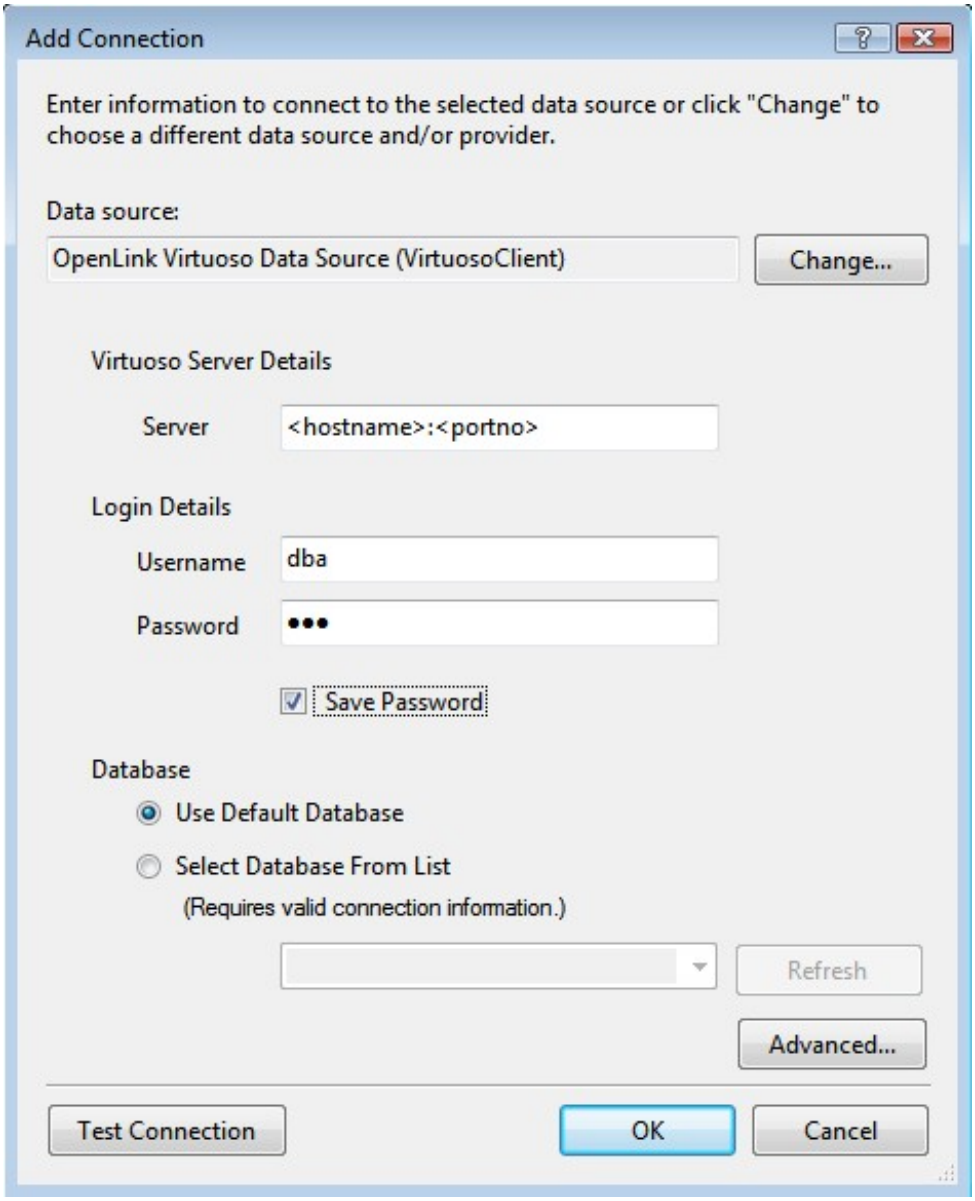
    Add Connection

    dialog, specify the

    hostname, portno, username and password

    for the target Virtuoso Server and check the Save Password check box.

    Figure 8.483. Connection Properties



14. Select the

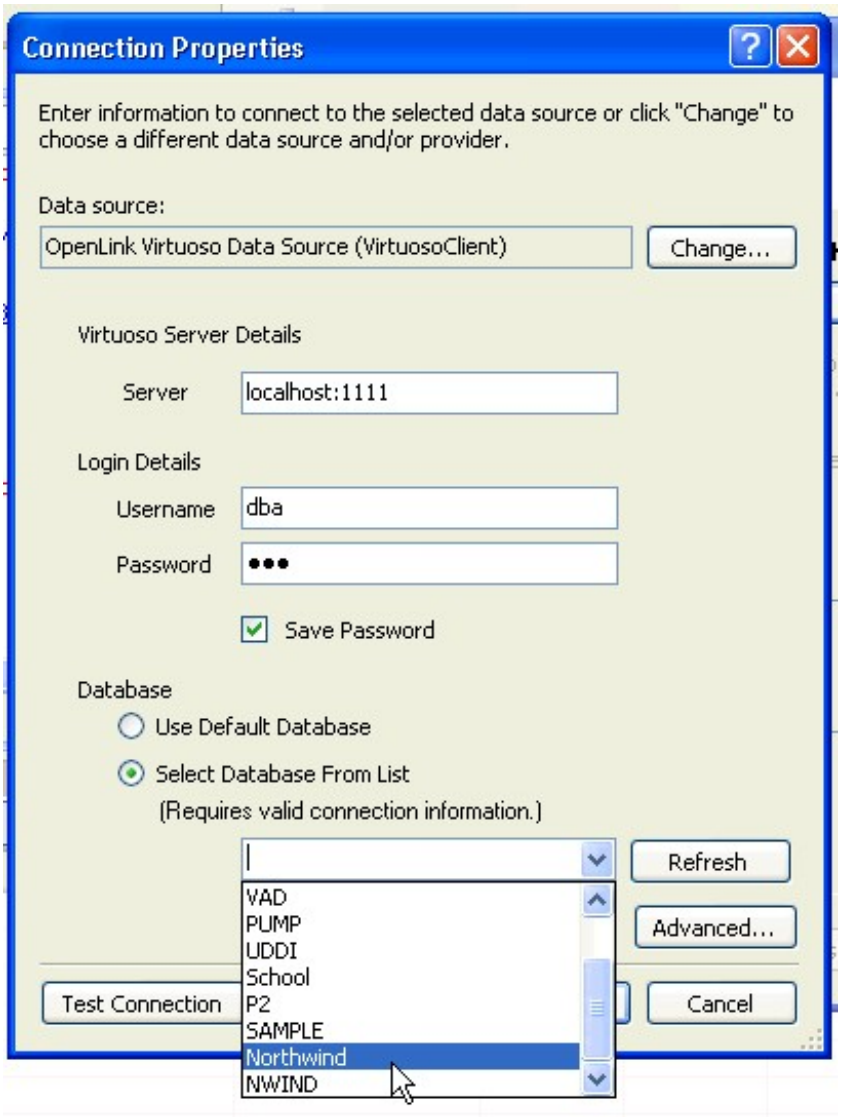
    Select Database From List

    radio button and choose the

    Northwind

    database from the drop down list.

Figure 8.484. Add connection

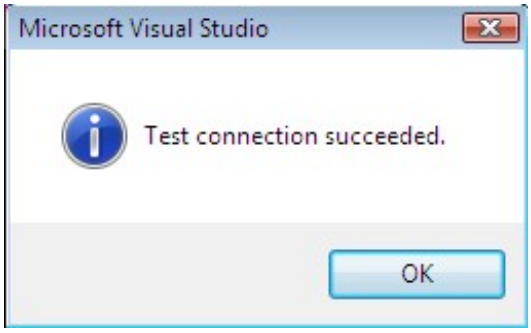


15. Press the

    Test Connection

    dialog to verify that the database is accessible.

    Figure 8.485. Test Connection



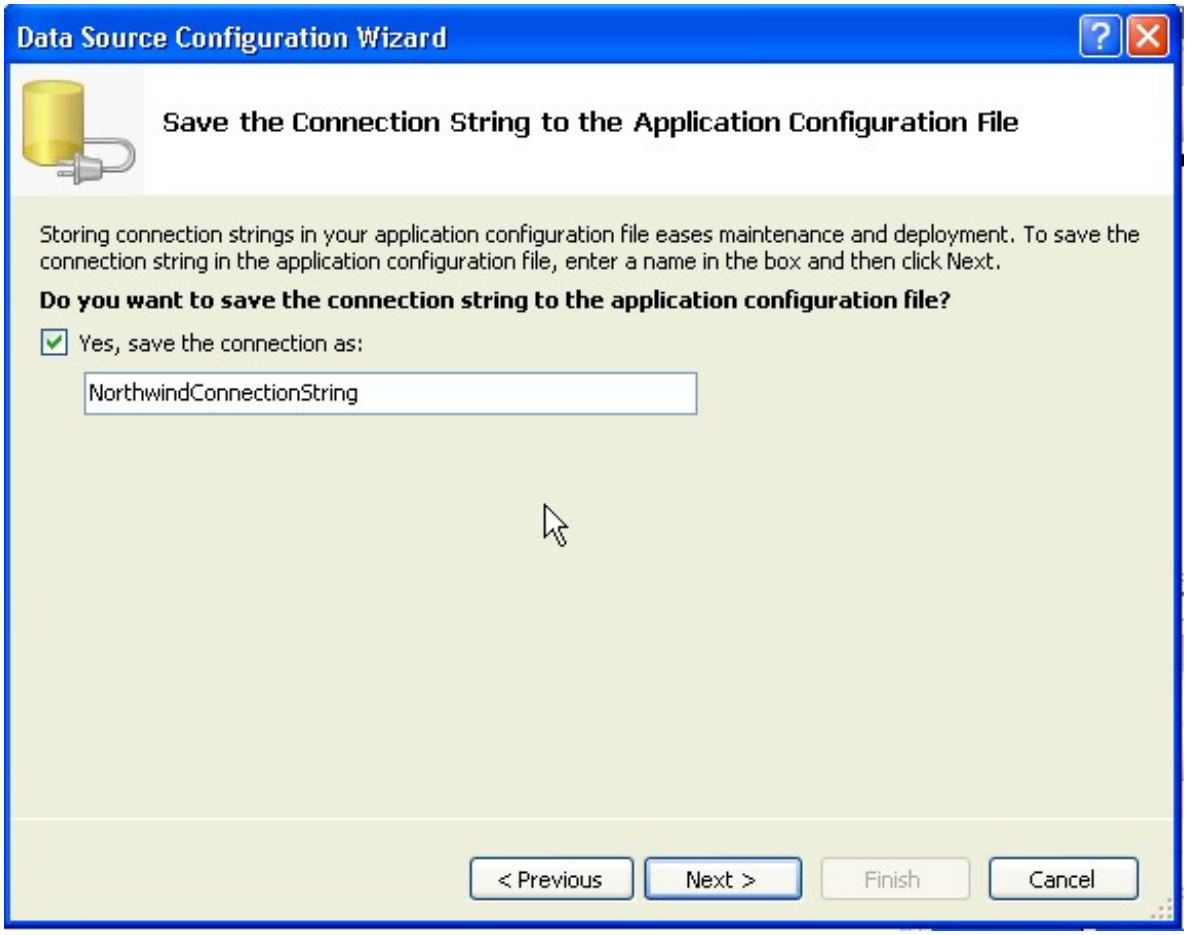
16. Leave the default connect string

    NorthwindConnectionString

    and click

Next

Figure 8.486. NorthwindConnectionString



17. From the list of available tables returned for the Northwind database, select the

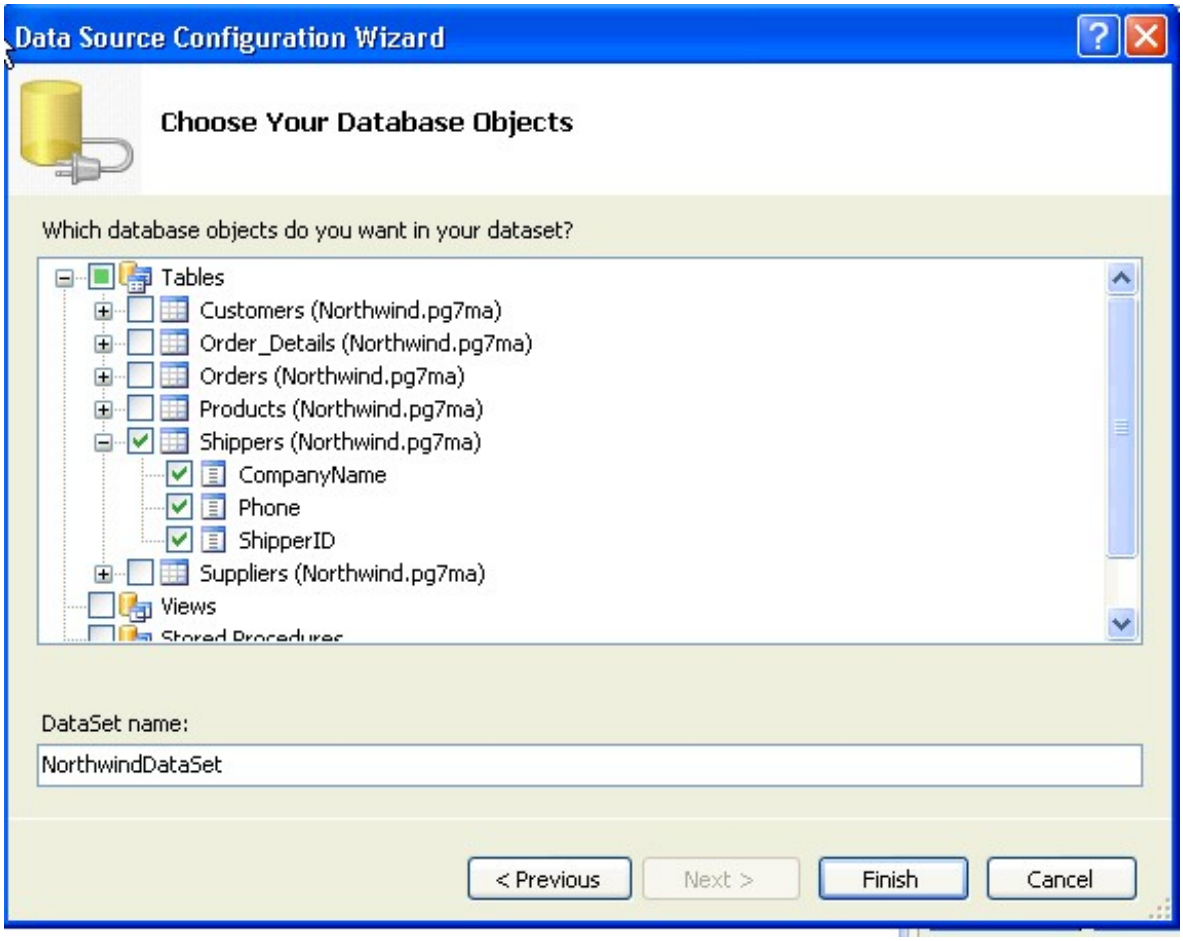
    Shippers

    table to be associated with the

    DataGridView

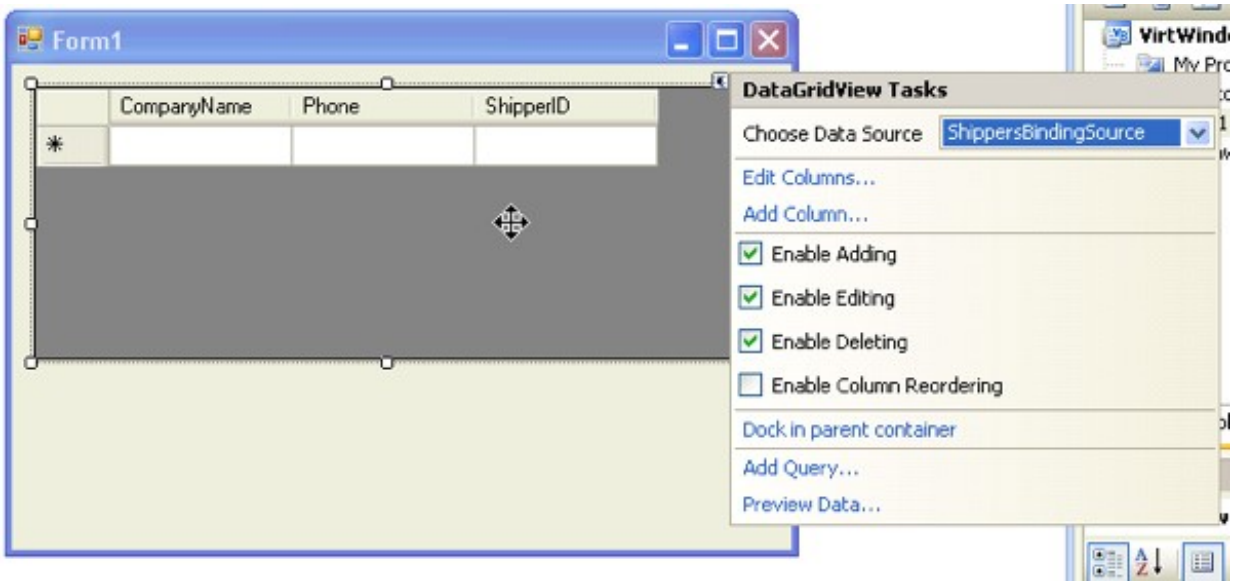
    control.

    Figure 8.487. Northwind database



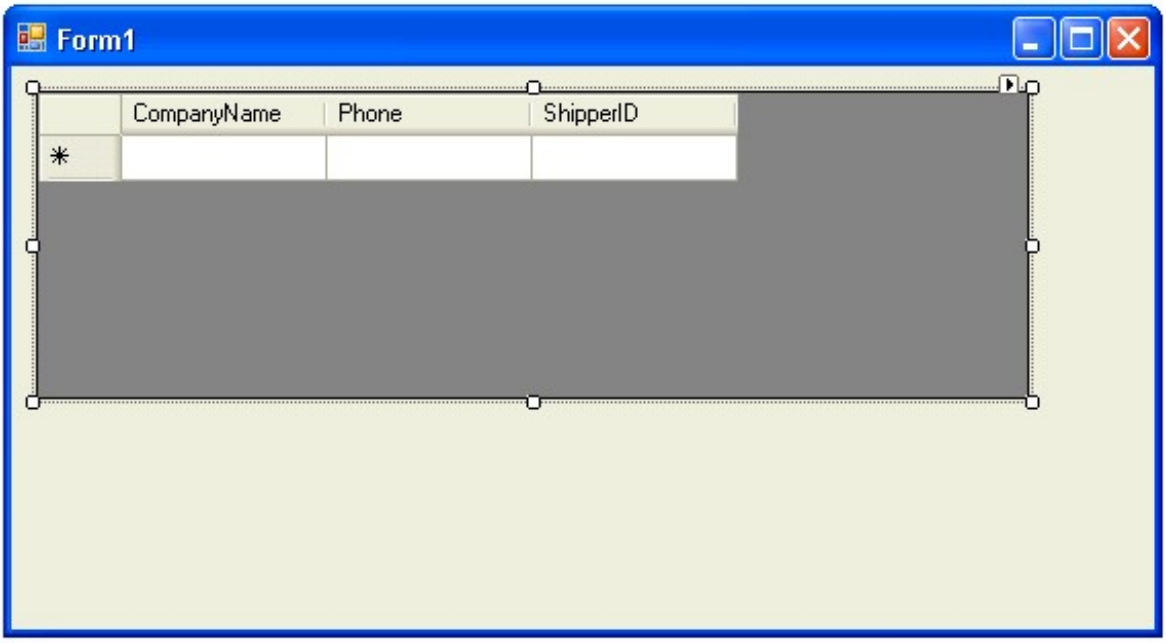
18. The columns names of the select table will be displayed in the DataGridView.

Figure 8.488. DataGridView



19. Resize the Form and DataGridView to allow all columns to be visible, if possible.

Figure 8.489. Resize the Form and DataGridView



20. To test the application, simply hit

    Ctrl+F5

    within Visual Studio or select

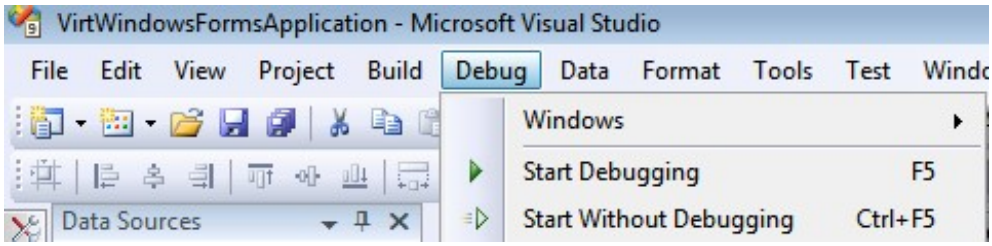
    Start Debugging

    from the

    Debug

    menu.

    Figure 8.490. Start Debugging



21. The data from the

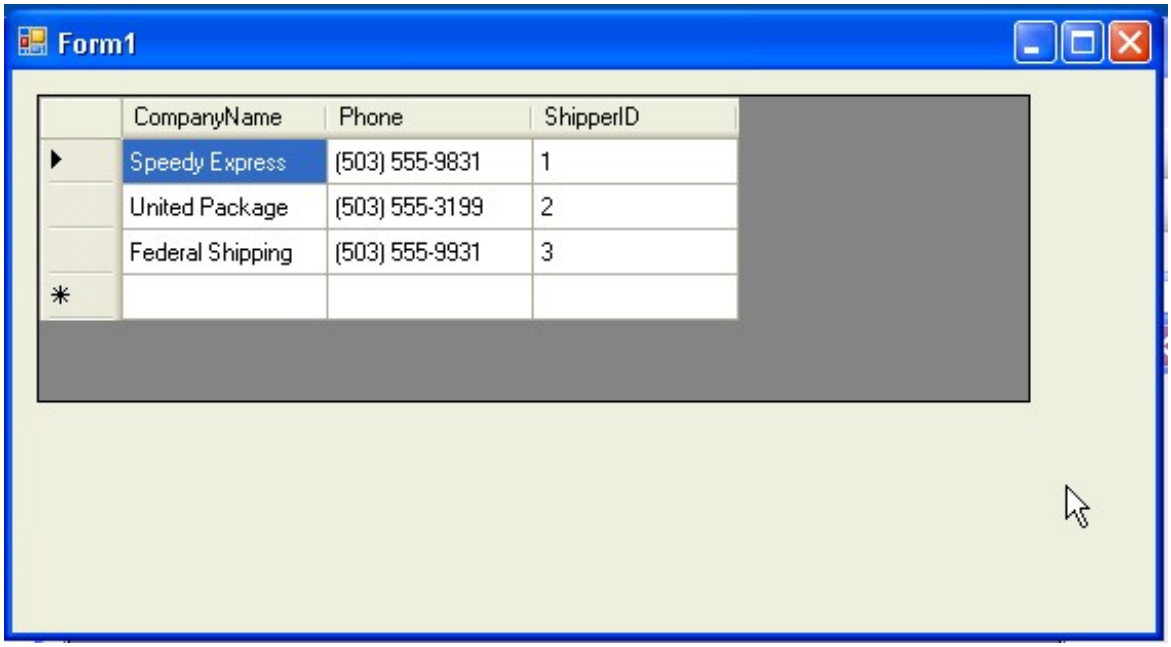
    Shippers

    table will be displayed in the

    DataGrid

    .

    Figure 8.491. DataGrid



The task is now complete.

8.10. Using Microsoft Entity Frameworks to Access ODBC to JDBC Bridge Schema Objects with Virtuoso

Abstract

This section details the steps required to provide Microsoft Entity Framework access to ODBC to JDBC Bridge Schema Objects using the OpenLink Virtuoso Universal Server. This is achieved by linking the required ODBC to JDBC Bridge Schema objects into Virtuoso using its built in Virtual Database engine, and then using the Virtuoso ADO.NET Entity Framework provider to query the remote ODBC to JDBC Bridge Schema objects linked into the Virtuoso Server.

Prerequisites. The following prerequisites must be in place for this solution to be possible.

ODBC to JDBC Bridge DBMS. An ODBC to JDBC Bridge DBMS hosting the required Schema Objects needs to be available. In this section we shall make and ODBC to JDBC bridge connection to another *Virtuoso Northwind Demo* database to demonstrate the process.

An ODBC to JDBC Bridge DBMS hosting the required Schema Objects needs to be available. In this section we shall make and ODBC to JDBC bridge connection to another *Virtuoso Northwind Demo* database to demonstrate the process.

ODBC to JDBC Bridge Driver. An ODBC to JDBC Bridge Driver is required for Linking the ODBC to JDBC Bridge Schema Objects into the Virtuoso Server. The *OpenLink ODBC to JDBC Bridge Driver* will be used in this section, for which a functional ODBC Data source name of "jdbcma" will be assumed to exist on the machine hosting the Virtuoso Server.

An ODBC to JDBC Bridge Driver is required for Linking the ODBC to JDBC Bridge Schema Objects into the Virtuoso Server. The *OpenLink ODBC to JDBC Bridge Driver* will be used in this section, for which a functional ODBC Data source name of "jdbcma" will be assumed to exist on the machine hosting the Virtuoso Server.

Virtuoso Universal Server. An Virtuoso installation including the Virtuoso Universal Server and ADO.NET Entity Framework Provider is required. The Virtuoso components used must be Release 5.10.x or above, this being the minimum version containing support for Microsoft Entity Frameworks

An Virtuoso installation including the Virtuoso Universal Server and ADO.NET Entity Framework Provider is required. The Virtuoso components used must be Release 5.10.x or above, this being the minimum version containing support for Microsoft Entity Frameworks

Microsoft Visual Studio 2008 SP1. Microsoft Visual Studio 2008 Service Pack 1 is required, this being the only version containing the necessary Entity Framework support available at the time of writing.

Microsoft Visual Studio 2008 Service Pack 1 is required, this being the only version containing the necessary Entity Framework support available at the time of writing.

Tasks

Ensure ODBC to JDBC Bridge Primary Keys (PKs) are not nullable. The Visual Studio 2008 Entity Data Model (EDM) requires that all primary keys are *NOT* Nullable, and will fail to generate an EDM if any are. Thus ensure any tables to be used are defined as not nullable in the ODBC to JDBC Bridge database schema before attempting to generate an EDM. In the case of the ODBC to JDBC Bridge Virtuoso Northwind Demo database all tables are not nullable, thus this should not be an issue in this case.

The Visual Studio 2008 Entity Data Model (EDM) requires that all primary keys are *NOT* Nullable, and will fail to generate an EDM if any are. Thus ensure any tables to be used are defined as not nullable in the ODBC to JDBC Bridge database schema before attempting to generate an EDM. In the case of the ODBC to JDBC Bridge Virtuoso Northwind Demo database all tables are not nullable, thus this should not be an issue in this case.

8.10.1. Install and configure OpenLink ODBC Driver for ODBC to JDBC Bridge

The Virtuoso Virtual Database engine uses ODBC as the connectivity mechanism for linking remote database objects into its local schema. Thus an ODBC to JDBC Bridge Driver must be available with a suitably configured DSN for connecting to the target database. The OpenLink ODBC to JDBC Bridge Drivers have been used in this document, although in theory any ODBC to JDBC Bridge Driver can be used.

Installation instructions for the OpenLink ODBC Driver for ODBC to JDBC Bridge are available from:

- Product Installation & Basic Configuration (ODBC)

8.10.2. Install and configure OpenLink Virtuoso Universal Server

- Install and configure OpenLink Virtuoso Universal Server

8.10.3. Linking ODBC to JDBC Bridge tables into OpenLink Virtuoso

1. Start the Virtuoso Web User Interface

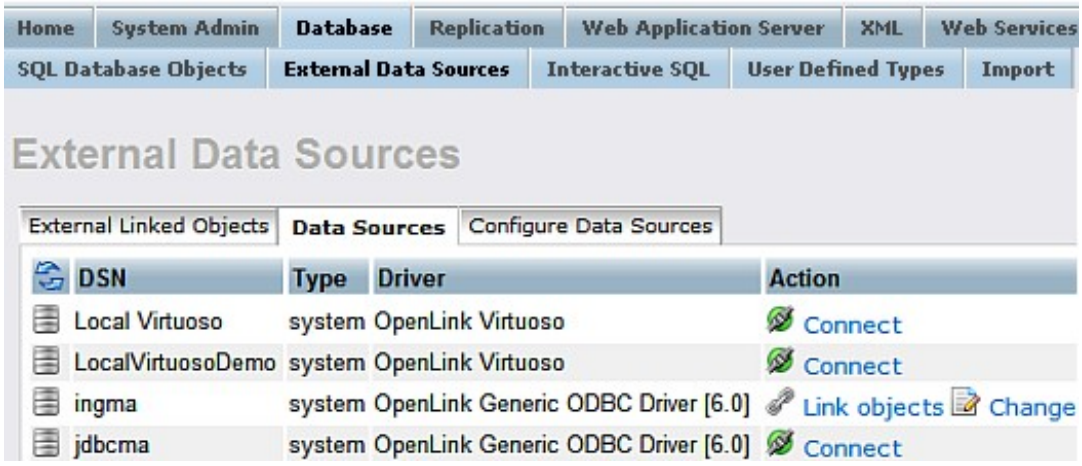
   Figure 8.492. Start

2. Select the "Conductor" link in the Left Frame and Login as the "dba" user.

Figure 8.493. Conductor

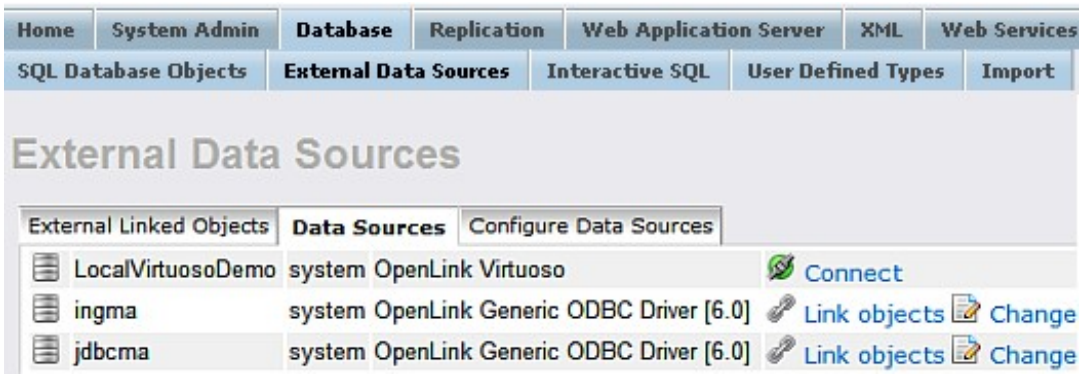
3. Select the "Databases" -> "External Data Source" -> "Data Sources" tab menu items

Figure 8.494. Databases



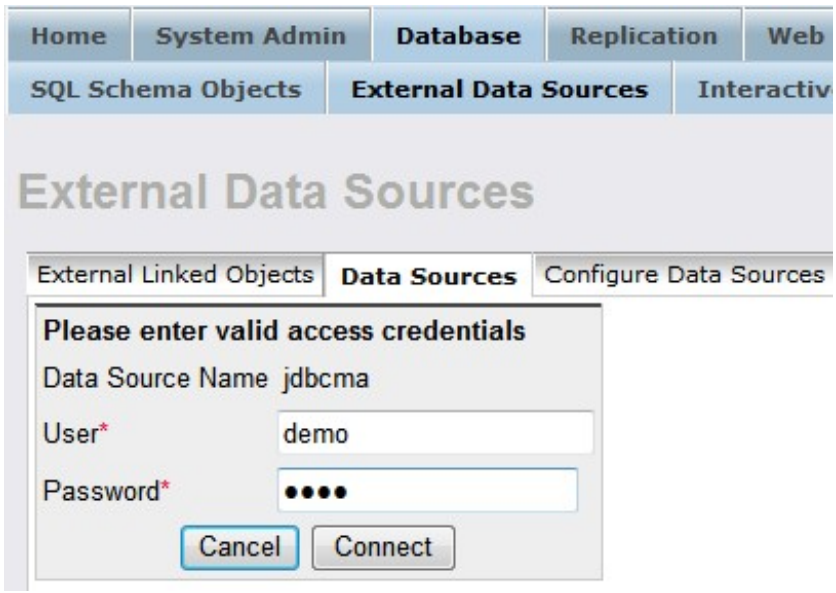
4. Select the "Connect" button for the "jdbcma" ODBC to JDBC Bridge DSN.

Figure 8.495. Connect



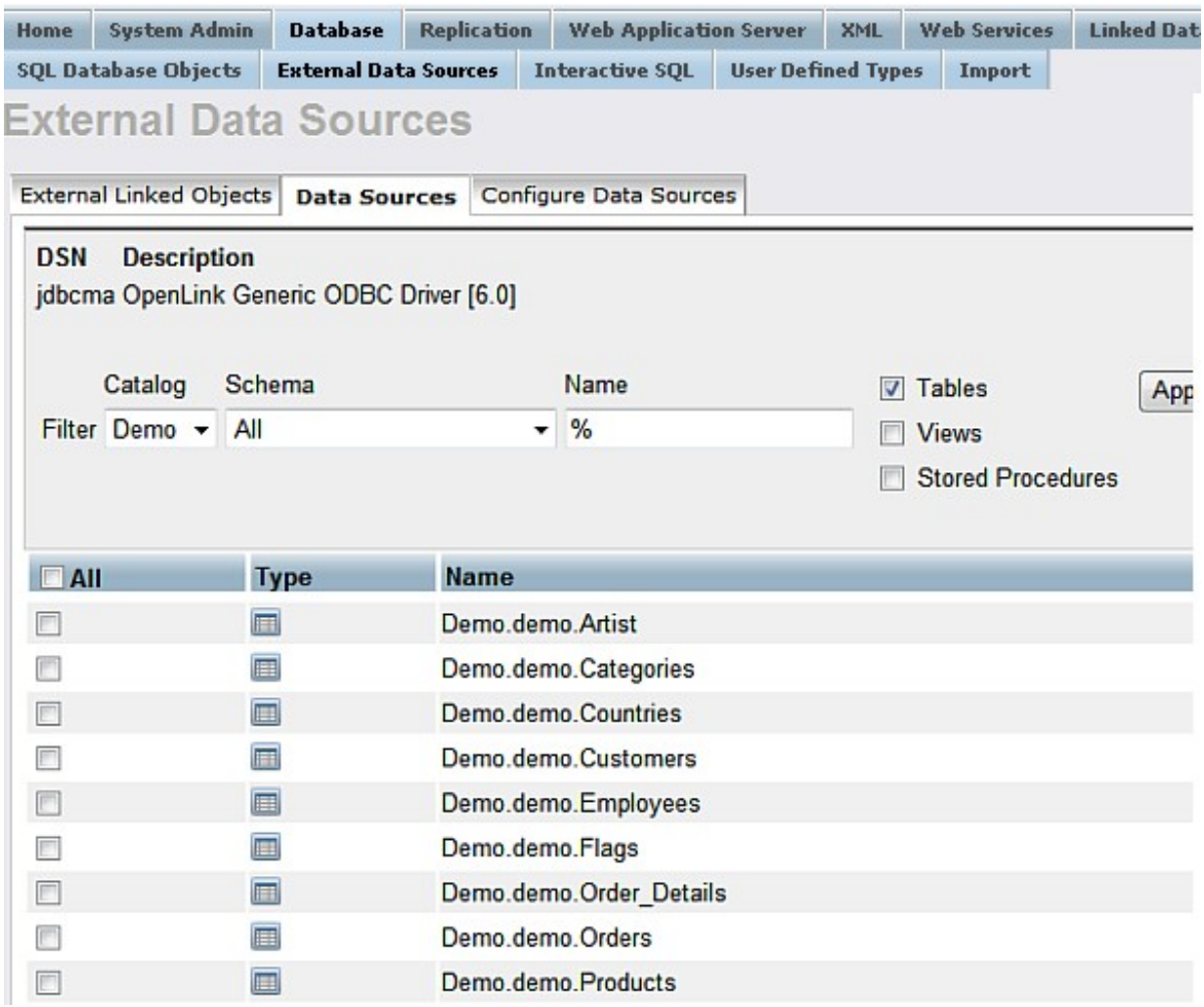
5. On successful connection Select the "Link Objects" button to obtain a list of available tables

Figure 8.496. Link Objects



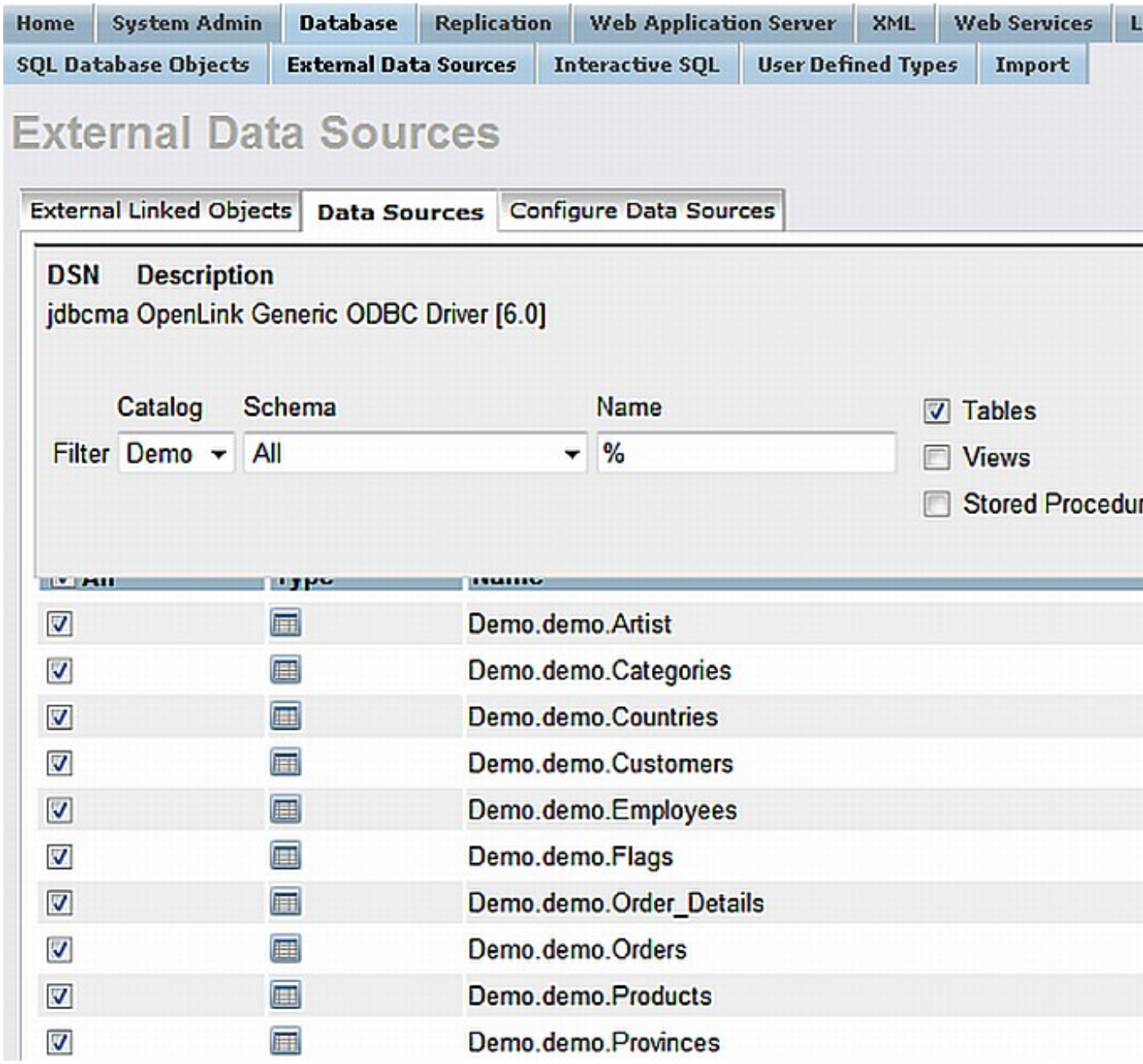
6. Select all the tables that are part of the "JDBC" catalog.

Figure 8.497. Select tables



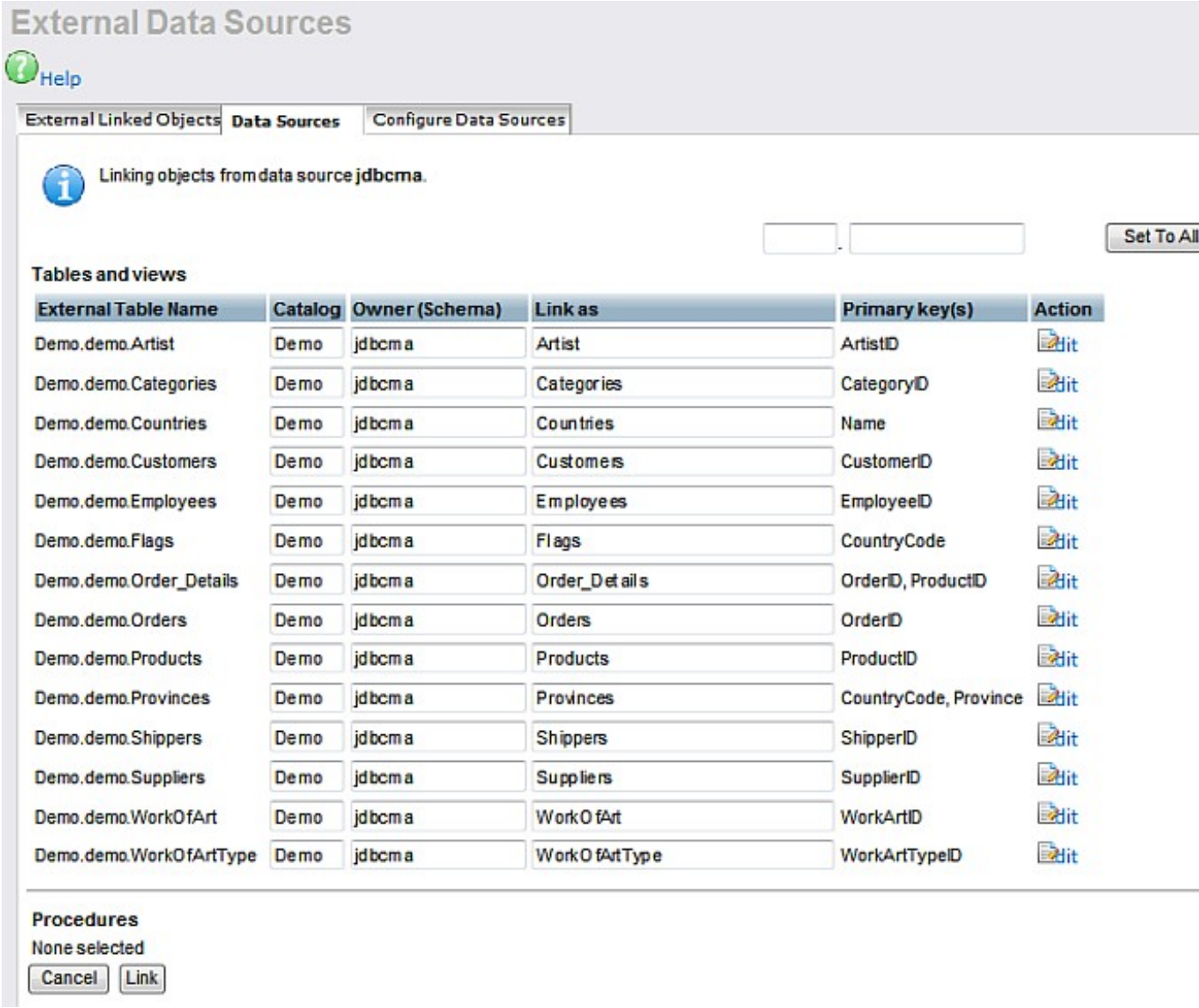
7. Select all the "JDBC" Catalog tables to be linked into the database and select "Link" button.

Figure 8.498. "Link" button



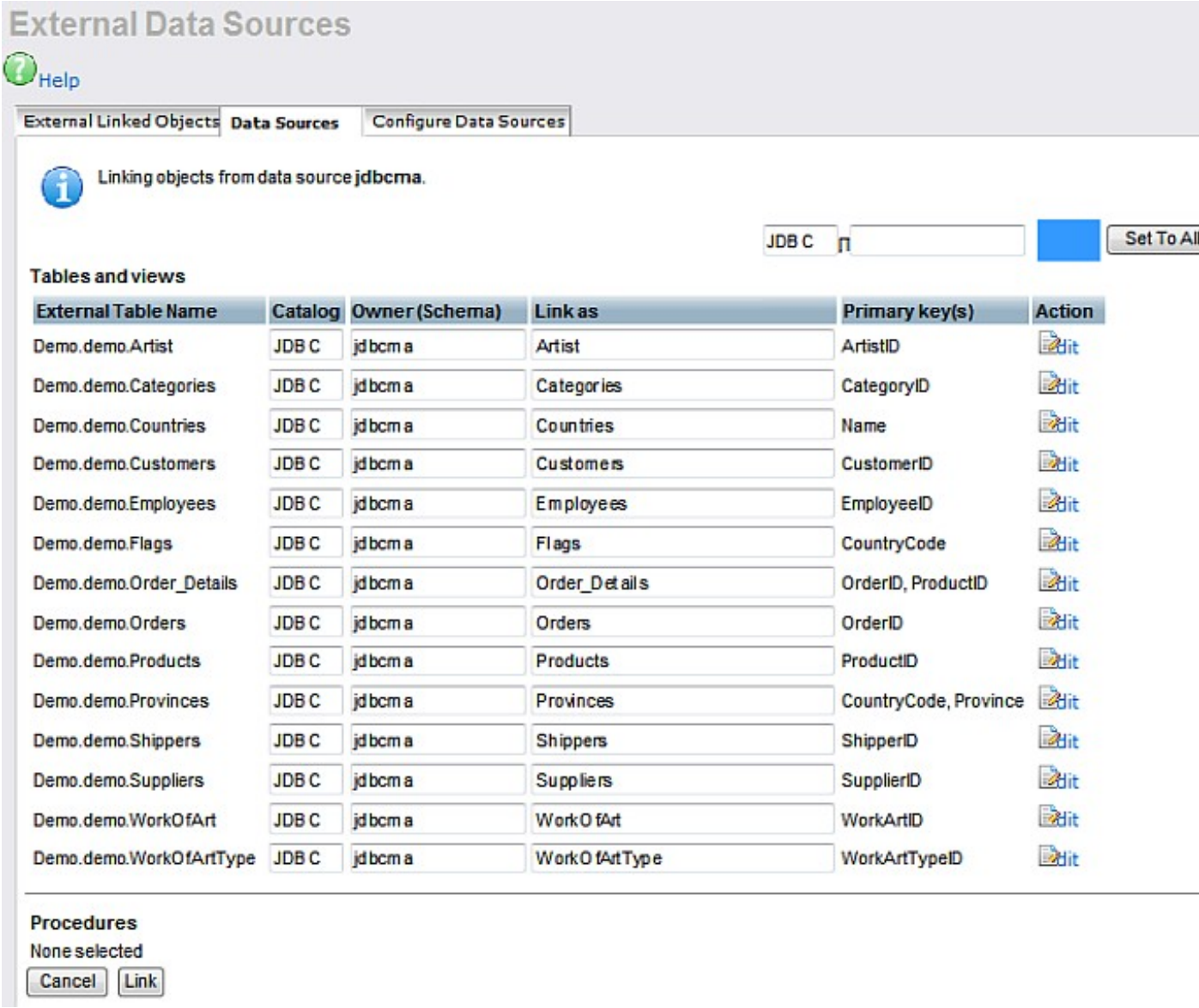
8. Change the Catalog for each table to be "JDBC" using the "Set All" button.

Figure 8.499. "Link" button



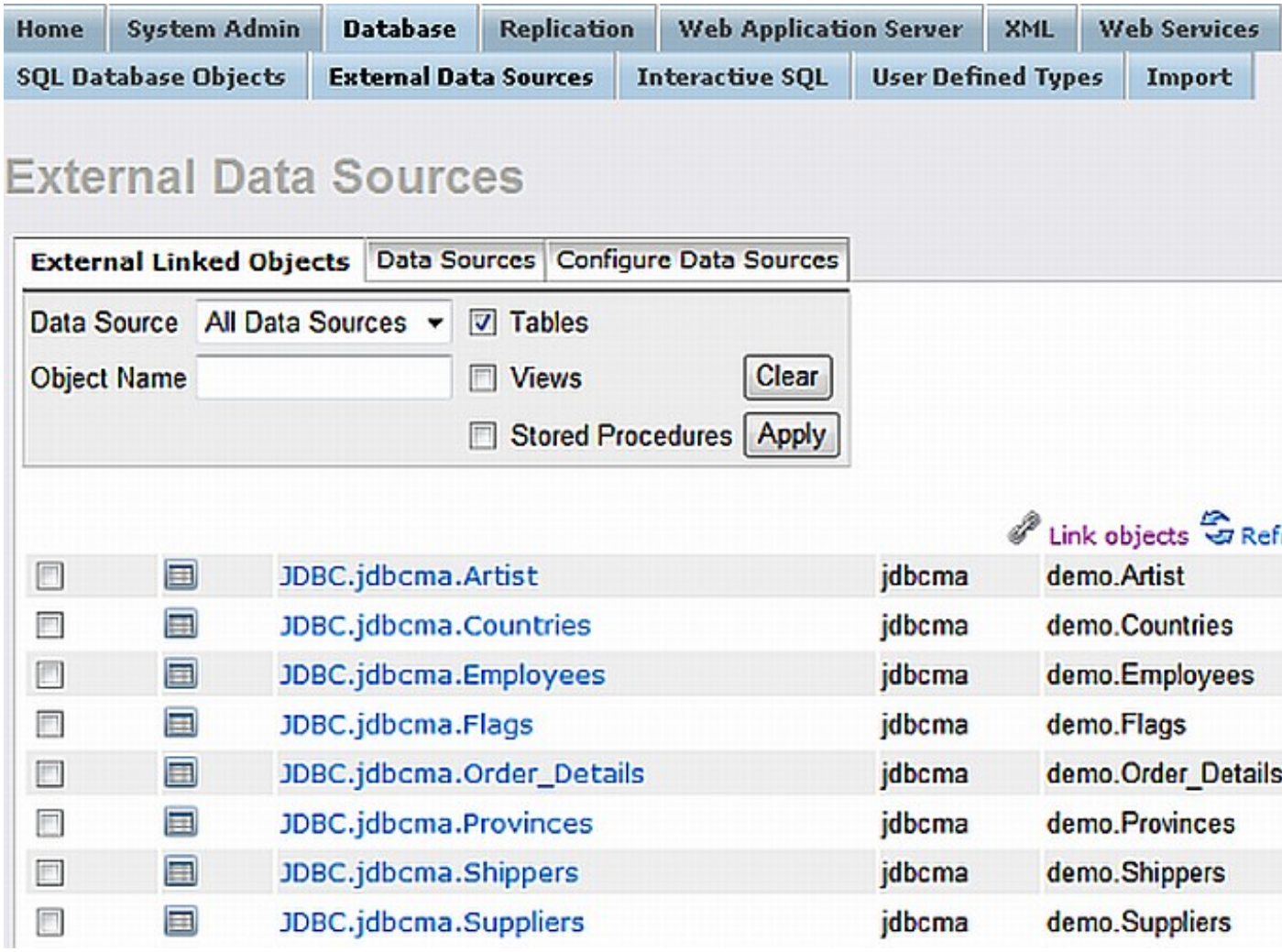
9. Select the "Link" button to link the selected tables into Virtuoso

Figure 8.500. "Link" button



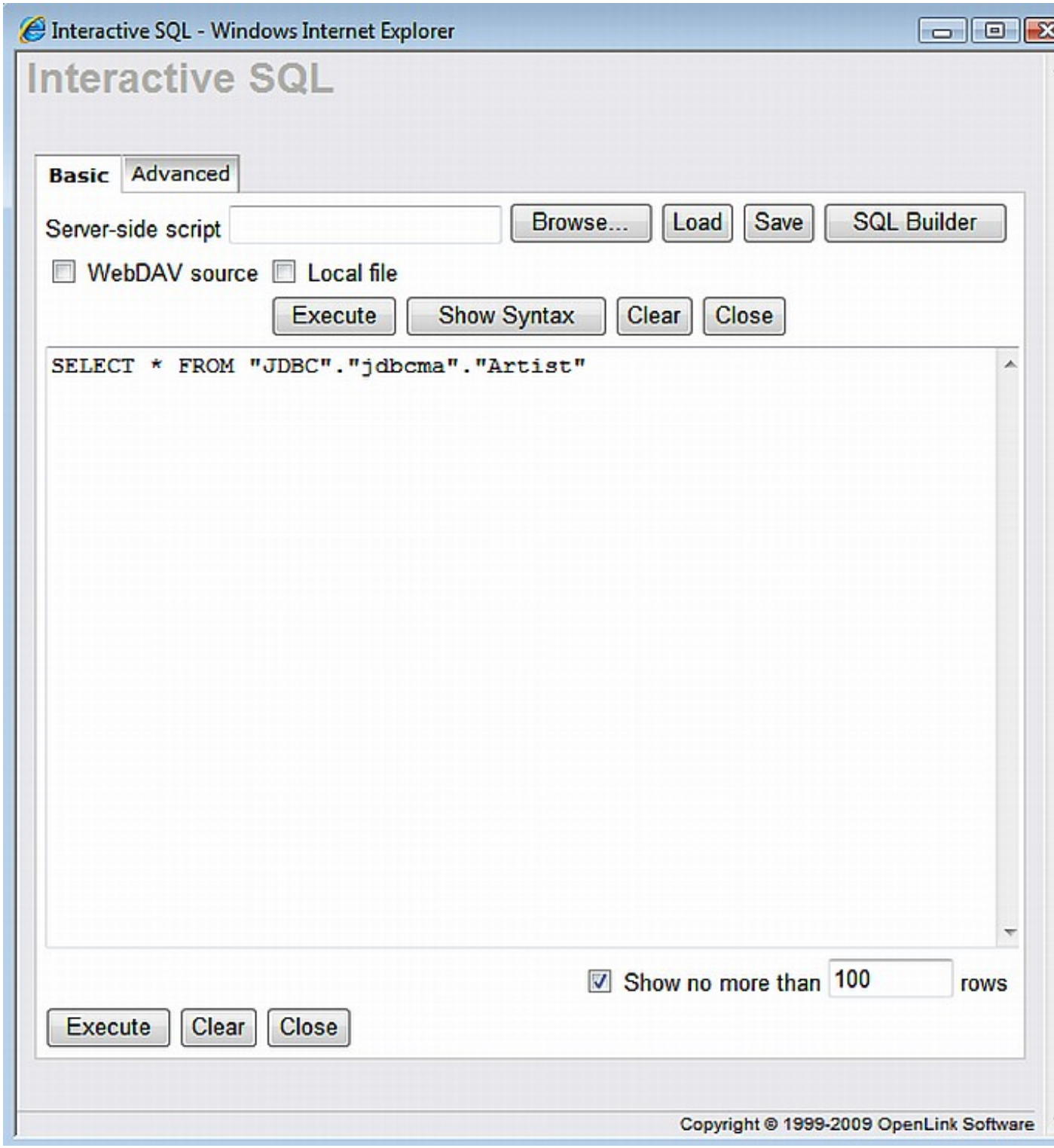
10. On completion of the Link process the tables will be displayed in the "External Linked Objects" tab.

Figure 8.501. Completion



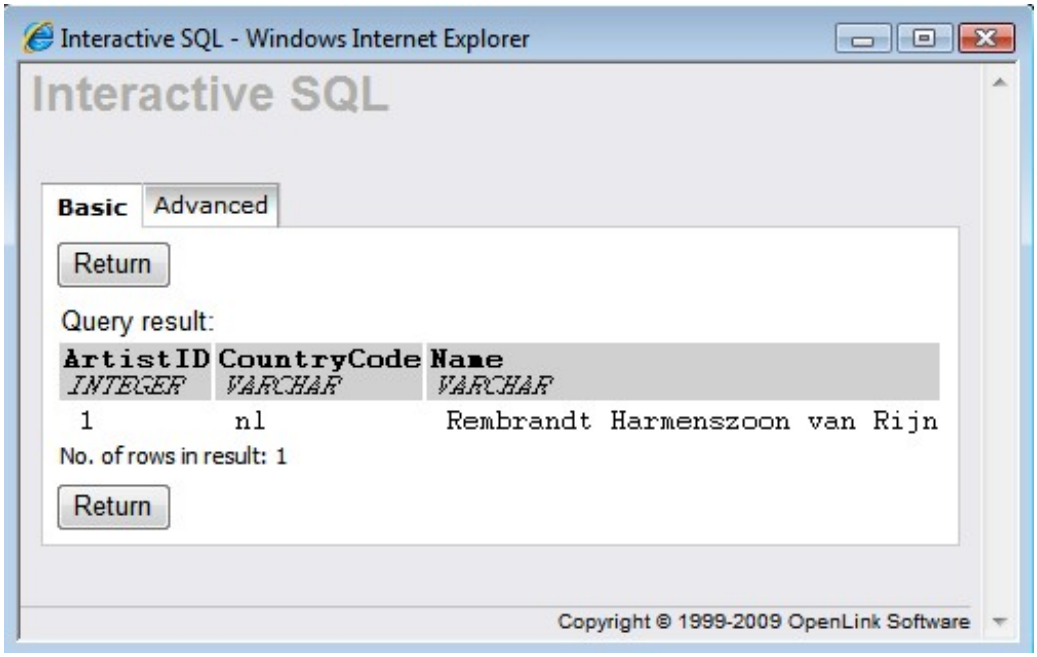
11. The linked tables can be queried by clicking on the hyperlink in the "Local Name" column of the "External Linked Objects" tab above, which loads the Virtuoso "Interactive SQL" interface with the required SQL "Select" for retrieving the remote table data. We shall use the "JDBC.jdbcma.Artist" table to demonstrate this.

Figure 8.502. Completion



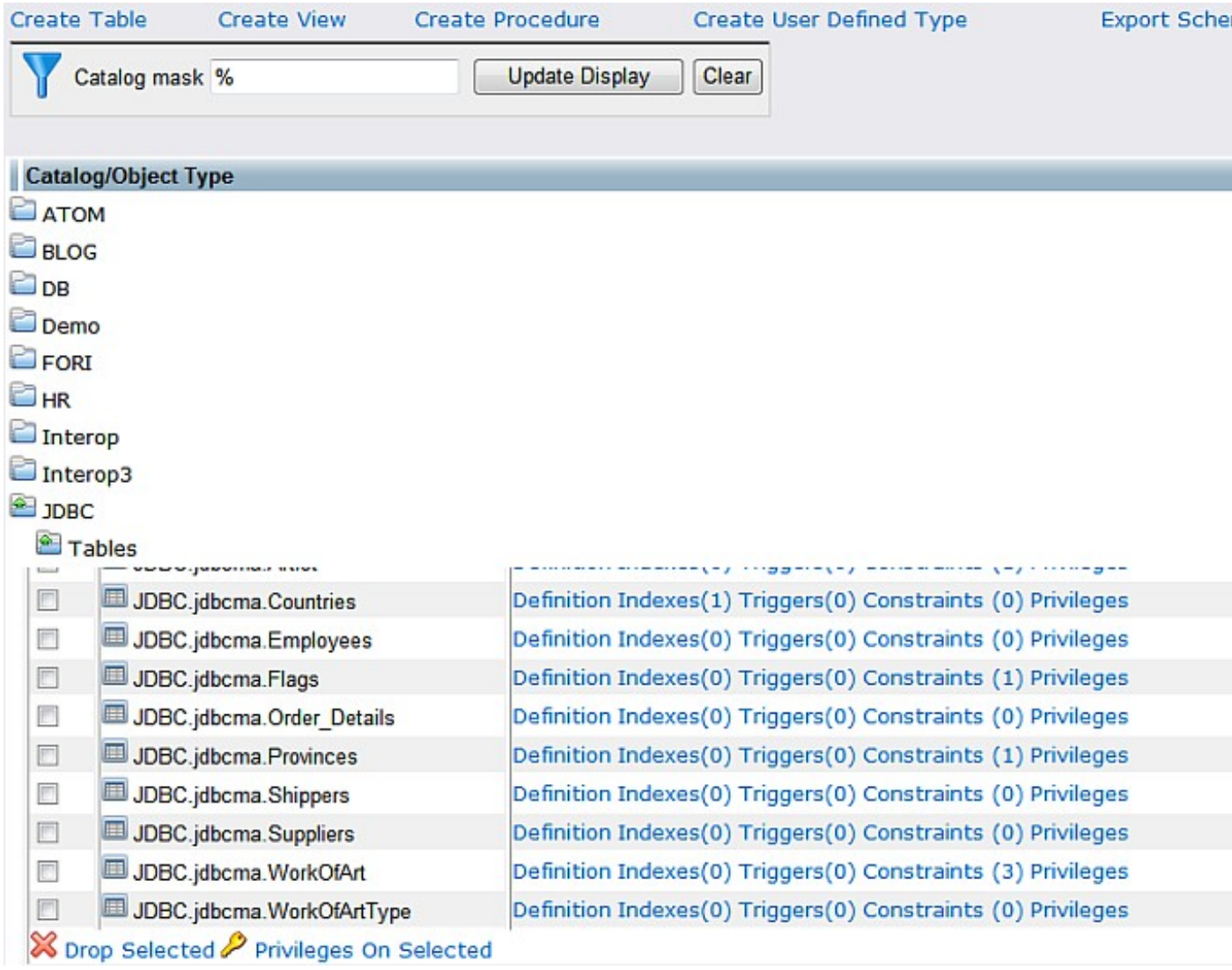
12. Then click the "Execute" button to run the query and retrieve the results from the remote table.

Figure 8.503. Completion



13. The tables can also be viewed as part of the Virtuoso "SQL Schema Objects" under the "JDBC" catalog name.

Figure 8.504. view tables



The Link process is now complete enabling the tables to be queried as if part of the Virtuoso Schema.

8.10.4. Creating EDM in Visual Studio 2008

The following steps can be used to create an Entity Data Model (EDM) for the ODBC to JDBC Bridge Northwind database:

1. Launch the Visual Studio 2008 SP1 IDE.

   Figure 8.505. Visual Studio 2008 SP1 IDE

2. Create a

   Web Application

   project by going to the

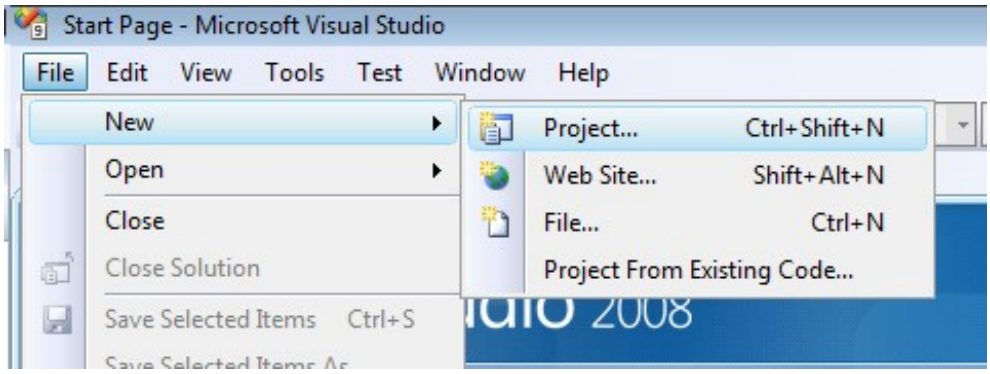
   File

   menu in Visual Studio and choosing

   New Project

   .

   Figure 8.506. Web Application

3. When the New Project window appears, choose either

   Visual Basic

   or

   Visual C#

as your programming language.

4. Within the language category, click on

   Web

   and select

   ASP.NET Web Application

   from the right-hand panel.
5. Choose a name for the project, for example

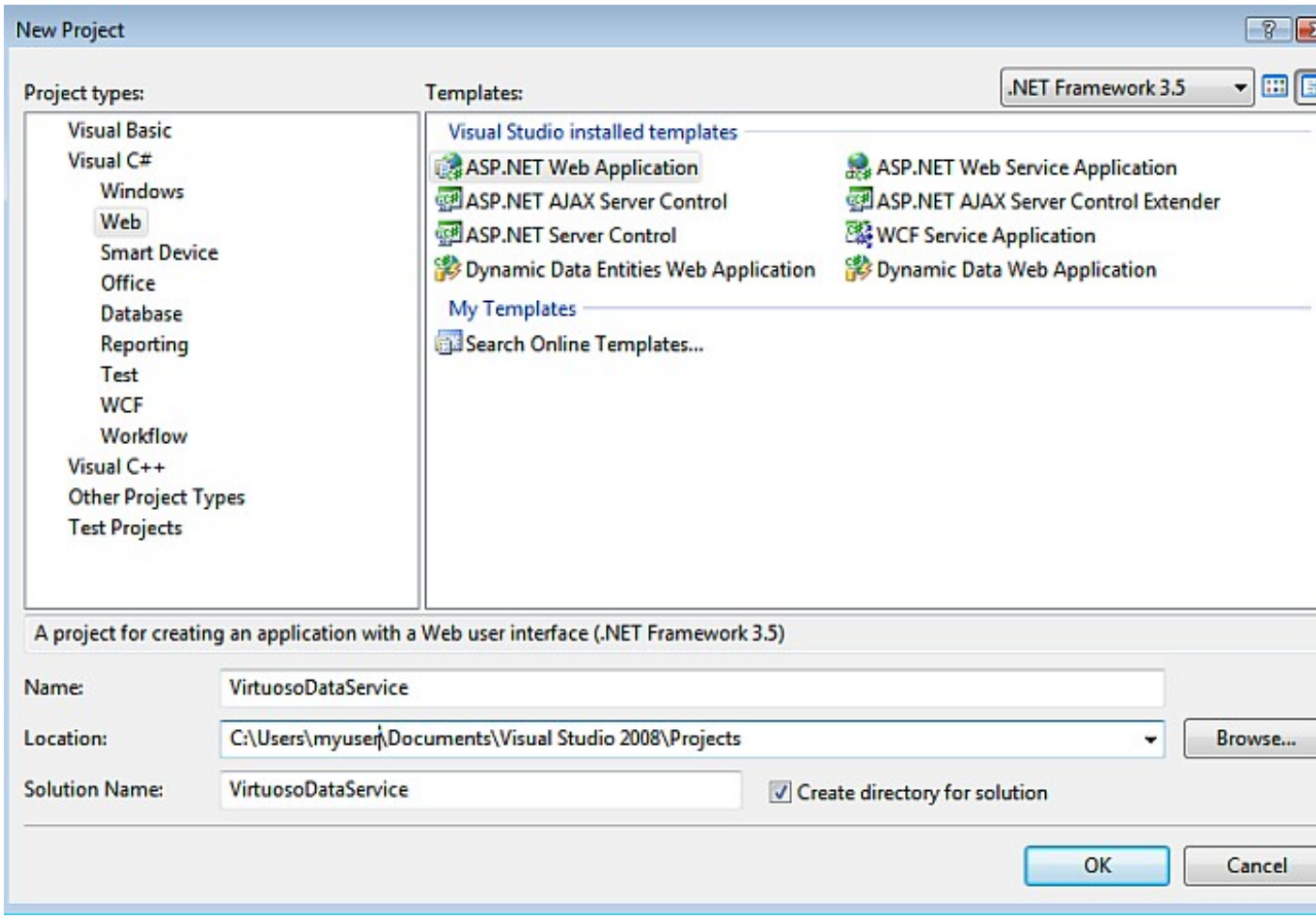
   VirtuosoDataService

   , and click

   OK

   .

Figure 8.507. name for the project

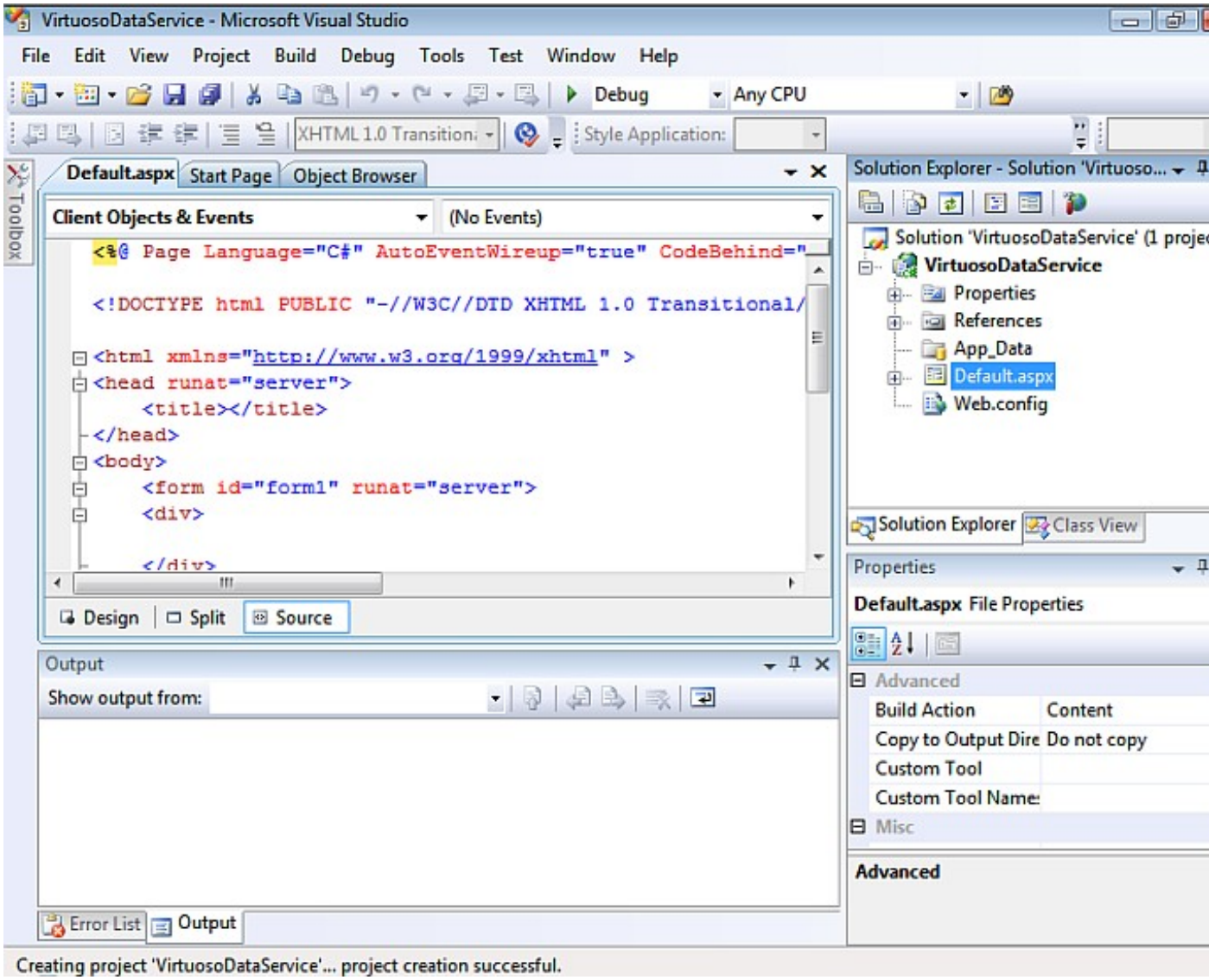


6. This will create a new project called

   VirtuosoDataService

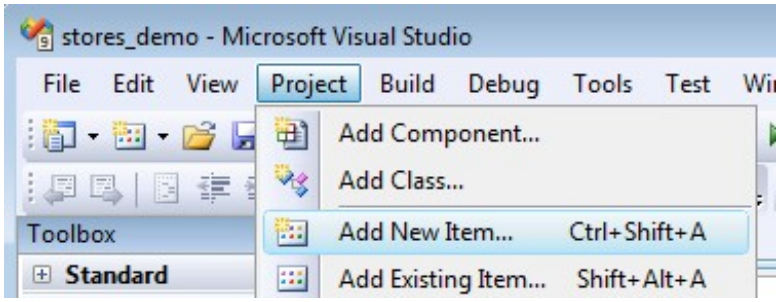
   .

Figure 8.508. create a new project



7. Select the Project -> Add New Item menu option.

Figure 8.509. VirtuosoDataService



8. The

Add New Item

dialog will appear. Choose the

ADO.NET Entity Data Model

template. Give it the name

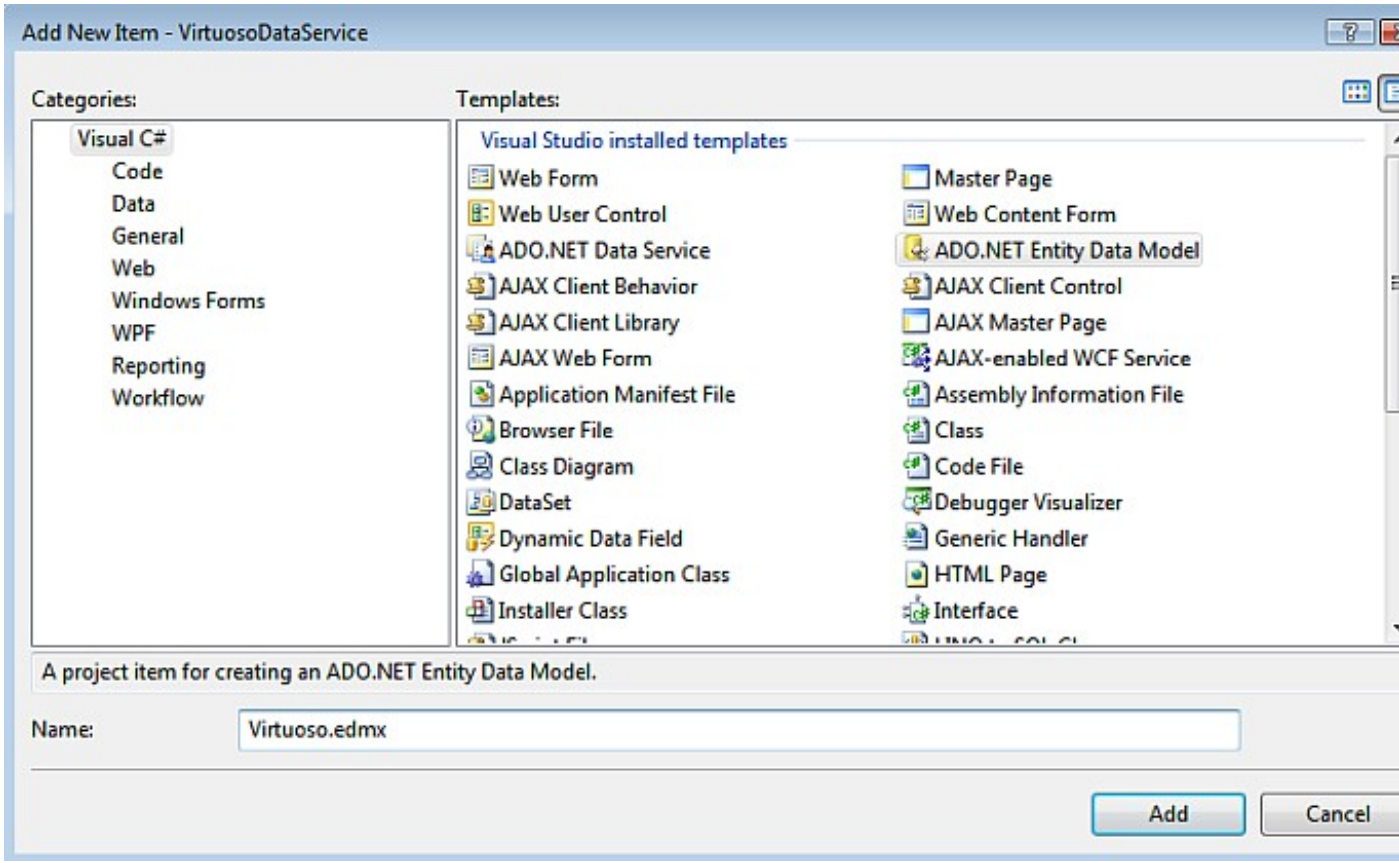
Virtuoso.edmx

and click

Add

to start the creation of the ADO.Net Entity Data Model.

Figure 8.510. Add New Item



9. In the

Entity Data Model Wizard

dialog

Choose Model Contents

page select the

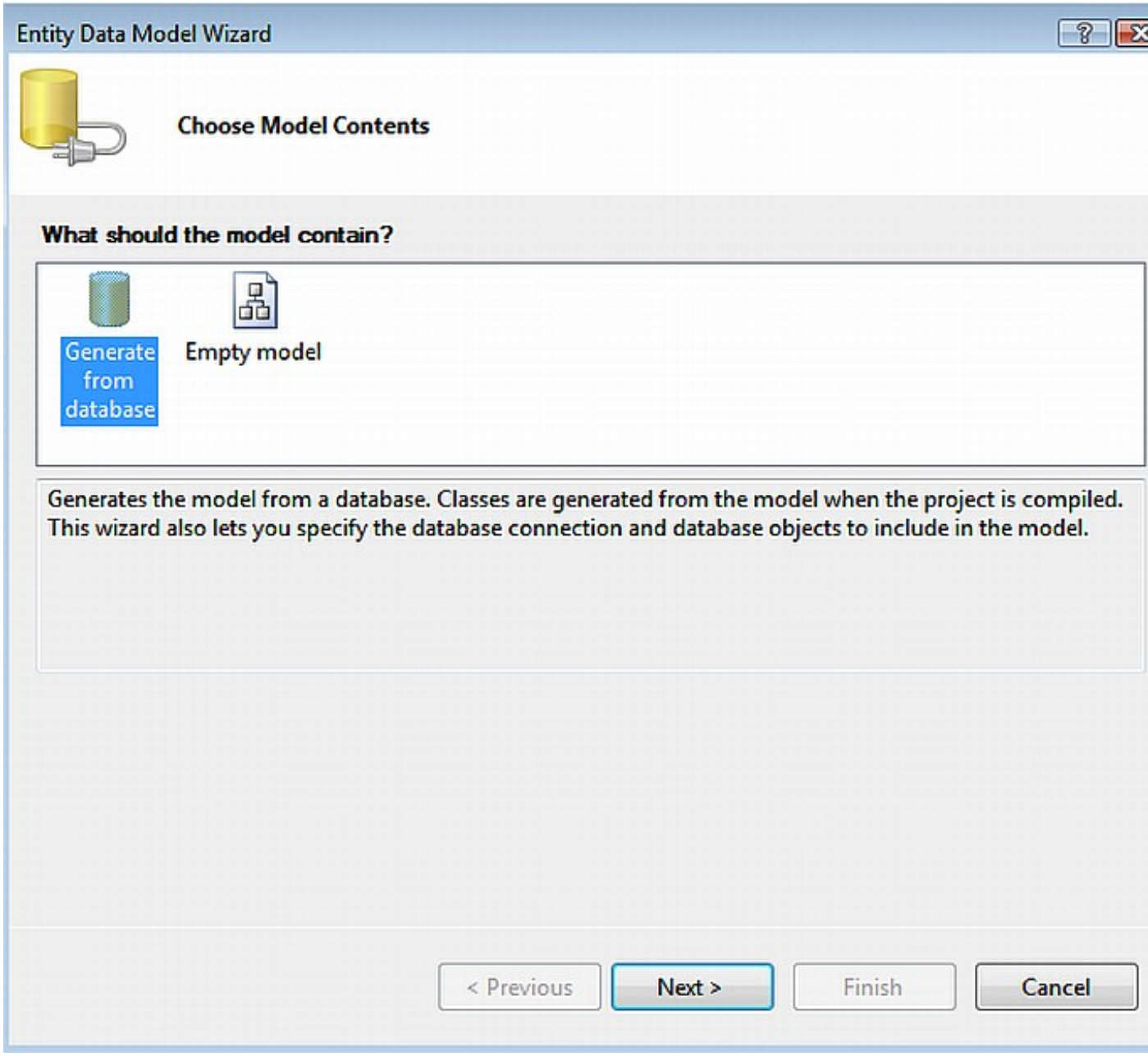
Generate from Database

model type and click

Next

.

Figure 8.511. Choose Model Contents



10. In the

    Entity Data Model Wizard

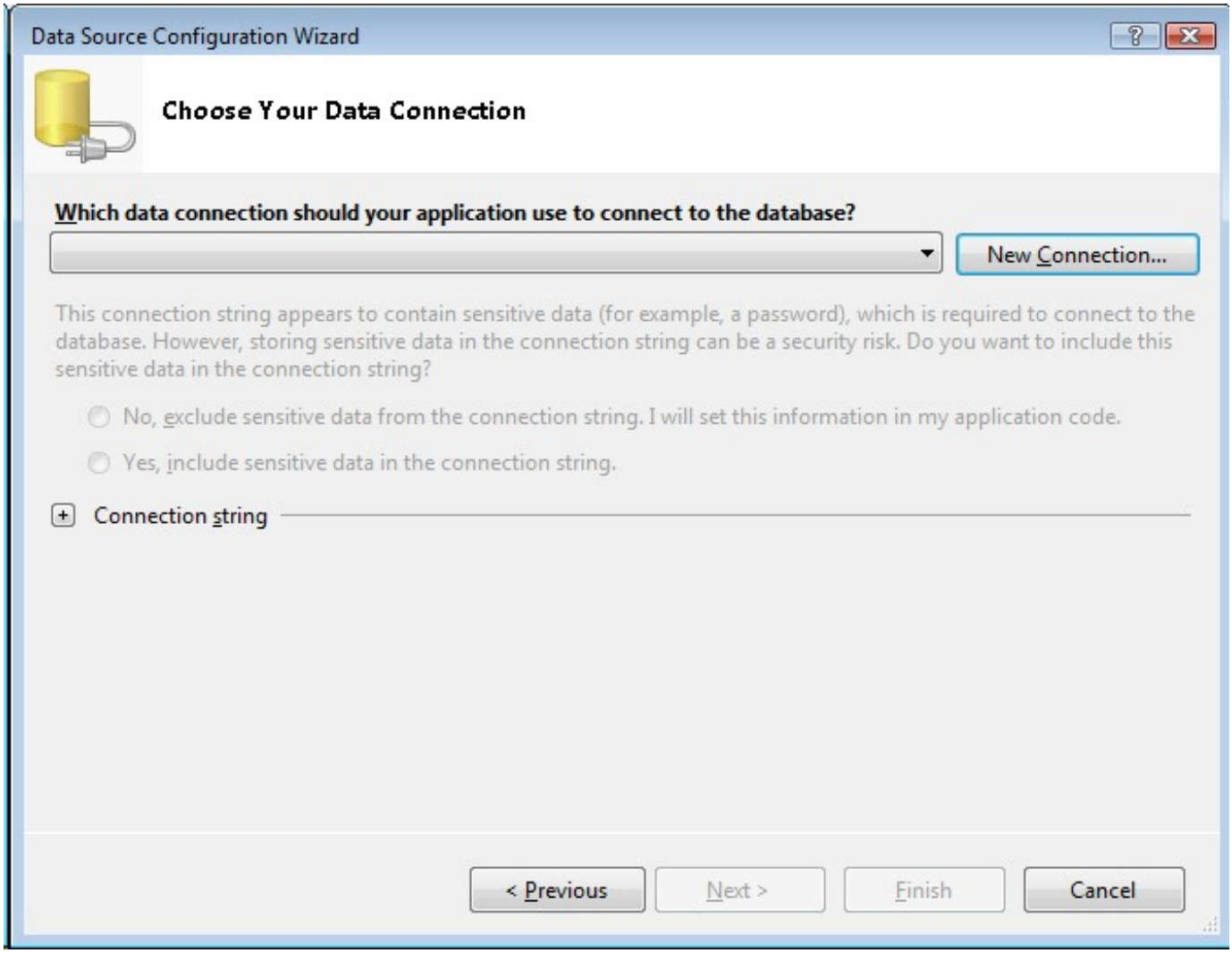
    dialog

    Choose your Data Connection

    page, select the

    New Connection button

Figure 8.512. Entity Data Model Wizard



11. In the

    Choose Data Source

    dialog, select the OpenLink

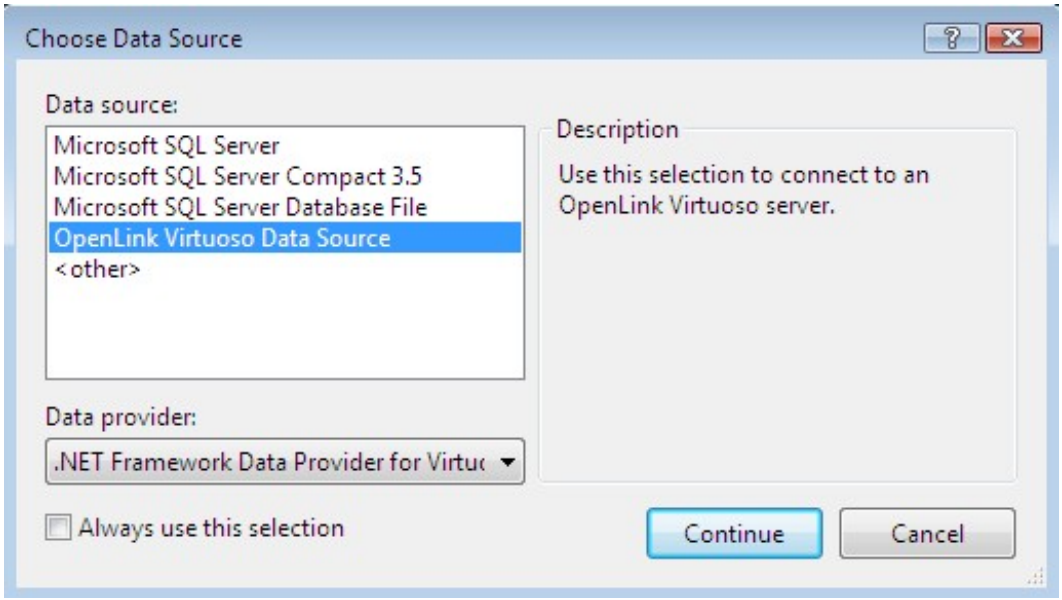
    Virtuoso Data Source

    from the list and click

    Continue

    .

Figure 8.513. Choose Data Source



12. In the

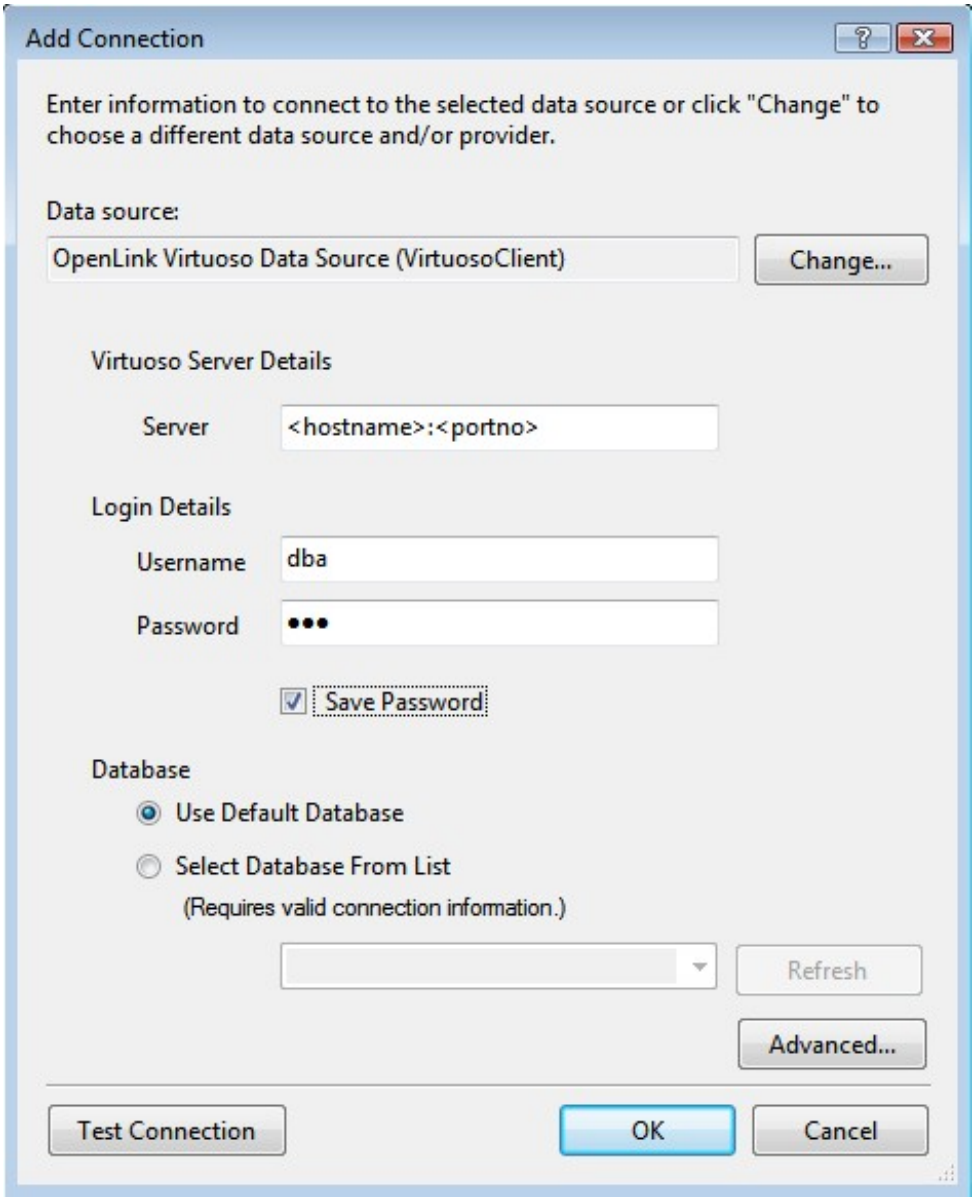
    Add Connection

    dialog, specify the

    hostname, portno, username and password

    for the target Virtuoso Server and check the Save Password check box.

    Figure 8.514. Connection Properties



13. Select the

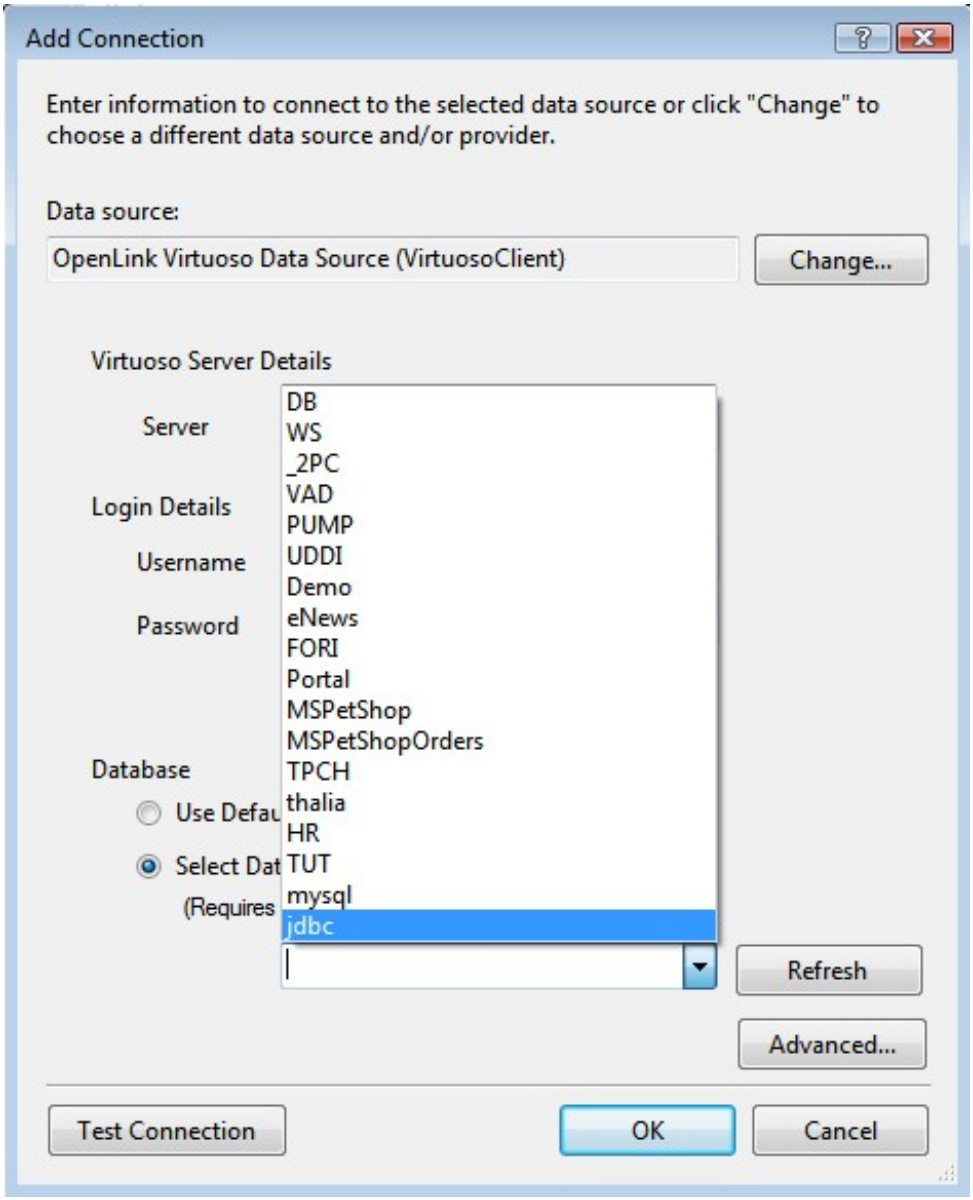
    Select Database From List

    radio button and choose the

    JDBC

    database from the drop down list.

Figure 8.515. Add connection

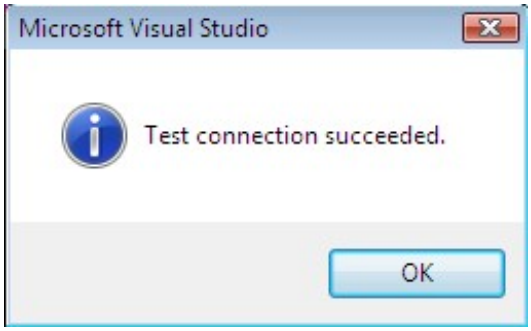


14. Press the

    Test Connection

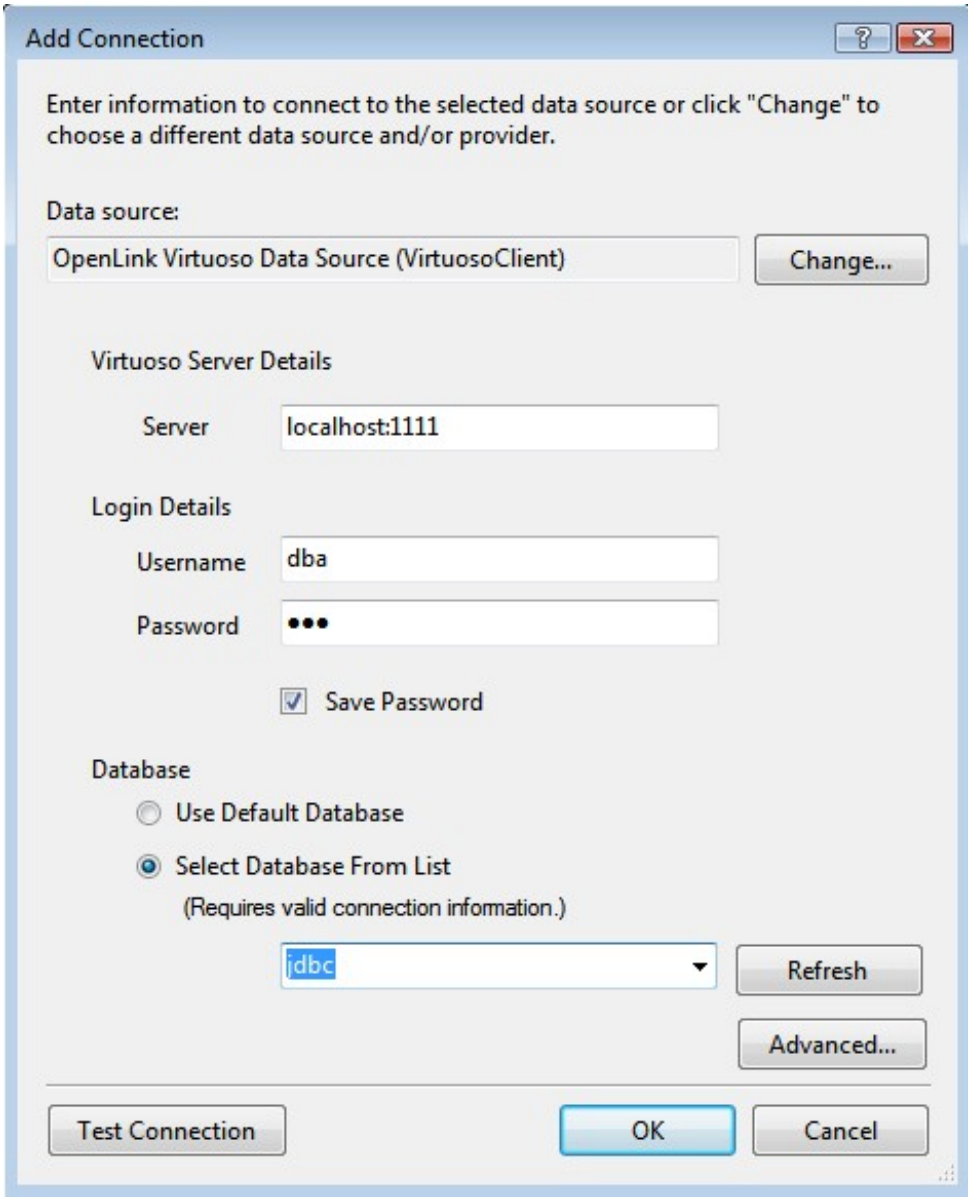
    dialog to verify that the database is accessible.

    Figure 8.516. Test Connection



15. Click OK to add the connection.

    Figure 8.517. Test Connection



16. Set the

    entity connect string

    name to

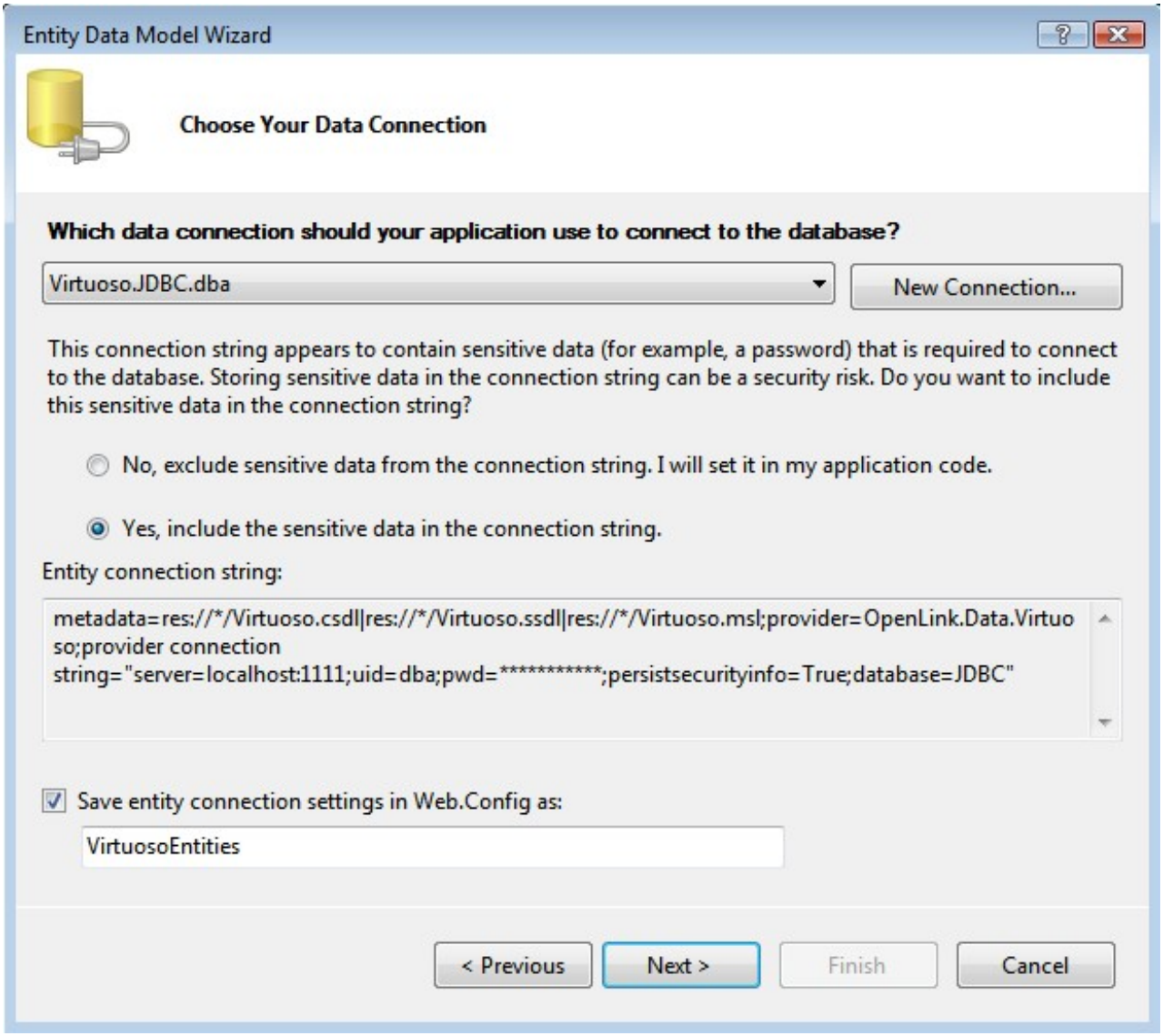
    VirtuosoEntities

    (note this name as it is required in the section on creating and ADO.Net Data Service) and click

    Next

    .

    Figure 8.518. entity connect string



17. In the

    Choose your Database Objects

    page select the

    Tables

    check box to select all tables in the JDBC catalog for addition to the Entity Data Model. Set the

    Model Namespace

    to

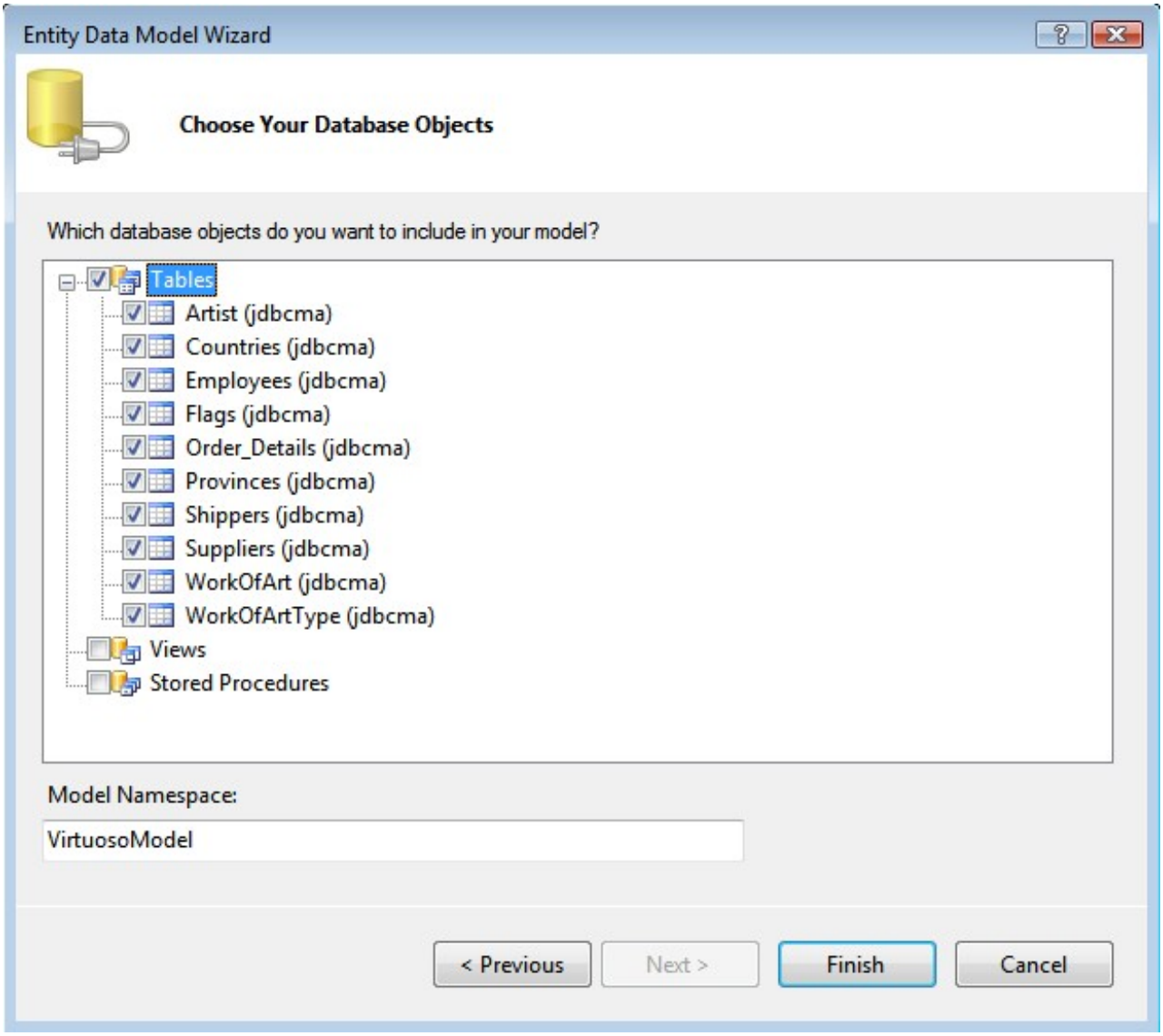
    VirtuosoModel

    and click

    Finish

    .

Figure 8.519. Database Objects

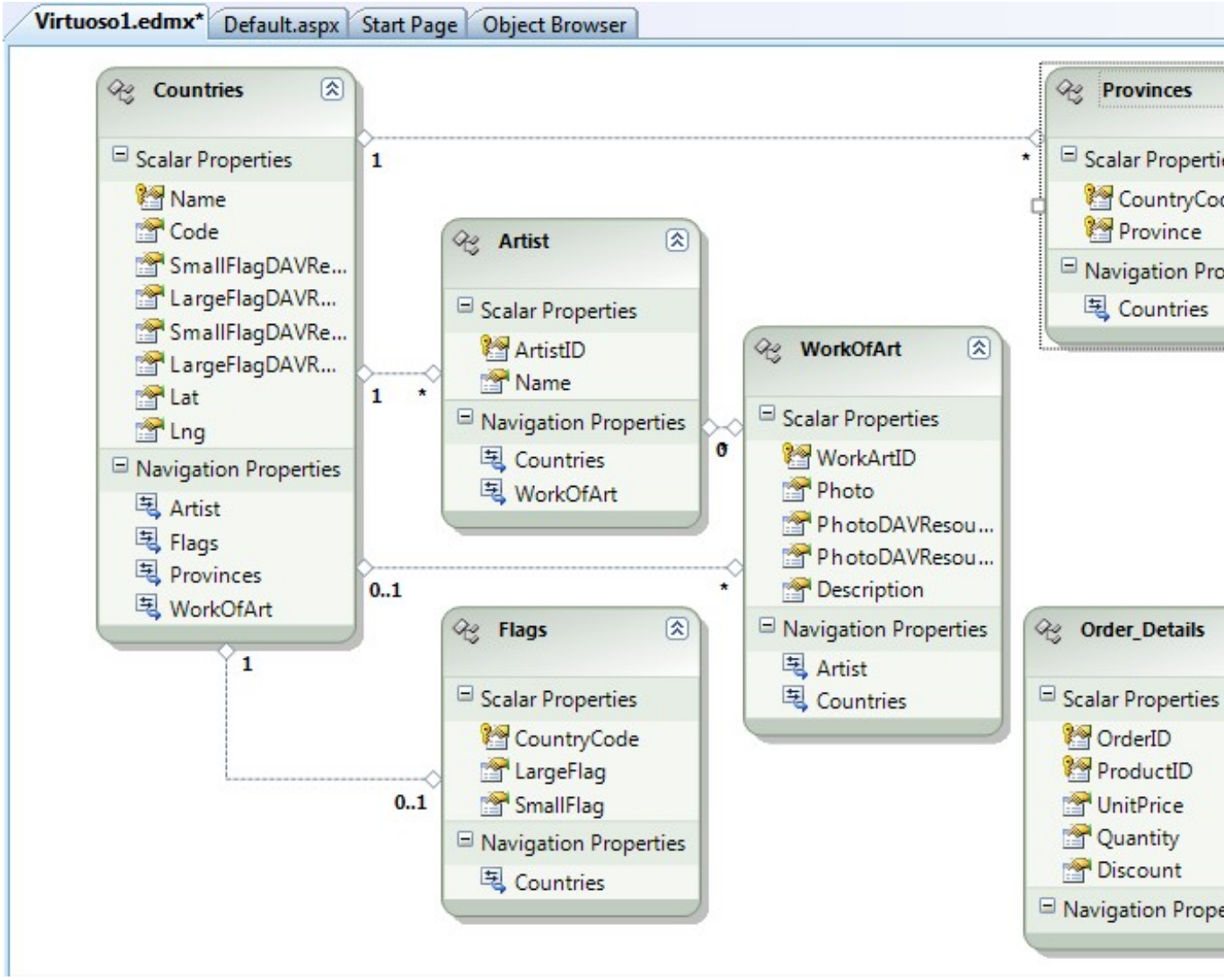


18. The

    Virtuoso.edmx

    EDM will be created with the tables and relationships displayed in the Visual Studio IDE

    Figure 8.520. Virtuoso.edmx



Creation for the Entity Data Model for the ODBC to JDBC Bridge Northwind database is now complete.

8.10.5. Using EDM to create Entity Framework based applications

Now that a Microsoft Entity Data Model has been created for the Northwind database, Entity Framework applications can be created to make use of it.

Entity Frameworks based ADO.NET Data Service

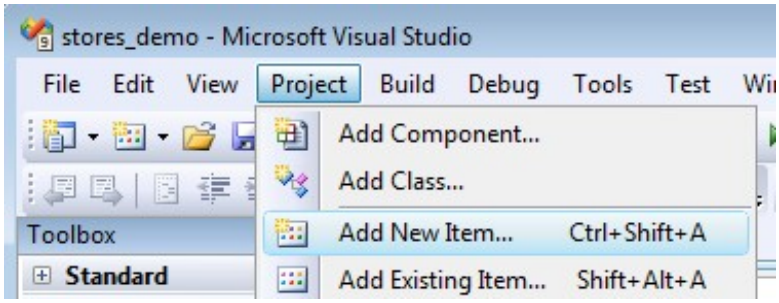
An ADO.Net Data Service for the ODBC to JDBC Bridge tables can be created using the Entity Data Model created in the Creating EDM in Visual Studio 2008 section .

1. Open the

   VirtuosoDataService

   project created in the Creating EDM in Visual Studio 2008 section .
2. Select the Project -> Add New Item menu option.

Figure 8.521. VirtuosoDataService



3. The

   Add New Item

   dialog will appear. Choose the

   ADO.NET Data Service template

   . Give it the name

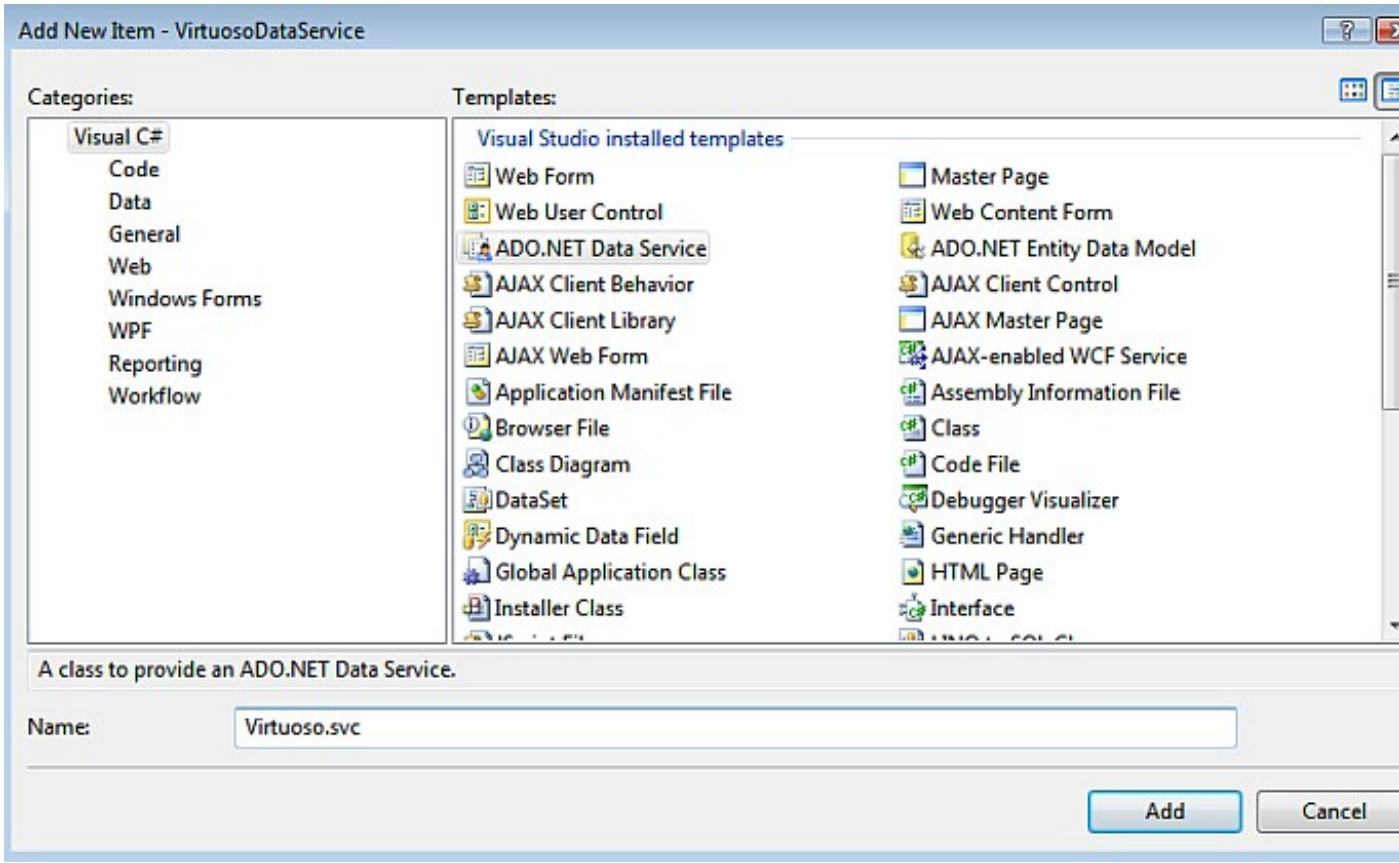
   Virtuoso.svc

   , and click

   Add

   to create the ADO.Net Data Service.

   Figure 8.522. Add New Item



4. In the newly created

   Virtuoso.svc.cs

Data Service file, add the data source class name of

VirtuosoEntities

(note this is the name set in the Creating EDM in Visual Studio 2008 section) as the

DataService

name. Enable the access to the Data Service by adding the entry

config.SetEntitySetAccessRule("", EntitySetRights.All);*

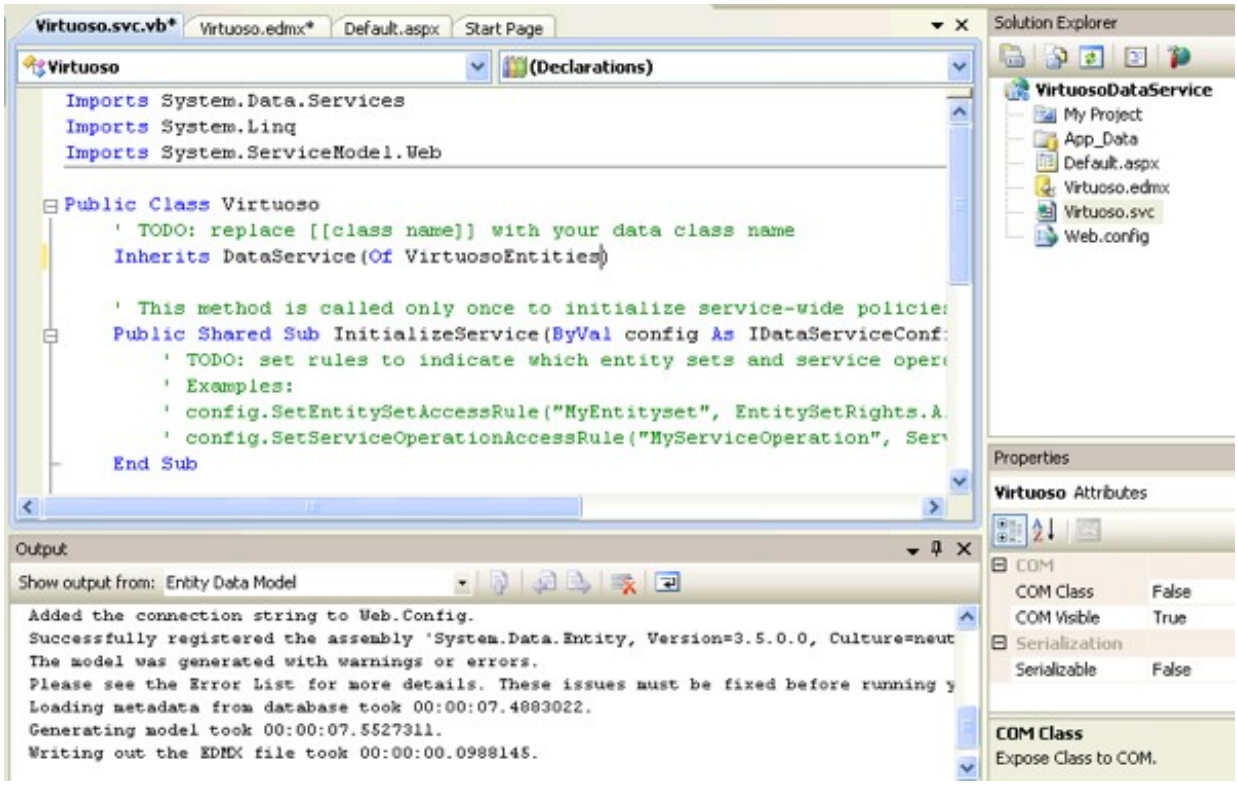
in the

InitializeService

method.

```
// C#

using System;
using System.Web;
using System.Collections.Generic;
using System.ServiceModel.Web;
using System.Linq;
using System.Data.Services;

namespace SimpleDataService
{
    public class Northwind: DataService<VirtuosoDemoEntities>
    {
        public static void InitializeService(IDataServiceConfiguration  config)
        {
            config.SetEntitySetAccessRule("*", EntitySetRights.All);
        }
    }
}
```

Figure 8.523. Virtuoso.svc.cs

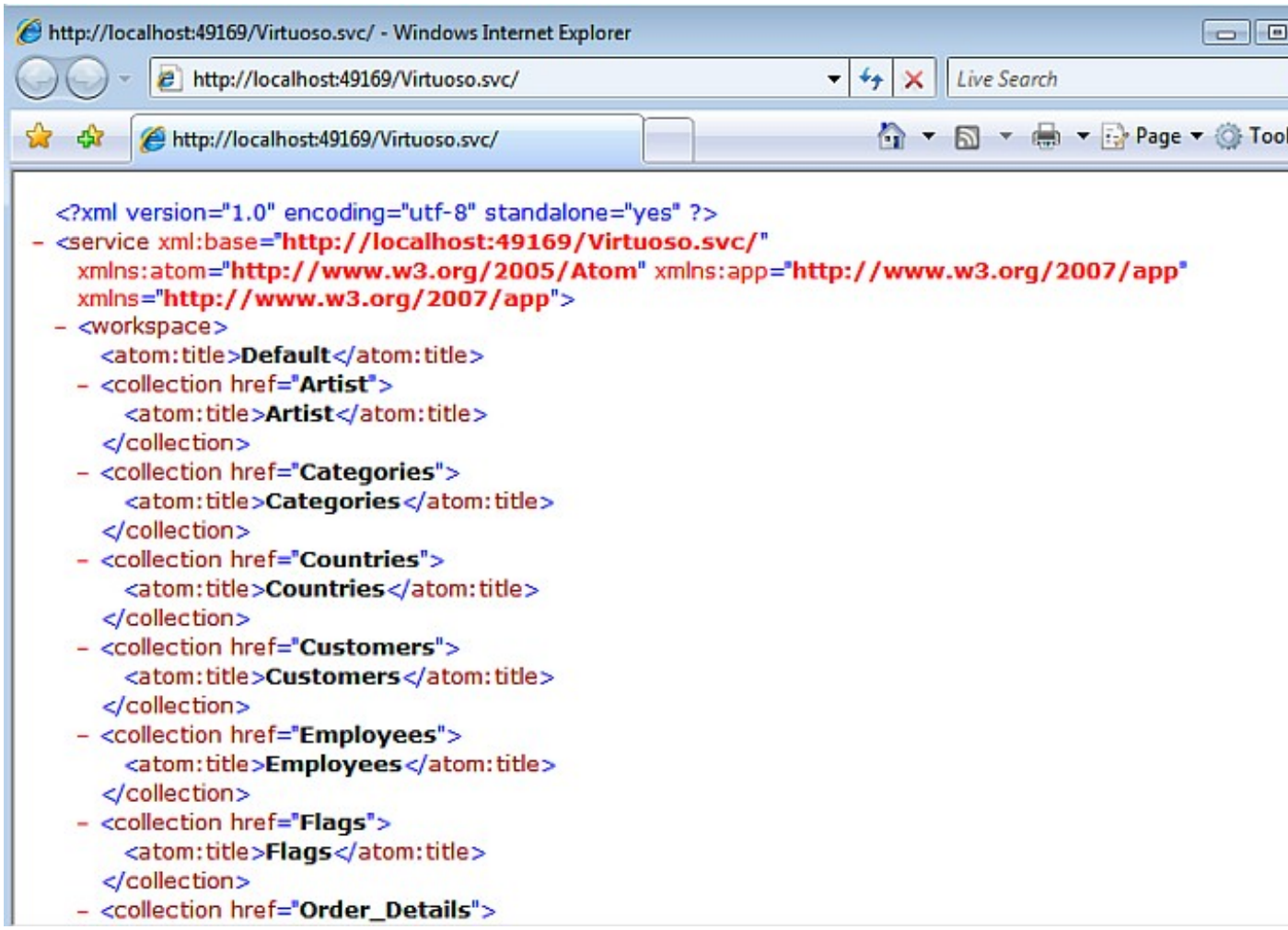


5. To test the Data Service, simply hit

   Ctrl+F5

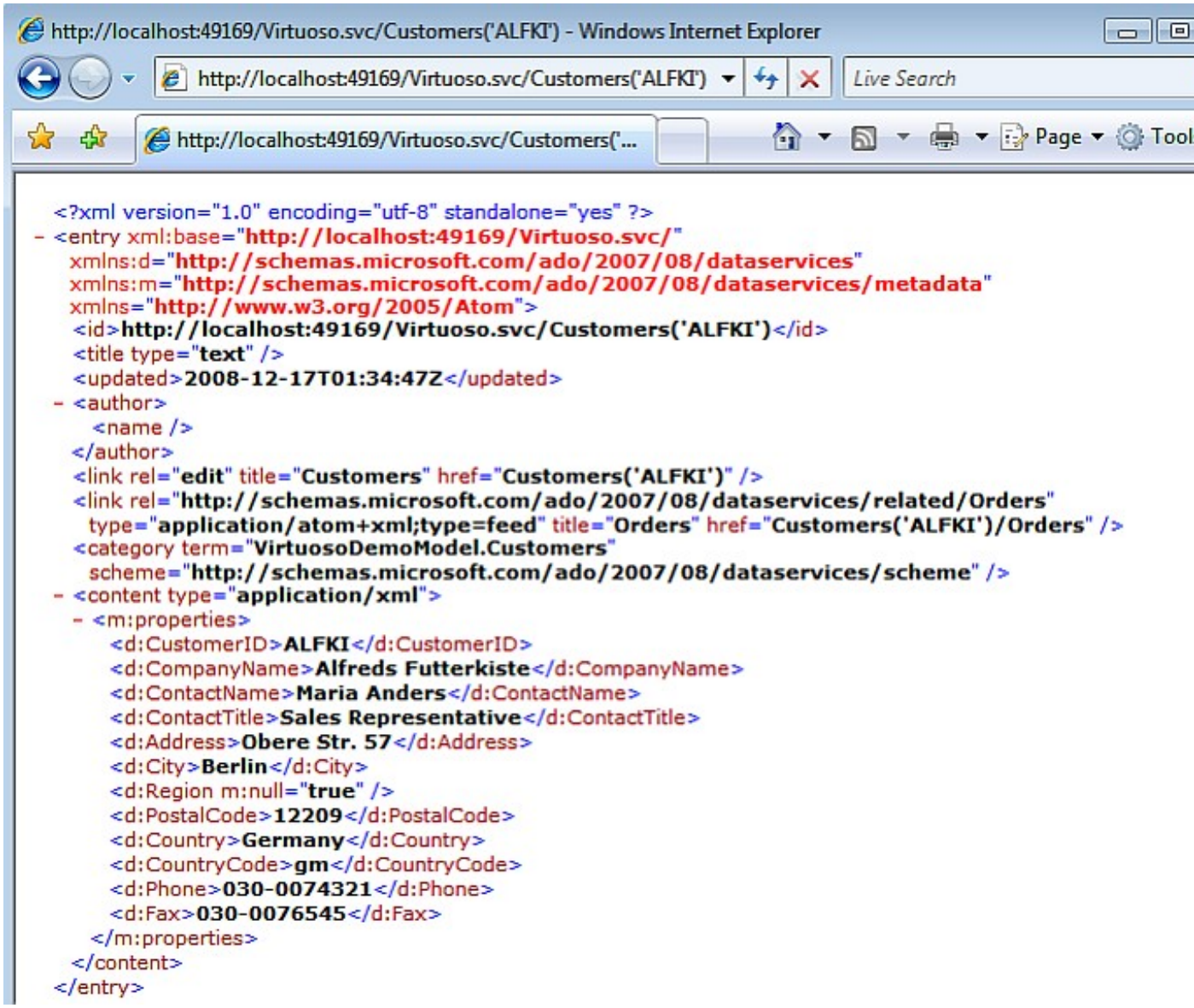
   within Visual Studio. This will start the development web server, run the Data Services server inside and load a Web browser page displaying the list of available tables/entities for the Northwind Demo database catalog.

   Figure 8.524. Data Service test



6. To access a specific entity instance like the Customers table ALFKI record, use this convention http://host/vdir/Virtuoso.svc/Customers('ALFKI') .

   Figure 8.525. Customers



Notes:

1. *Important*

   - To view

   Atom

   (the default format returned by an ADO.NET Data Service) in Internet Explorer, you must first ensure that

   Feed Reading View

   is turned

   off

   . This can be done on the

   Content tab

   of

Tools in Internet Options

.

2. If a Data Services entity instance URI page fails to load you can turn

   Verbose

   errors on by adding

   config.UseVerboseErrors = true;

   in the

   virtuoso.svc.cs InitializeService

   method to obtain more detailed information from the server as to why the page failed to load:

```
public static void InitializeService(IDataServiceConfiguration config)
{
config.UseVerboseErrors = true;
config.SetEntitySetAccessRule("*", EntitySetRights.All);
}
```

Visual Studio Windows DataGrid Form Application

This section details the steps required to create a simple Visual Studio 2008 Windows Form application, with associated DataGridView control for displaying data in selected tables from the target database.

1. Launch the Visual Studio 2008 SP1 IDE.

   Figure 8.526. Visual Studio 2008 SP1 IDE

2. Create a

   Web Application

   project by going to the

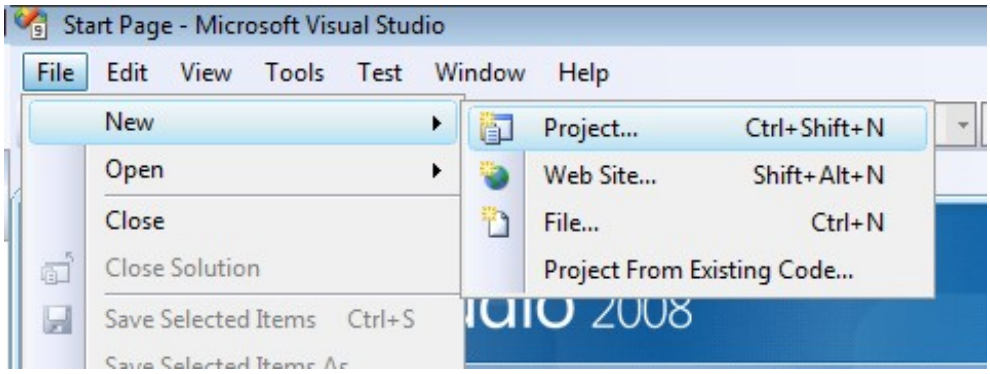
   File

   menu in Visual Studio and choosing

New Project

.

Figure 8.527. Web Application



3. When the New Project window appears, choose either

   Visual Basic

   or

   Visual C#

   as your programming language.
4. Within the language category, click on

   Windows

   and select

   Windows Form Application

   from the right-hand panel.
5. Choose a name for the project, for example

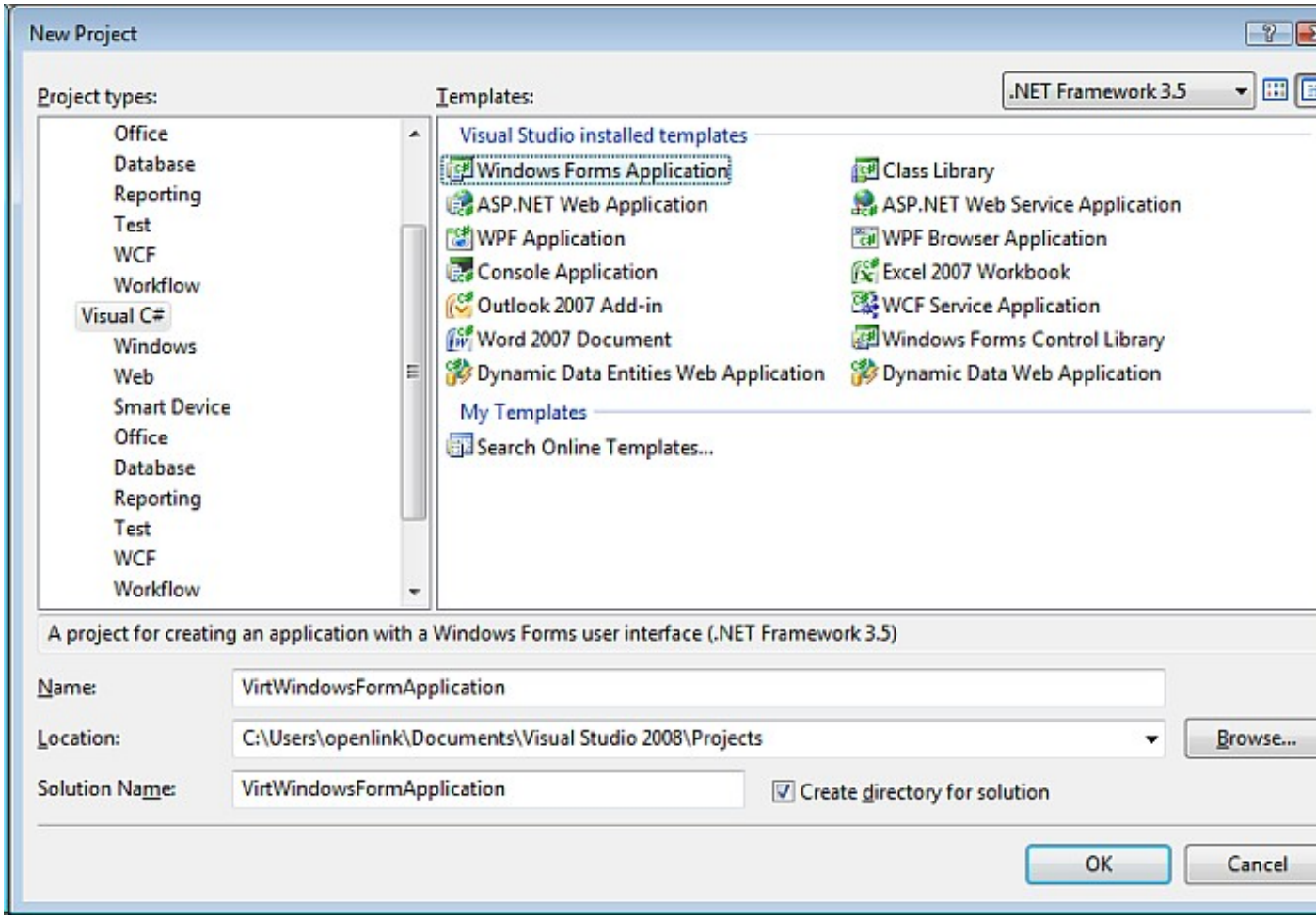
   VirtWindowsFormApplication

   , and click

   OK

   .

Figure 8.528. Web Application



6. In the

   Toolbox

   , expand

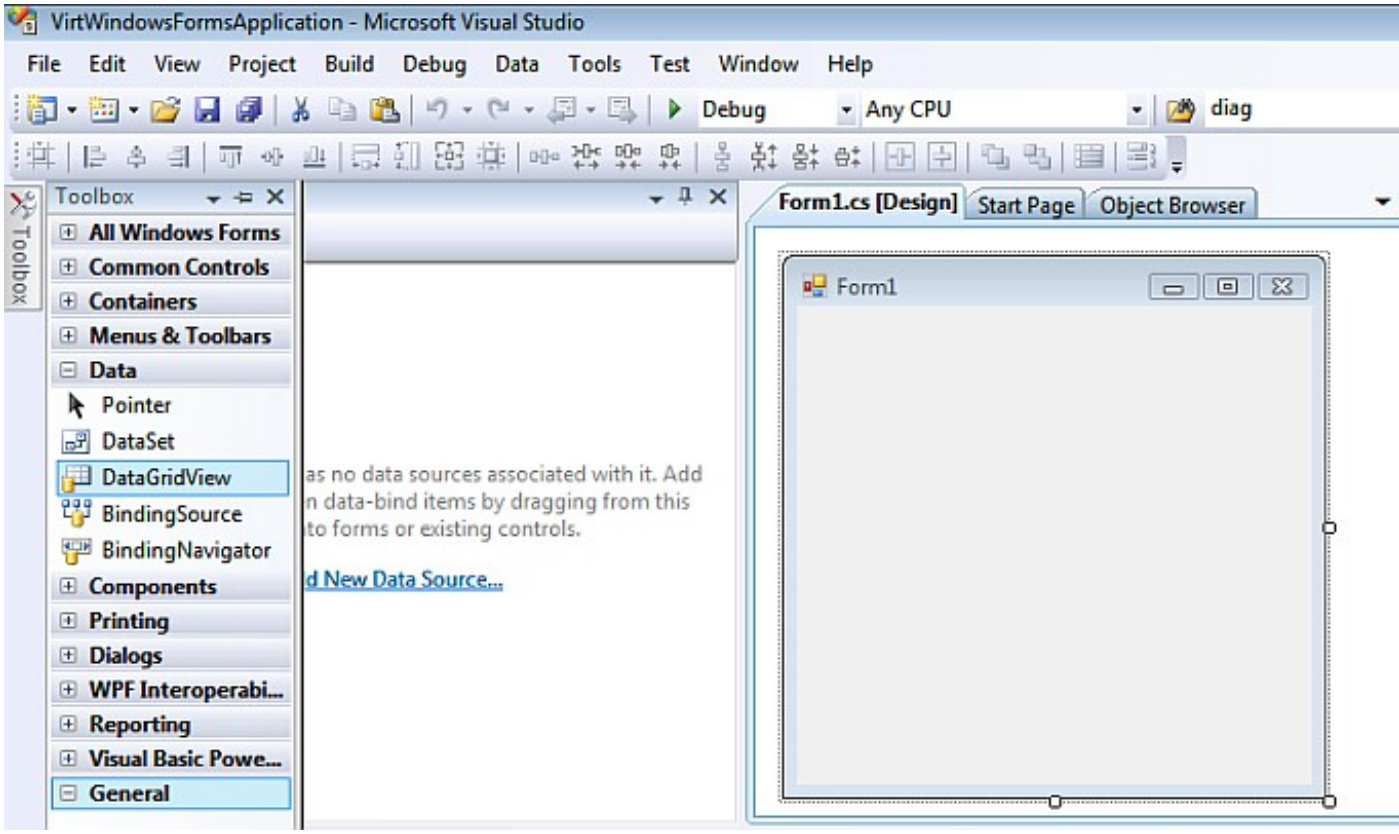
   Data Controls

   , and drag the

   DataGridView

   control onto the form.

   Figure 8.529. Toolbox



7. Click on the little

   arrow

   in the top right of the

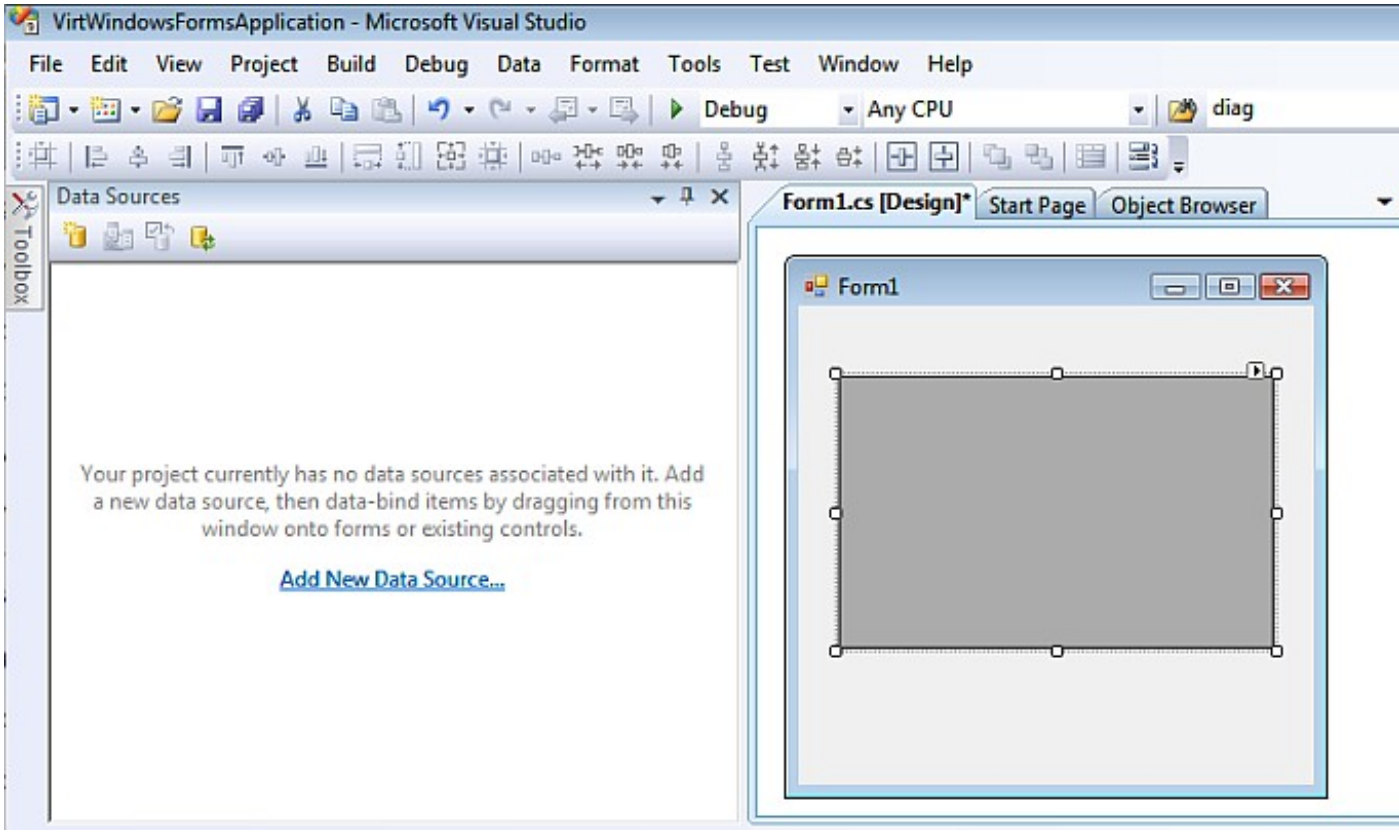
   DataGridView

   control. This loads the

   DataGridView Task

   menu.

Figure 8.530. DataGridView Task

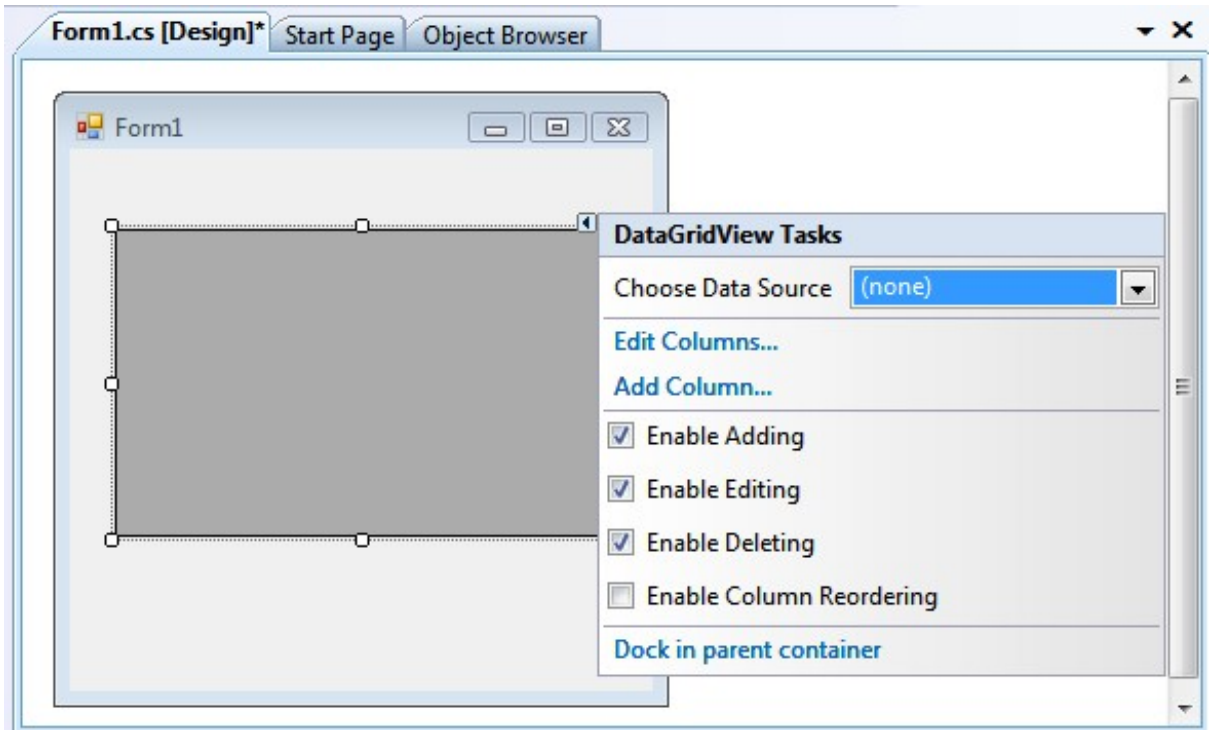


8. Click on the

   Choose Data Source

   list box.

   Figure 8.531. Choose Data Source

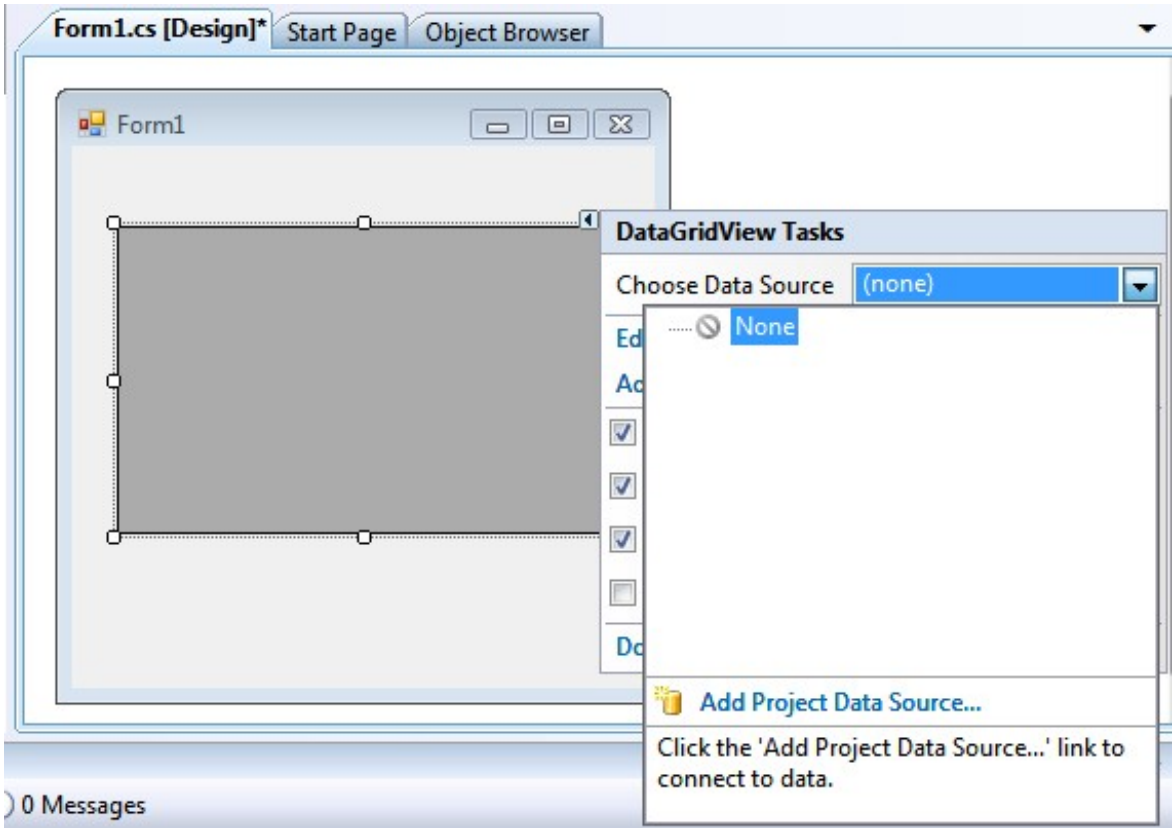


9. Click on the

   Add Project Data Source

link to connect to a data source.

Figure 8.532. Add Project Data Source



10. In the

    Data Source Configuration Wizard

    dialog

    Choose Data Source Type

    page select the

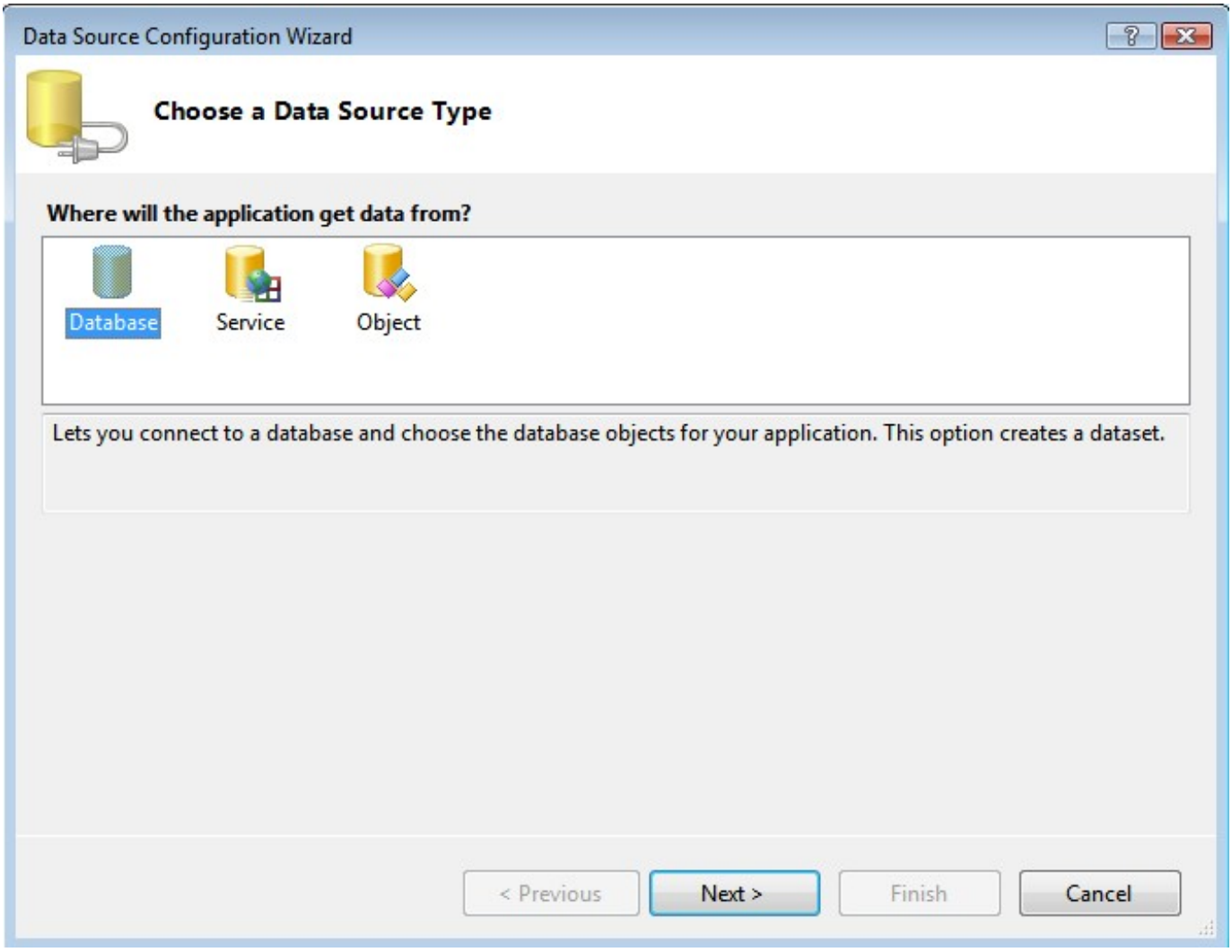
    Database

    data source type and click

    Next

    .

Figure 8.533. Data Source Type



11. In the

    Data Source Configuration Wizard

    dialog

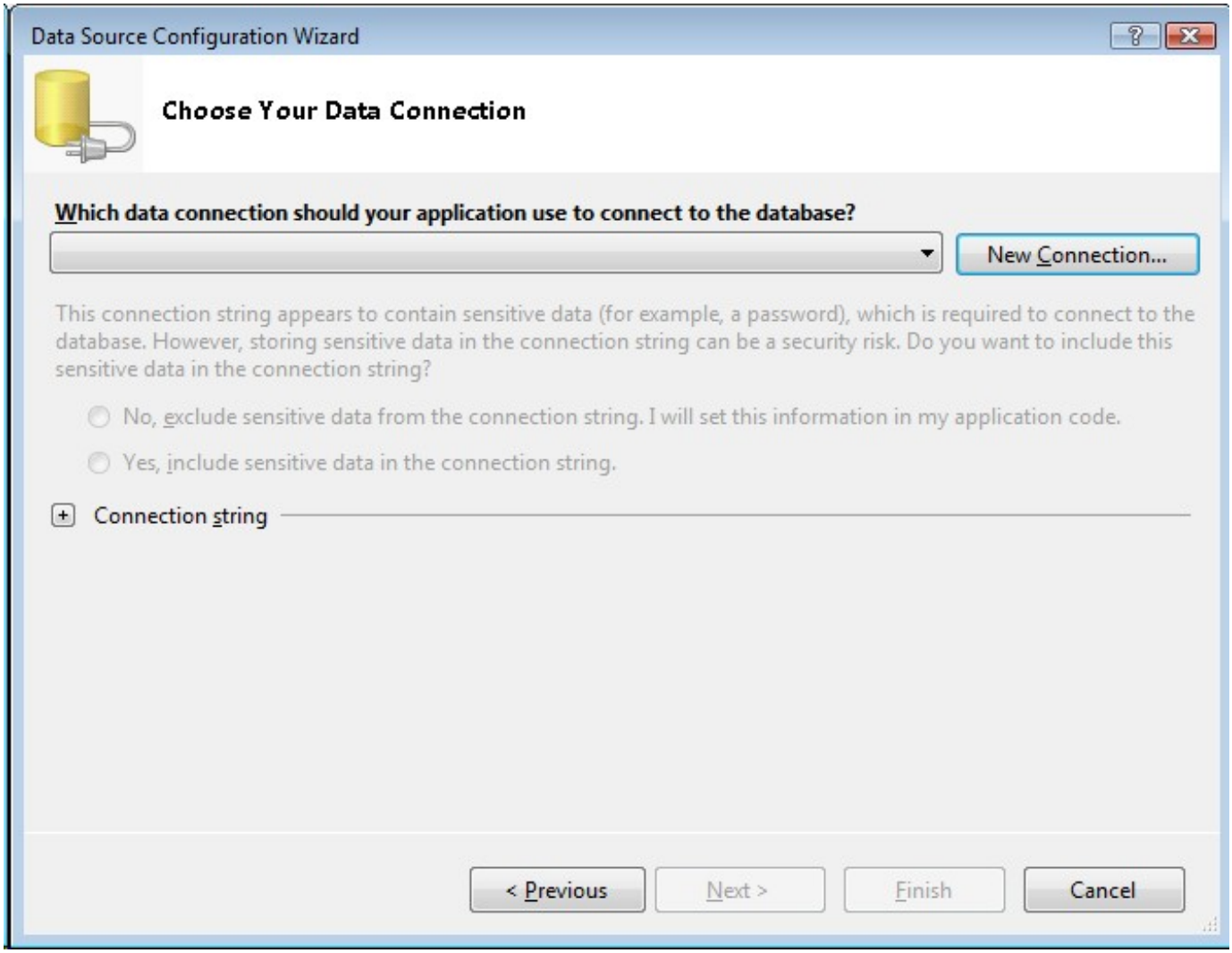
    Choose your Data Connection

    page, select the

    New Connection

    button

Figure 8.534. Data Source Configuration Wizard



12. In the

    Choose Data Source

    dialog, select the OpenLink

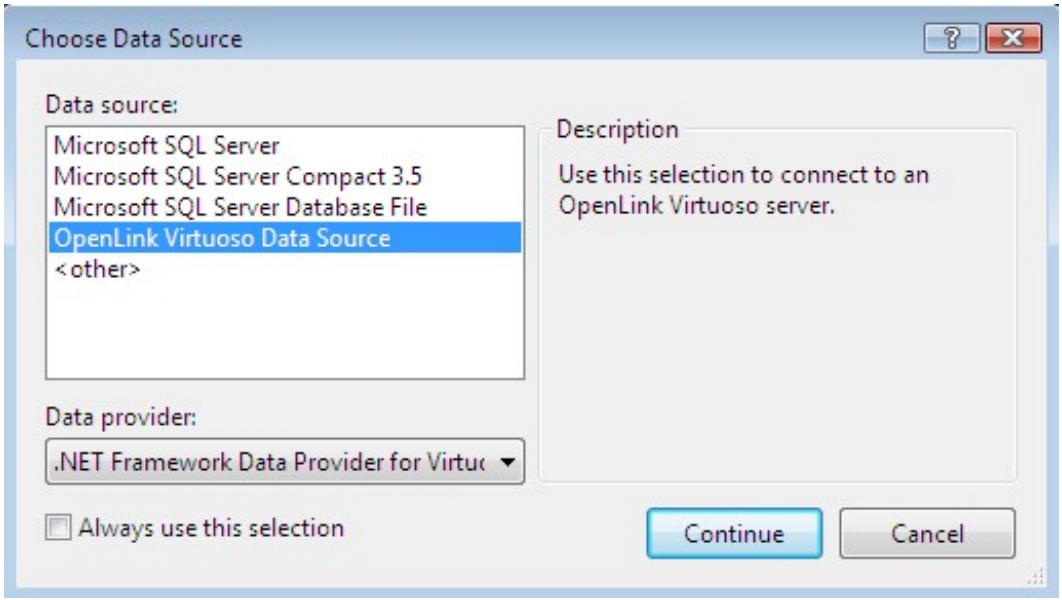
    Virtuoso Data Source

    from the list and click

    Continue

    .

Figure 8.535. Data Source



13. In the

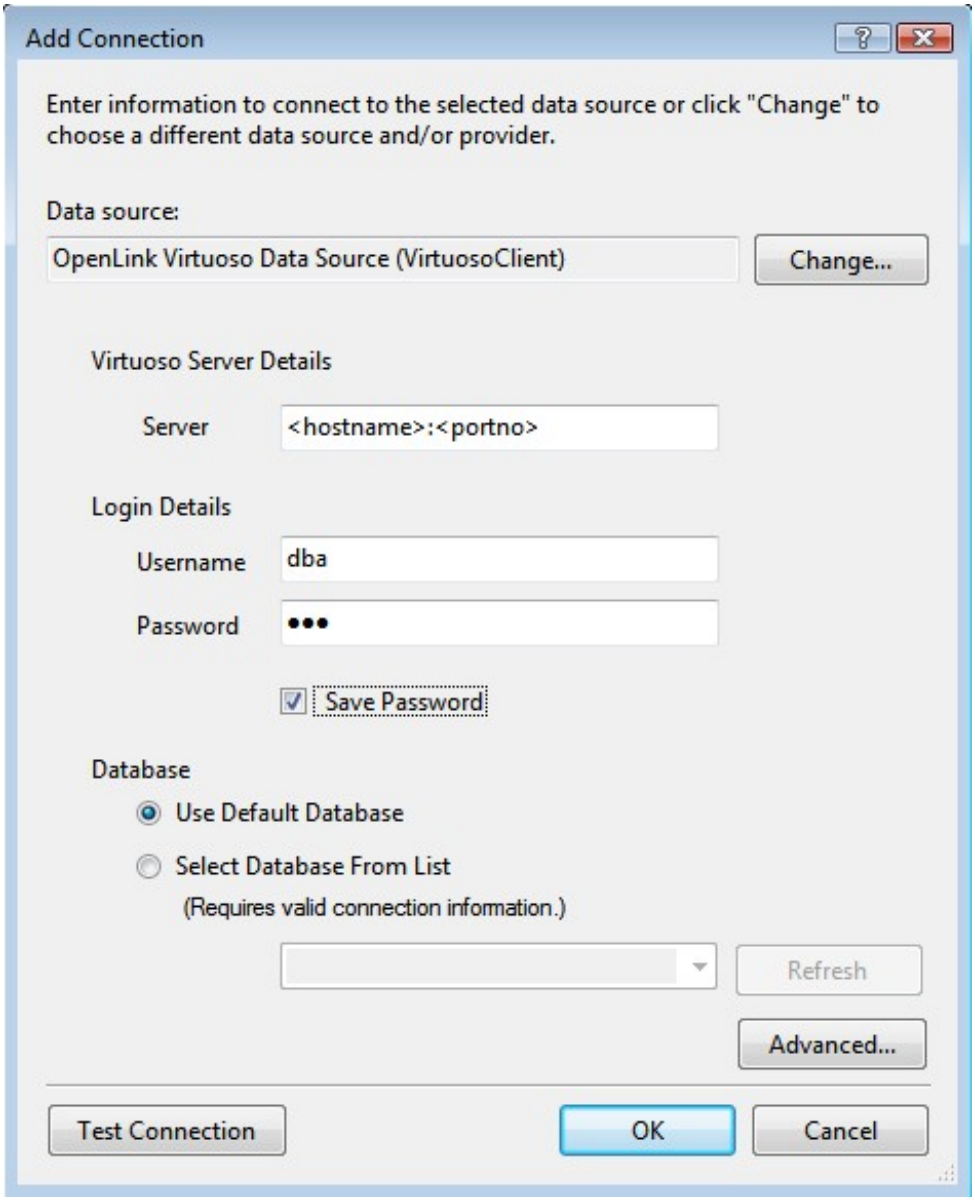
    Add Connection

    dialog, specify the

    hostname, portno, username and password

    for the target Virtuoso Server and check the Save Password check box.

    Figure 8.536. Connection Properties



14. Select the

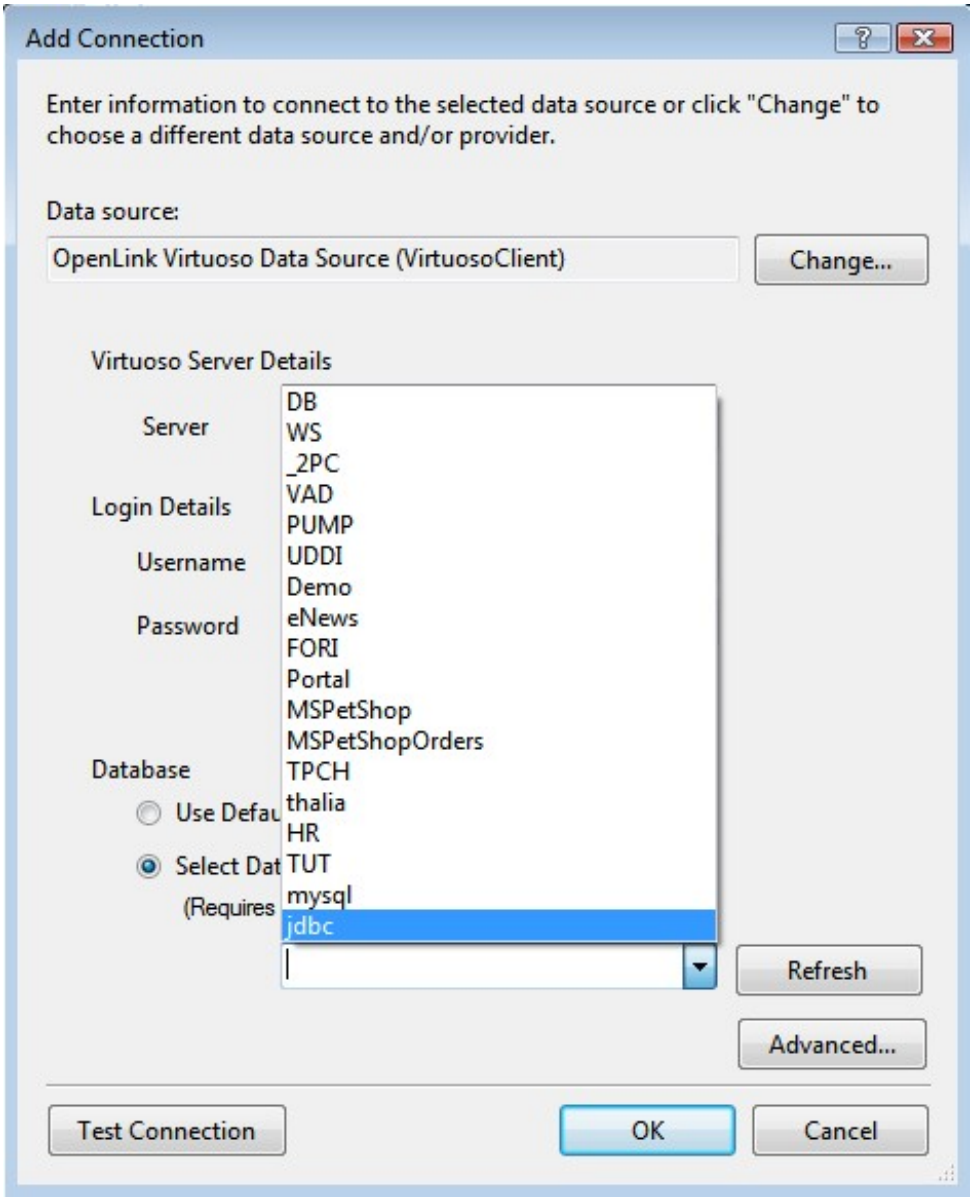
    Select Database From List

    radio button and choose the

    jdbc

    database from the drop down list.

    Figure 8.537. Add connection

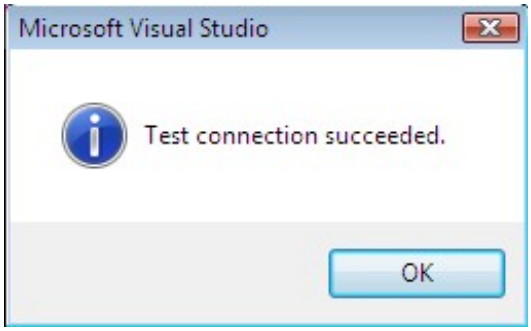


15. Press the

    Test Connection

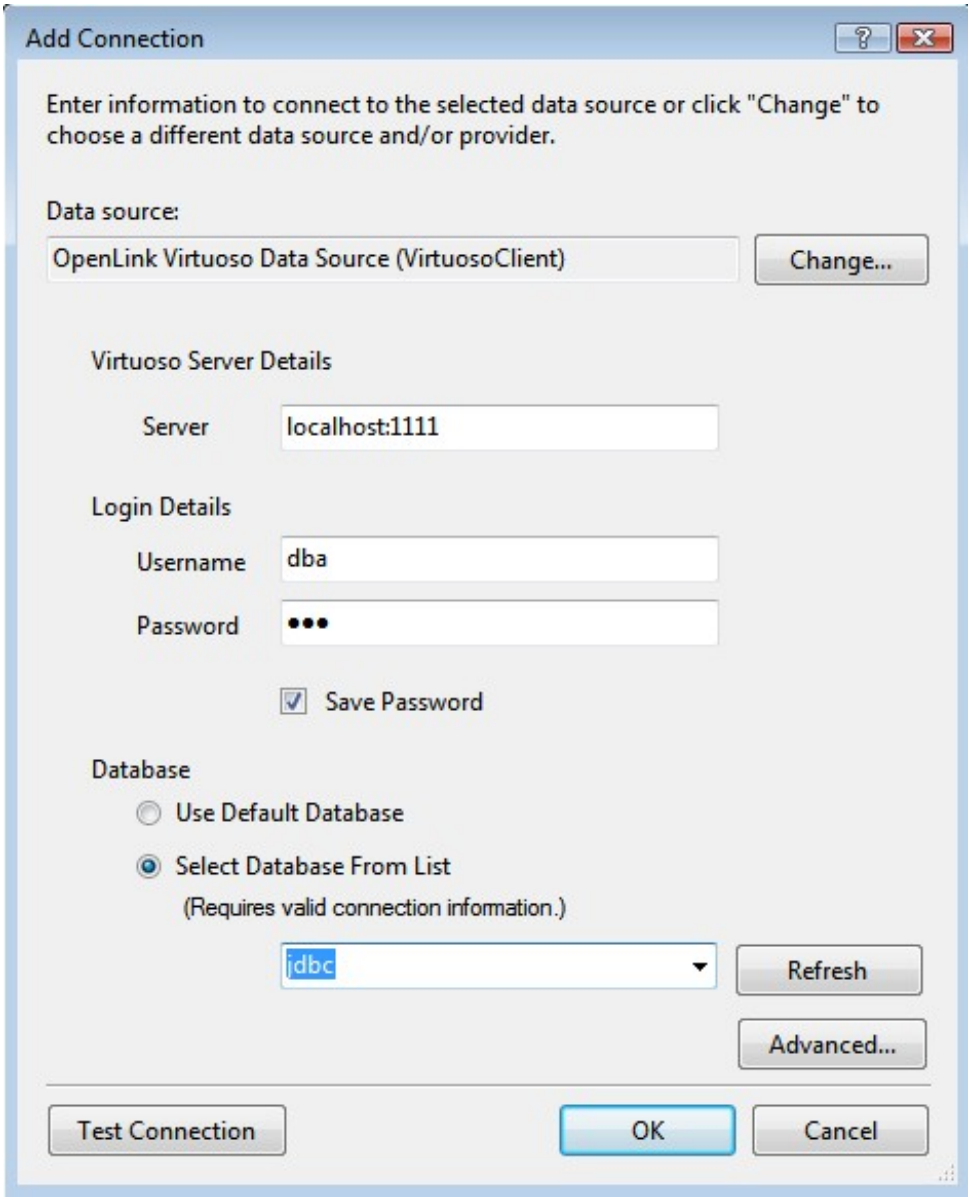
    dialog to verify that the database is accessible.

Figure 8.538. Test Connection



16. Click OK to add the connection.

Figure 8.539. Test Connection



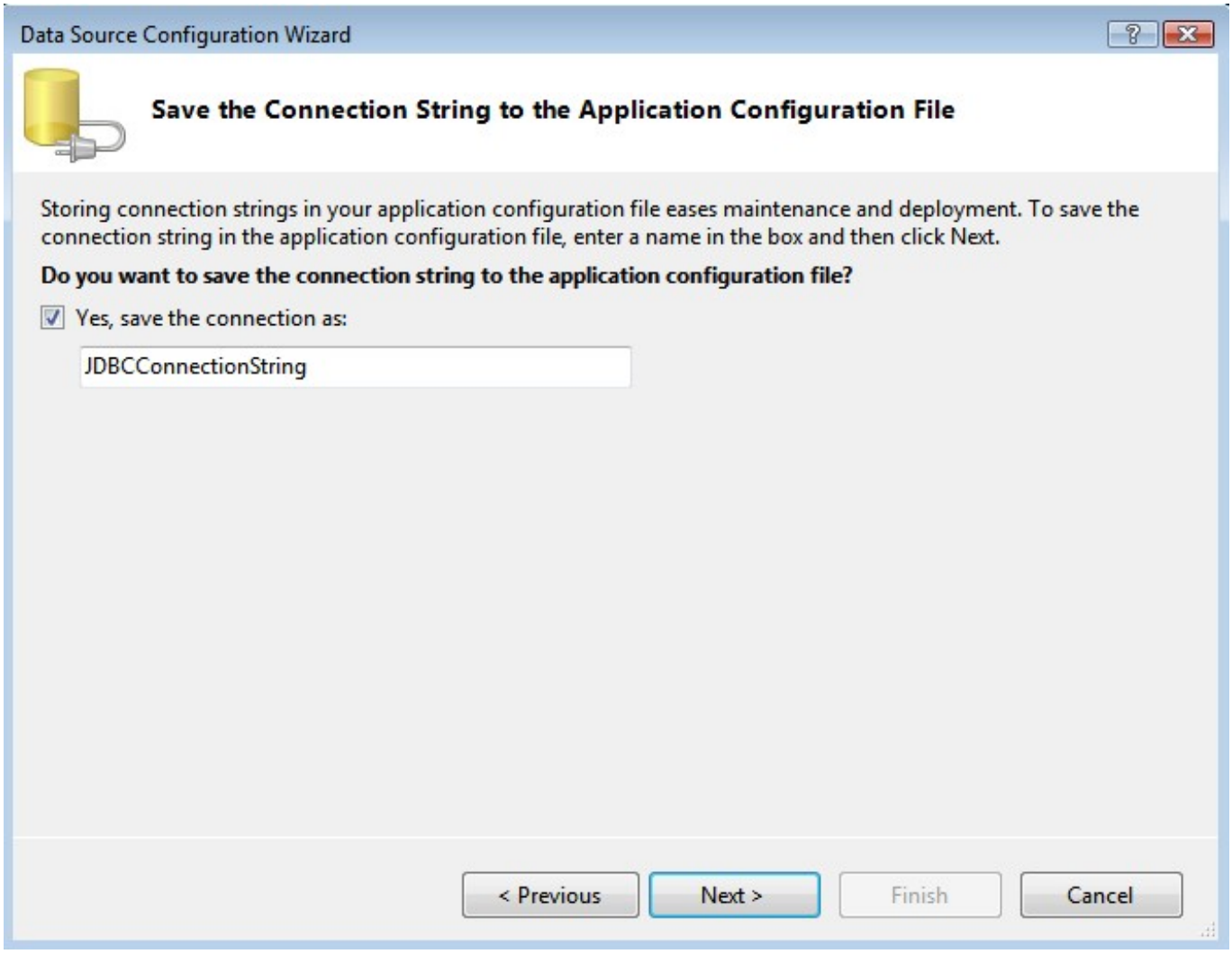
17. Leave the default connect string

    JDBCConnectionString

    and click

    Next

    Figure 8.540. JDBCConnectionString



18. From the list of available tables returned for the JDBC database, select the

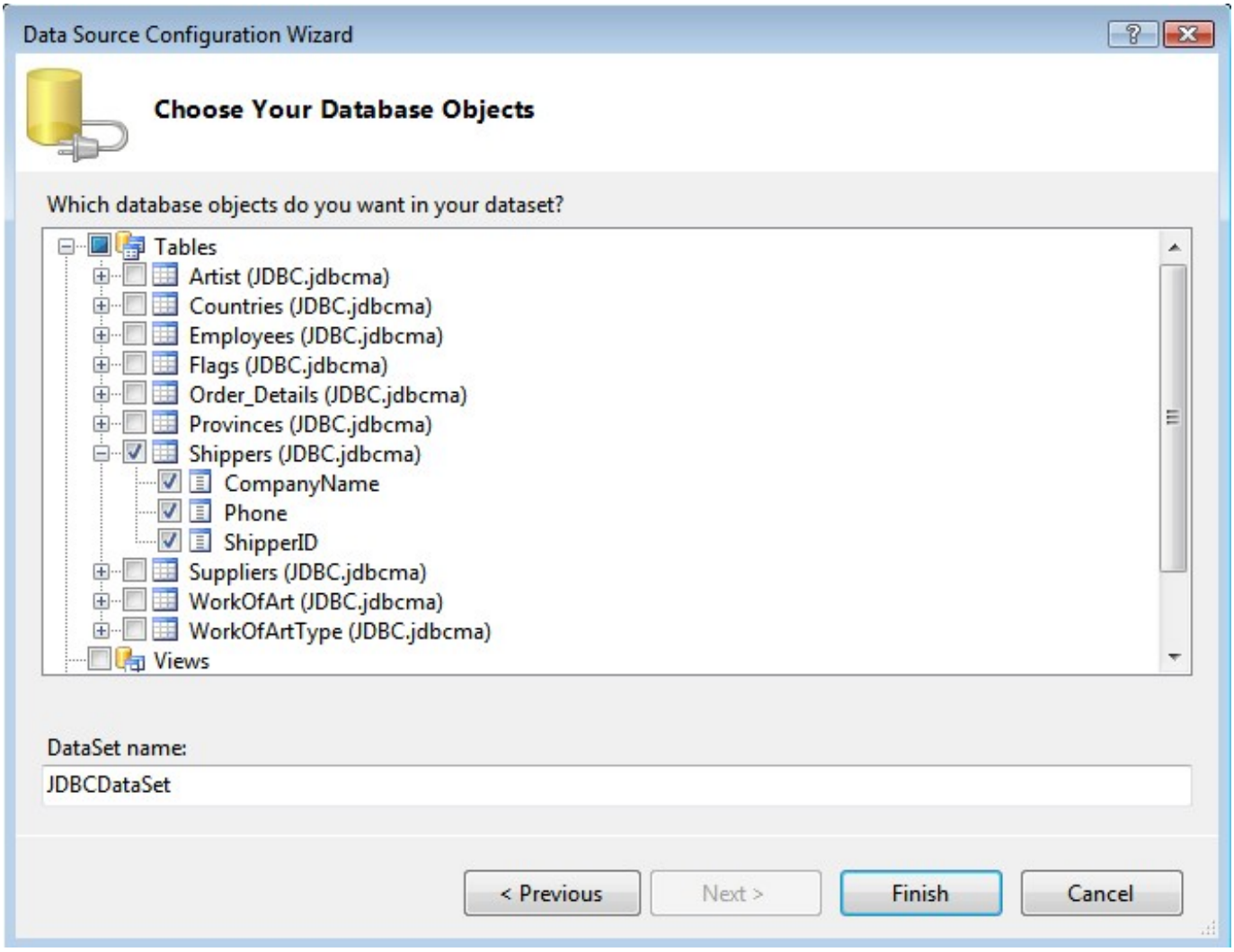
    Shippers

    table to be associated with the

    DataGridView

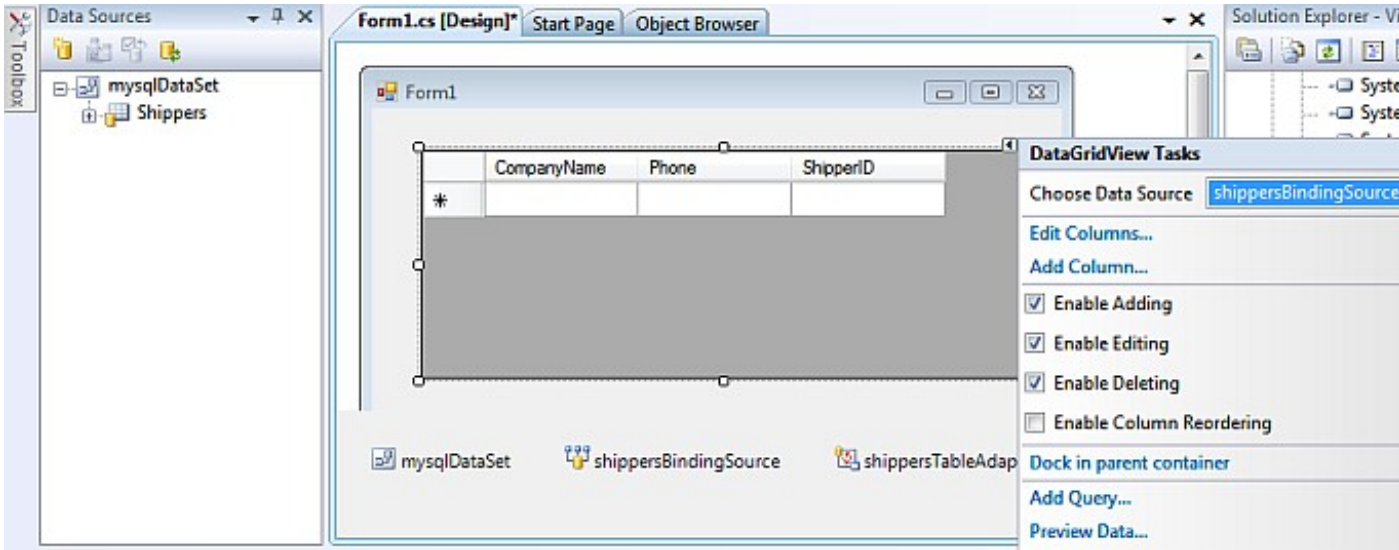
    control.

    Figure 8.541. JDBC database



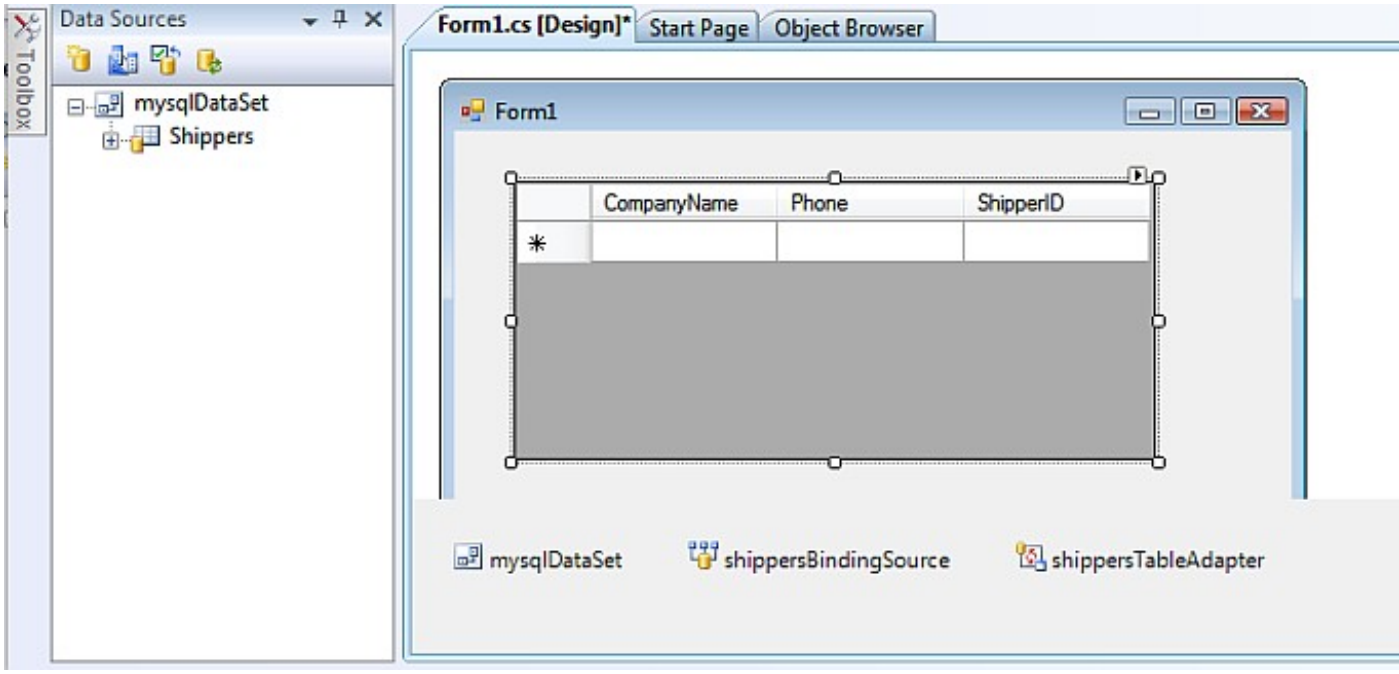
19. The columns names of the select table will be displayed in the DataGridView.

Figure 8.542. DataGridView



20. Resize the Form and DataGridView to allow all columns to be visible, if possible.

Figure 8.543. Resize the Form and DataGridView



21. To test the application, simply hit

    Ctrl+F5

    within Visual Studio or select

    Start Debugging

    from the

    Debug

    menu.

    Figure 8.544. Start Debugging

22. The data from the

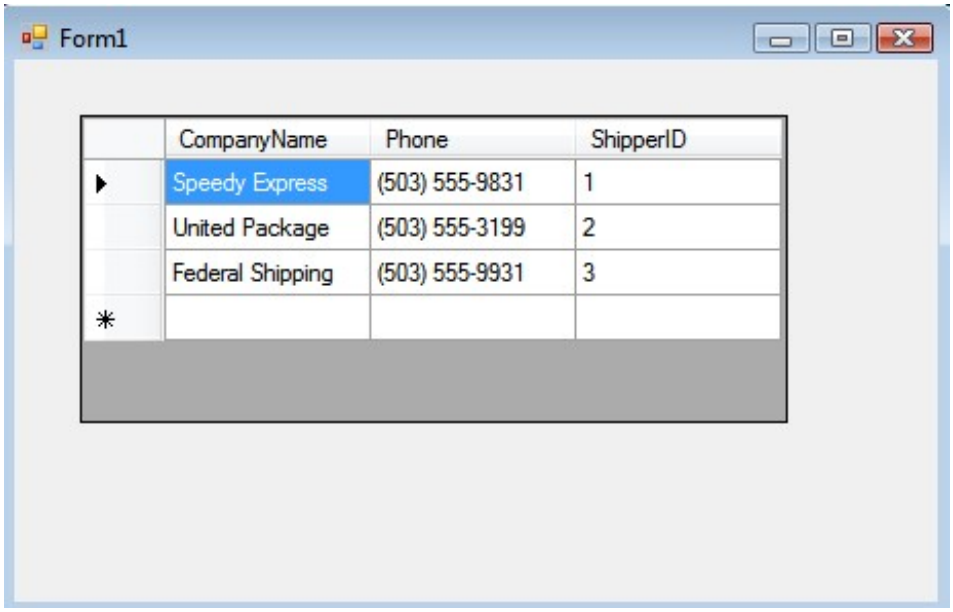
    Shippers

    table will be displayed in the

    DataGrid

    .

    Figure 8.545. DataGrid



The task is now complete.

8.11. Using Microsoft Entity Frameworks to Access ODBC to ODBC Bridge Schema Objects with Virtuoso

Abstract

This section details the steps required to provide Microsoft Entity Framework access to ODBC to ODBC Bridge Schema Objects using the OpenLink Virtuoso Universal Server. This is achieved by linking the required ODBC to ODBC Bridge Schema objects into Virtuoso using its built in Virtual Database engine, and then using the Virtuoso ADO.NET Entity Framework provider to query the remote ODBC to ODBC Bridge Schema objects linked into the Virtuoso Server.

Prerequisites. The following prerequisites must be in place for this solution to be possible.

ODBC to ODBC Bridge DBMS. An ODBC to ODBC Bridge DBMS hosting the required Schema Objects needs to be available. In this section we shall make and ODBC to ODBC bridge connection to another *Virtuoso Northwind Demo* database to demonstrate the process.

An ODBC to ODBC Bridge DBMS hosting the required Schema Objects needs to be available. In this section we shall make and ODBC to ODBC bridge connection to another *Virtuoso Northwind Demo* database to demonstrate the process.

ODBC to ODBC Bridge Driver. An ODBC to ODBC Bridge Driver is required for Linking the ODBC to ODBC Bridge Schema Objects into the Virtuoso Server. The *OpenLink ODBC to ODBC Bridge Driver* will be used in this section, for which a functional ODBC Data source name of "odbcma" will be assumed to exist on the machine hosting the Virtuoso Server.

An ODBC to ODBC Bridge Driver is required for Linking the ODBC to ODBC Bridge Schema Objects into the Virtuoso Server. The *OpenLink ODBC to ODBC Bridge Driver* will be used in this section, for which a functional ODBC Data source name of "odbcma" will be assumed to exist on the machine hosting the Virtuoso Server.

Virtuoso Universal Server. An Virtuoso installation including the Virtuoso Universal Server and ADO.NET Entity Framework Provider is required. The Virtuoso components used must be Release 5.10.x or above, this being the minimum version containing support for Microsoft Entity Frameworks

An Virtuoso installation including the Virtuoso Universal Server and ADO.NET Entity Framework Provider is required. The Virtuoso components used must be Release 5.10.x or above, this being the minimum version containing support for Microsoft Entity Frameworks

Microsoft Visual Studio 2008 SP1. Microsoft Visual Studio 2008 Service Pack 1 is required, this being the only version containing the necessary Entity Framework support available at the time of writing.

Microsoft Visual Studio 2008 Service Pack 1 is required, this being the only version containing the necessary Entity Framework support available at the time of writing.

Tasks

Ensure ODBC to ODBC Bridge Primary Keys (PKs) are not nullable. The Visual Studio 2008 Entity Data Model (EDM) requires that all primary keys are *NOT* Nullable, and will fail to generate an EDM if any are. Thus ensure any tables to be used are defined as not nullable in the ODBC to ODBC Bridge database schema before attempting to generate an EDM. In the case of the ODBC to ODBC Bridge Virtuoso Northwind Demo database all tables are not nullable, thus this should not be an issue in this case.

The Visual Studio 2008 Entity Data Model (EDM) requires that all primary keys are *NOT* Nullable, and will fail to generate an EDM if any are. Thus ensure any tables to be used are defined as not nullable in the ODBC to ODBC Bridge database schema before attempting to generate an EDM. In the case of the ODBC to ODBC Bridge Virtuoso Northwind Demo database all tables are not nullable, thus this should not be an issue in this case.

8.11.1. Install and configure OpenLink ODBC Driver for ODBC to ODBC Bridge

The Virtuoso Virtual Database engine uses ODBC as the connectivity mechanism for linking remote database objects into its local schema. Thus an ODBC to ODBC Bridge Driver must be available with a suitably configured DSN for connecting to the target database. The OpenLink ODBC to ODBC Bridge Drivers have been used in this document, although in theory any ODBC to ODBC Bridge Driver can be used.

Installation instructions for the OpenLink ODBC Driver for ODBC to ODBC Bridge are available from: Product Installation & Basic Configuration (ODBC)

Install and configure OpenLink Virtuoso Universal Server

1. Install and configure OpenLink Virtuoso Universal Server

8.11.2. Linking ODBC to ODBC Bridge tables into OpenLink Virtuoso

1. Start the Virtuoso Web User Interface

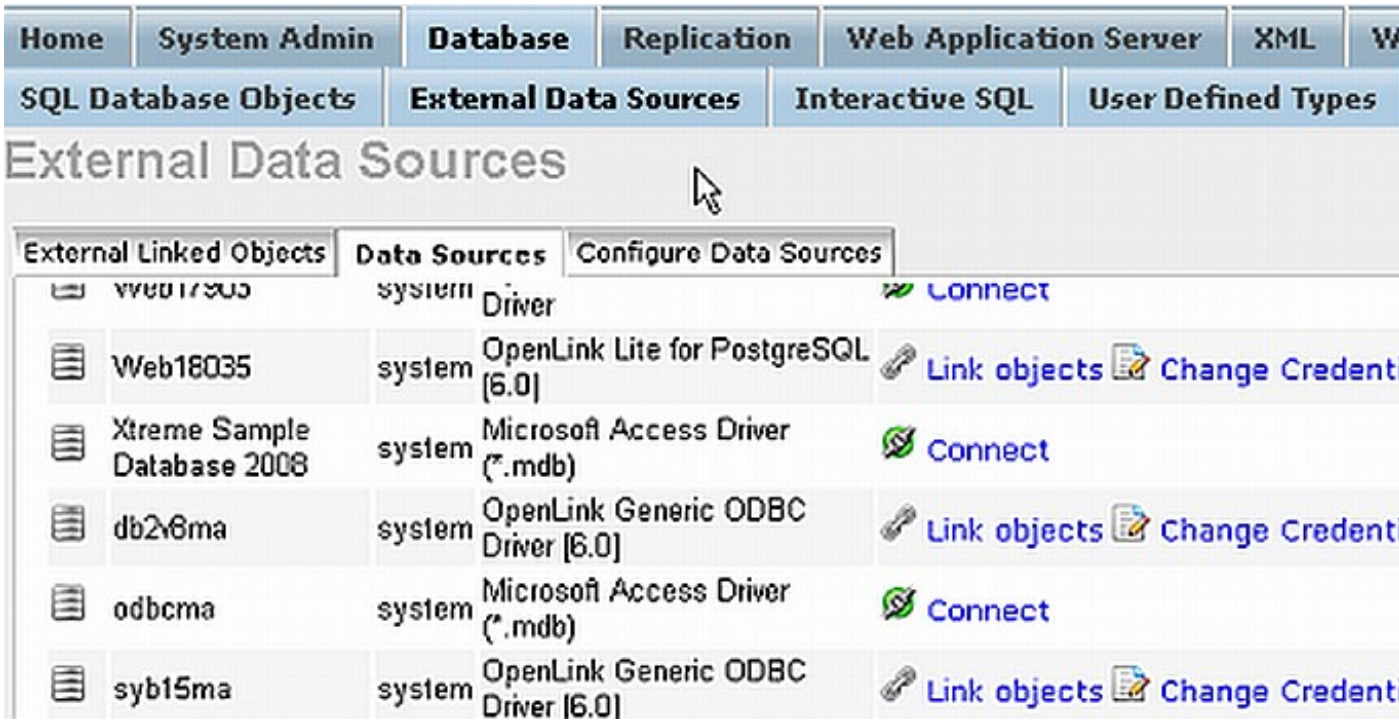
   Figure 8.546. Start

2. Select the "Conductor" link in the Left Frame and Login as the "dba" user.

Figure 8.547. Conductor

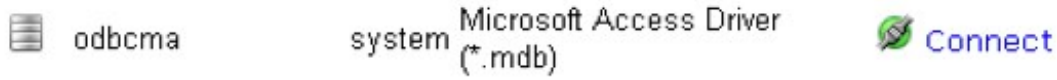
3. Select the "Databases" -> "External Data Source" -> "Data Sources" tab menu items

Figure 8.548. Databases



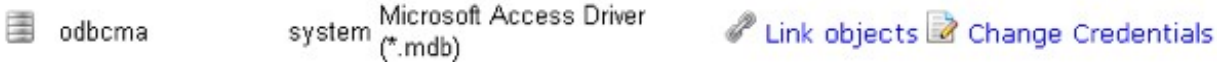
4. Select the "Connect" button for the "odbcma" ODBC to ODBC Bridge DSN.

Figure 8.549. Connect



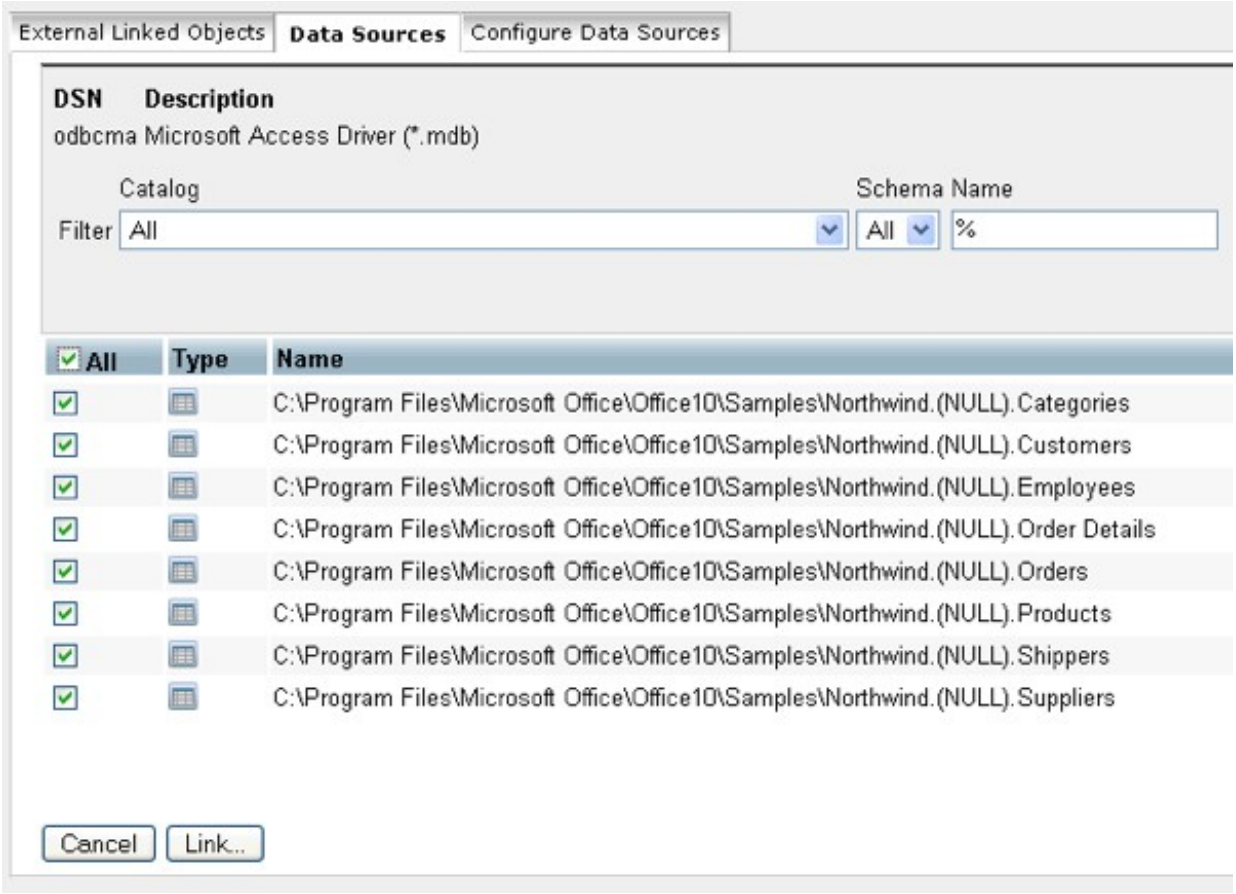
5. On successful connection Select the "Link Objects" button to obtain a list of available tables

Figure 8.550. Link Objects



6. Select all the tables that are part of the "Northwind" catalog.

Figure 8.551. Select tables

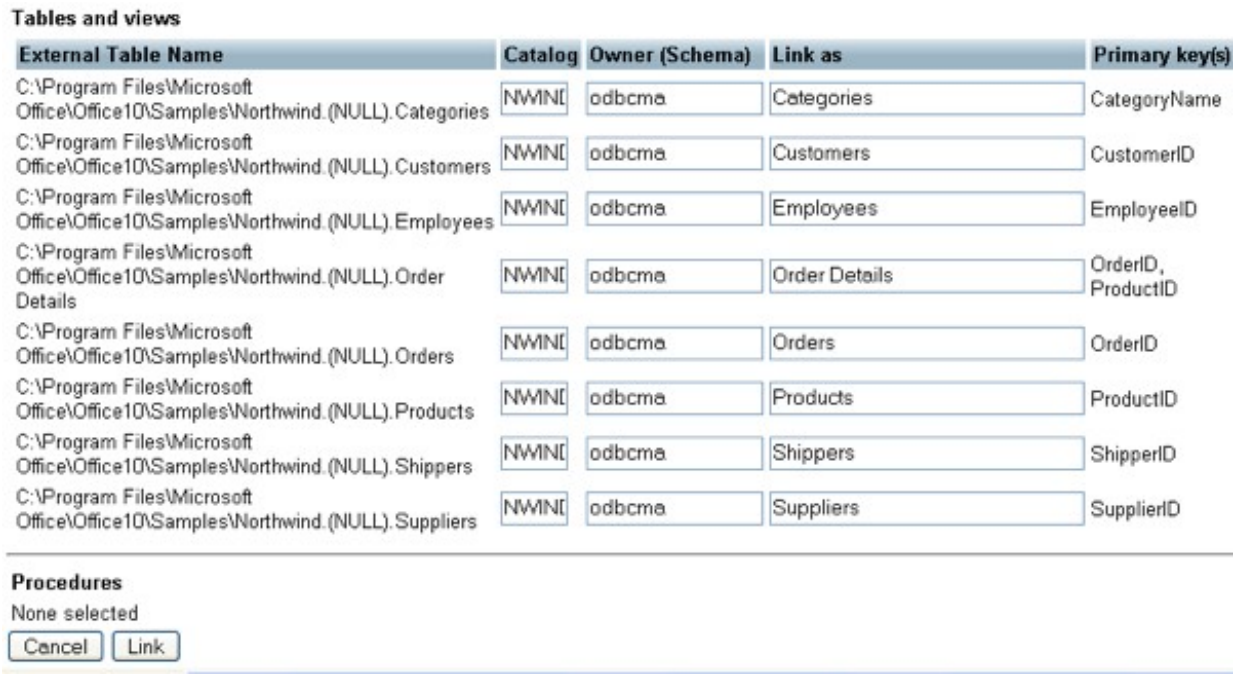


7. Change the Catalog for each table to be "NWIND" using the "Set All" button.

Figure 8.552. "Link" button

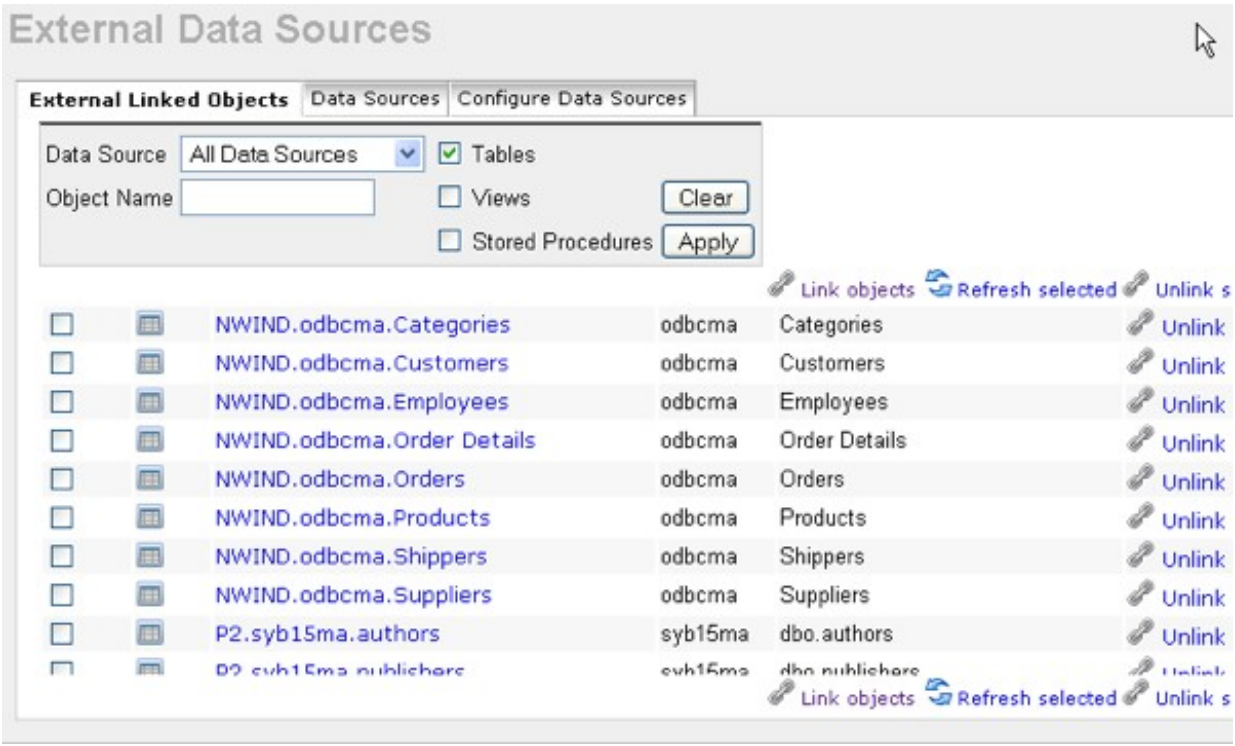
8. All the catalog names are changed to be "NWIND".
9. Select the "Link" button to link the selected tables into Virtuoso

Figure 8.553. "Link" button



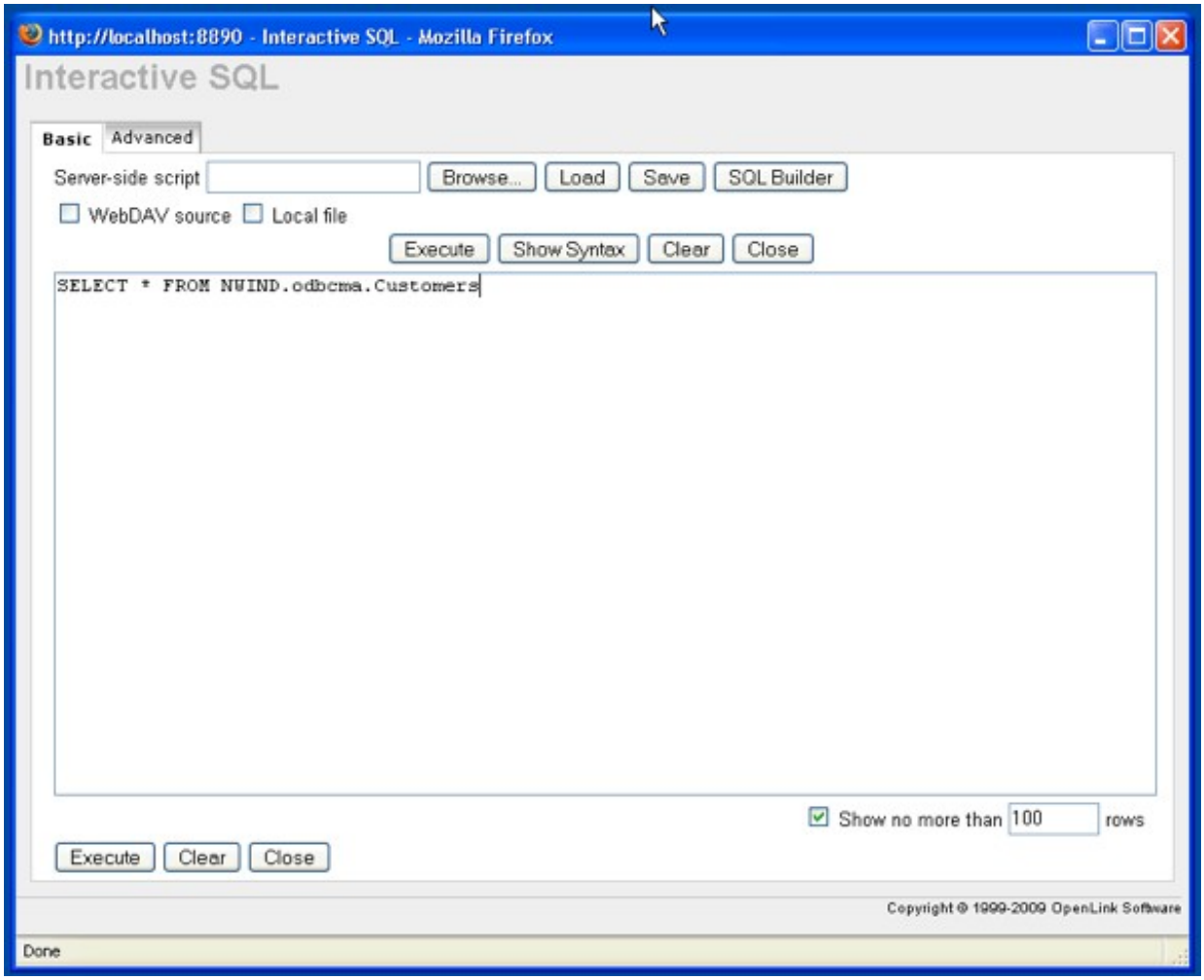
10. On completion of the Link process the tables will be displayed in the "External Linked Objects" tab.

Figure 8.554. Completion



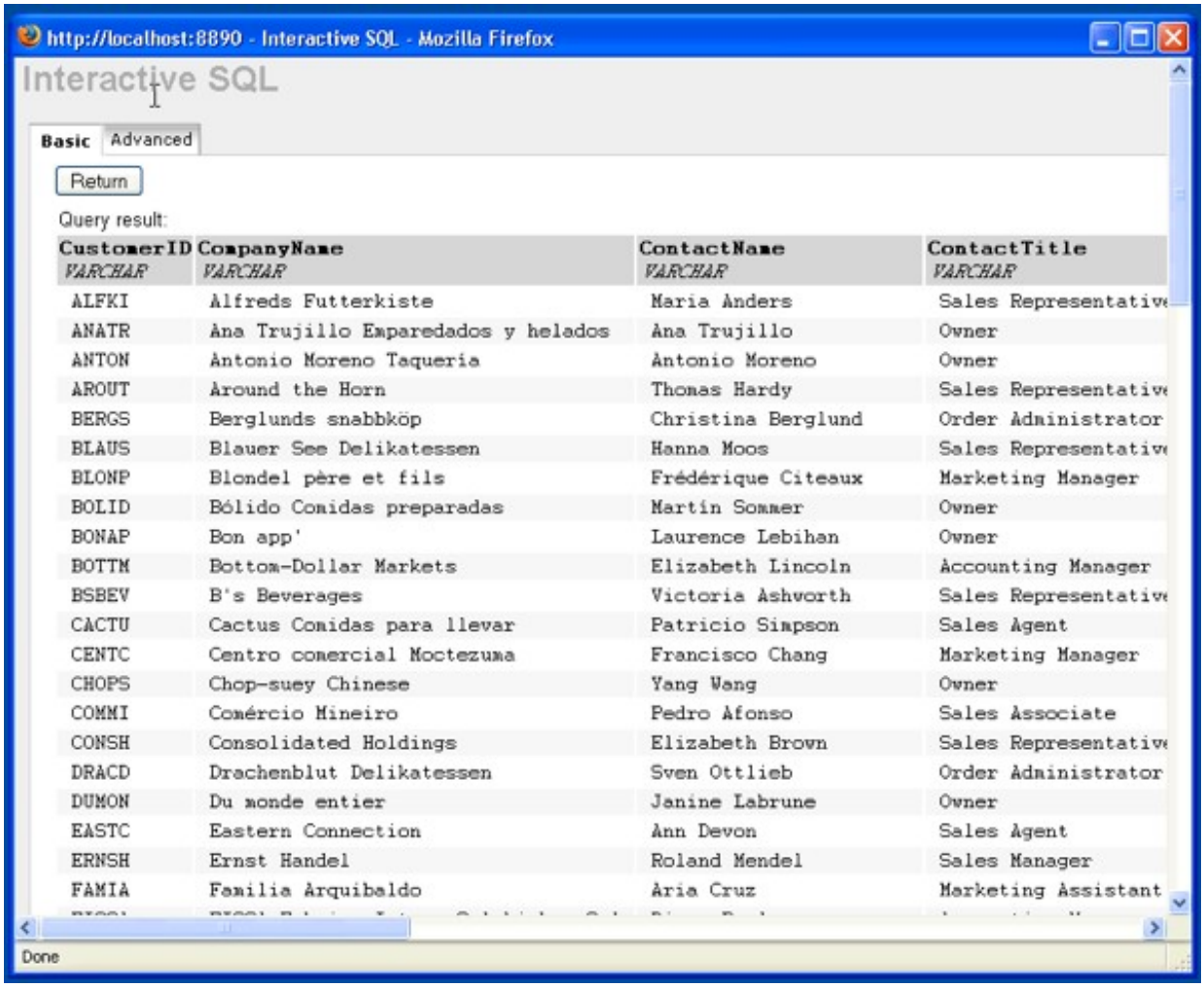
11. The linked tables can be queried by clicking on the hyperlink in the "Local Name" column of the "External Linked Objects" tab above, which loads the Virtuoso "Interactive SQL" interface with the required SQL "Select" for retrieving the remote table data. We shall use the "NWIND.odbcma.Customers" table to demonstrate this.

Figure 8.555. Completion



12. Then click the "Execute" button to run the query and retrieve the results from the remote table.

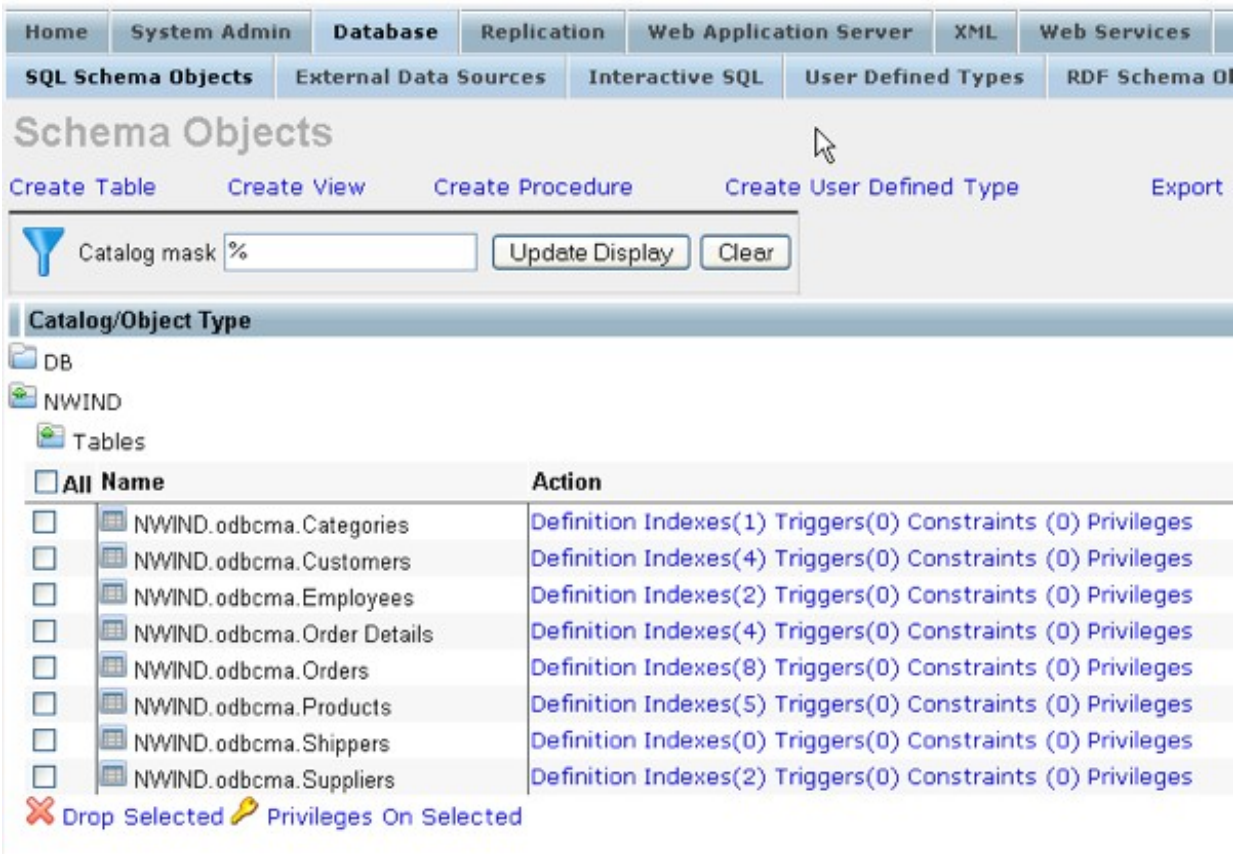
Figure 8.556. Completion



CustomerID VARCHAR	CompanyName VARCHAR	ContactName VARCHAR	ContactTitle VARCHAR
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner
ANTON	Antonio Moreno Taqueria	Antonio Moreno	Owner
AROUT	Around the Horn	Thomas Hardy	Sales Representative
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative
BLONP	Blondel père et fils	Frédérique Citeaux	Marketing Manager
BOLID	Bólido Comidas preparadas	Martín Sommer	Owner
BONAP	Bon app'	Laurence Lebihan	Owner
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager
BSBEV	B's Beverages	Victoria Ashworth	Sales Representative
CACTU	Cactus Comidas para llevar	Patricio Simpson	Sales Agent
CENTC	Centro comercial Moctezuma	Francisco Chang	Marketing Manager
CHOPS	Chop-suey Chinese	Yang Wang	Owner
COMMI	Comércio Mineiro	Pedro Afonso	Sales Associate
CONSH	Consolidated Holdings	Elizabeth Brovn	Sales Representative
DRACD	Drachenblut Delikatessen	Sven Ottlieb	Order Administrator
DUMON	Du monde entier	Janine Labrune	Owner
EASTC	Eastern Connection	Ann Devon	Sales Agent
ERNSH	Ernst Handel	Roland Mendel	Sales Manager
FAMIA	Familia Arquibaldo	Aria Cruz	Marketing Assistant

13. The tables can also be viewed as part of the Virtuoso "SQL Schema Objects" under the "NWIND" catalog name.

Figure 8.557. view tables



The Link process is now complete enabling the tables to be queried as if part of the Virtuoso Schema.

8.11.3. Creating EDM in Visual Studio 2008

The following steps can be used to create an Entity Data Model (EDM) for the Microsoft Access Northwind database:

1. Launch the Visual Studio 2008 SP1 IDE.

   Figure 8.558. Visual Studio 2008 SP1 IDE

2. Create a

   Web Application

   project by going to the

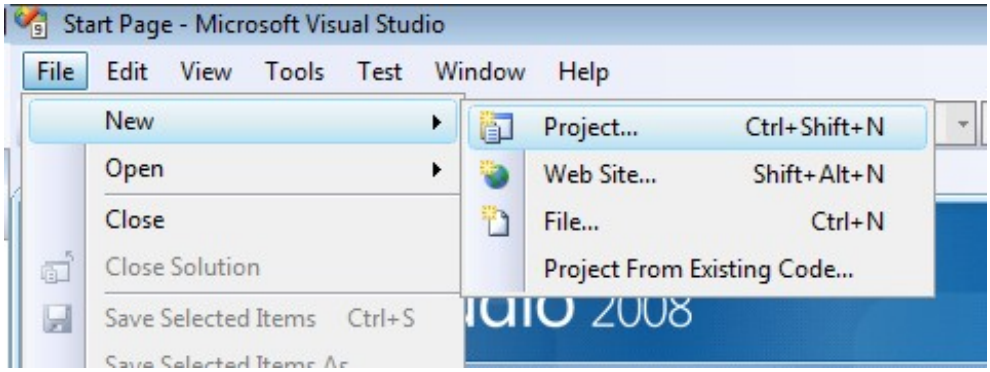
   File

menu in Visual Studio and choosing

New Project

.

Figure 8.559. Web Application



3. When the New Project window appears, choose either

   Visual Basic

   or

   Visual C#

   as your programming language.
4. Within the language category, click on

   Web

   and select

   ASP.NET Web Application

   from the right-hand panel.
5. Choose a name for the project, for example

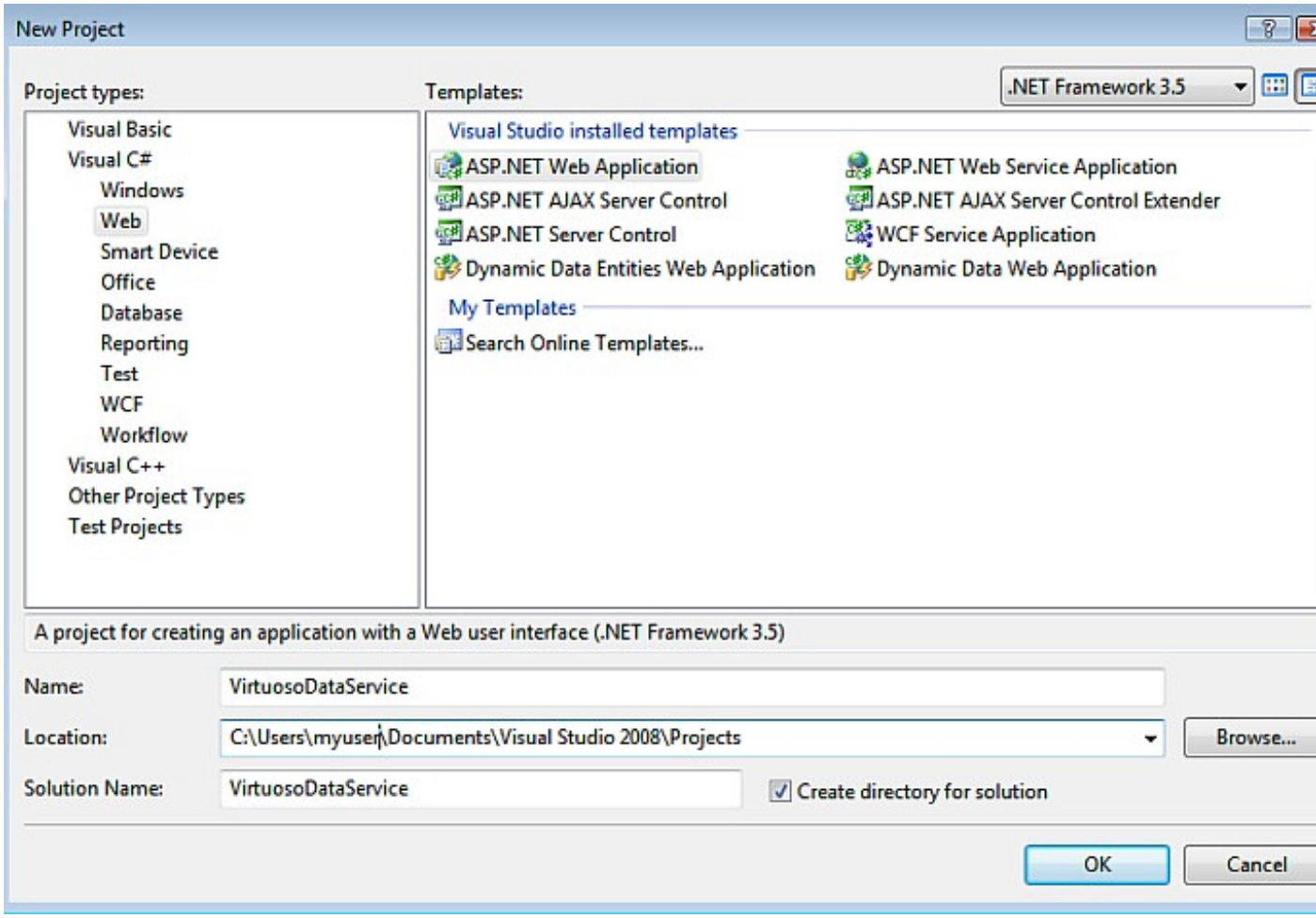
   VirtuosoDataService

   , and click

   OK

   .

Figure 8.560. name for the project

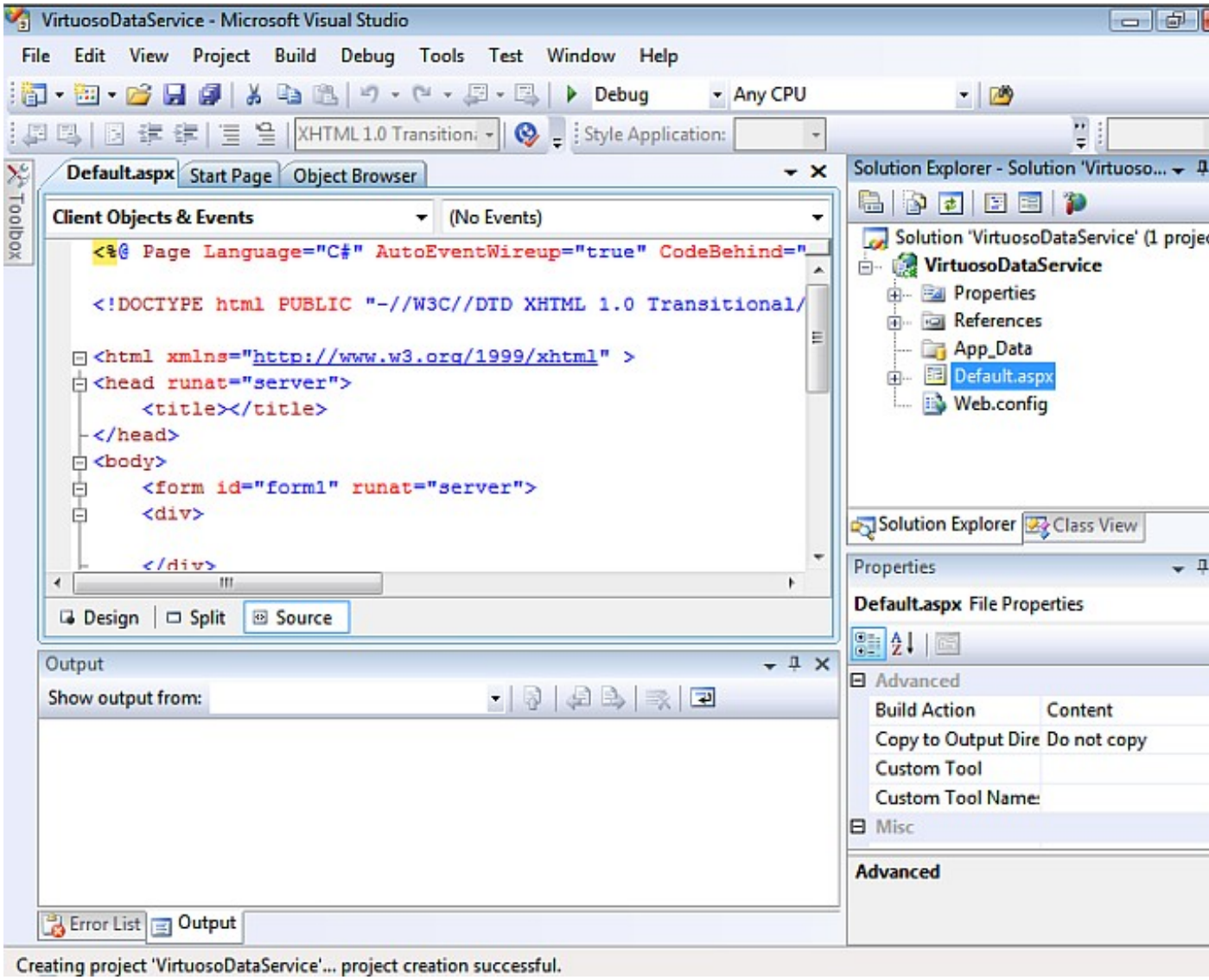


6. This will create a new project called

   VirtuosoDataService

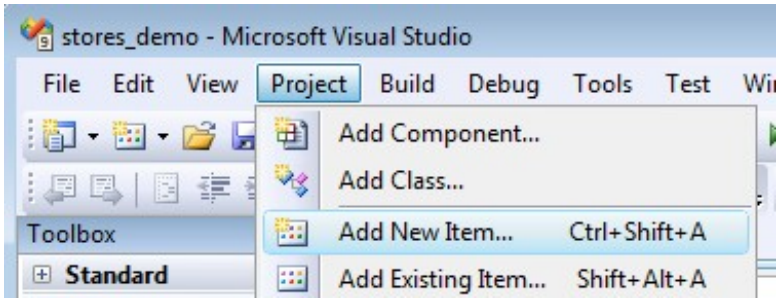
   .

Figure 8.561. create a new project



7. Select the Project -> Add New Item menu option.

Figure 8.562. VirtuosoDataService



8. The

Add New Item

dialog will appear. Choose the

ADO.NET Entity Data Model

template. Give it the name

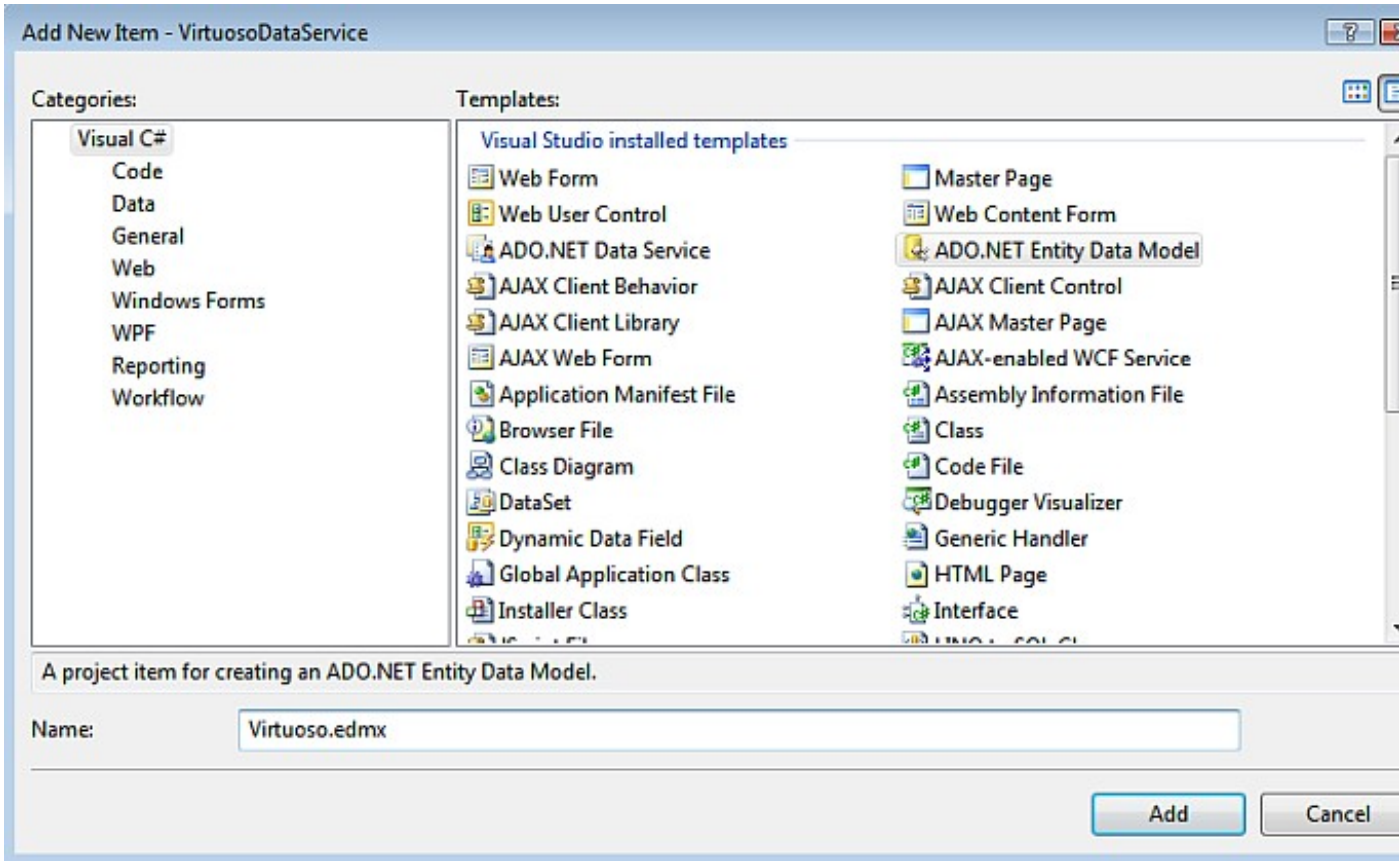
Virtuoso.edmx

and click

Add

to start the creation of the ADO.Net Entity Data Model.

Figure 8.563. Add New Item



9. In the

Entity Data Model Wizard

dialog

Choose Model Contents

page select the

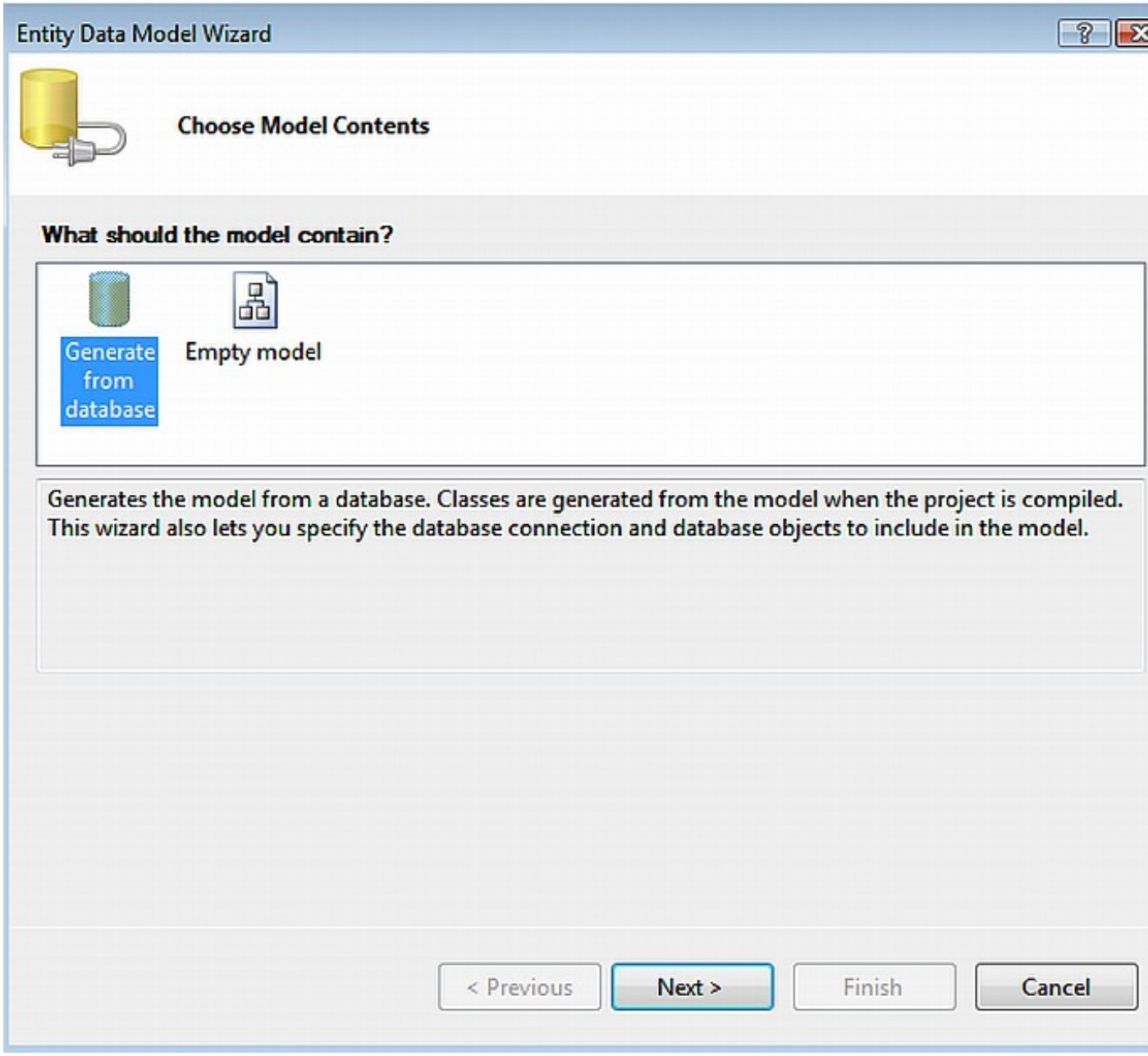
Generate from Database

model type and click

Next

.

Figure 8.564. Choose Model Contents



10. In the

    Entity Data Model Wizard

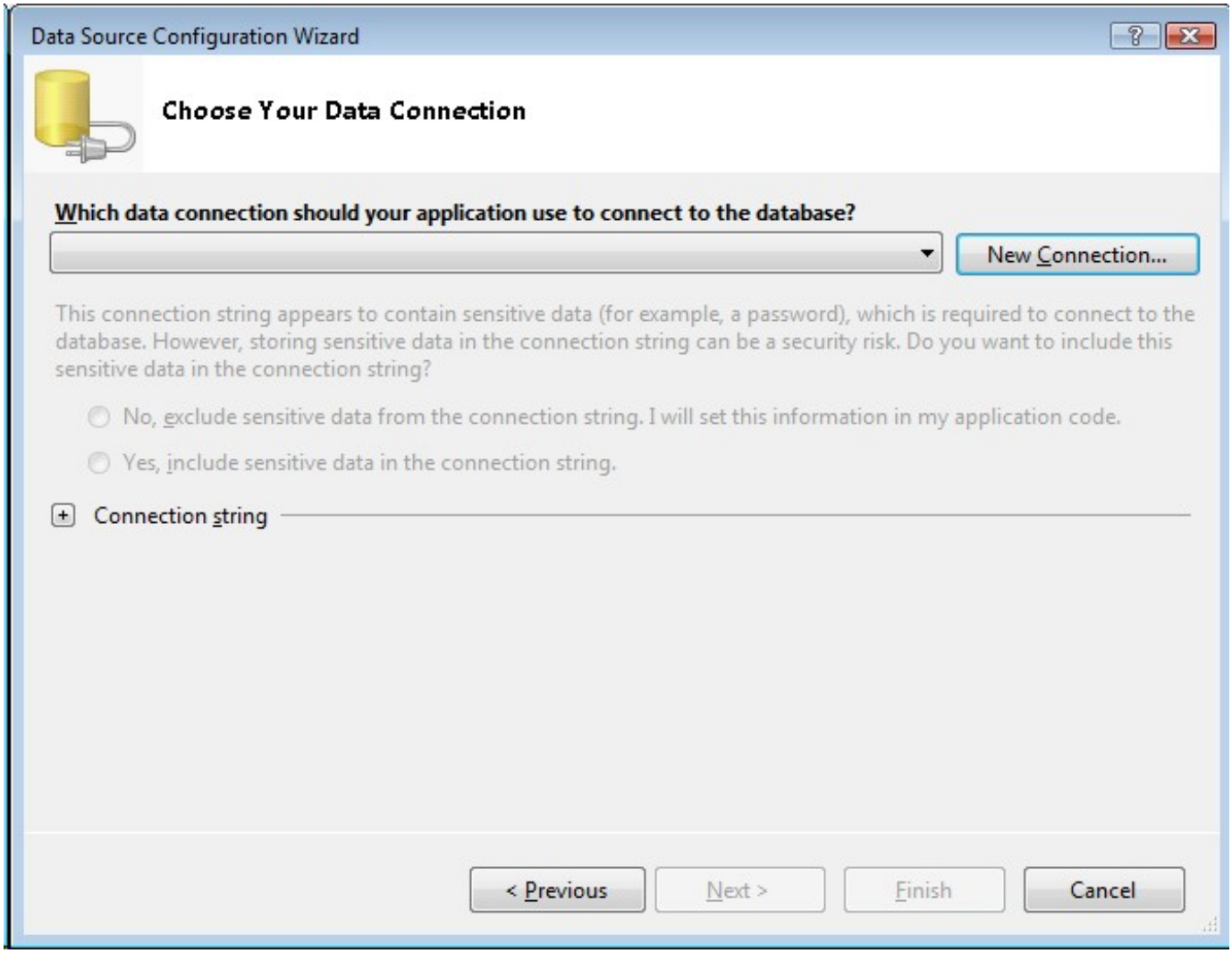
    dialog

    Choose your Data Connection

    page, select the

    New Connection button

Figure 8.565. Entity Data Model Wizard



11. In the

    Choose Data Source

    dialog, select the OpenLink

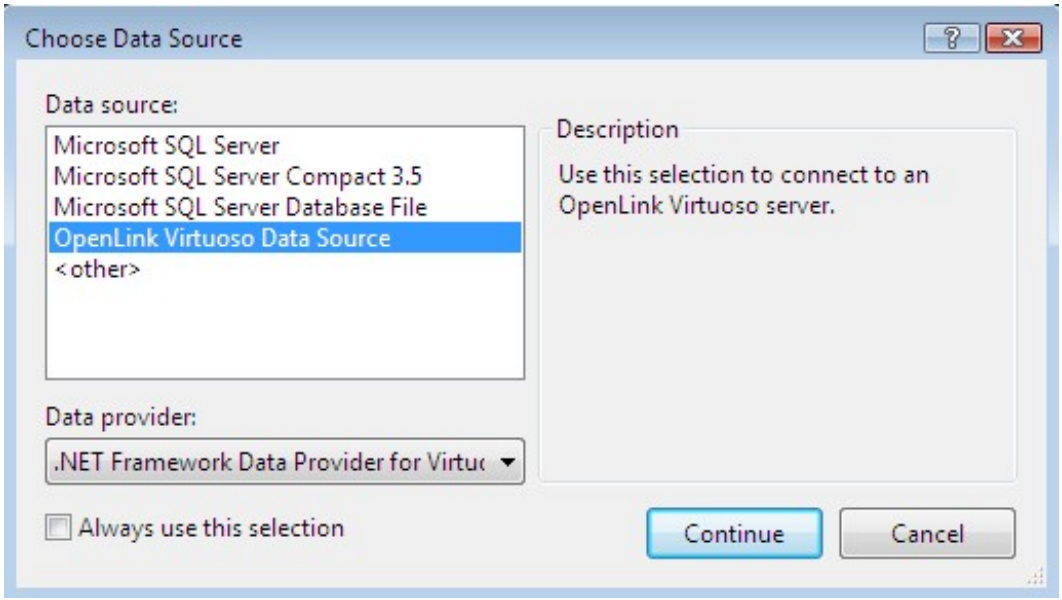
    Virtuoso Data Source

    from the list and click

    Continue

    .

    Figure 8.566. Choose Data Source



12. In the

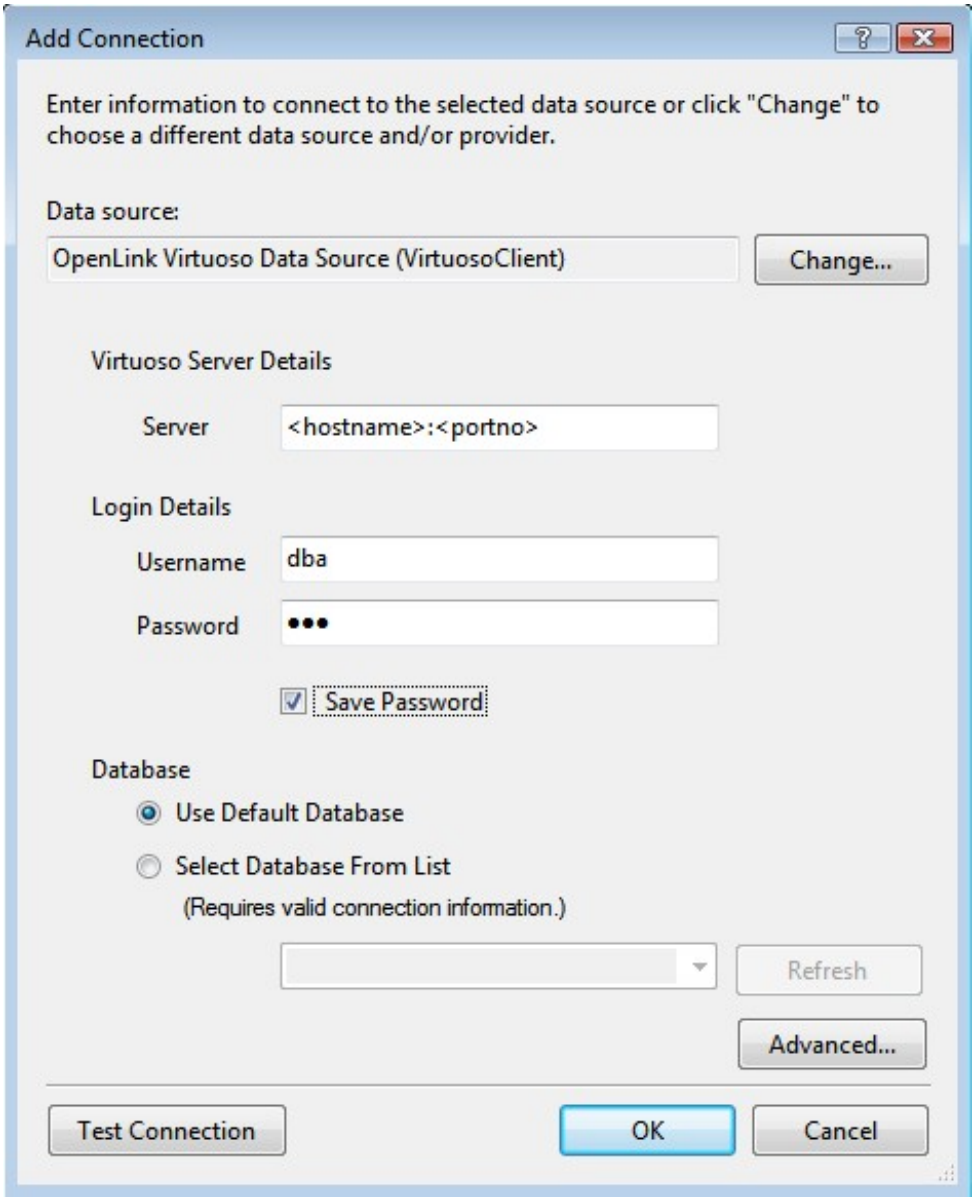
    Add Connection

    dialog, specify the

    hostname, portno, username and password

    for the target Virtuoso Server and check the Save Password check box.

    Figure 8.567. Connection Properties



13. Select the

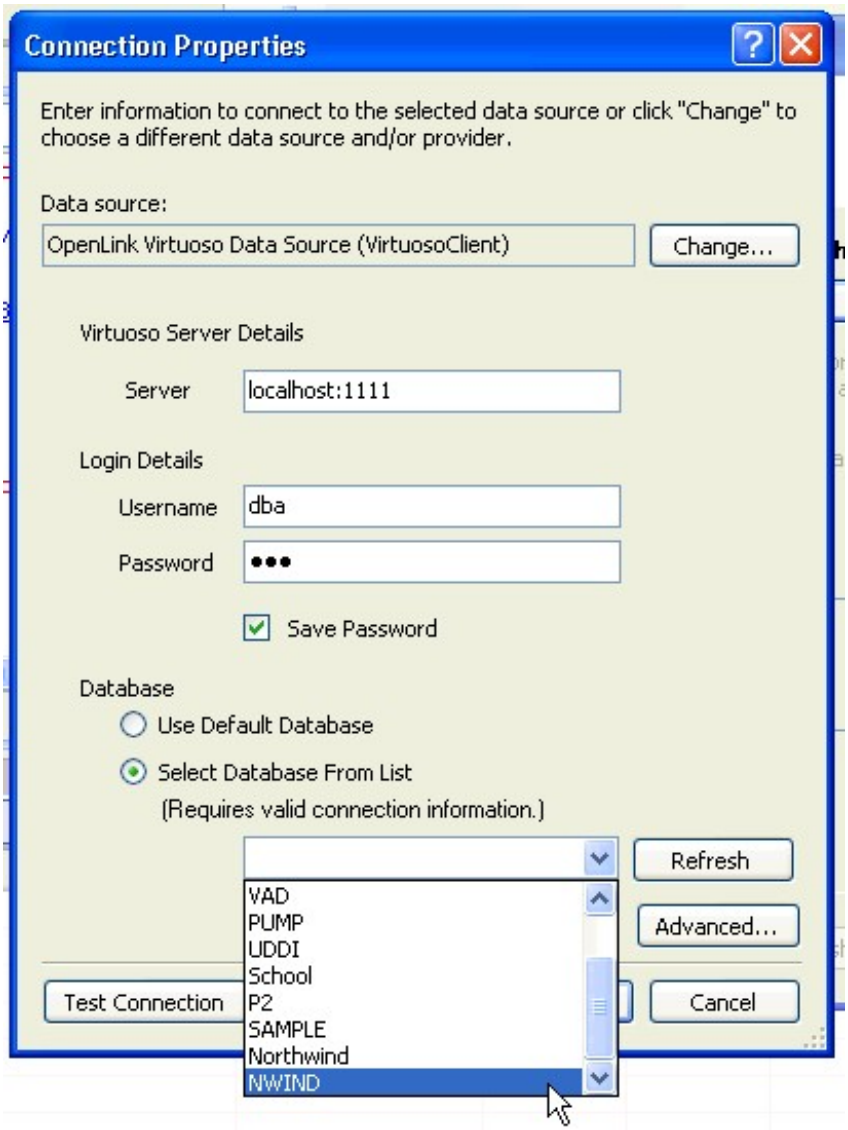
    Select Database From List

    radio button and choose the

    NWIND

    database from the drop down list.

Figure 8.568. Add connection

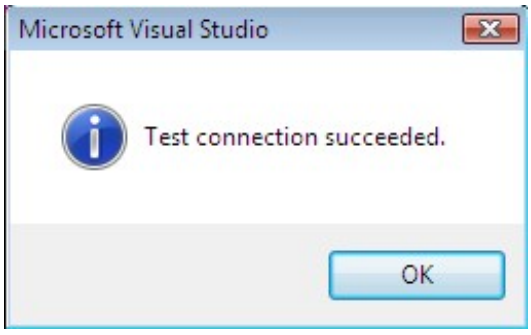


14. Press the

    Test Connection

    dialog to verify that the database is accessible.

    Figure 8.569. Test Connection



15. Set the

    entity connect string

    name to

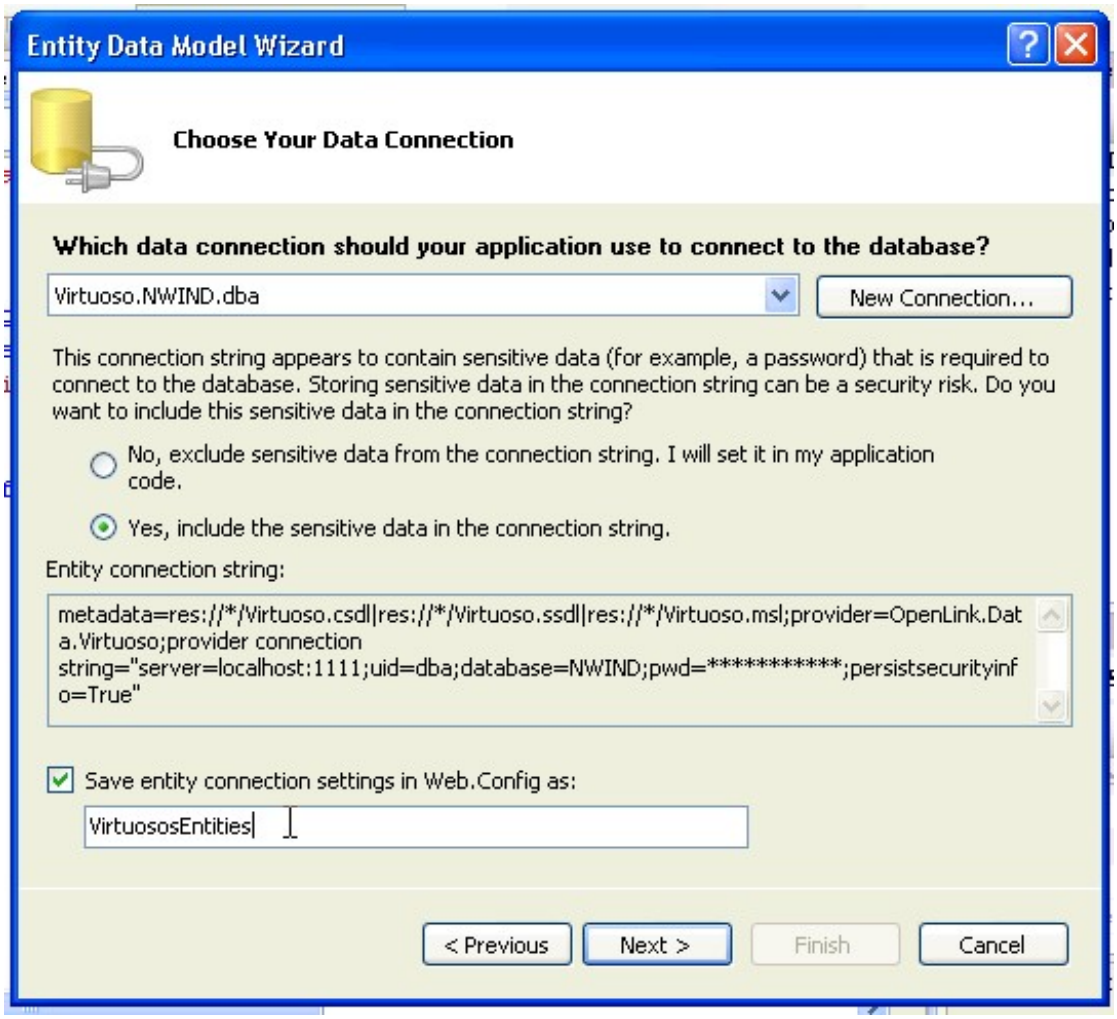
VirtuosoEntities

(note this name as it is required in the section on creating and ADO.Net Data Service) and click

Next

.

Figure 8.570. entity connect string



16. In the

    Choose your Database Objects

    page select the

    Tables

    check box to select all tables in the NWIND catalog for addition to the Entity Data Model. Set the

    Model Namespace

    to

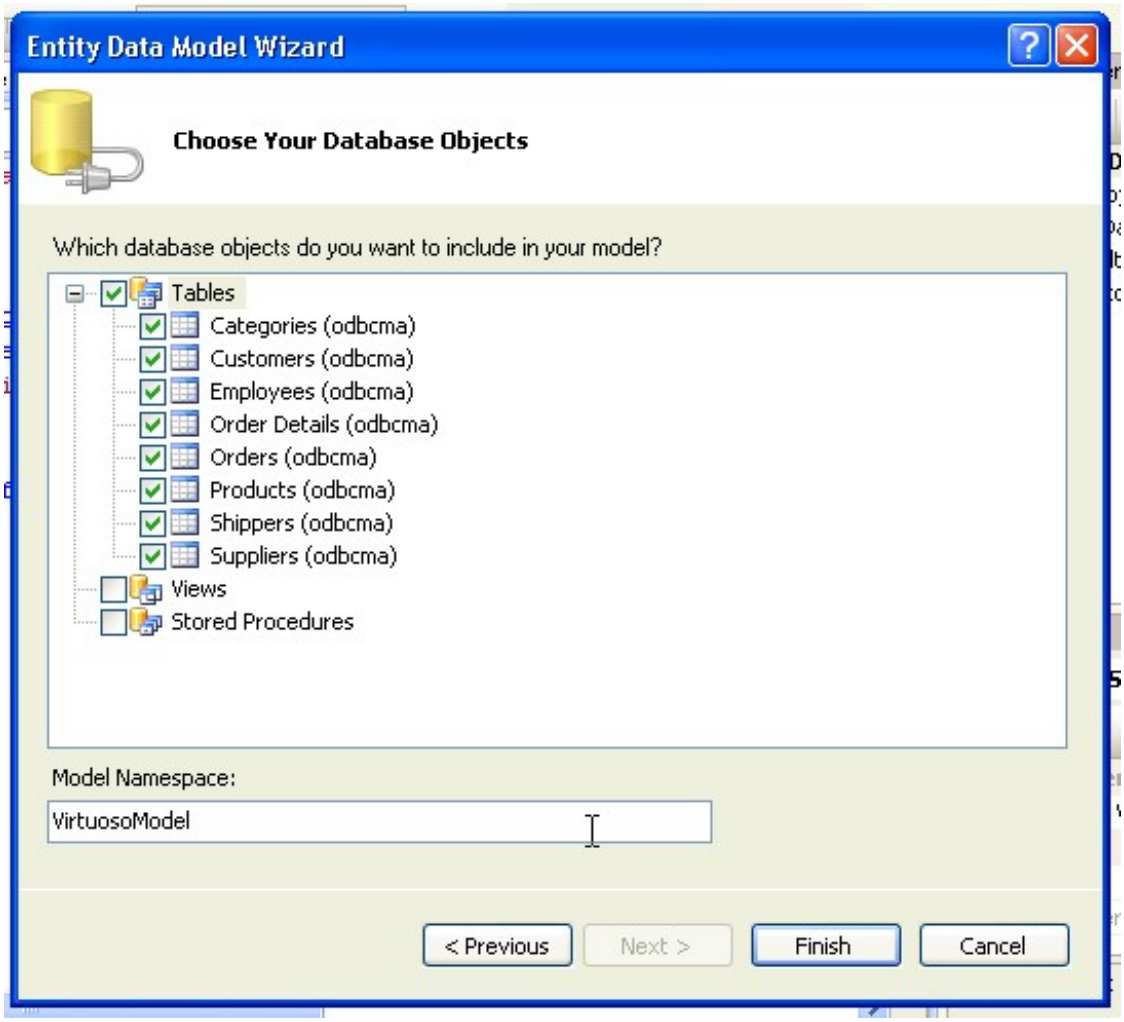
    VirtuosoModel

    and click

Finish

.

Figure 8.571. Database Objects

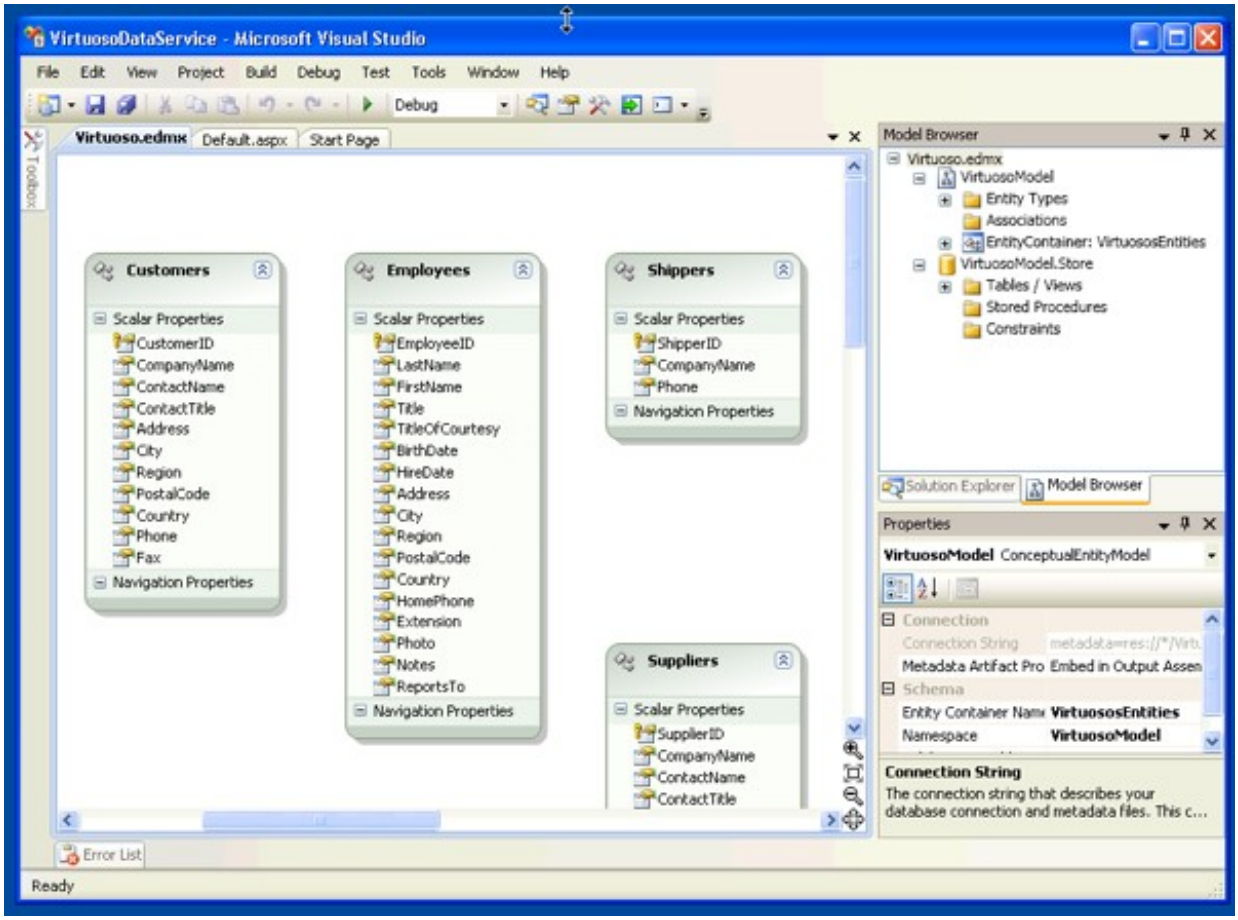


17. The

    Virtuoso.edmx

    EDM will be created with the tables and relationships displayed in the Visual Studio IDE

Figure 8.572. Virtuoso.edmx



Creation for the Entity Data Model for the Microsoft Access Northwind database is now complete.

8.11.4. Using EDM to create Entity Framework based applications

Now that a Microsoft Entity Data Model has been created for the Northwind database, Entity Framework applications can be created to make use of it.

Visual Studio Windows DataGrid Form Application

This section details the steps required to create a simple Visual Studio 2008 Windows Form application, with associated DataGridView control for displaying data in selected tables from the target database.

1. Launch the Visual Studio 2008 SP1 IDE.

   Figure 8.573. Visual Studio 2008 SP1 IDE

2. Create a

   Web Application

   project by going to the

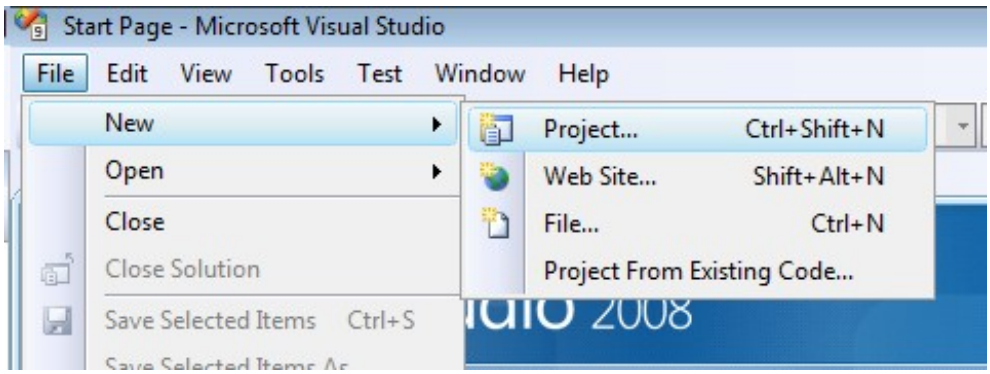
   File

   menu in Visual Studio and choosing

   New Project

   .

   Figure 8.574. Web Application



3. When the New Project window appears, choose either

   Visual Basic

   or

   Visual C#

   as your programming language.
4. Within the language category, click on

   Windows

   and select

   Windows Form Application

from the right-hand panel.

5. Choose a name for the project, for example

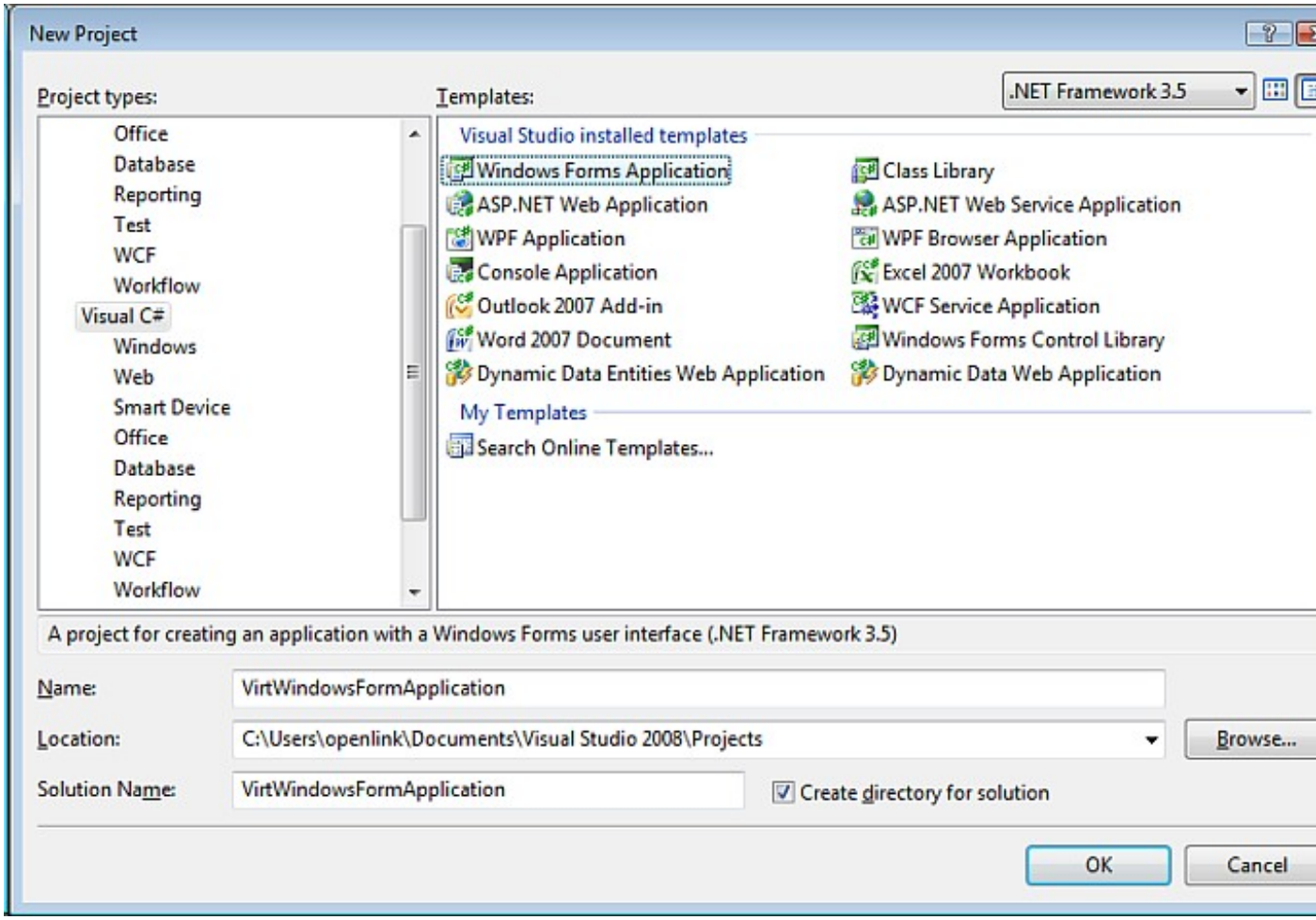
   VirtWindowsFormApplication

   , and click

   OK

   .

   Figure 8.575. Web Application



6. In the

   Toolbox

   , expand

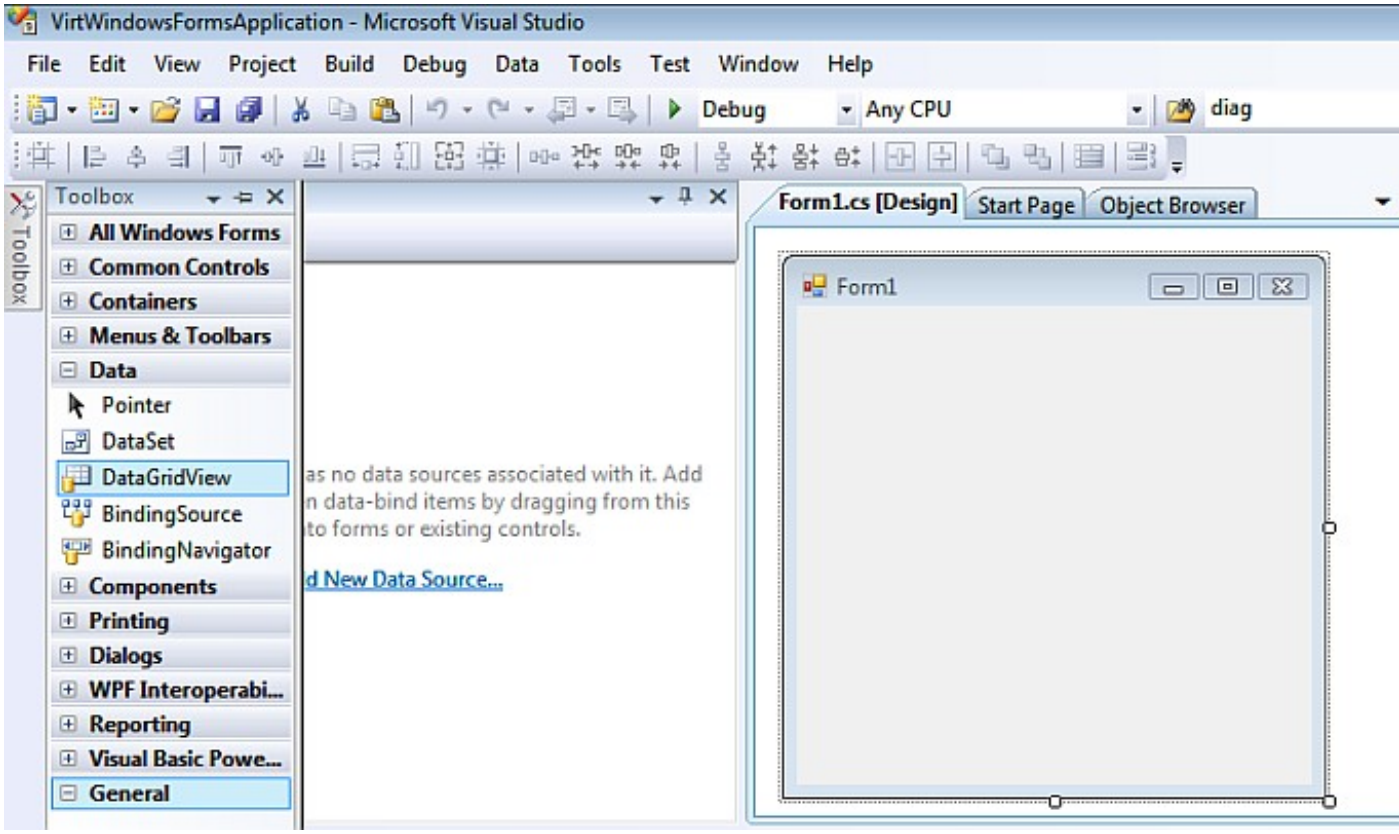
   Data Controls

   , and drag the

   DataGridView

   control onto the form.

   Figure 8.576. Toolbox



7. Click on the little

   arrow

   in the top right of the

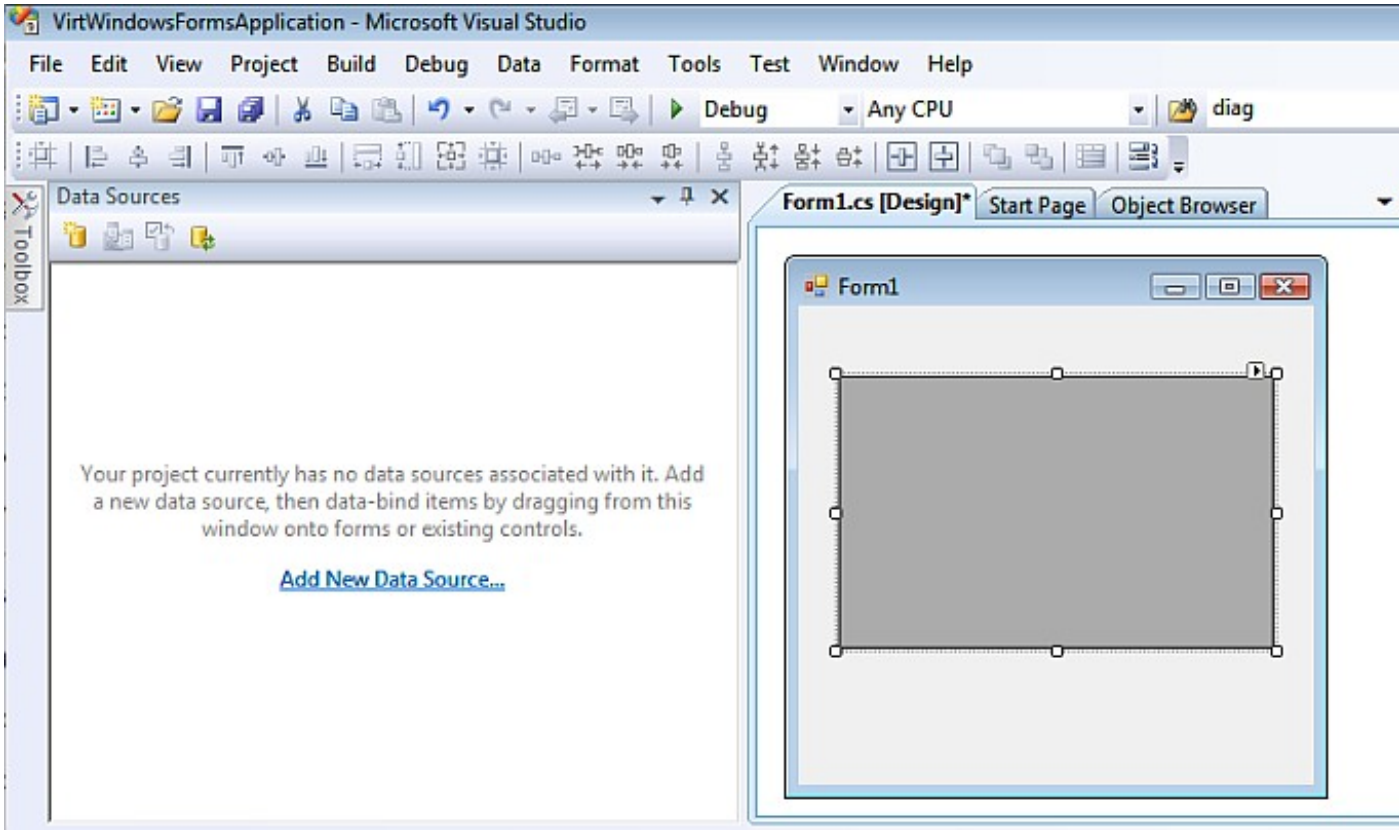
   DataGridView

   control. This loads the

   DataGridView Task

   menu.

Figure 8.577. DataGridView Task

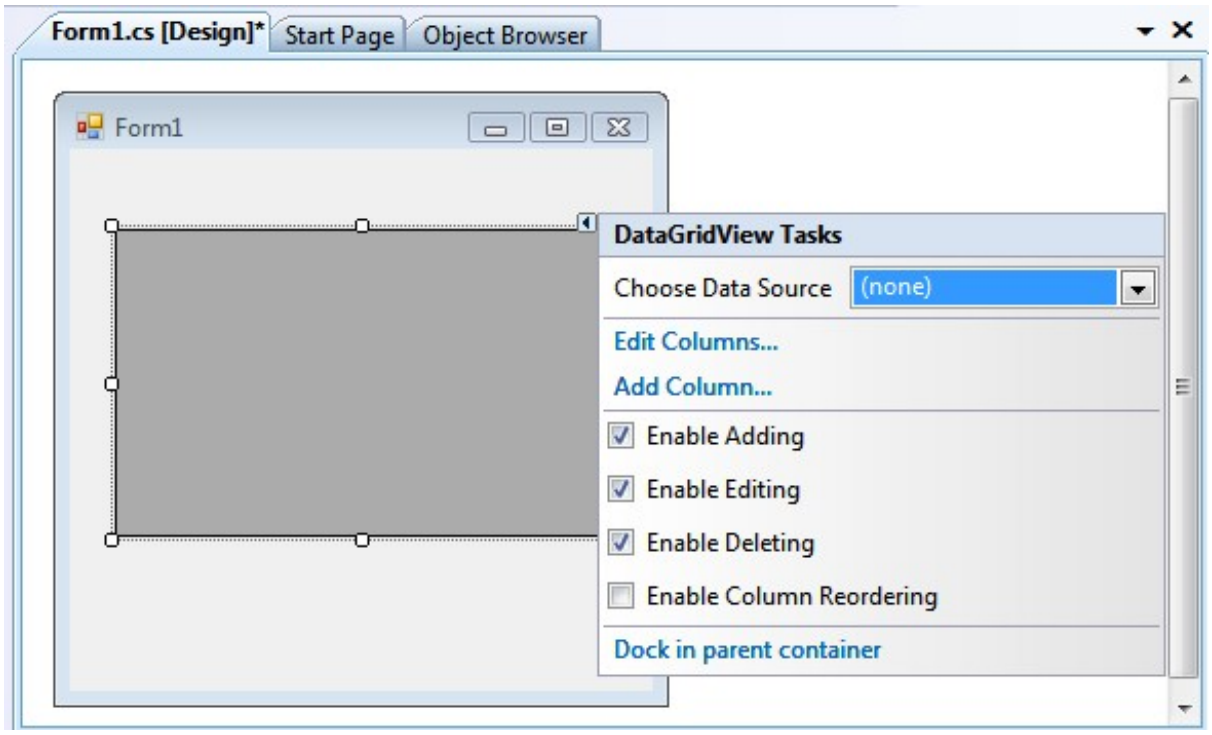


8. Click on the

   Choose Data Source

   list box.

Figure 8.578. Choose Data Source

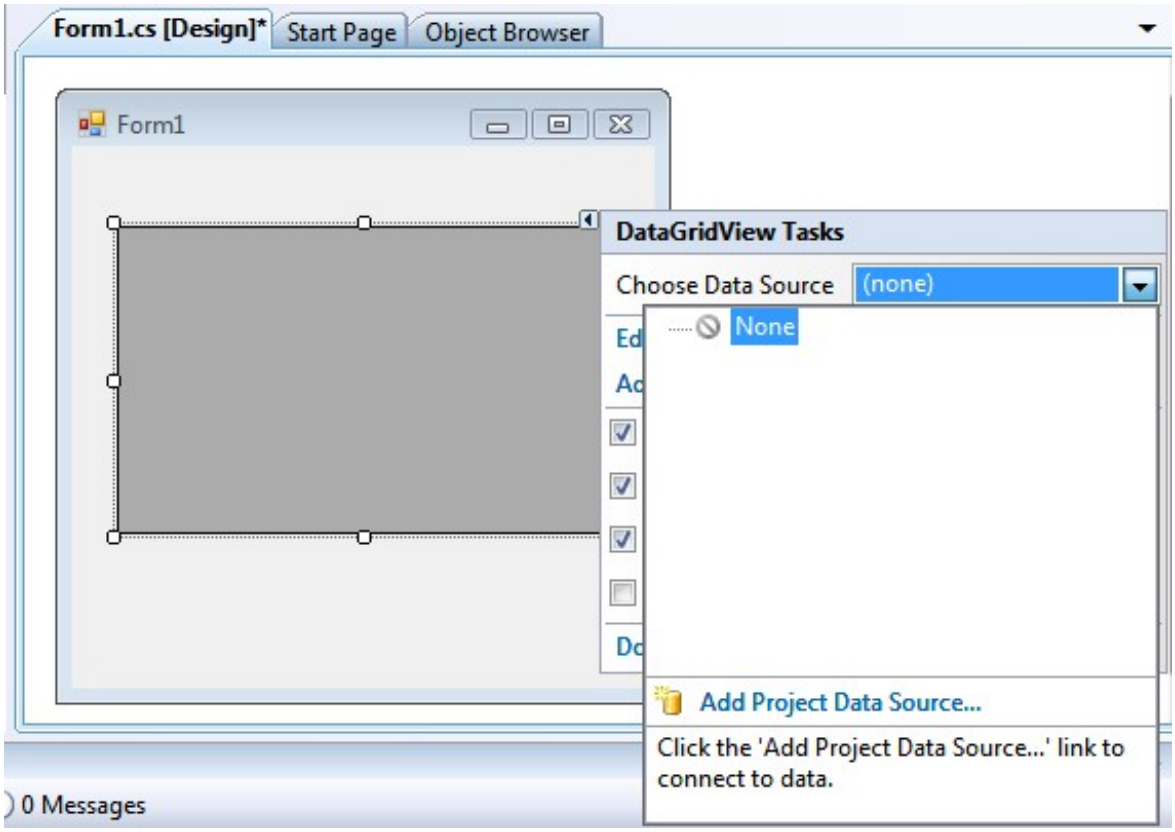


9. Click on the

   Add Project Data Source

link to connect to a data source.

Figure 8.579. Add Project Data Source



10. In the

    Data Source Configuration Wizard

    dialog

    Choose Data Source Type

    page select the

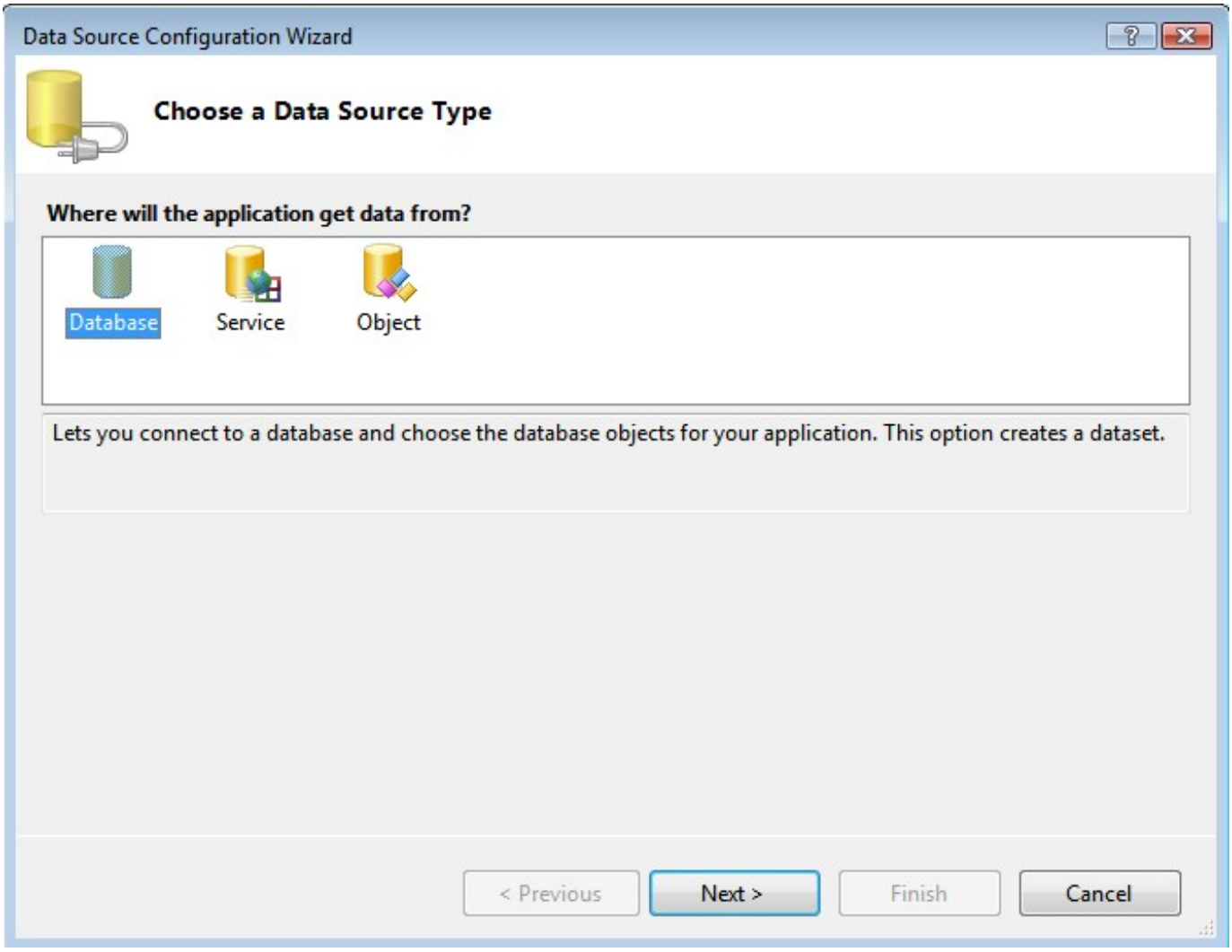
    Database

    data source type and click

    Next

    .

Figure 8.580. Data Source Type



11. In the

    Data Source Configuration Wizard

    dialog

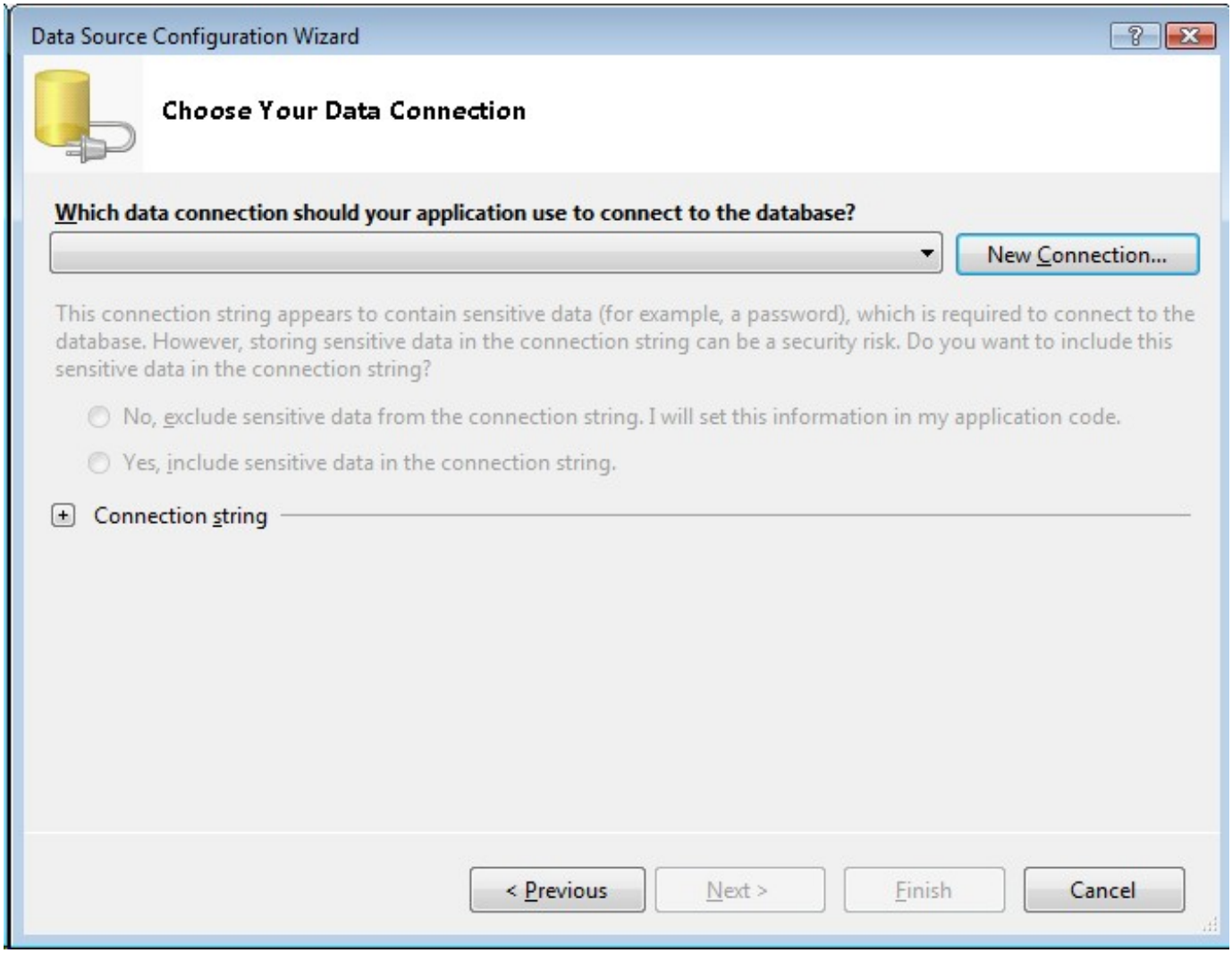
    Choose your Data Connection

    page, select the

    New Connection

    button

Figure 8.581. Data Source Configuration Wizard



12. In the

    Choose Data Source

    dialog, select the OpenLink

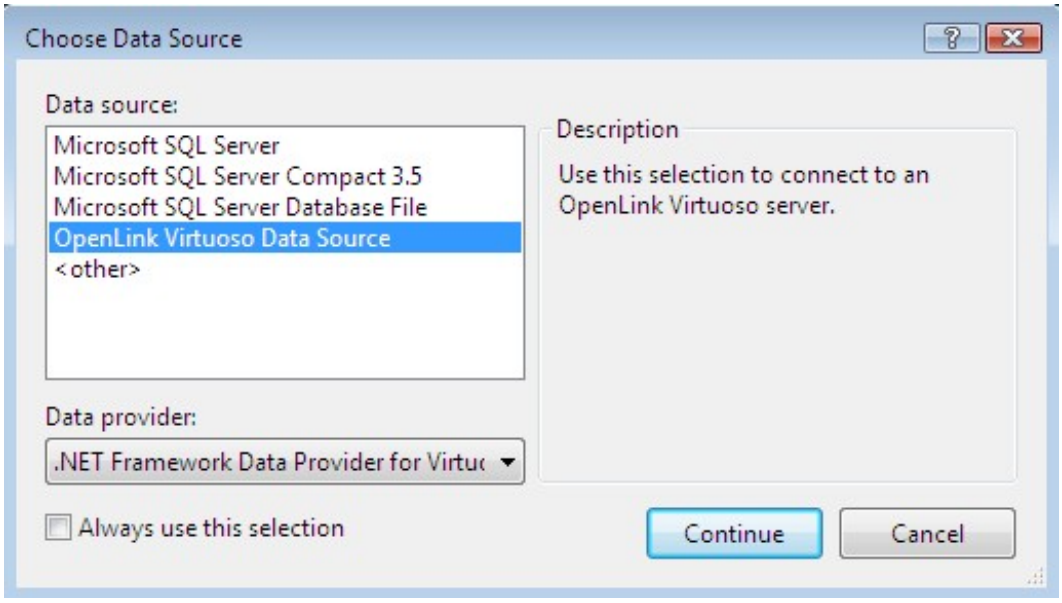
    Virtuoso Data Source

    from the list and click

    Continue

    .

Figure 8.582. Data Source



13. In the

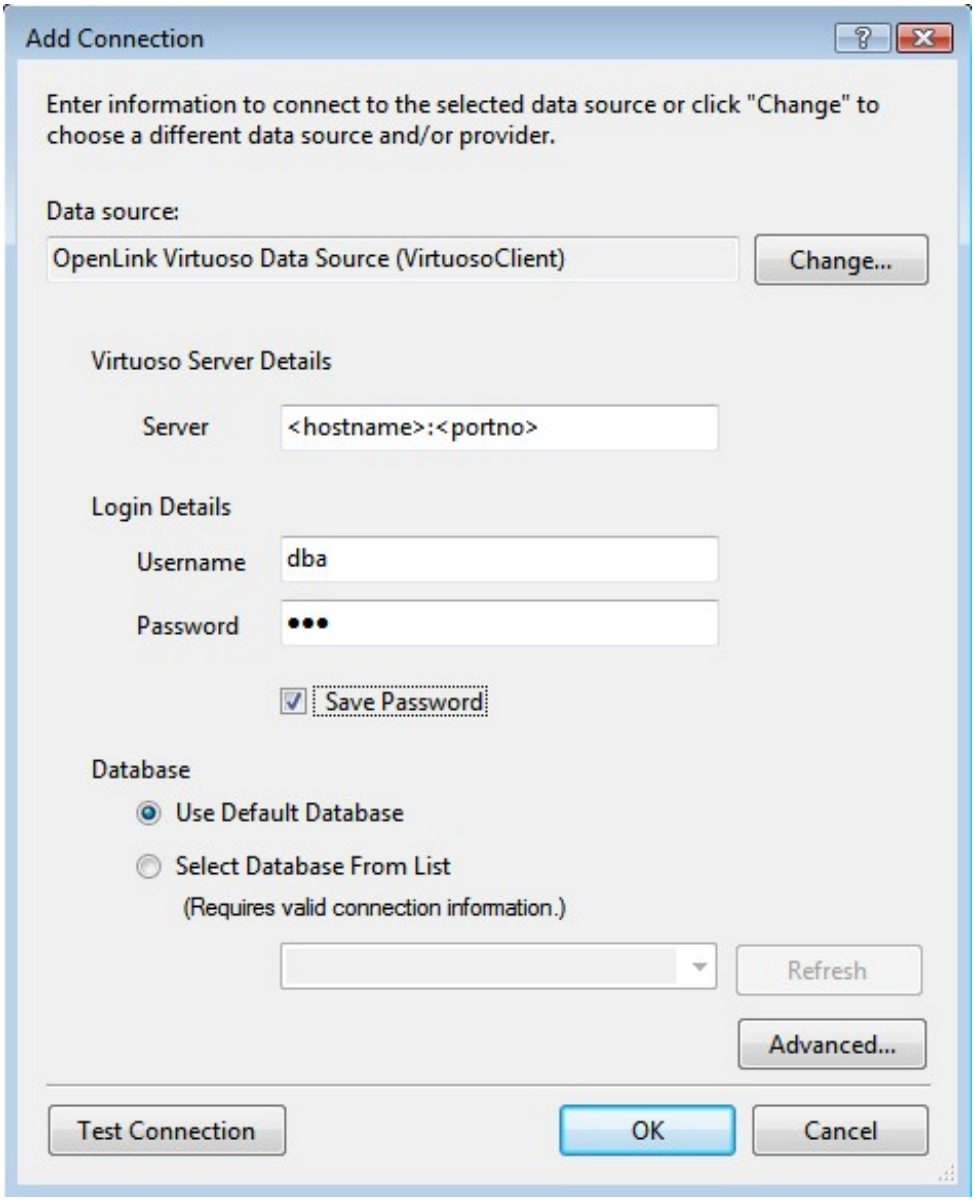
    Add Connection

    dialog, specify the

    hostname, portno, username and password

    for the target Virtuoso Server and check the Save Password check box.

    Figure 8.583. Connection Properties



14. Select the

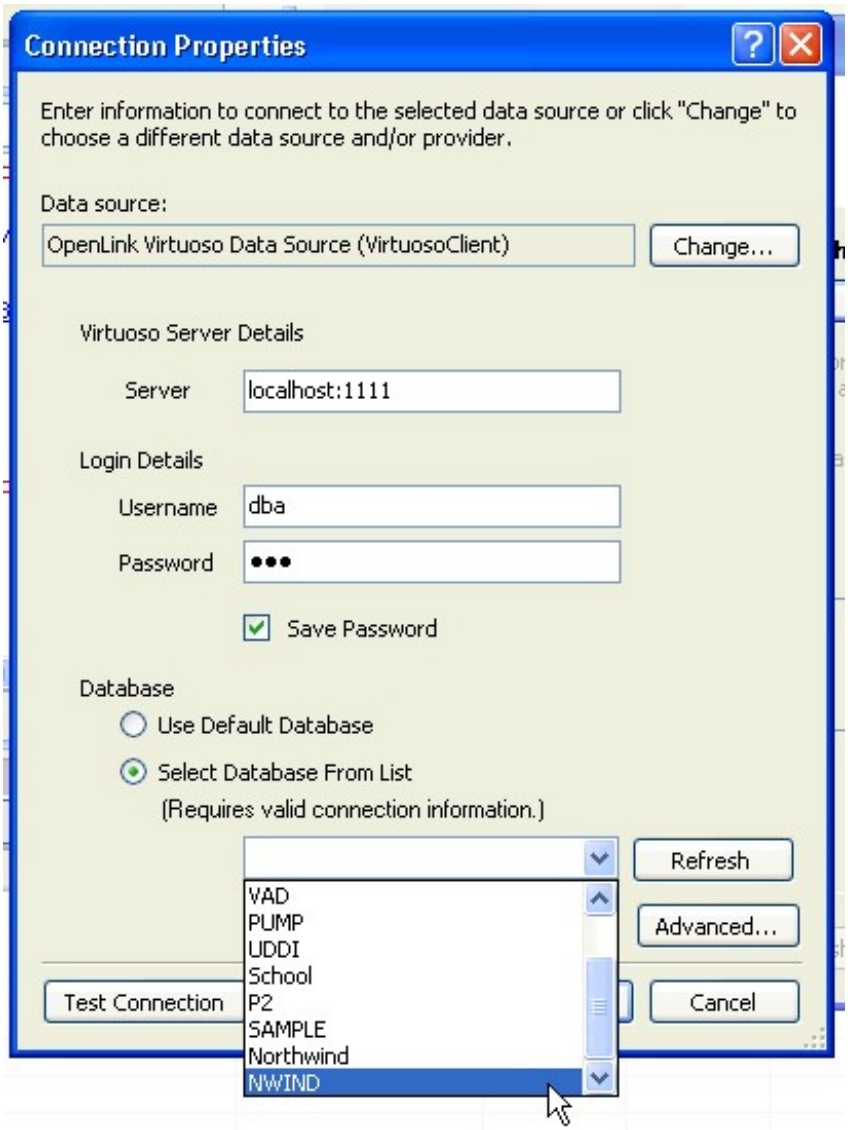
    Select Database From List

    radio button and choose the

    NWIND

    database from the drop down list.

Figure 8.584. Add connection

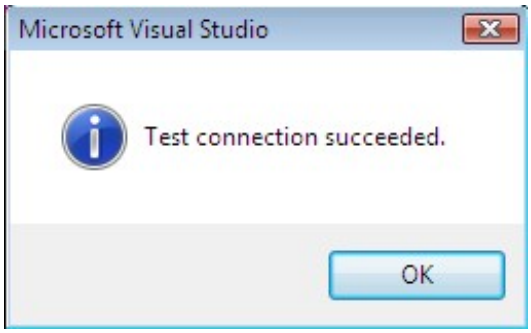


15. Press the

    Test Connection

    dialog to verify that the database is accessible.

    Figure 8.585. Test Connection



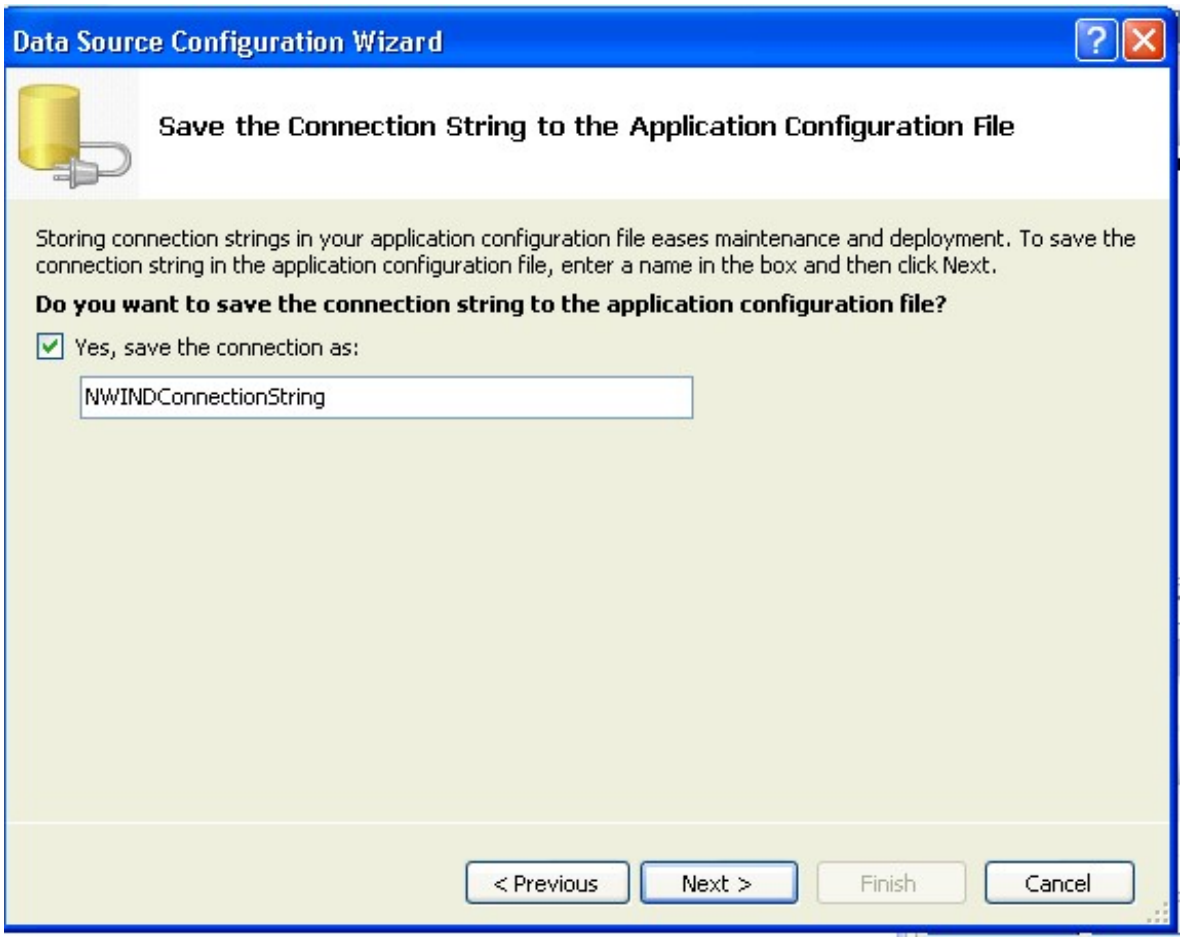
16. Leave the default connect string

    NWINDConnectionString

    and click

Next

Figure 8.586. NWINDConnectionString



17. From the list of available tables returned for the NWIND database, select the

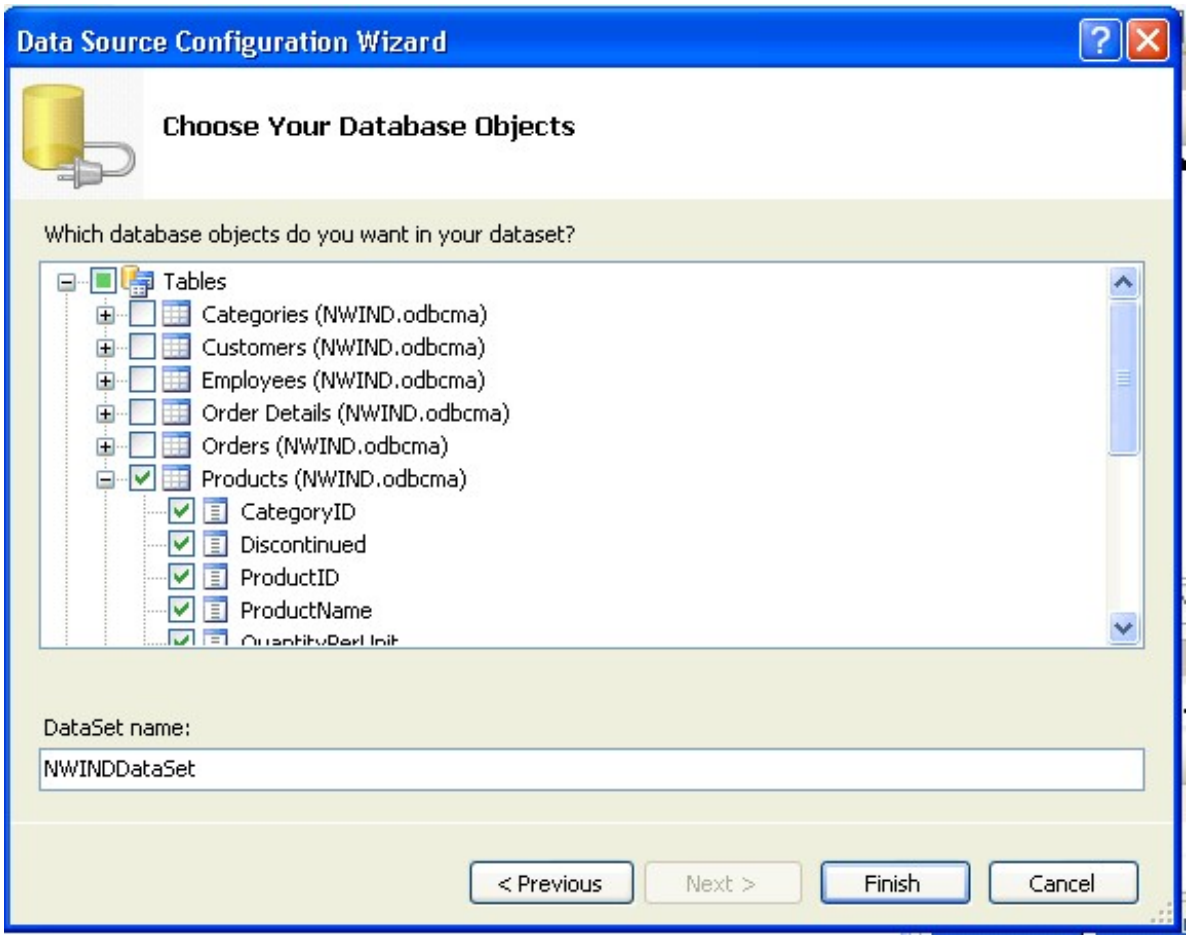
    JOBS

    table to be associated with the

    DataGridView

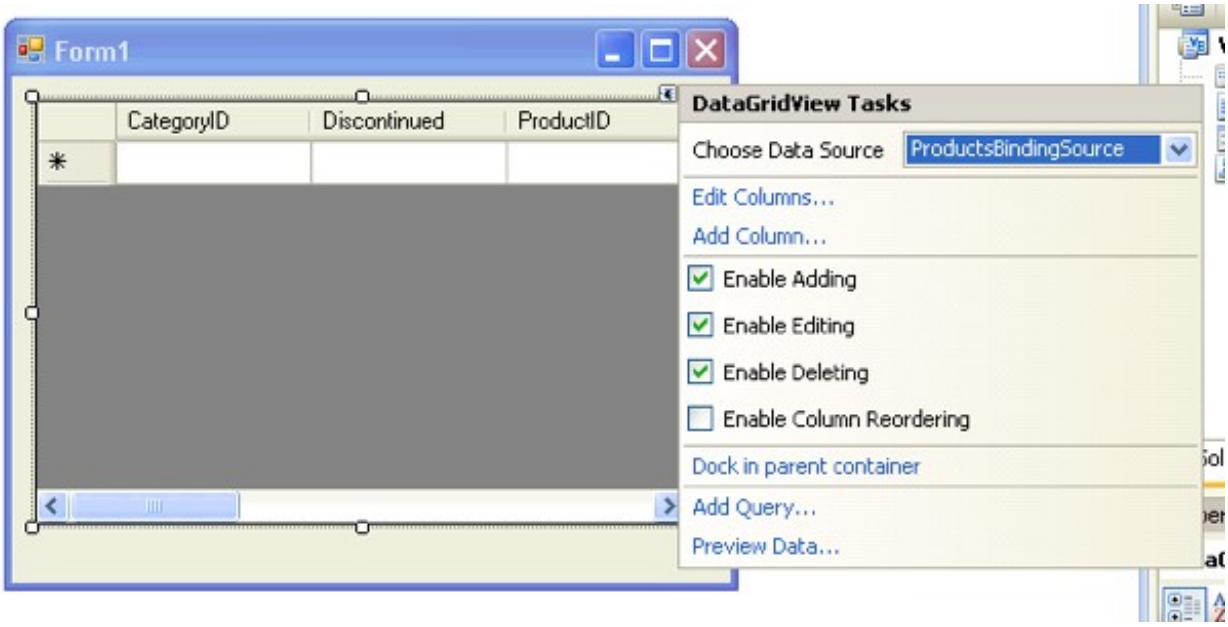
    control.

    Figure 8.587. Northwind database



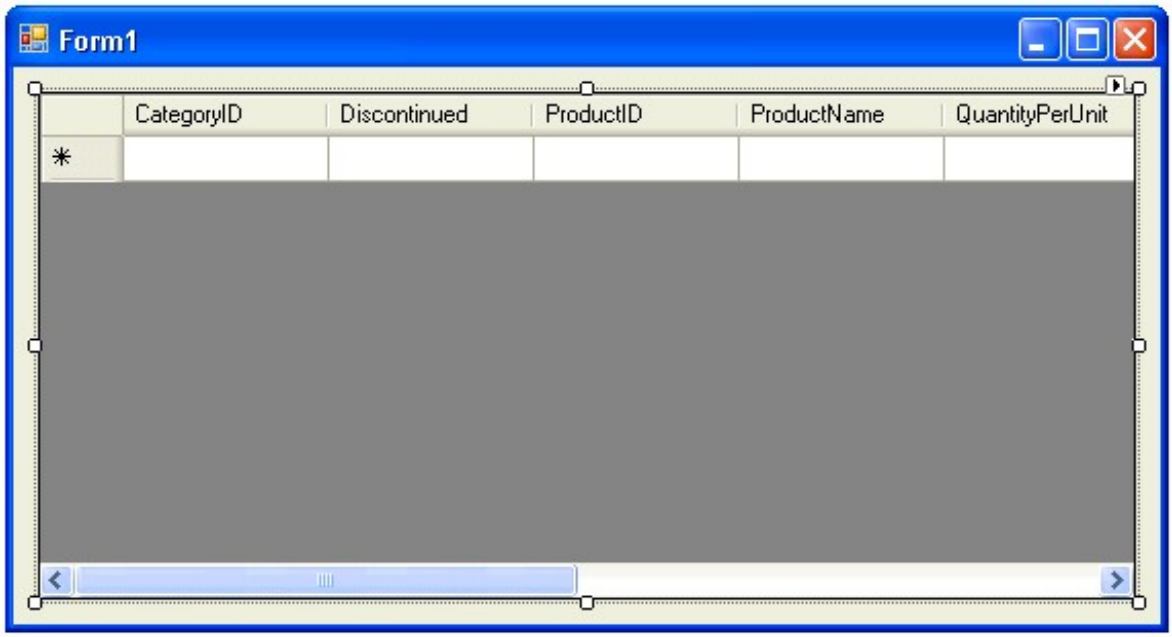
18. The columns names of the select table will be displayed in the DataGridView.

Figure 8.588. DataGridView



19. Resize the Form and DataGridView to allow all columns to be visible, if possible.

Figure 8.589. Resize the Form and DataGridView



20. To test the application, simply hit

    Ctrl+F5

    within Visual Studio or select

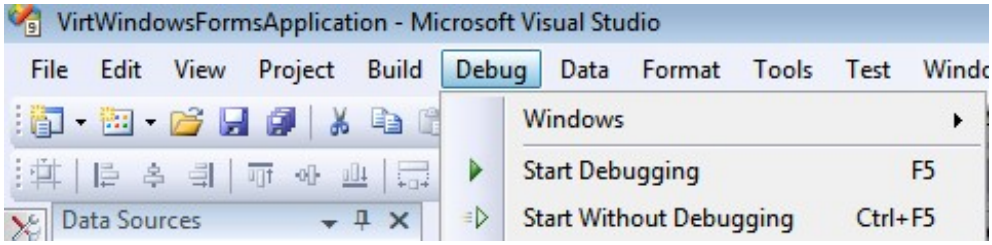
    Start Debugging

    from the

    Debug

    menu.

    Figure 8.590. Start Debugging



21. The data from the

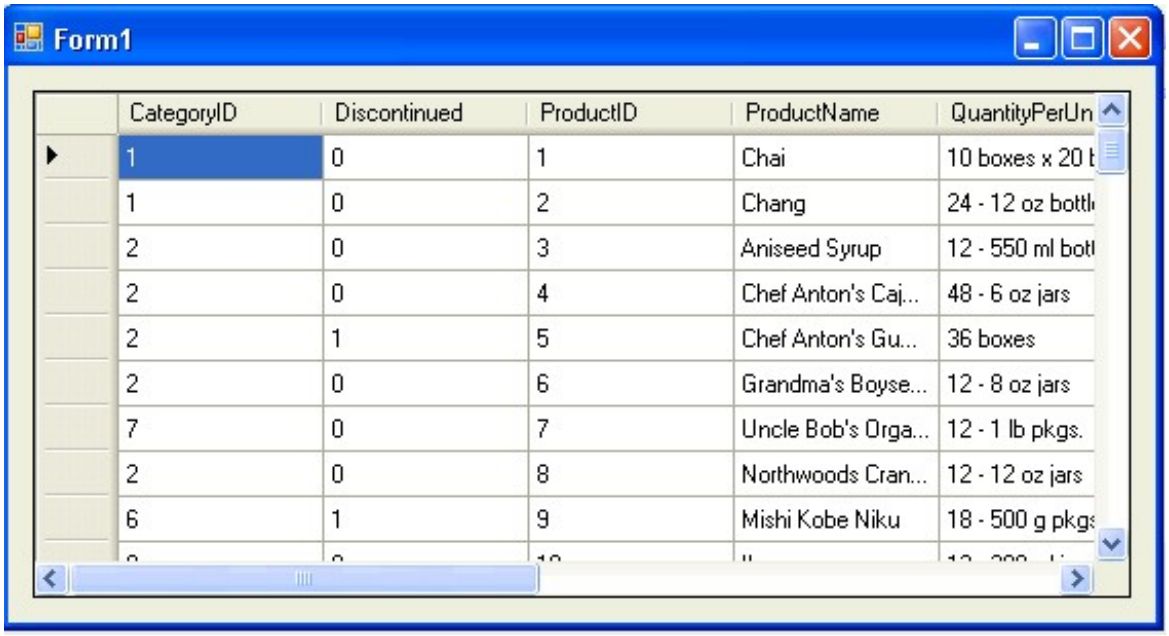
    Products

    table will be displayed in the

    DataGrid

    .

    Figure 8.591. DataGrid



The task is now complete.

8.12. Using Microsoft Entity Frameworks to Access Firebird Schema Objects with Virtuoso

Abstract

This section details the steps required to provide Microsoft Entity Framework access to Firebird Schema Objects using the OpenLink Virtuoso Universal Server. This is achieved by linking the required Firebird Schema objects into Virtuoso using its built in Virtual Database engine, and then using the Virtuoso ADO.NET Entity Framework provider to query the remote Firebird Schema objects linked into the Virtuoso Server.

Prerequisites. The following prerequisites must be in place for this solution to be possible.

Firebird DBMS. A Firebird DBMS hosting the required Schema Objects needs to be available. In this section the *Firebird employee* database will be used to demonstrate the process.

A Firebird DBMS hosting the required Schema Objects needs to be available. In this section the *Firebird employee* database will be used to demonstrate the process.

ODBC Driver for Firebird. A Firebird ODBC Driver is required for Linking the Firebird Schema Objects into the Virtuoso Server. The *native Firebird ODBC Driver* will be used in this section, for which a functional ODBC Data source name of "fire" will be assumed to exist on the machine hosting the Virtuoso Server.

A Firebird ODBC Driver is required for Linking the Firebird Schema Objects into the Virtuoso Server. The *native Firebird ODBC Driver* will be used in this section, for which a functional ODBC Data source name of "fire" will be assumed to exist on the machine hosting the Virtuoso Server.

Virtuoso Universal Server. An Virtuoso installation including the Virtuoso Universal Server and ADO.NET Entity Framework Provider is required. The Virtuoso components used must be Release 5.10.x or above, this being the minimum version containing support for Microsoft Entity Frameworks

An Virtuoso installation including the Virtuoso Universal Server and ADO.NET Entity Framework Provider is required. The Virtuoso components used must be Release 5.10.x or above, this being the minimum version containing support for Microsoft Entity Frameworks

Microsoft Visual Studio 2008 SP1. Microsoft Visual Studio 2008 Service Pack 1 is required, this being the only version containing the necessary Entity Framework support available at the time of writing.

Microsoft Visual Studio 2008 Service Pack 1 is required, this being the only version containing the necessary Entity Framework support available at the time of writing.

Tasks

Ensure Firebird Primary Keys (PKs) are not nullable. The Visual Studio 2008 Entity Data Model (EDM) requires that all primary keys are *NOT* Nullable, and will fail to generate an EDM if any are. Thus ensure any tables to be used are defined as not nullable in the Firebird database schema before attempting to generate an EDM. In the case of the Firebird employee database all tables are not nullable, thus this should not be an issue in this case.

The Visual Studio 2008 Entity Data Model (EDM) requires that all primary keys are *NOT* Nullable, and will fail to generate an EDM if any are. Thus ensure any tables to be used are defined as not nullable in the Firebird database schema before attempting to generate an EDM. In the case of the Firebird employee database all tables are not nullable, thus this should not be an issue in this case.

8.12.1. Install and configure the Firebird ODBC Driver

The Virtuoso Virtual Database engine uses ODBC as the connectivity mechanism for linking remote database objects into its local schema. Thus a Firebird ODBC Driver must be available with a suitably configured DSN for connecting to the target database.

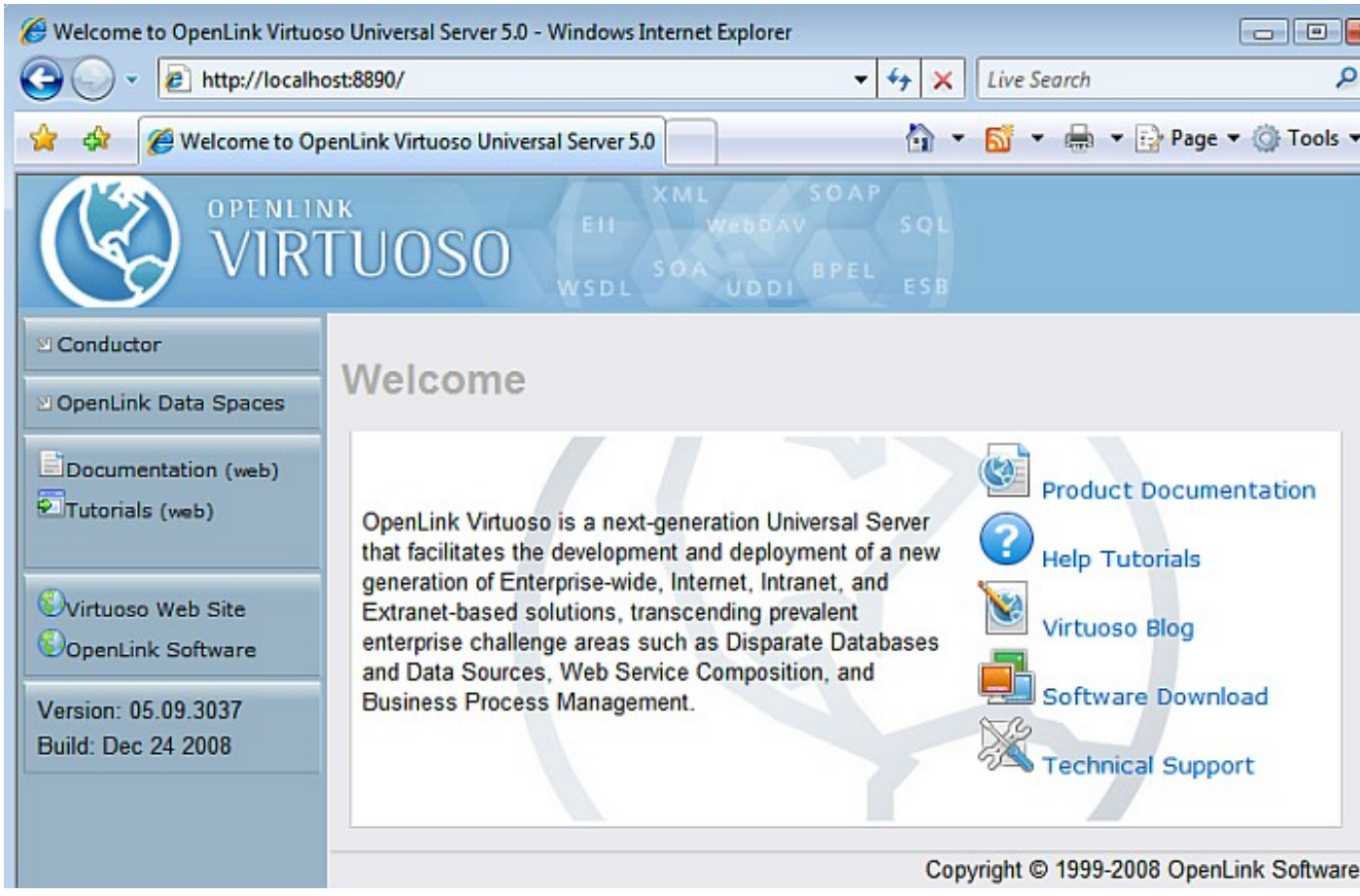
8.12.2. Install and configure OpenLink Virtuoso Universal Server

- Install and configure OpenLink Virtuoso Universal Server

8.12.3. Linking Firebird tables into OpenLink Virtuoso

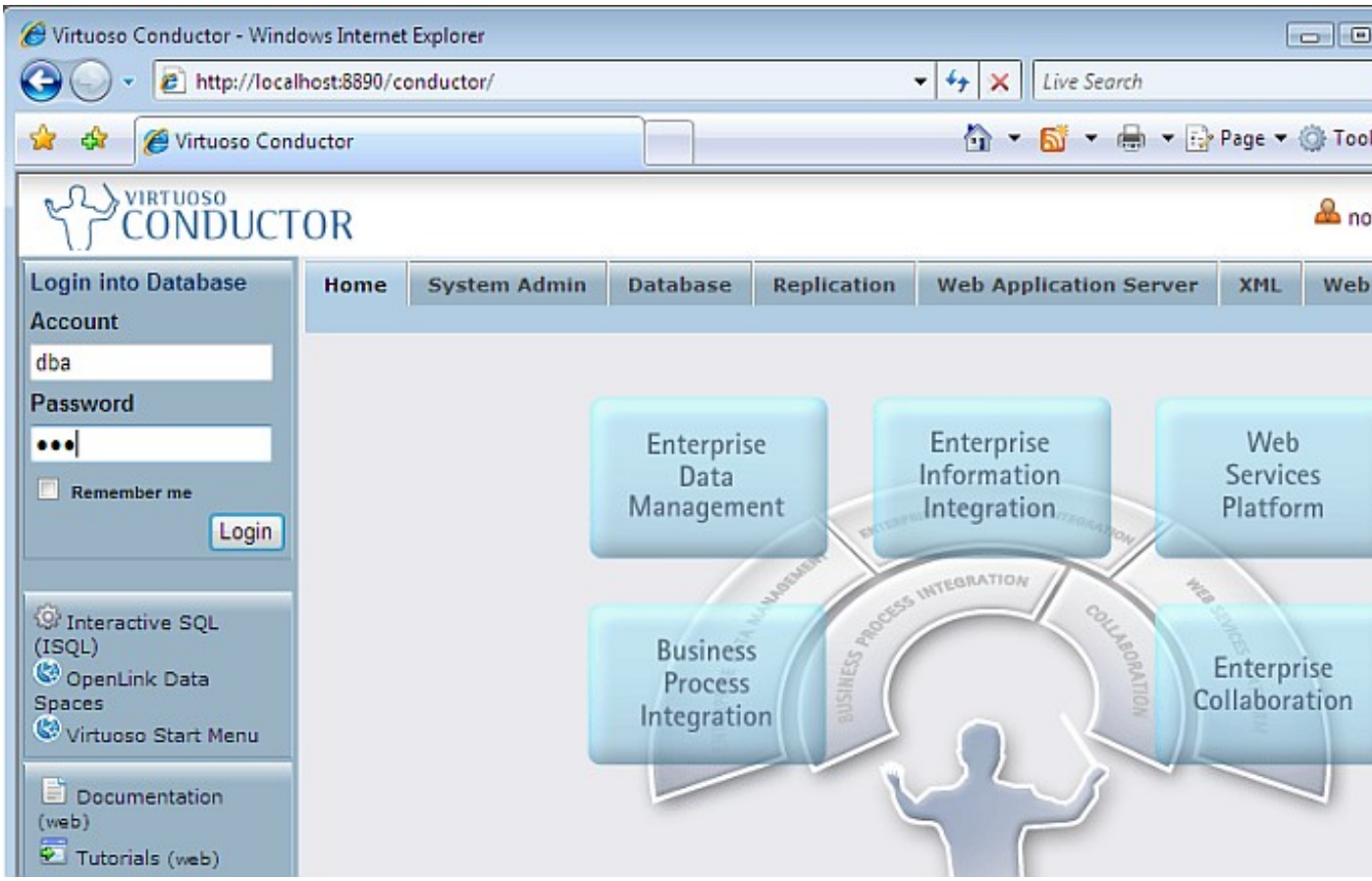
1. Start the Virtuoso Web User Interface

   Figure 8.592. Start



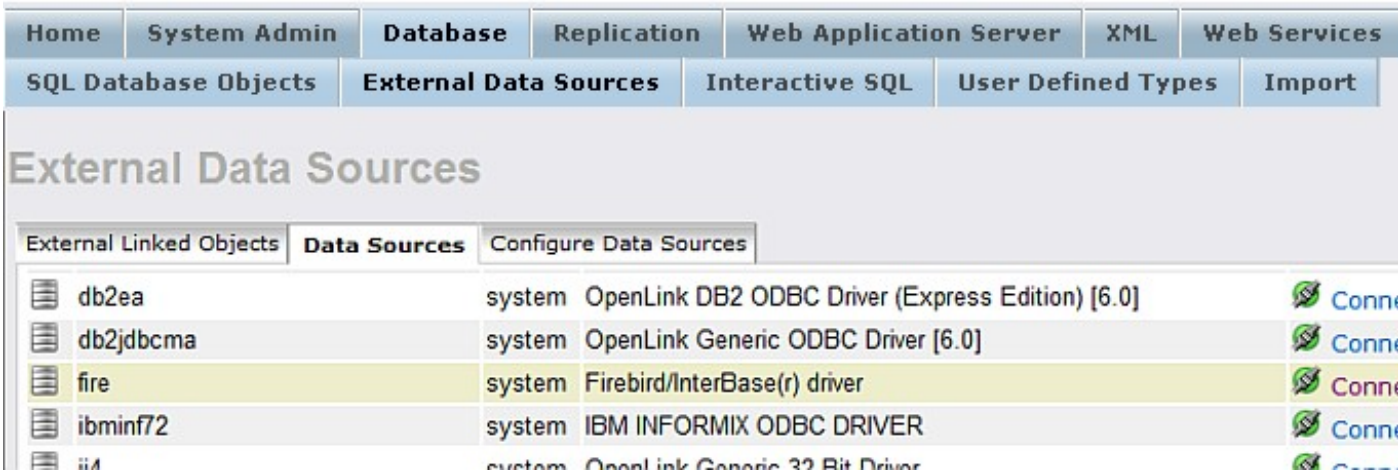
2. Select the "Conductor" link in the Left Frame and Login as the "dba" user.

Figure 8.593. Conductor



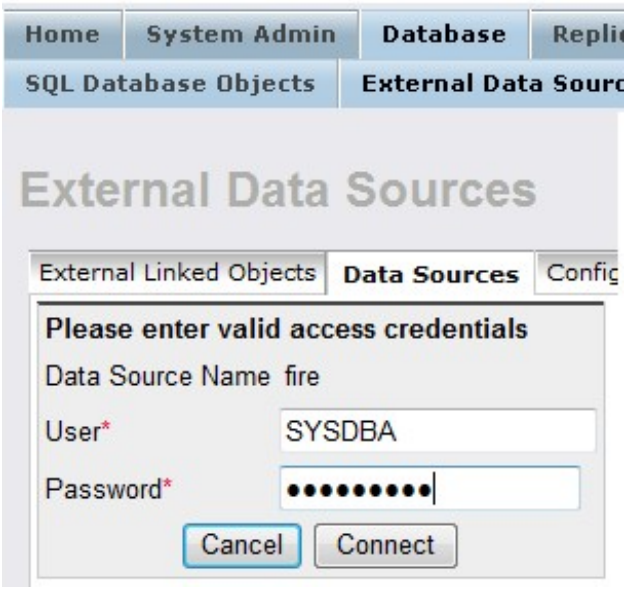
3. Select the "Databases" -> "External Data Source" -> "Data Sources" tab menu items

Figure 8.594. Databases



4. Select the "Connect" button for the "fire" Firebird DSN.

Figure 8.595. Connect

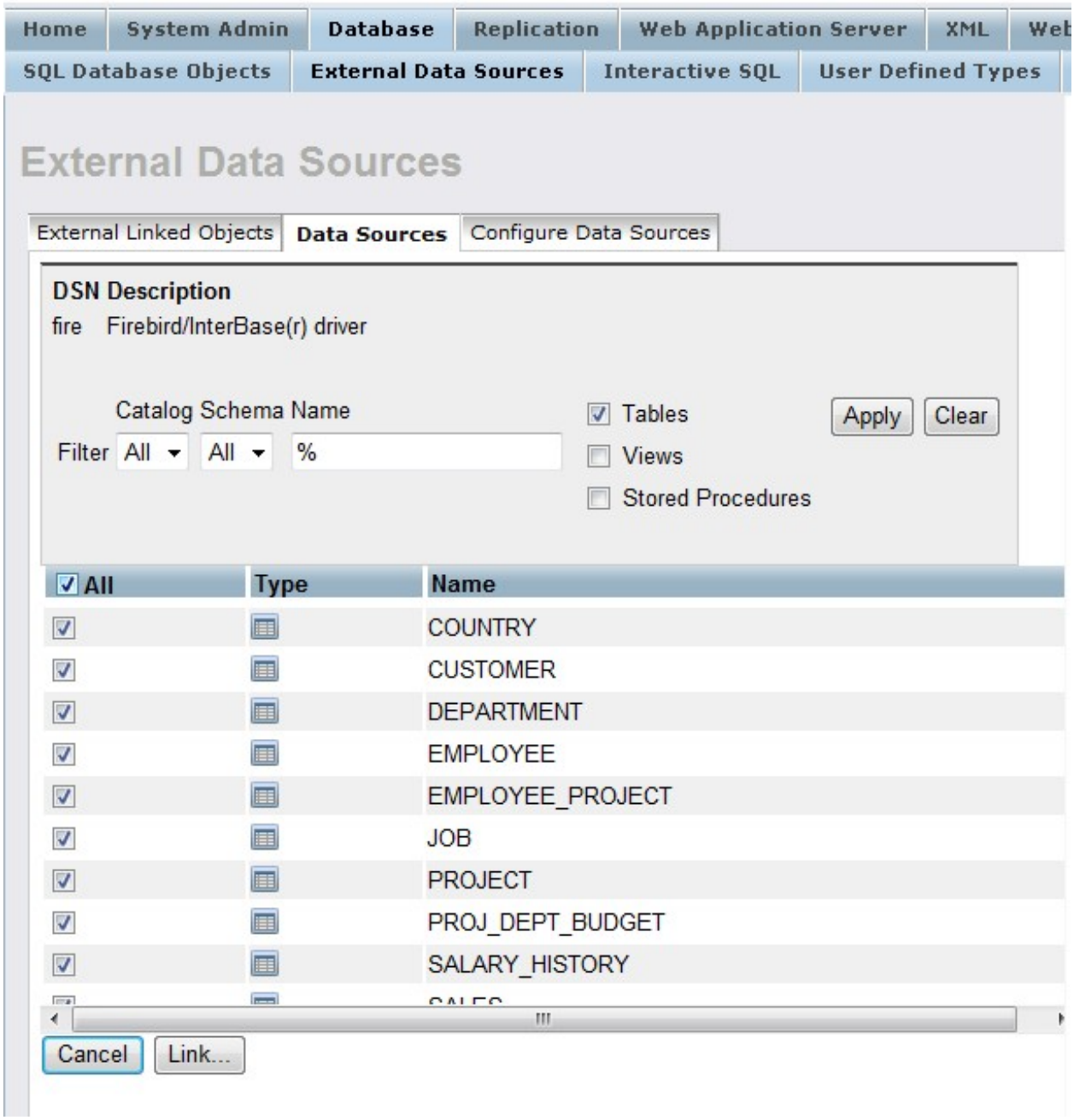


5. On successful connection Select the "Link Objects" button to obtain a list of available tables

Figure 8.596. Link Objects

6. Select all the tables that are part of the "employee" catalog.

Figure 8.597. Select all tables

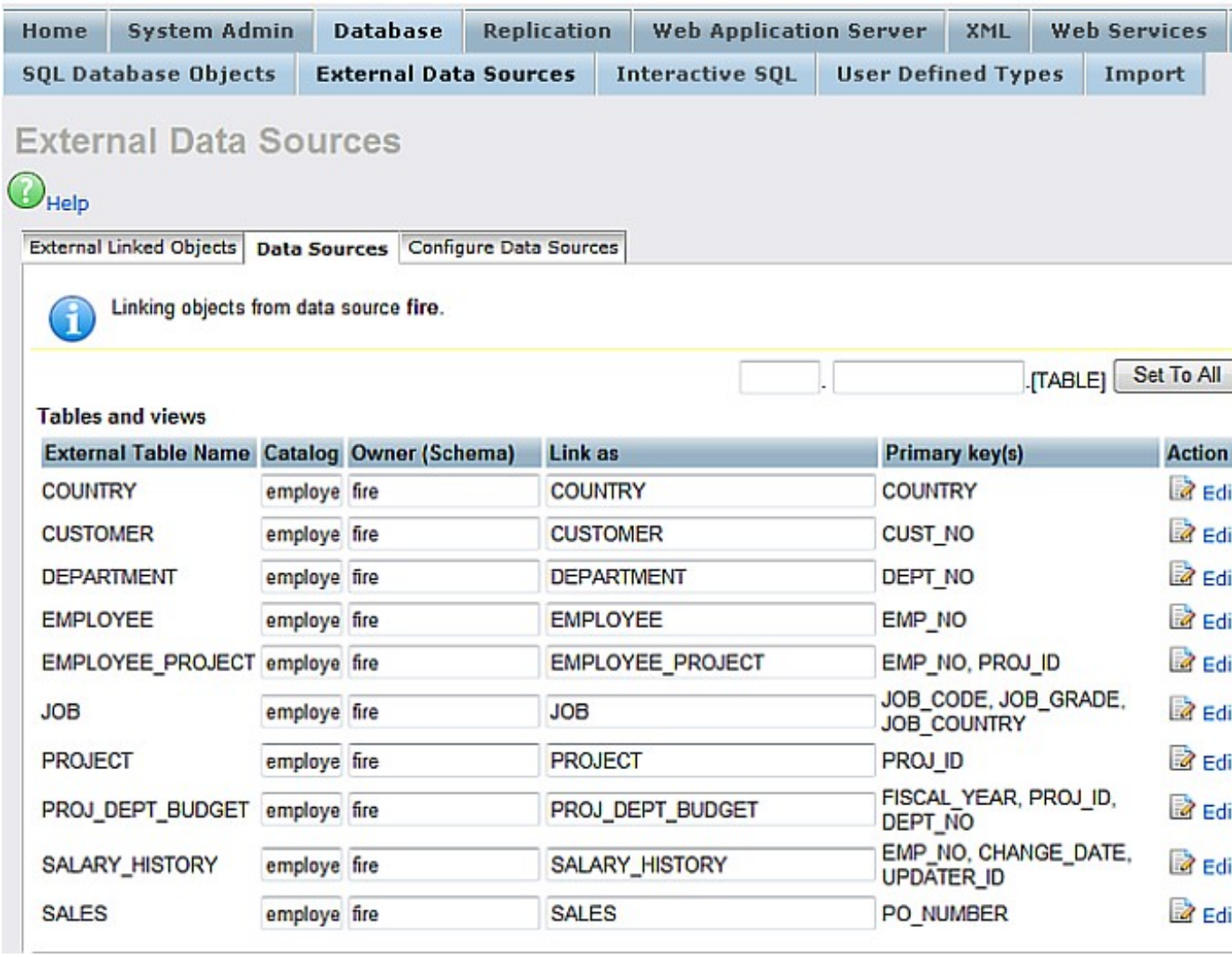


7. Change the Catalog for each table to be "employee" using the "Set All" button.

Figure 8.598. Catalog

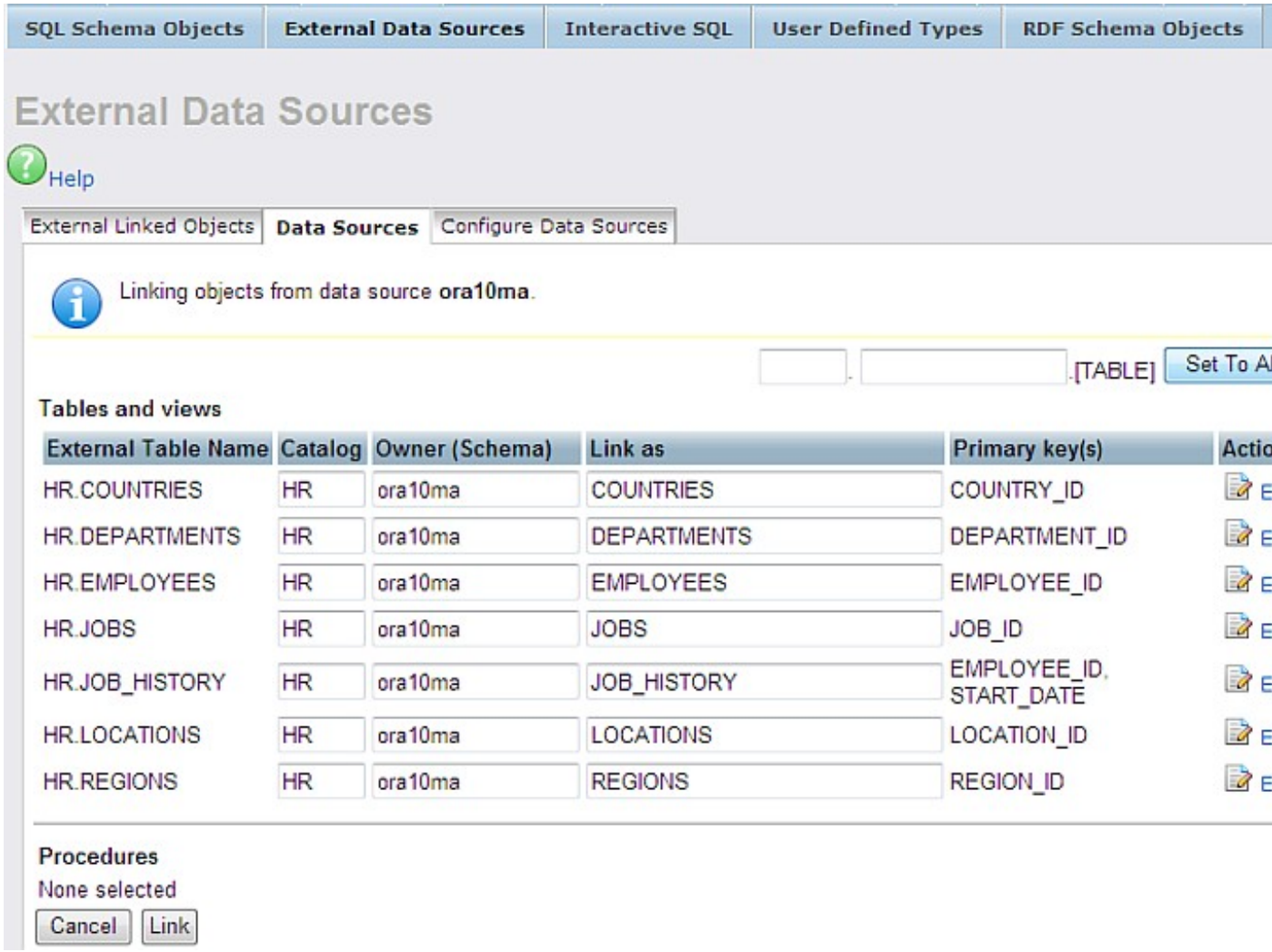
8. All the catalog names are changed to be "employee".

Figure 8.599. catalog names



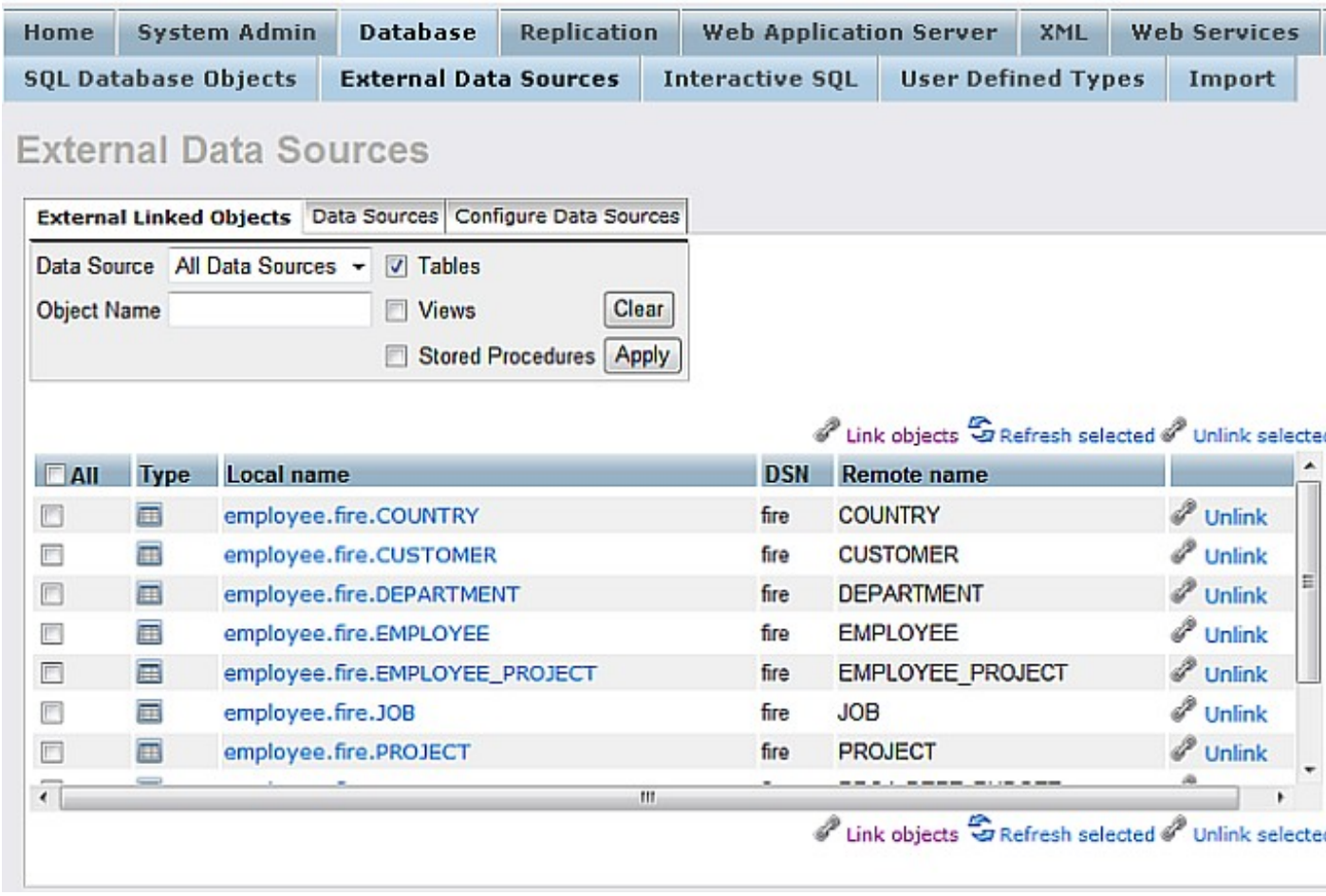
9. Select the "Link" button to link the selected tables into Virtuoso

Figure 8.600. "Link" button



10. On completion of the Link process the tables will be displayed in the "External Linked Objects" tab.

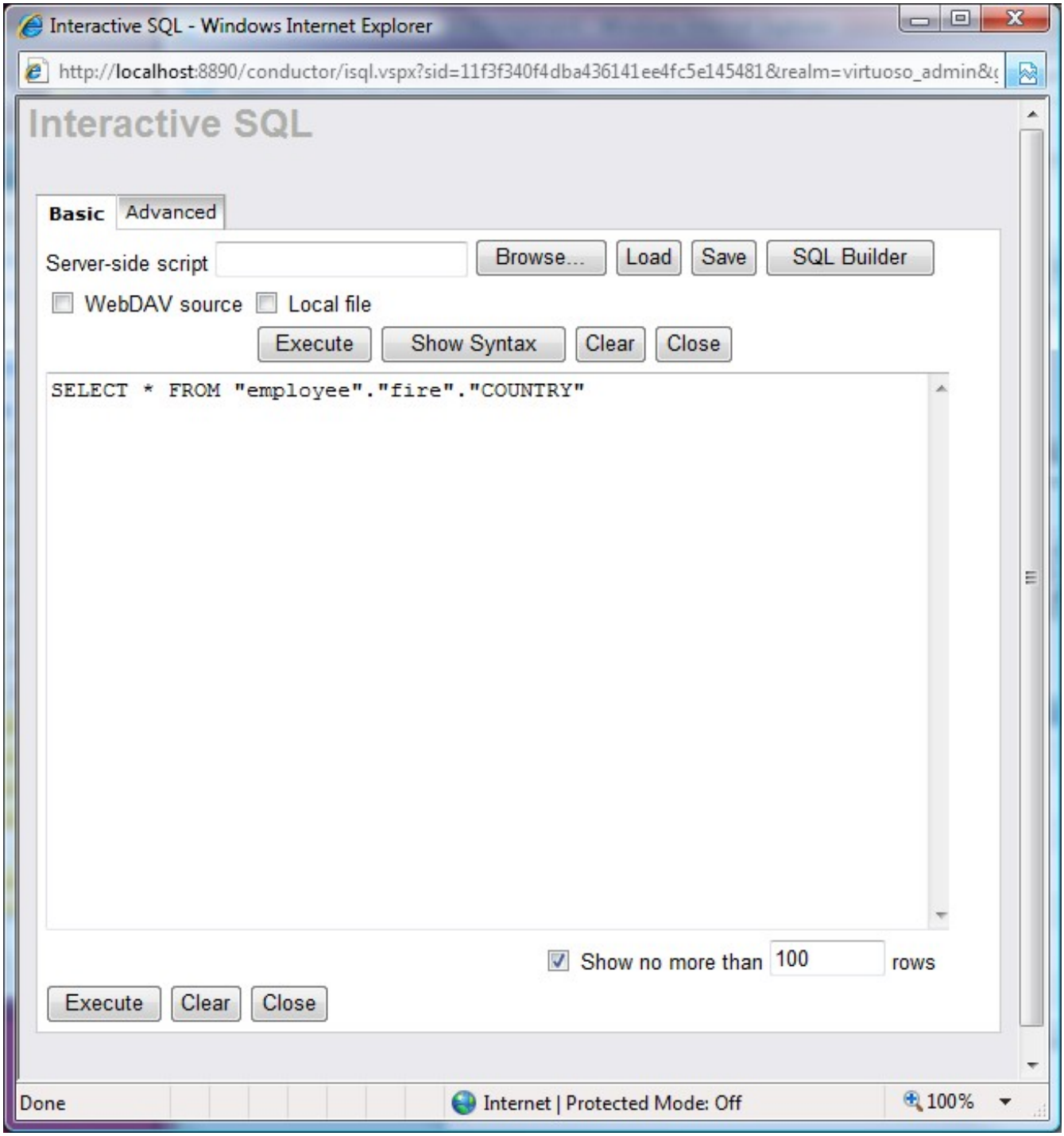
Figure 8.601. Completion



11. At this point you can test the remotely linked tables by clicking on the link that accompanies each table. e.g. employee.fire.COUNTRY.

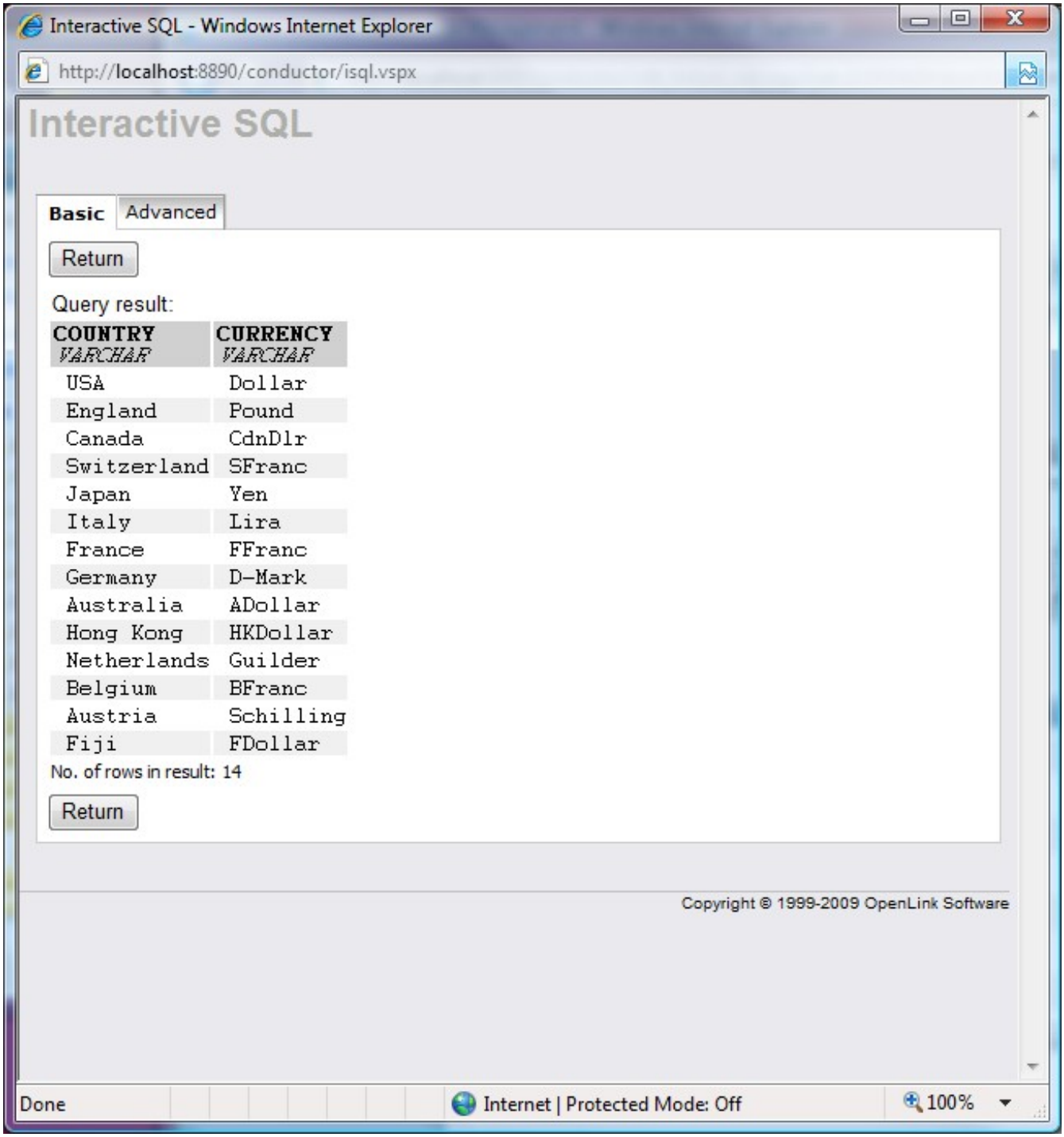
    This will display the Interactive ISQL interface which will have been already populated with a suitable SQL statement.

Figure 8.602. remotely linked tables



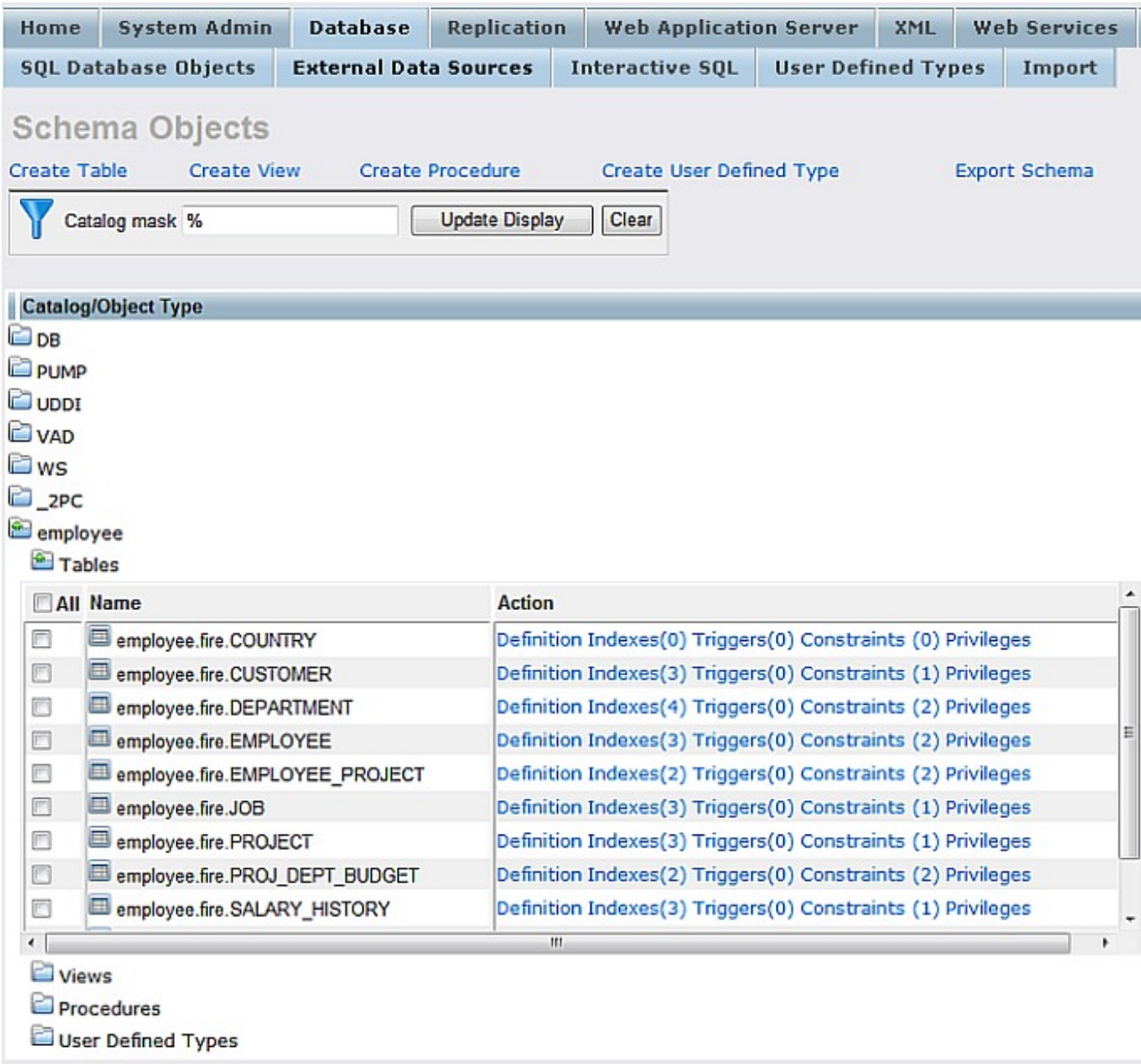
12. Select Execute to see data from the remotely linked table.

Figure 8.603. Execute



13. The tables can also be viewed as part of the Virtuoso "SQL Schema Objects" under the "employee" catalog name.

Figure 8.604. SQL Schema Objects



The Link process is now complete enabling the tables to be queried as if part of the Virtuoso Schema.

8.12.4. Creating EDM in Visual Studio 2008

The following steps can be used to create an Entity Data Model (EDM) for the Firebird employee database:

1. Launch the Visual Studio 2008 SP1 IDE.

   Figure 8.605. Visual Studio 2008 SP1 IDE

2. Create a

   Web Application

   project by going to the

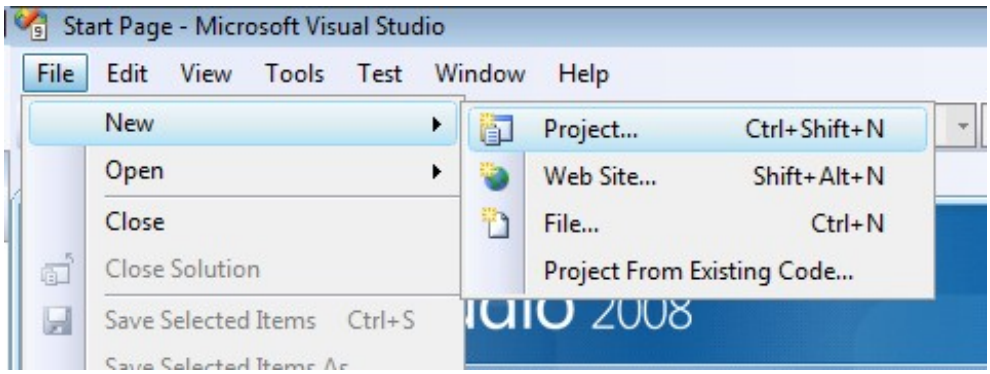
   File

   menu in Visual Studio and choosing

   New Project

   .

   Figure 8.606. Web Application



3. When the New Project window appears, choose either

   Visual Basic

   or

   Visual C#

   as your programming language.
4. Within the language category, click on

   Web

   and select

   ASP.NET Web Application

from the right-hand panel.

5. Choose a name for the project, for example

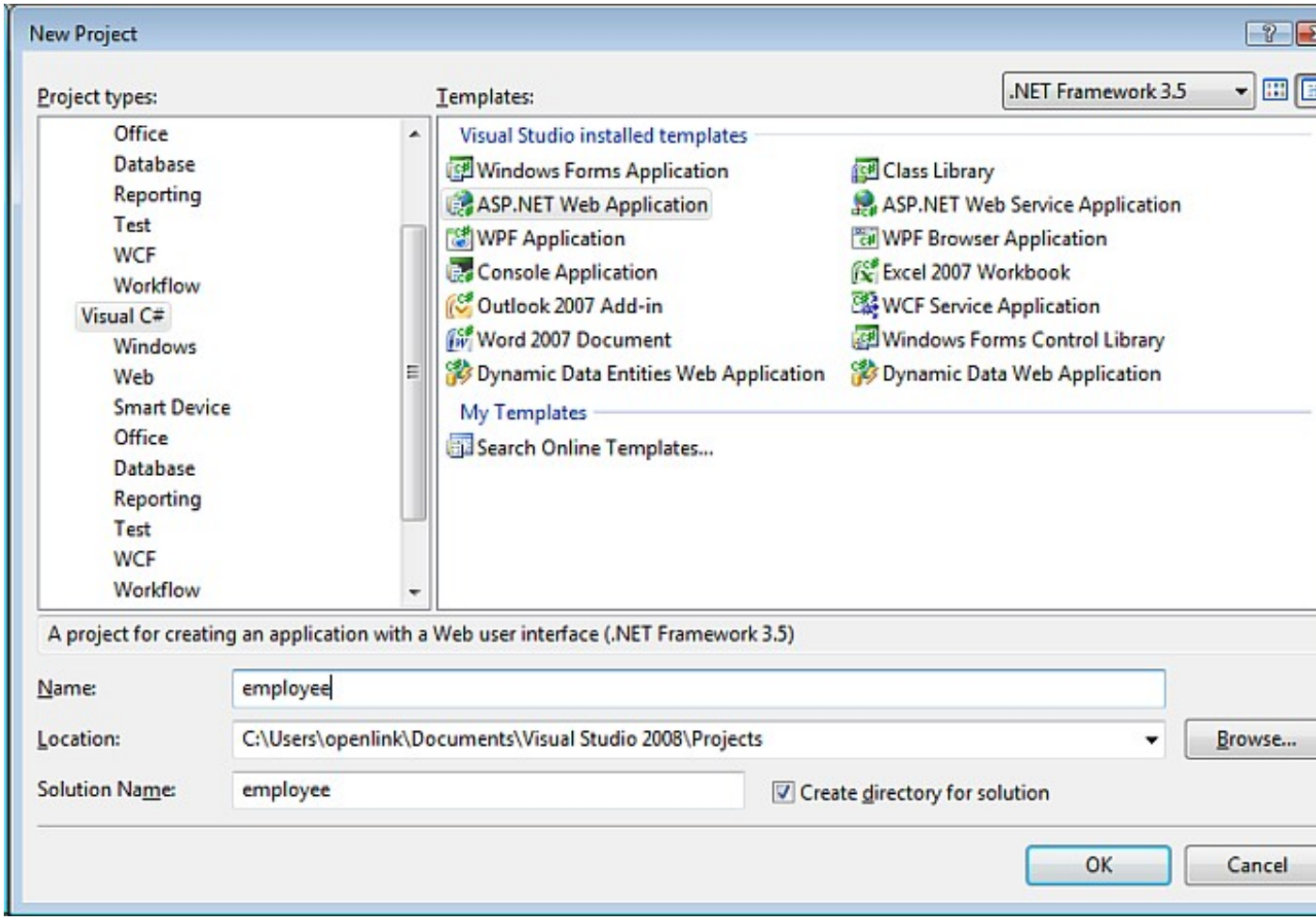
   employee

   , and click

   OK

   .

Figure 8.607. name for the project

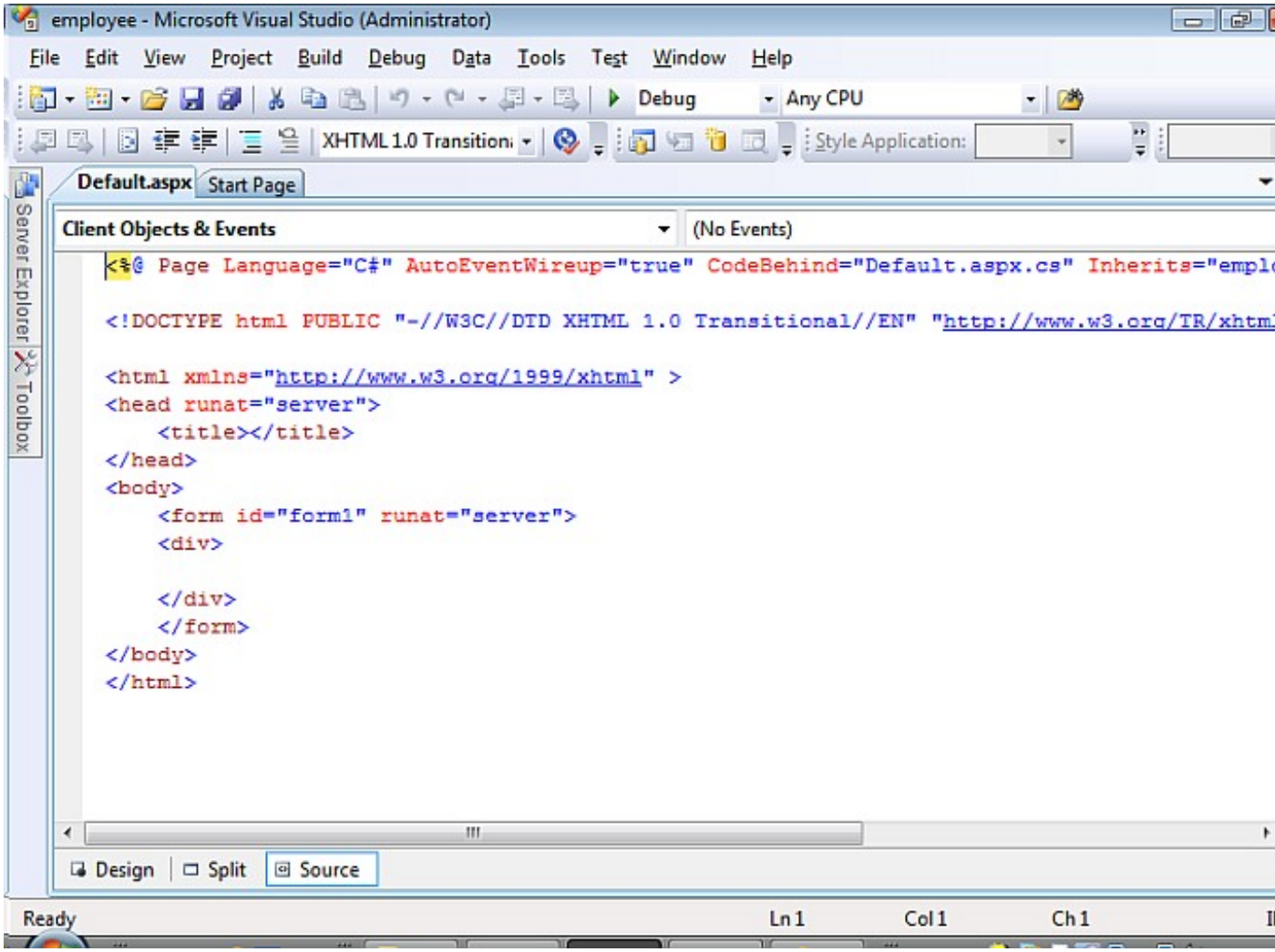


6. This will create a new project called

   employee

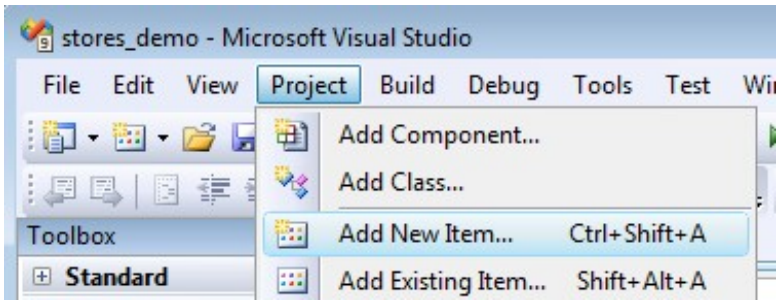
   .

Figure 8.608. create a new project



7. Select the Project -> Add New Item menu option.

Figure 8.609. employee



8. The

Add New Item

dialog will appear. Choose the

ADO.NET Entity Data Model

template. Give it the name

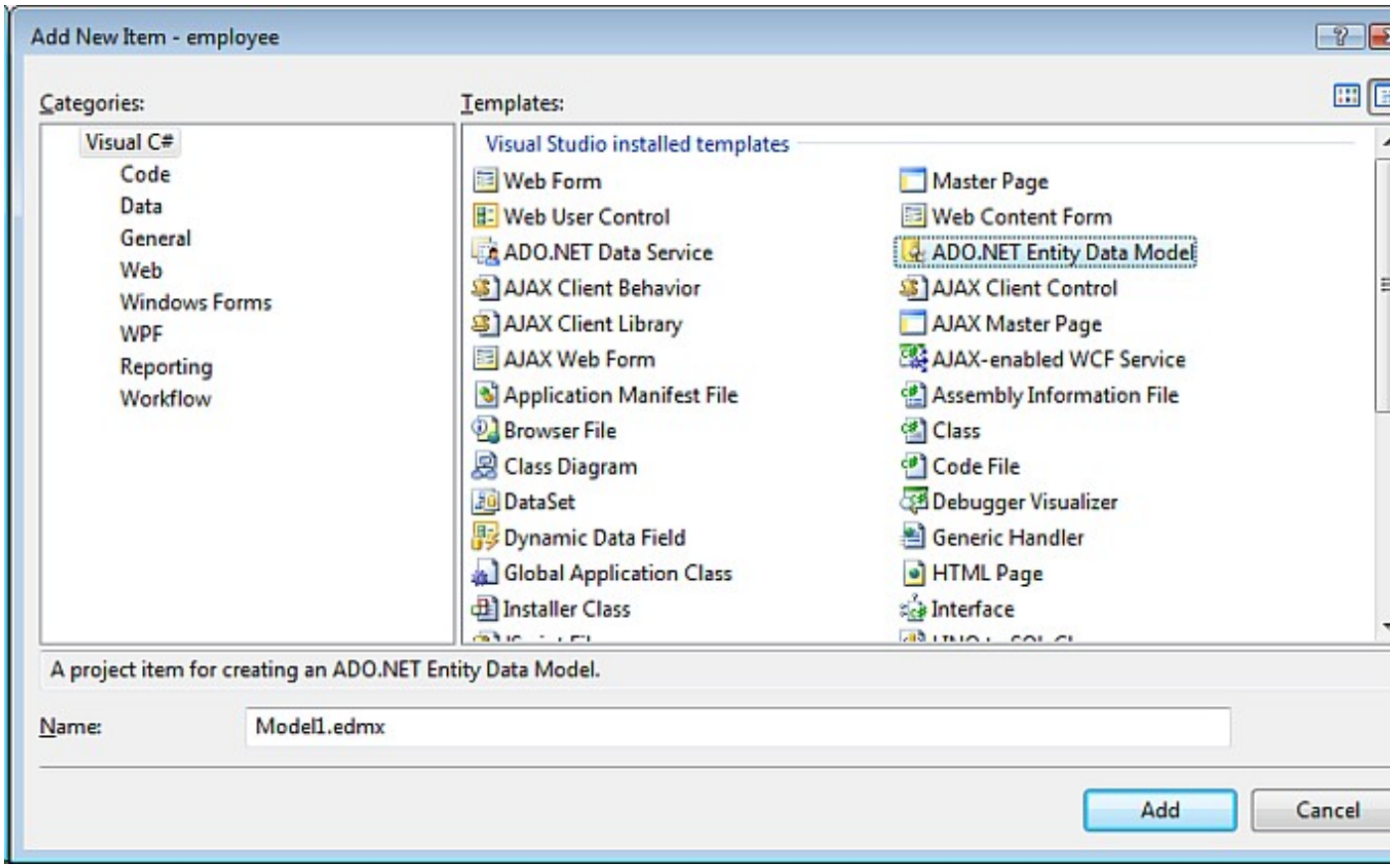
Model1.edmx

and click

Add

to start the creation of the ADO.Net Entity Data Model.

Figure 8.610. Add New Item



9. In the

Entity Data Model Wizard

dialog

Choose Model Contents

page select the

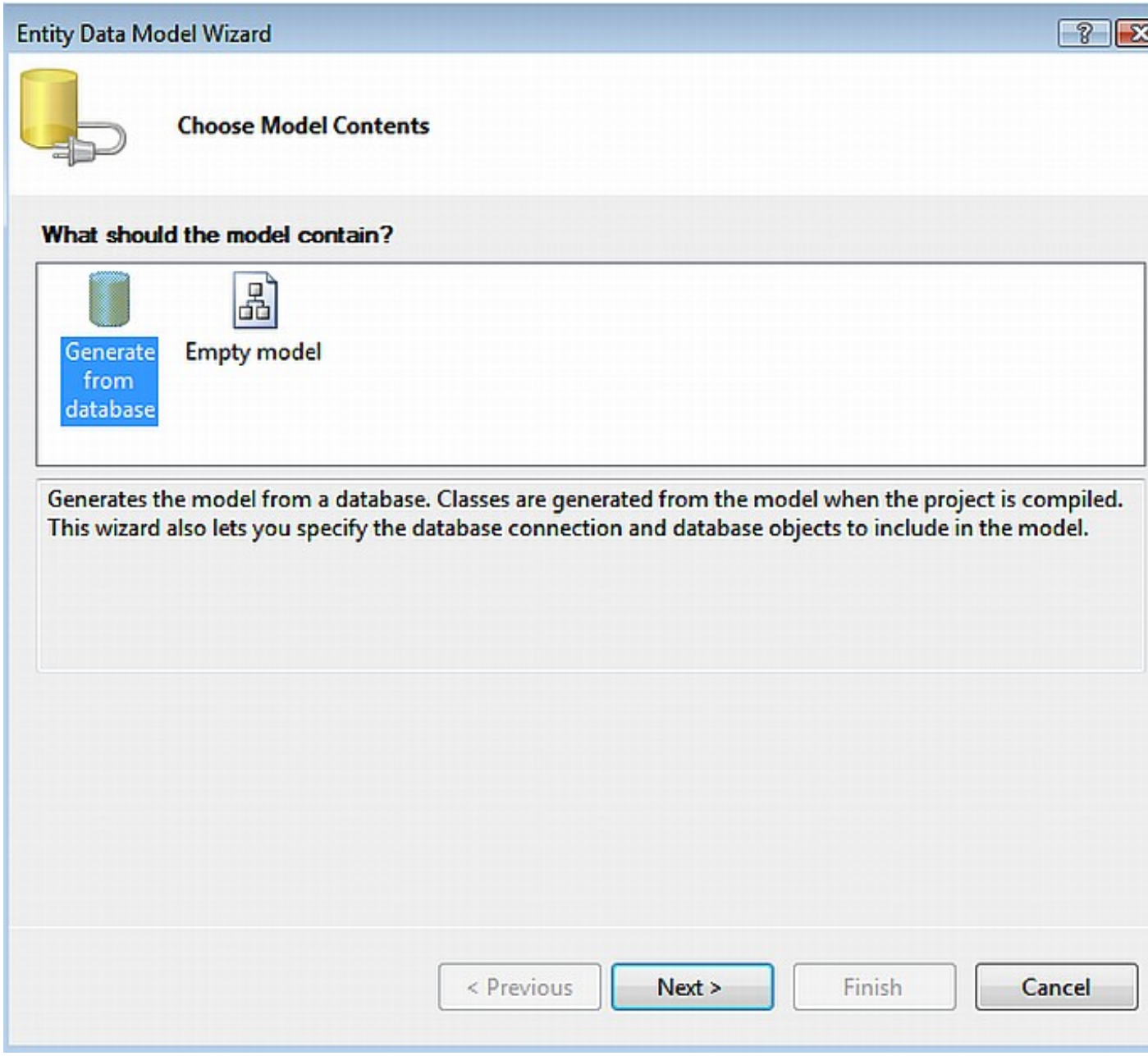
Generate from Database

model type and click

Next

.

Figure 8.611. Choose Model Contents



10. In the

    Entity Data Model Wizard

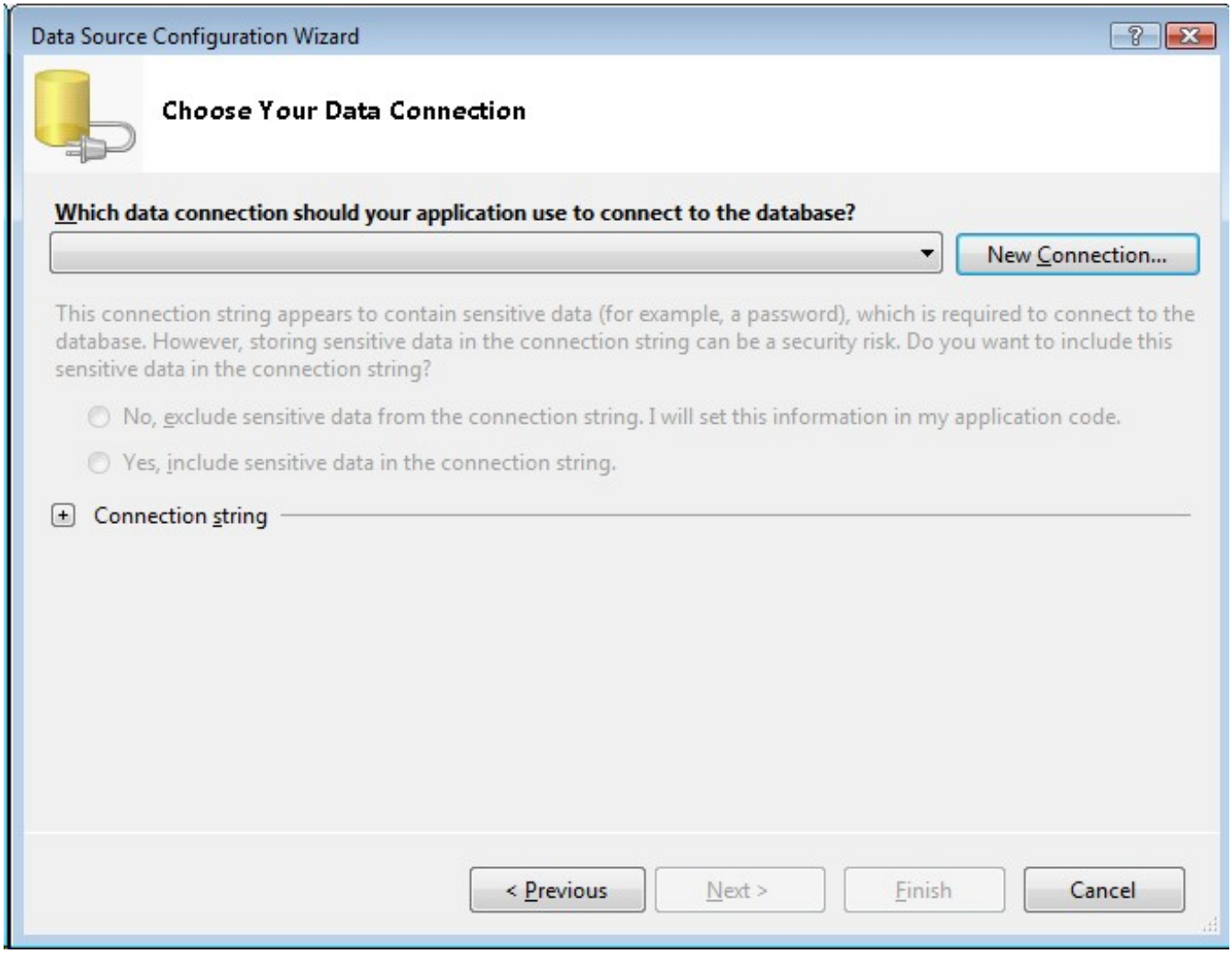
    dialog

    Choose your Data Connection

    page, select the

    New Connection button

Figure 8.612. Entity Data Model Wizard



11. In the

    Choose Data Source

    dialog, select the OpenLink

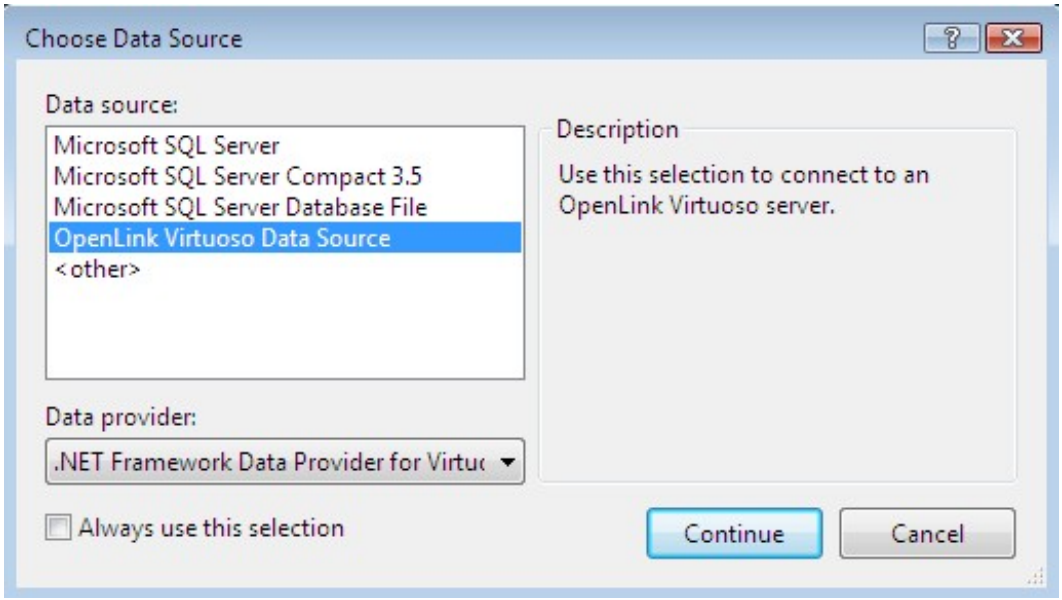
    Virtuoso Data Source

    from the list and click

    Continue

    .

    Figure 8.613. Choose Data Source



12. In the

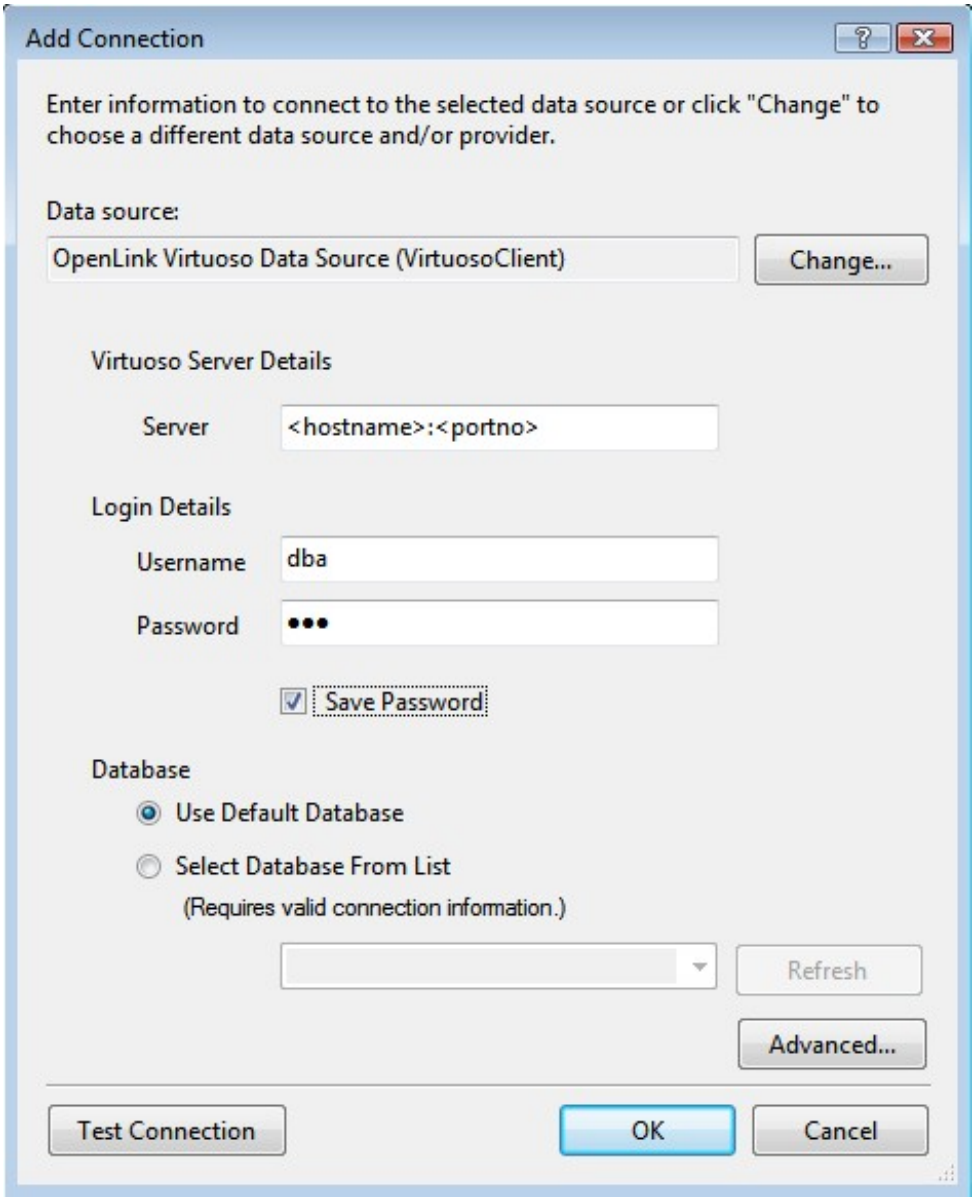
    Add Connection

    dialog, specify the

    hostname, portno, username and password

    for the target Virtuoso Server and check the Save Password check box.

    Figure 8.614. Connection Properties



13. Select the

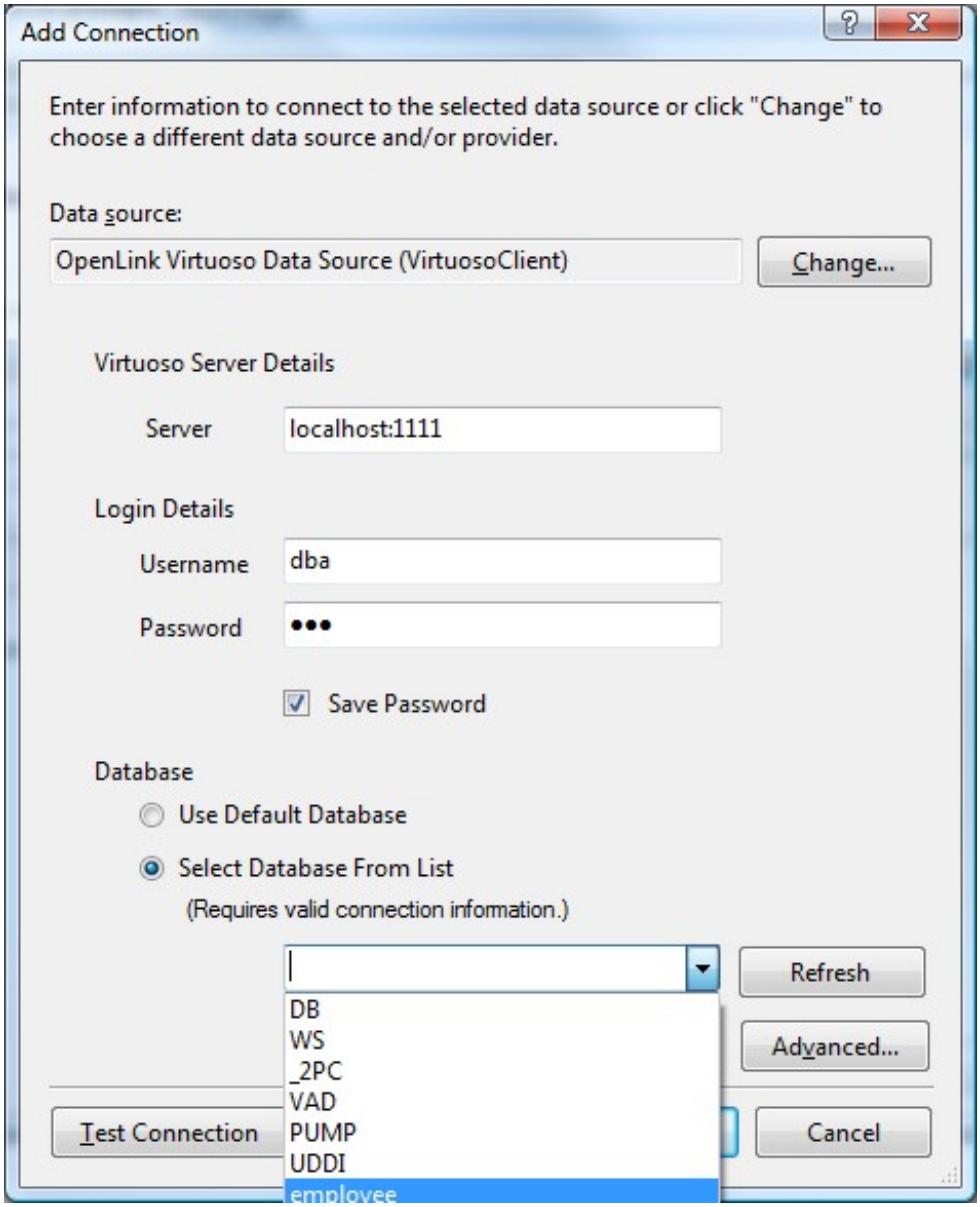
    Select Database From List

    radio button and choose the

    employee

    database from the drop down list.

Figure 8.615. Add connection

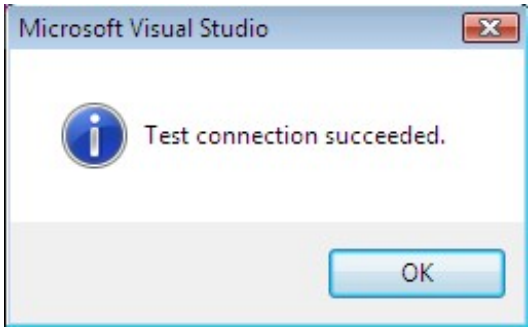


14. Press the

    Test Connection

    dialog to verify that the database is accessible.

    Figure 8.616. Test Connection



15. Set the

    entity connect string

name to

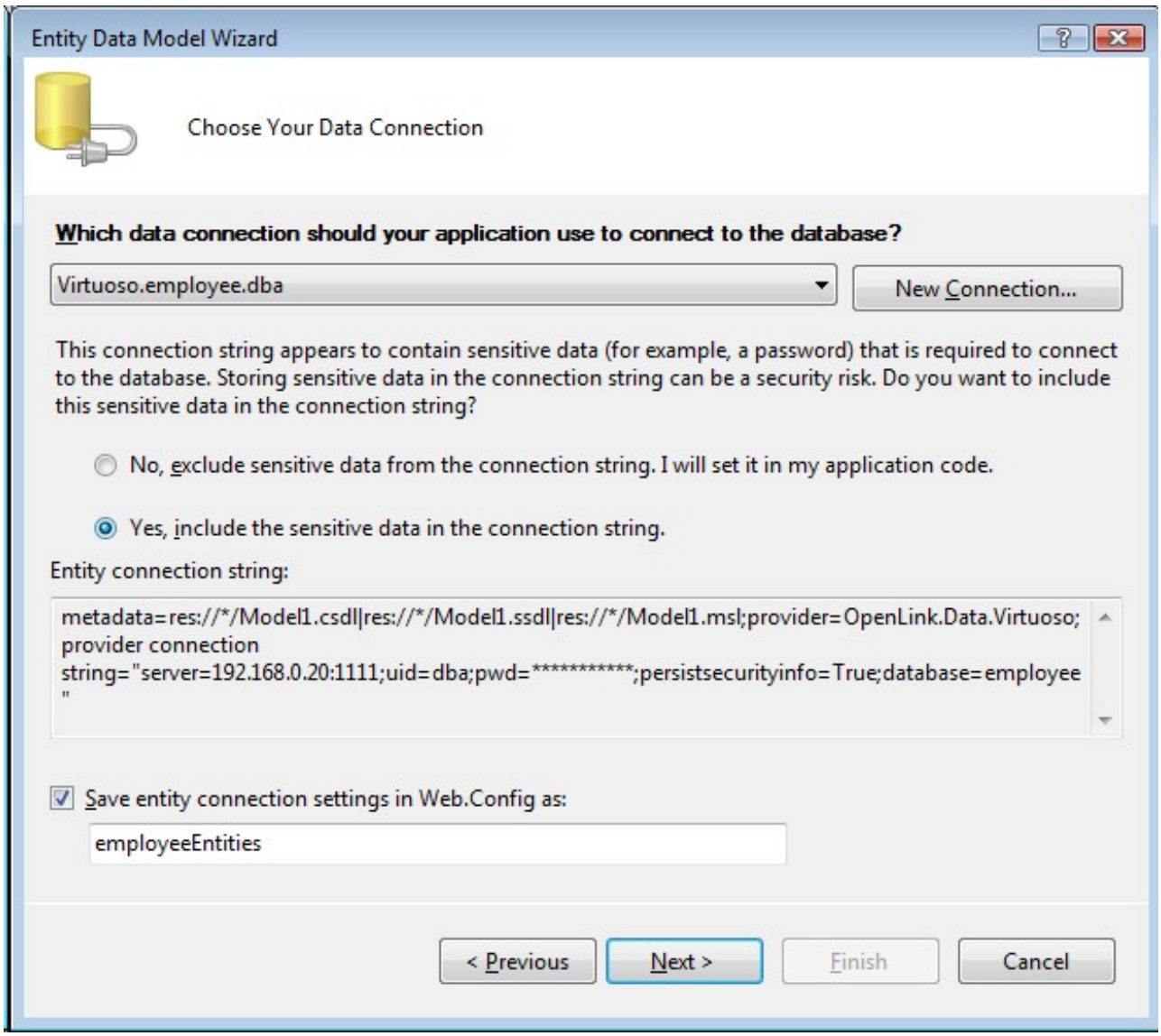
employeeEntities

(note this name as it is required in the section on creating and ADO.Net Data Service) and click

Next

.

Figure 8.617. entity connect string



16. In the

Choose your Database Objects

page select the

Tables

check box to select all tables in the employee catalog for addition to the Entity Data Model. Set the

Model Namespace

to

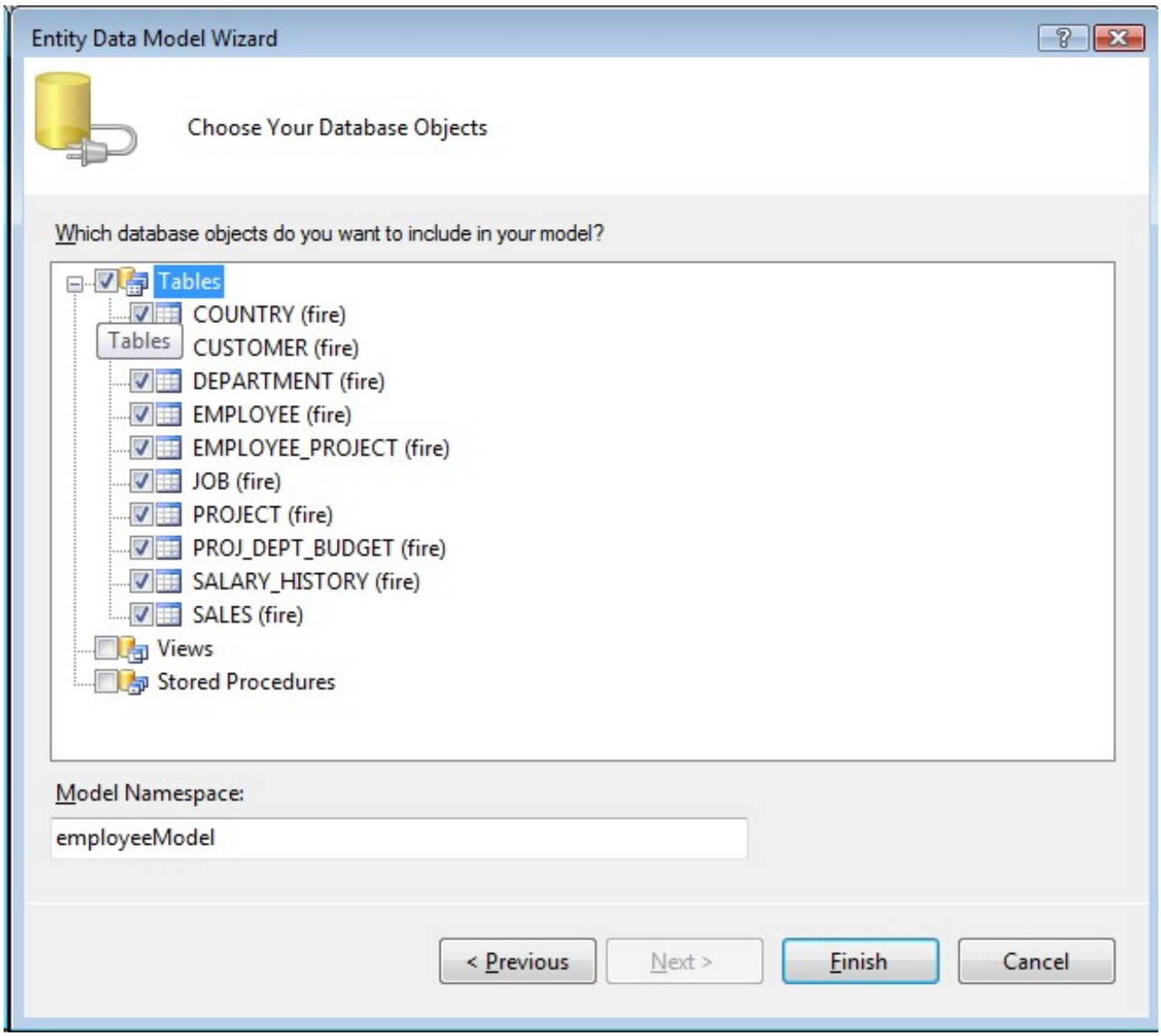
employeeModel

and click

Finish

.

Figure 8.618. Database Objects

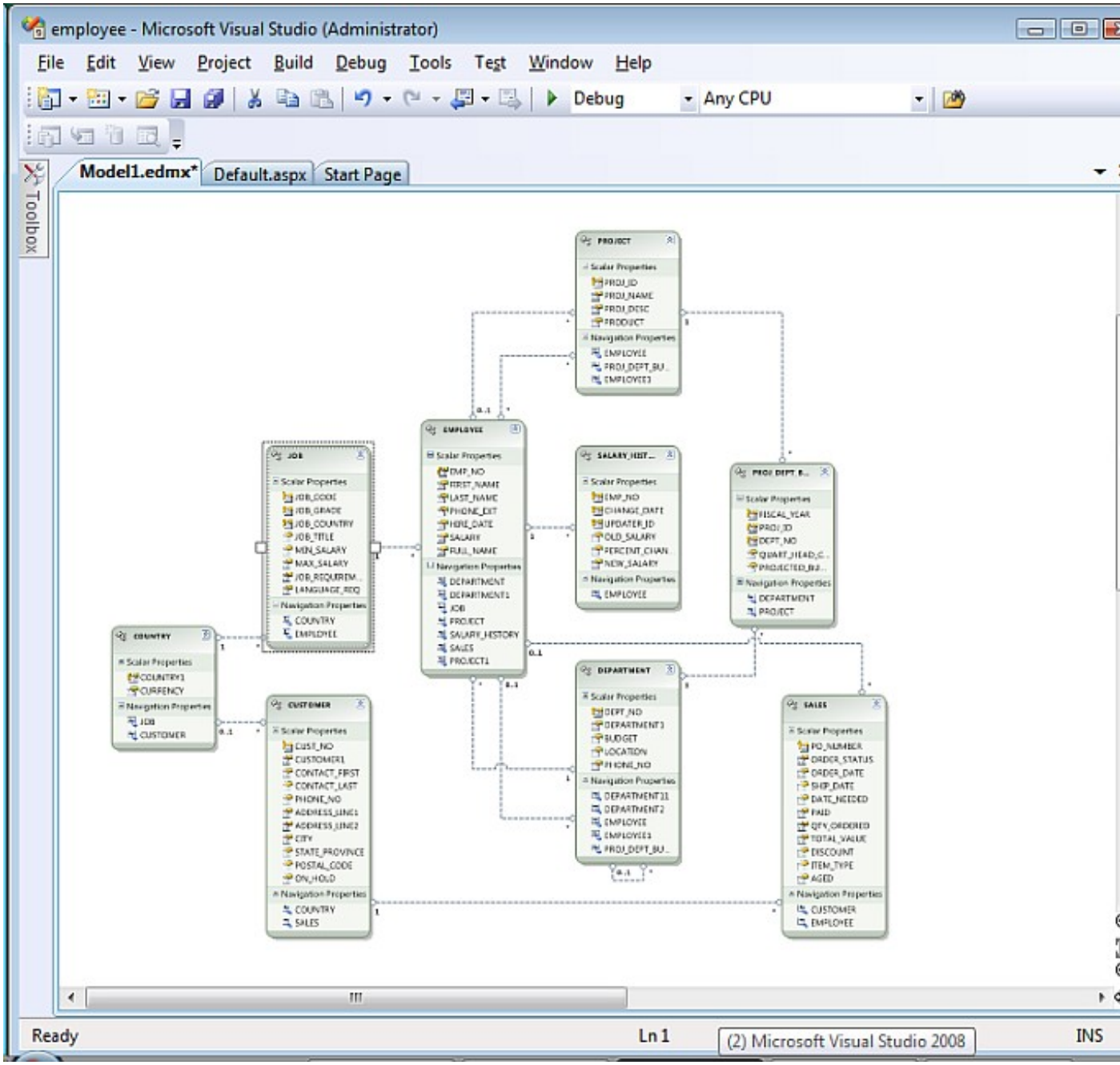


17. The

Model1.edmx

EDM will be created with the tables and relationships displayed in the Visual Studio IDE

Figure 8.619. Model1.edmx



Creation for the Entity Data Model for the Firebird employee database is now complete.

8.12.5. Using EDM to create Entity Framework based applications

Now that a Microsoft Entity Data Model has been created for the Firebird employee database, Entity Framework applications can be created to make use of it.

Entity Frameworks based ADO.NET Data Service

An ADO.Net Data Service for the Firebird tables can be created using the Entity Data Model created in the Creating EDM in Visual Studio 2008 section .

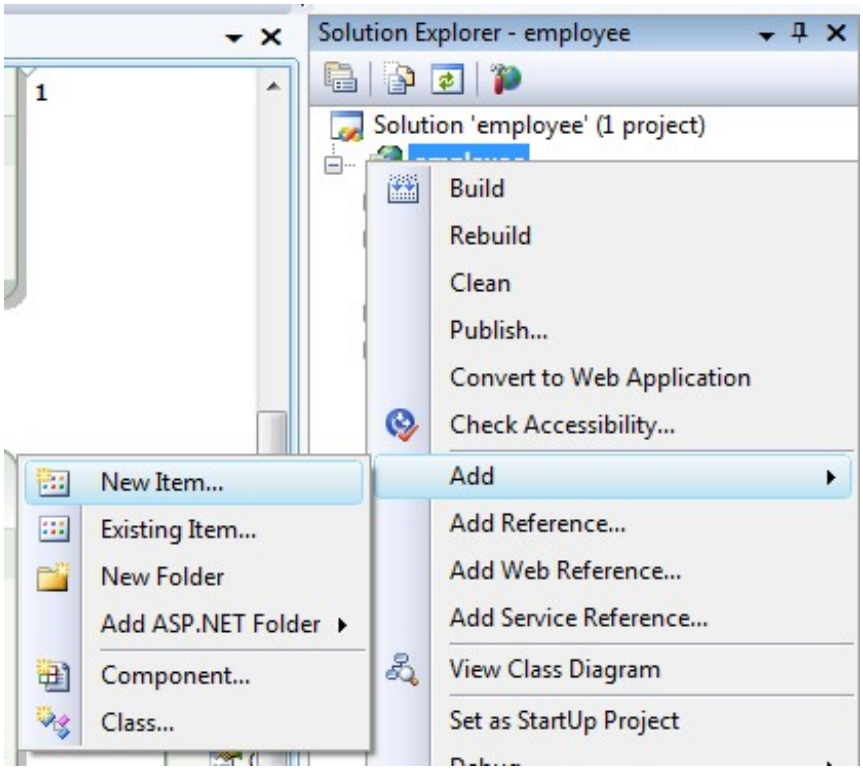
1. Open the

   employee

   project created in the Creating EDM in Visual Studio 2008 section .

2. Right click on the employee project name in the Solution Explorer pane, then select the Project -> Add New Item menu option.

Figure 8.620. employee



3. The

Add New Item

dialog will appear. Choose the

ADO.NET Data Service template

. Give it the name

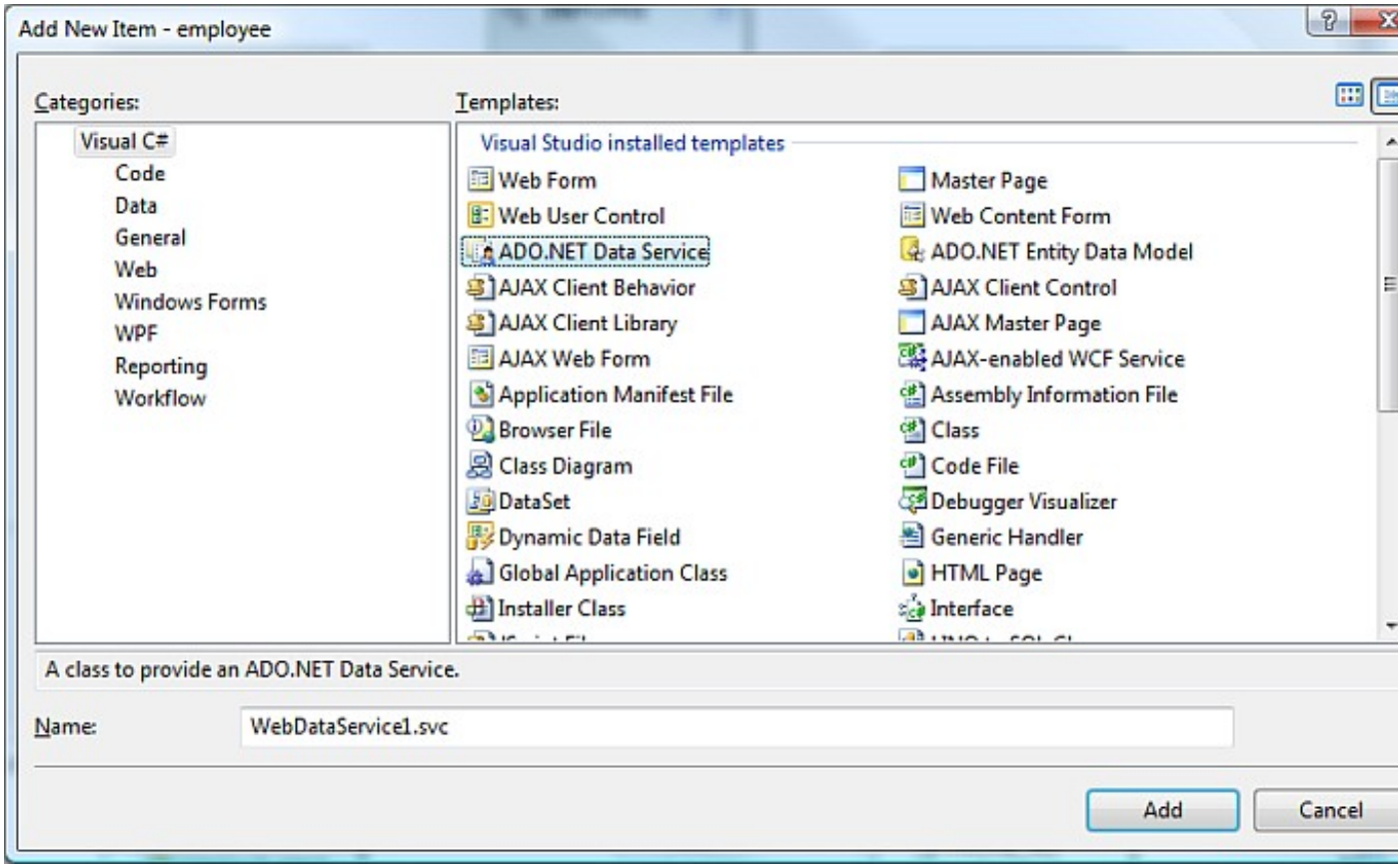
WebDataService1.svc

, and click

Add

to create the ADO.Net Data Service.

Figure 8.621. Add New Item



4. In the newly created

   WebDataService1.svc.cs

   Data Service file, add the data source class name of

   employeeEntities

   (note this is the name set in the Creating EDM in Visual Studio 2008 section) as the

   DataService

   name. Enable the access to the Data Service by adding the entry

   config.SetEntitySetAccessRule("", EntitySetRights.All);*

   in the

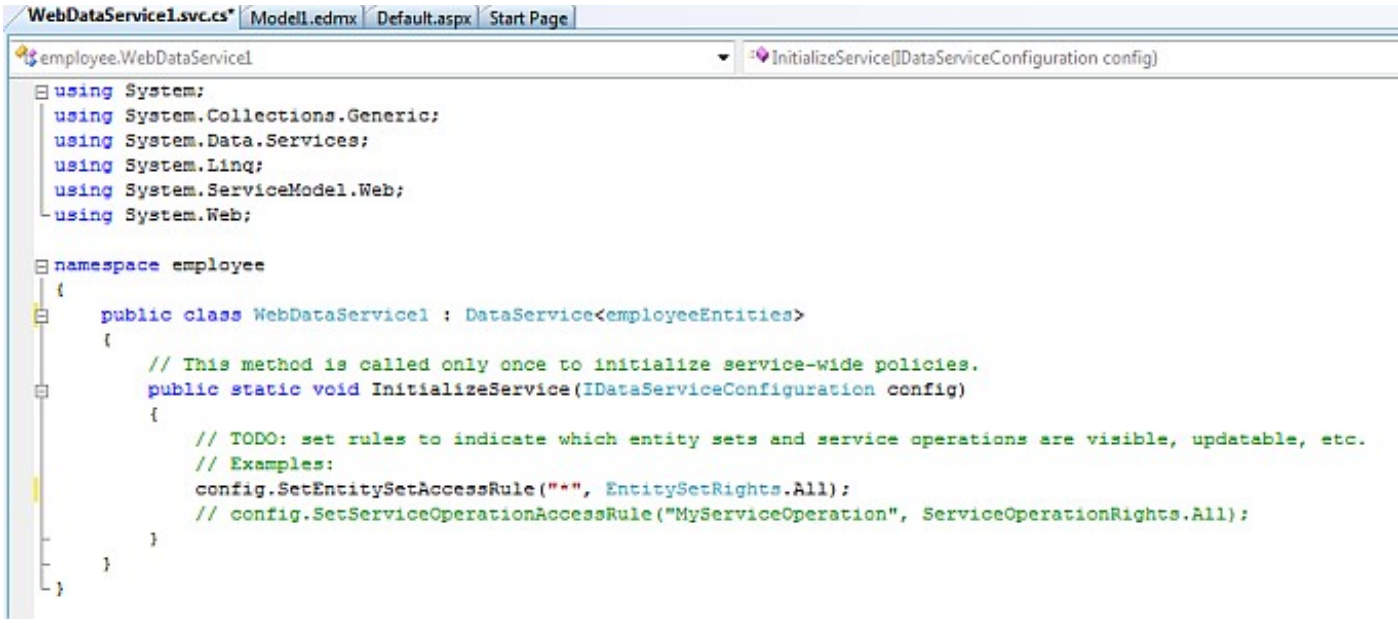
   InitializeService

   method.

```
// C#

using System;
using System.Web;
using System.Collections.Generic;
using System.ServiceModel.Web;
using System.Linq;
using System.Data.Services;

namespace SimpleDataService
{
    public class Northwind : DataService<employeeEntities>
    {
        public static void InitializeService(IDataServiceConfiguration  config)
```

```
            {
                config.SetEntitySetAccessRule("*", EntitySetRights.All);
            }
        }
    }
```

Figure 8.622. WebDataService1.svc.cs

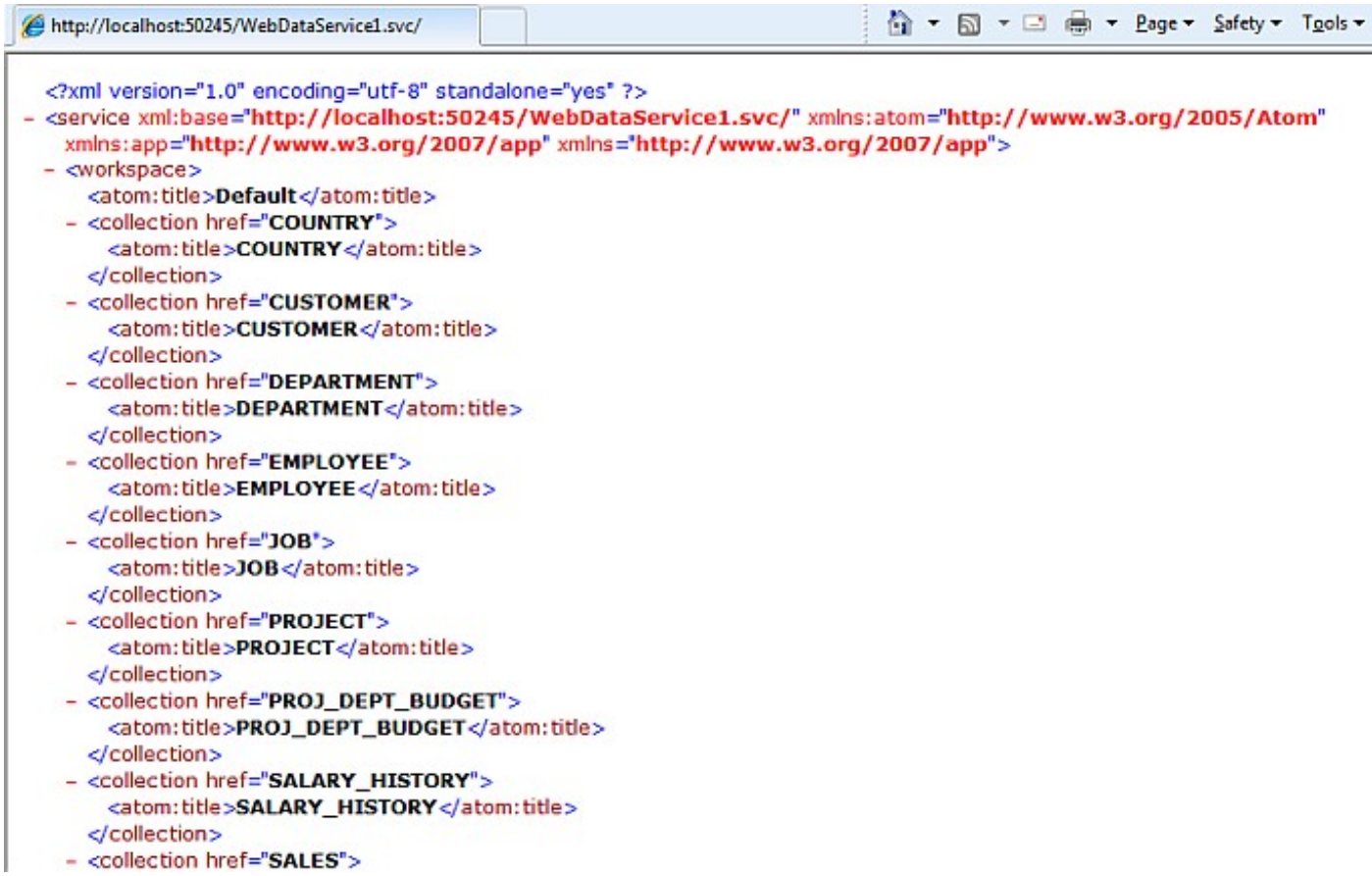


5. To test the Data Service, simply hit

   Ctrl+F5

   within Visual Studio. This will start the development web server, run the Data Services server inside and load a Web browser page displaying the list of available tables/entities for the employee database catalog.

   Figure 8.623. Data Service test



6. To access a specific entity instance like the

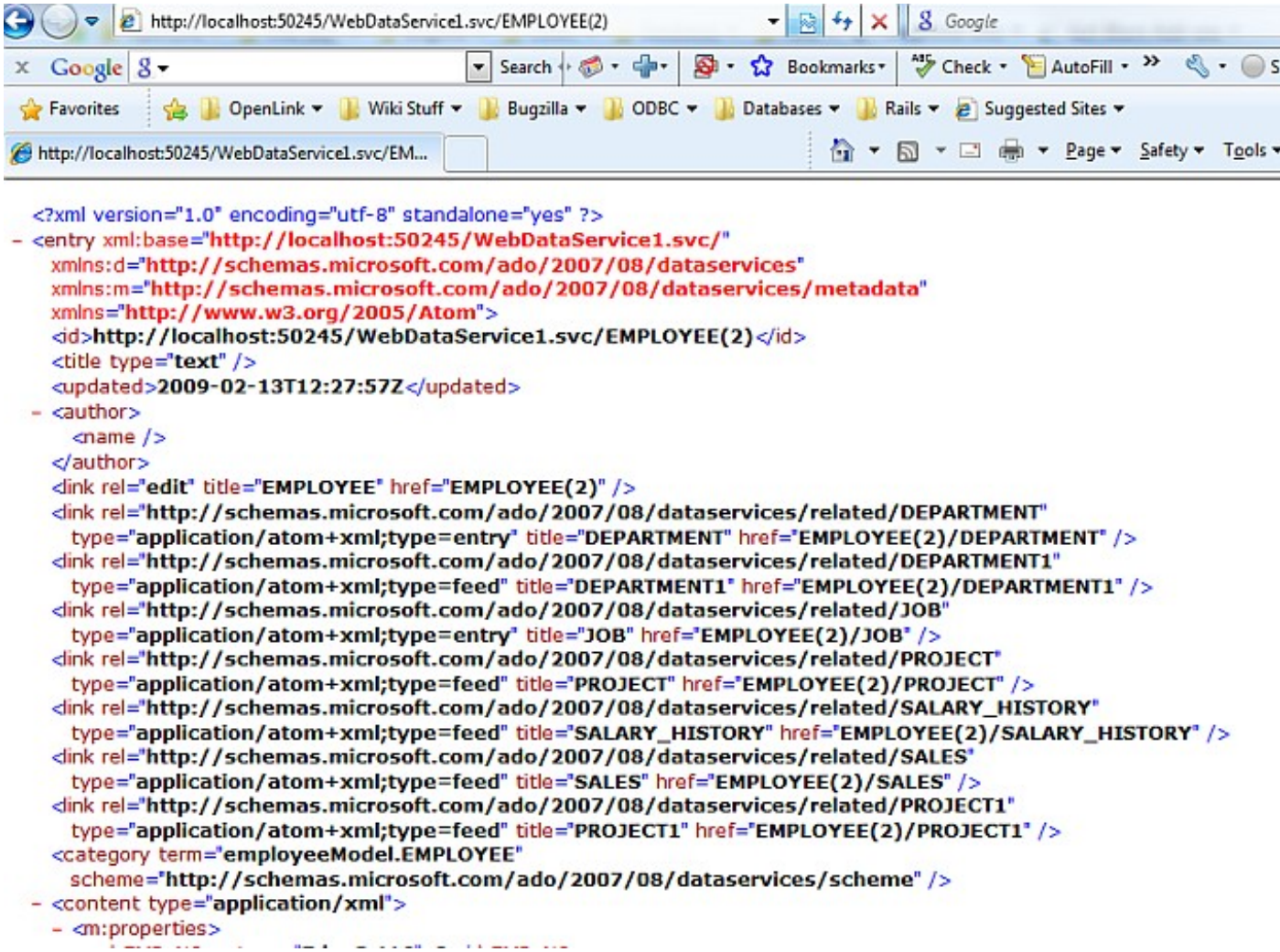
   EMPLOYEE

   table employee number

   2

   record, use this convention http://host/vdir/WebDataService1.svc/EMPLOYEE(2) .

   Figure 8.624. EMPLOYEES



Notes:

1. *Important*

   - To view

   Atom

   (the default format returned by an ADO.NET Data Service) in Internet Explorer, you must first ensure that

   Feed Reading View

   is turned

   off

   . This can be done on the

   Content tab

   of

   Tools in Internet Options

   .
2. If a Data Services entity instance URI page fails to load you can turn

Verbose

errors on by adding

config.UseVerboseErrors = true;

in the

virtuoso.svc.cs InitializeService

method to obtain more detailed information from the server as to why the page failed to load:

```
public static void InitializeService(IDataServiceConfiguration config)
{
config.UseVerboseErrors = true;
config.SetEntitySetAccessRule("*", EntitySetRights.All);
}
```

Visual Studio Windows DataGrid Form Application

This section details the steps required to create a simple Visual Studio 2008 Windows Form application, with associated DataGridView control for displaying data in selected tables from the target database.

1. Launch the Visual Studio 2008 SP1 IDE.

   Figure 8.625. Visual Studio 2008 SP1 IDE

2. Create a

   Web Application

   project by going to the

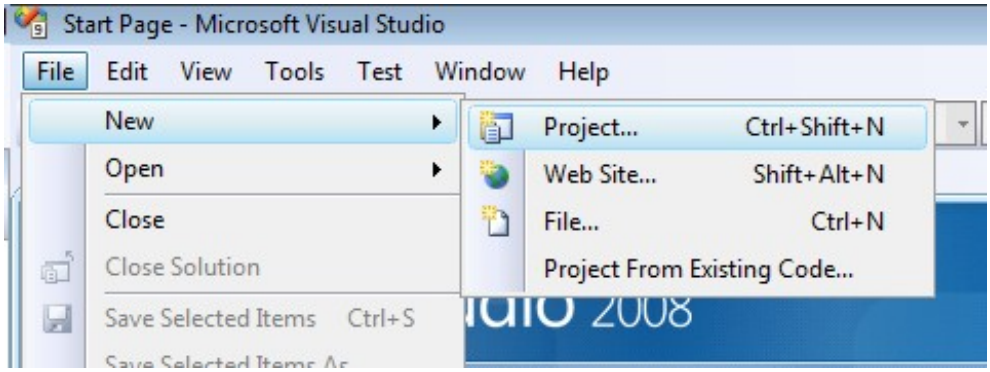
   File

   menu in Visual Studio and choosing

   New Project

   .

   Figure 8.626. Web Application



3. When the New Project window appears, choose either

   Visual Basic

   or

   Visual C#

   as your programming language.
4. Within the language category, click on

   Windows

   and select

   Windows Form Application

   from the right-hand panel.
5. Choose a name for the project, for example

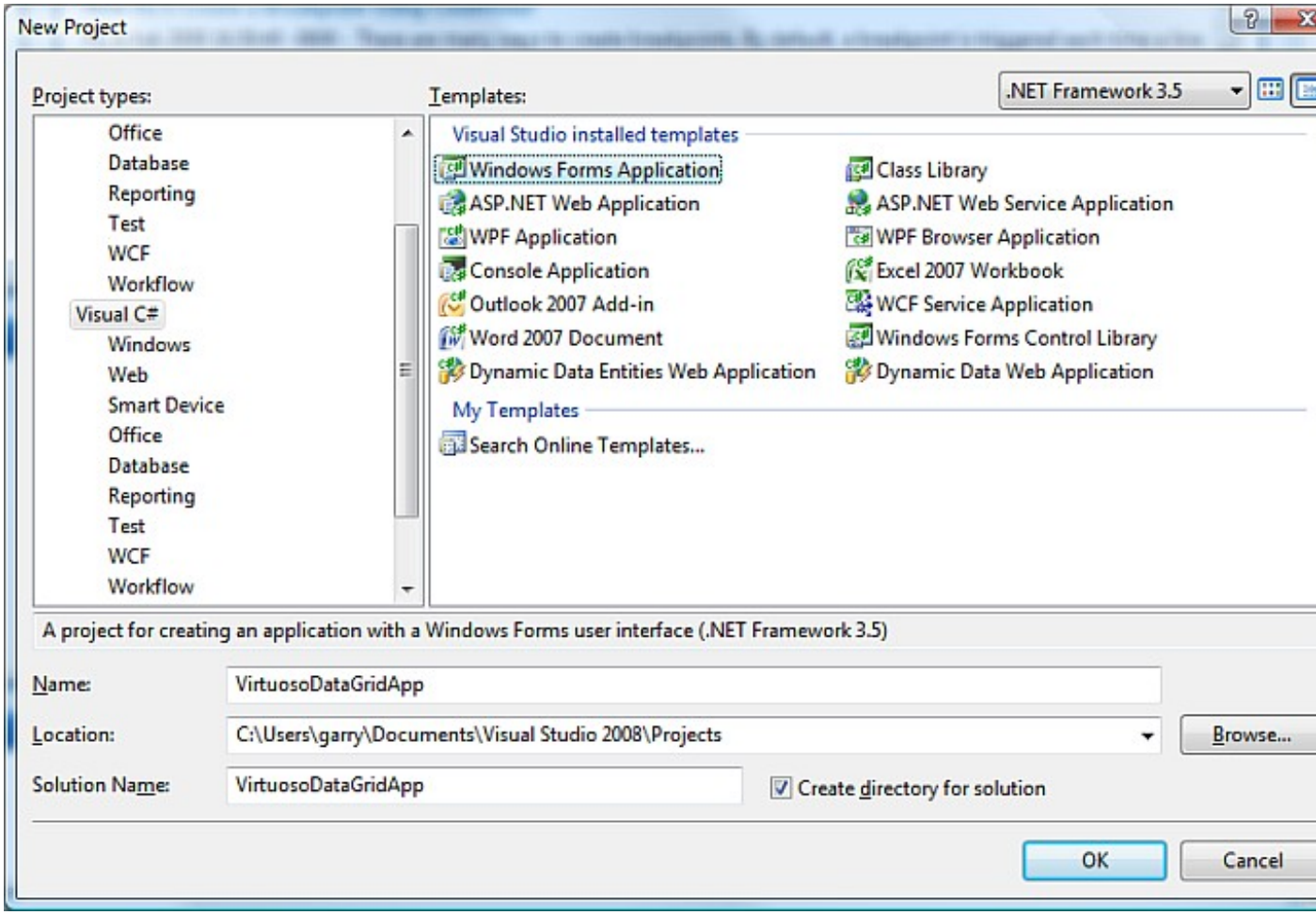
   VirtuosoDataGridApp

   , and click

   OK

   .

Figure 8.627. Web Application



6. In the

   Toolbox

   , expand

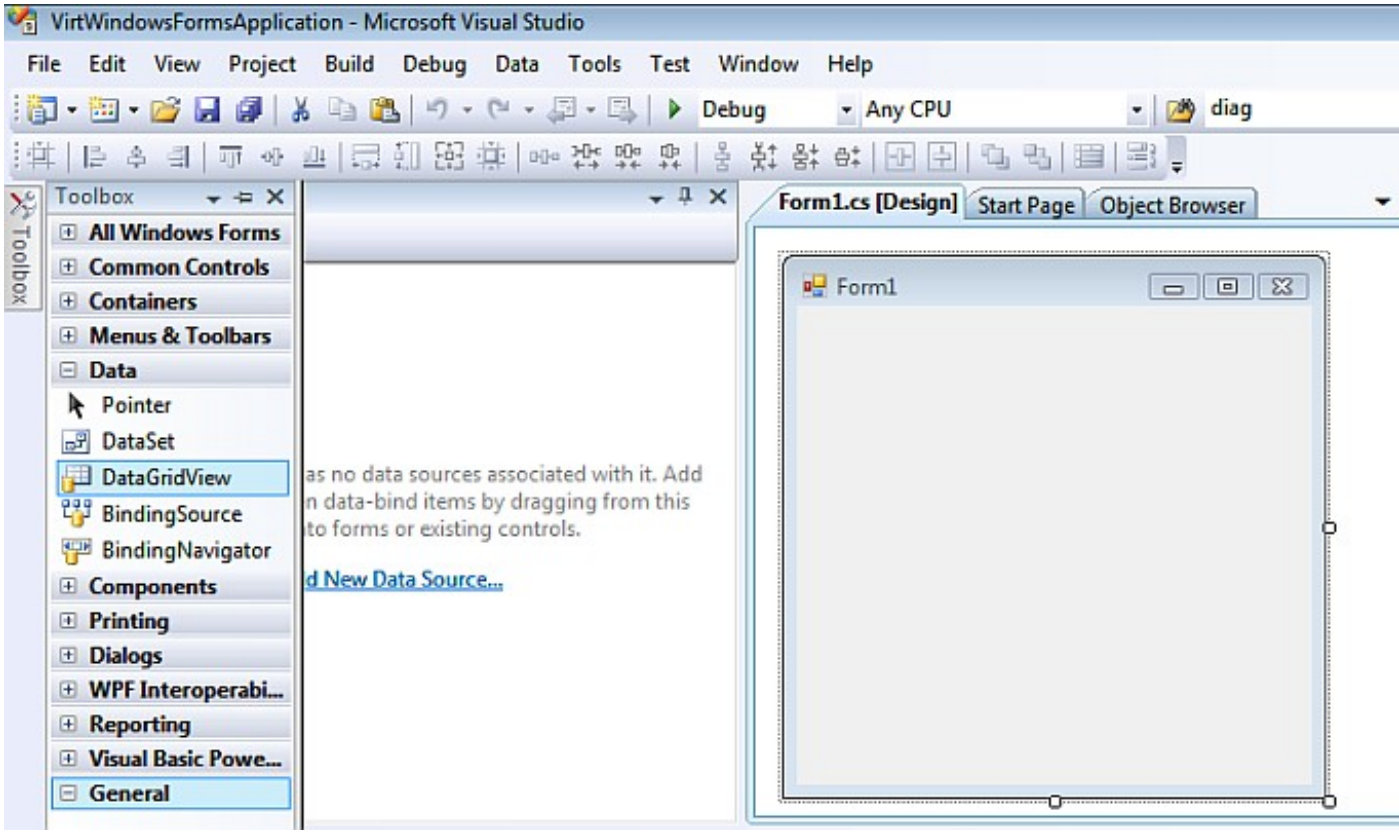
   Data Controls

   , and drag the

   DataGridView

   control onto the form.

   Figure 8.628. Toolbox



7. Click on the little

   arrow

   in the top right of the

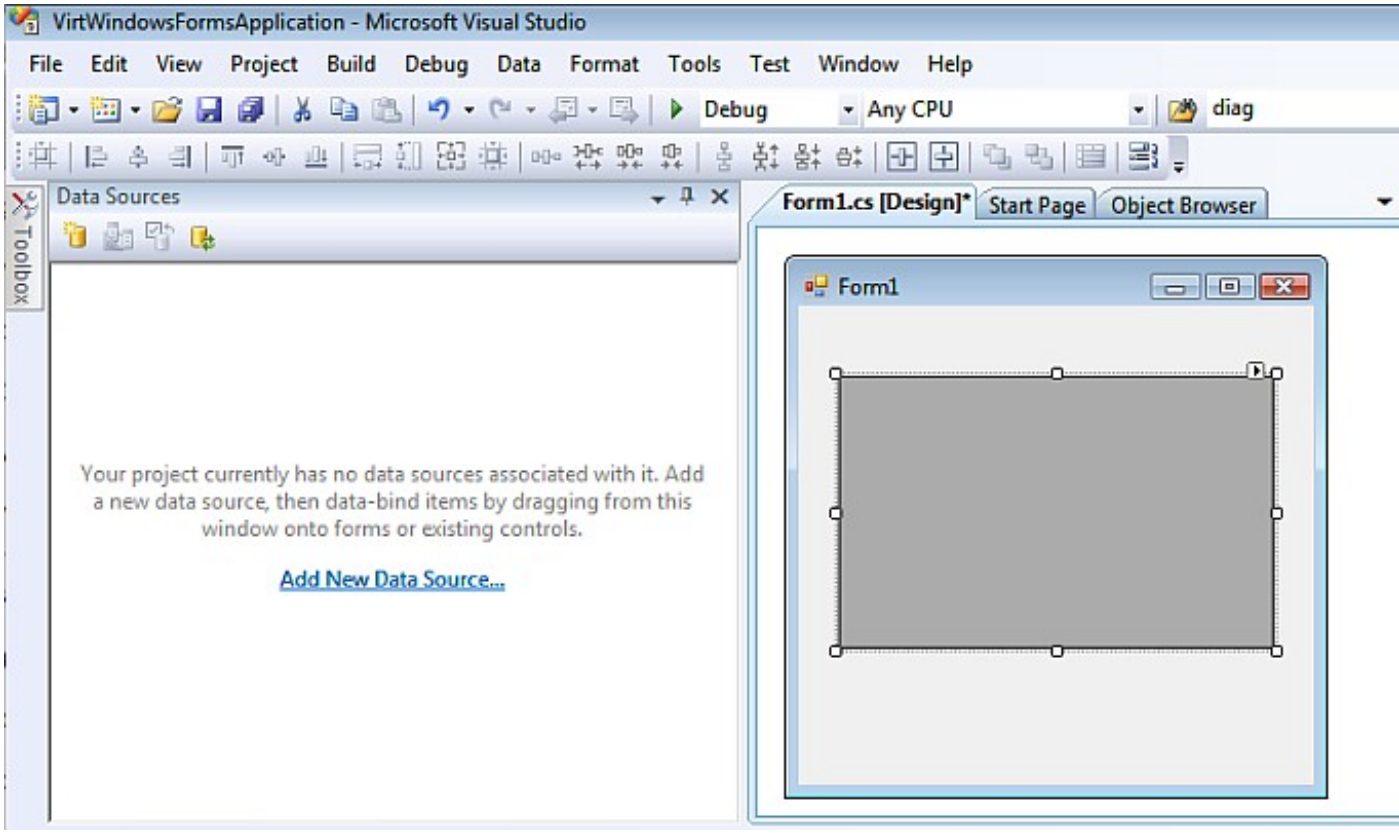
   DataGridView

   control. This loads the

   DataGridView Task

   menu.

Figure 8.629. DataGridView Task

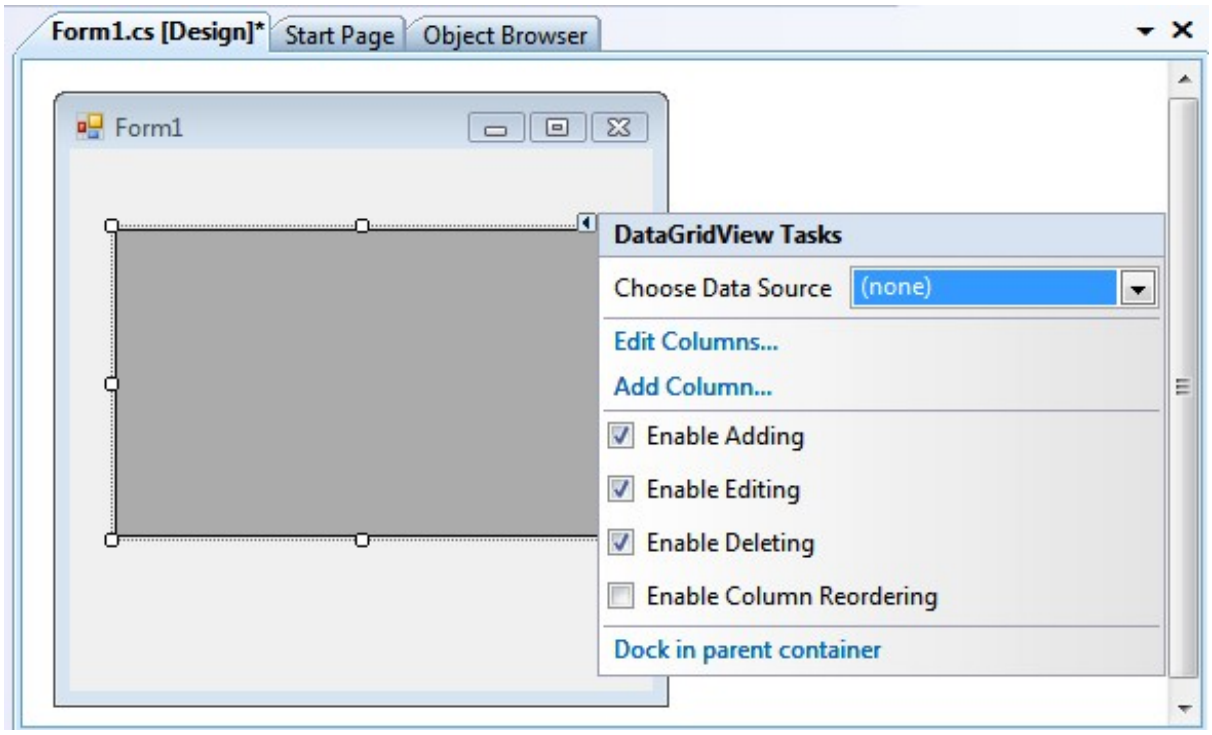


8. Click on the

   Choose Data Source

   list box.

   Figure 8.630. Choose Data Source

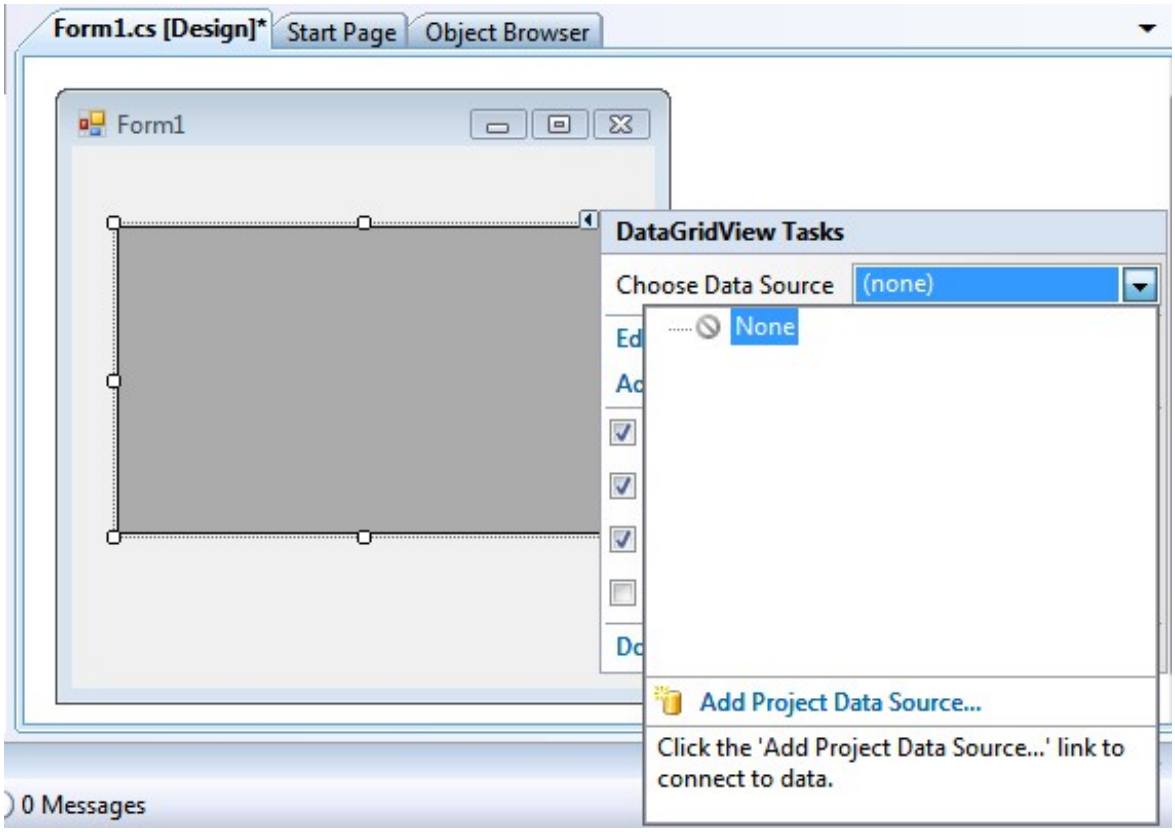


9. Click on the

   Add Project Data Source

link to connect to a data source.

Figure 8.631. Add Project Data Source



10. In the

    Data Source Configuration Wizard

    dialog

    Choose Data Source Type

    page select the

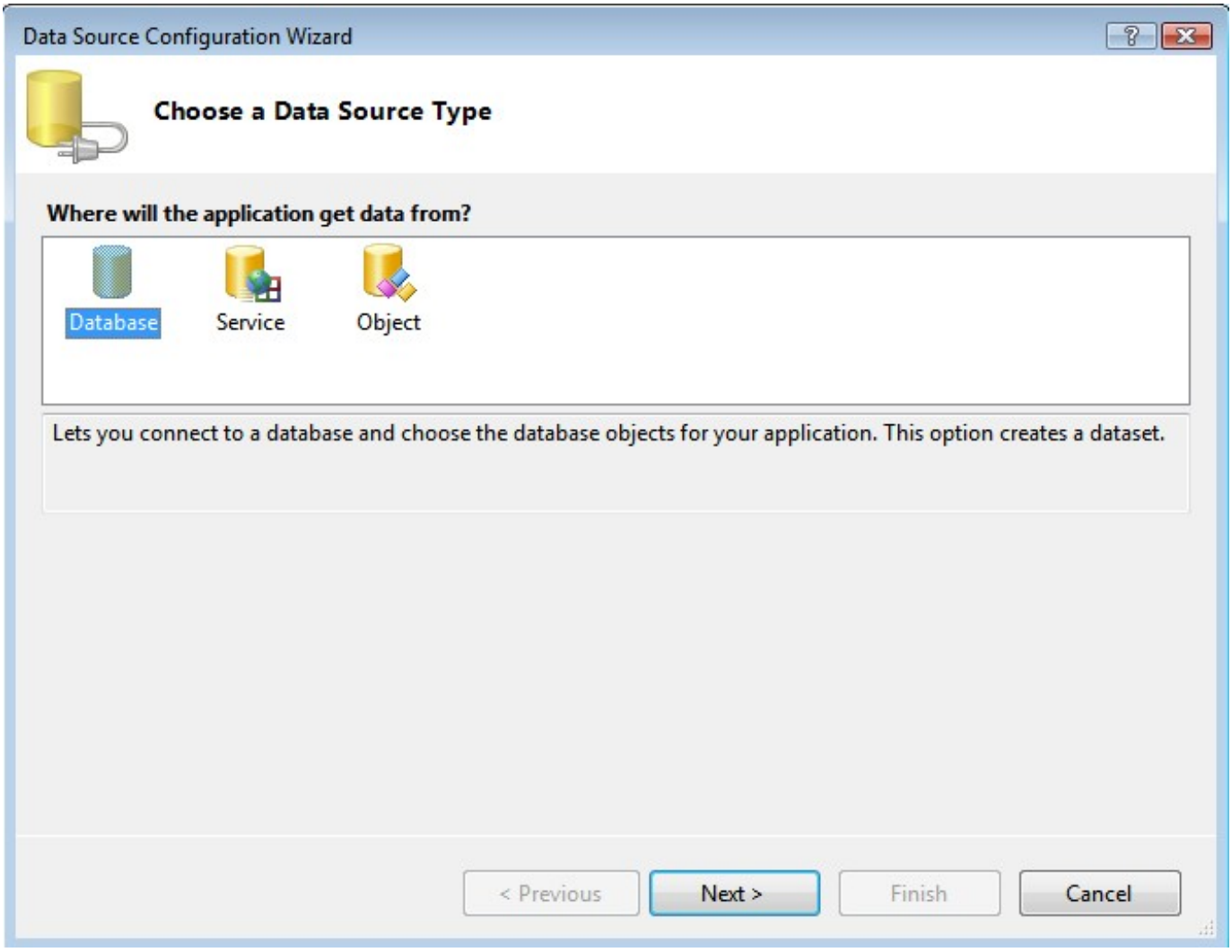
    Database

    data source type and click

    Next

    .

Figure 8.632. Data Source Type



11. In the

    Data Source Configuration Wizard

    dialog

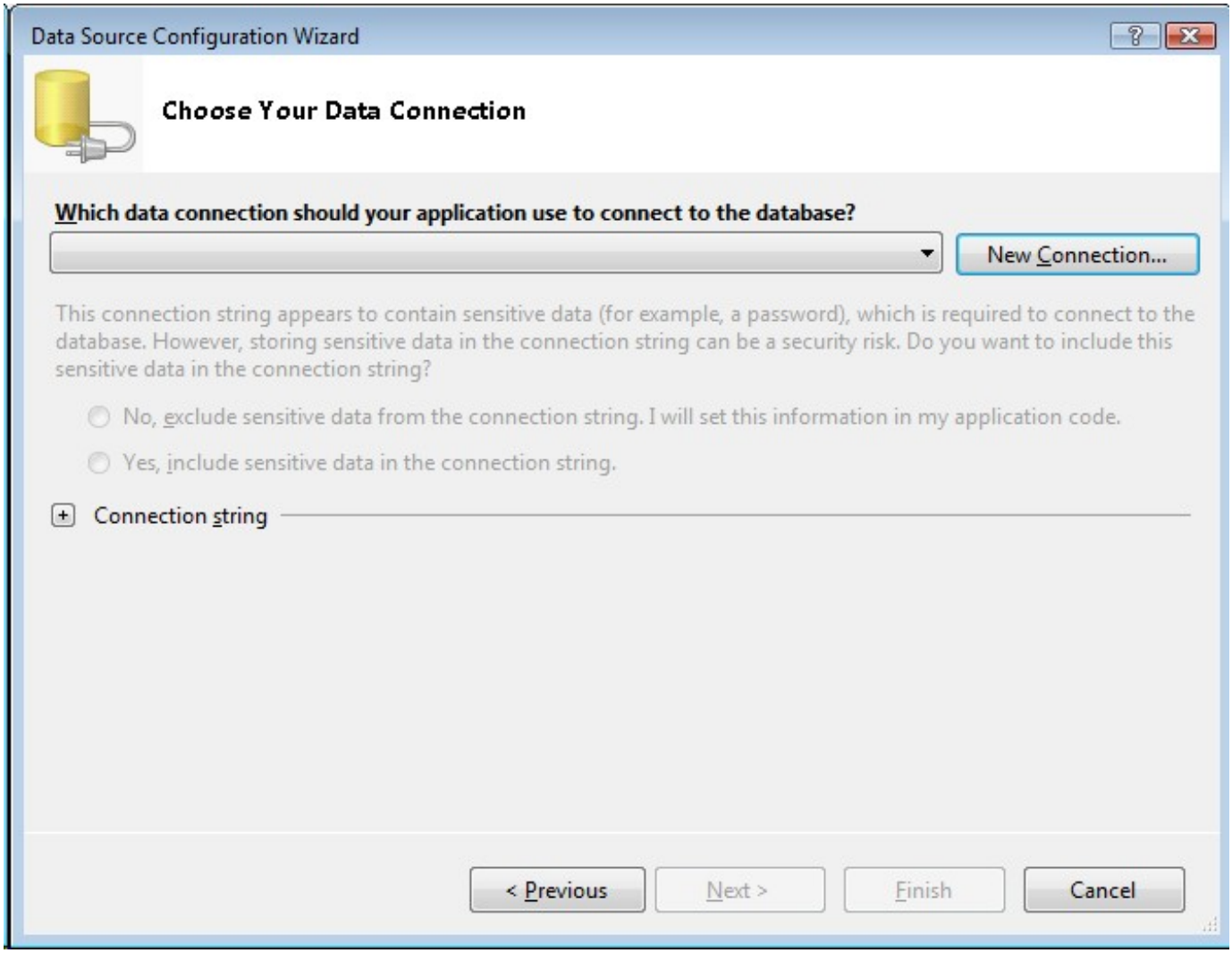
    Choose your Data Connection

    page, select the

    New Connection

    button

Figure 8.633. Data Source Configuration Wizard



12. In the

    Choose Data Source

    dialog, select the OpenLink

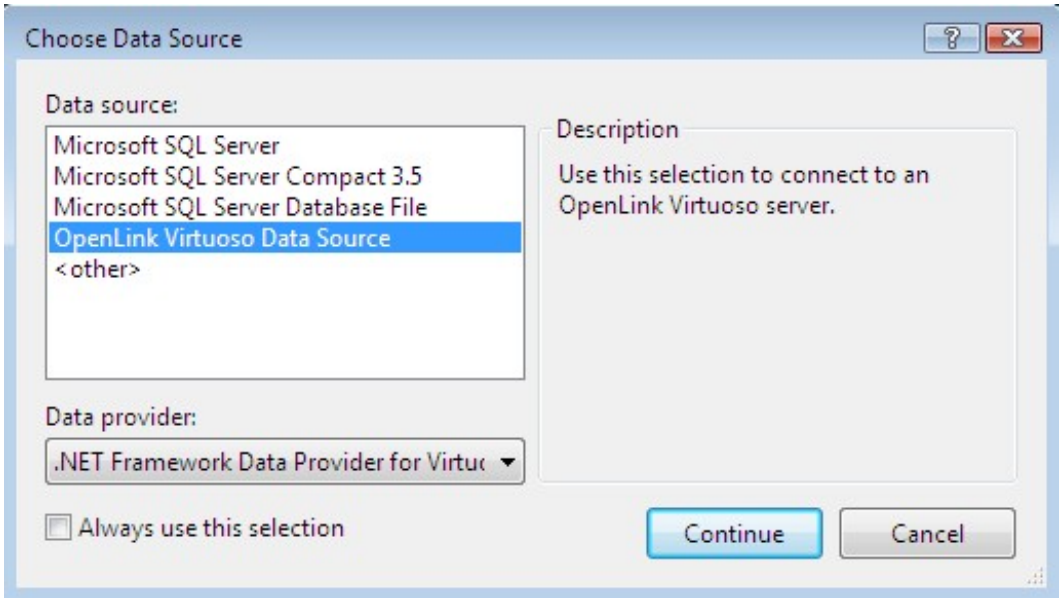
    Virtuoso Data Source

    from the list and click

    Continue

    .

Figure 8.634. Data Source



13. In the

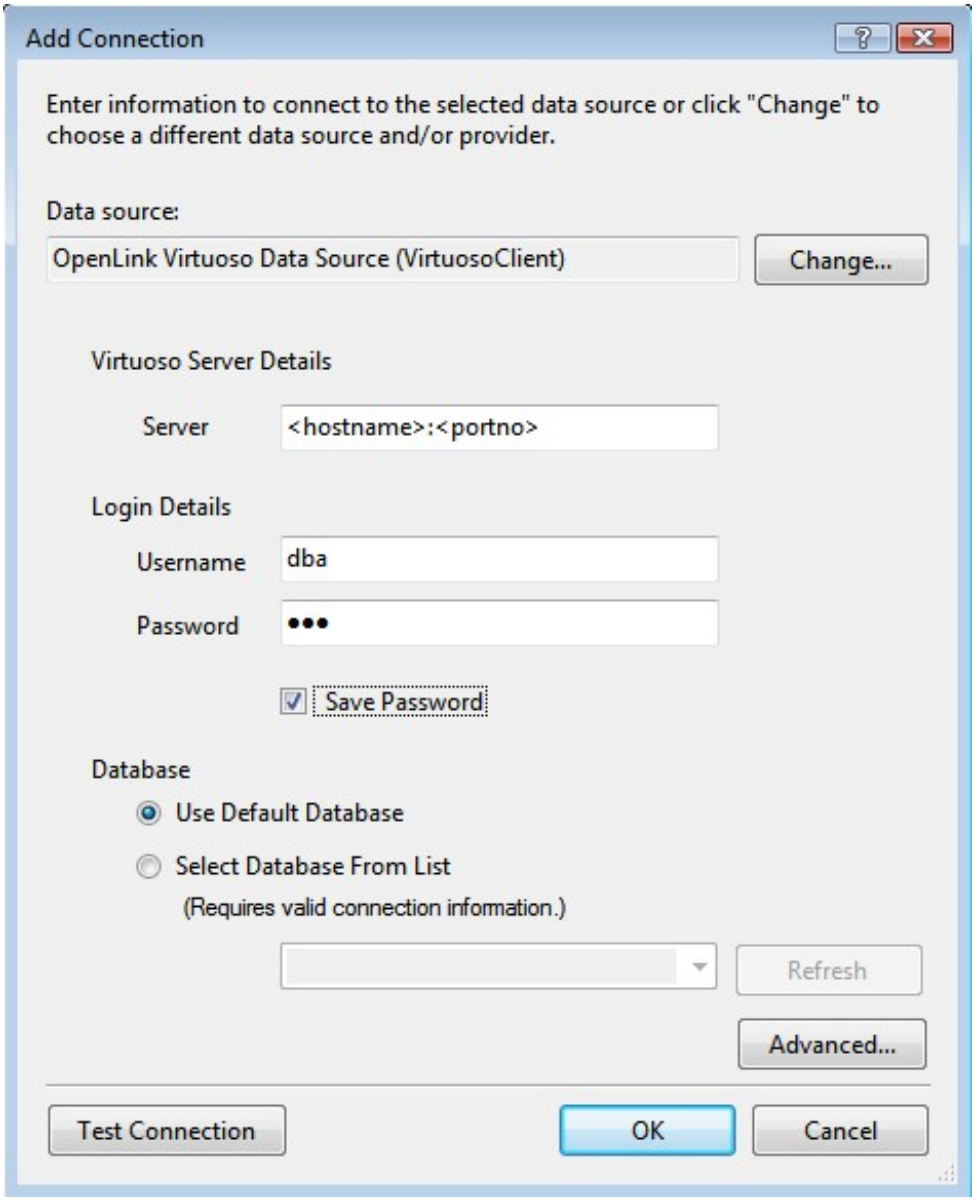
    Add Connection

    dialog, specify the

    hostname, portno, username and password

    for the target Virtuoso Server and check the Save Password check box.

    Figure 8.635. Connection Properties



14. Select the

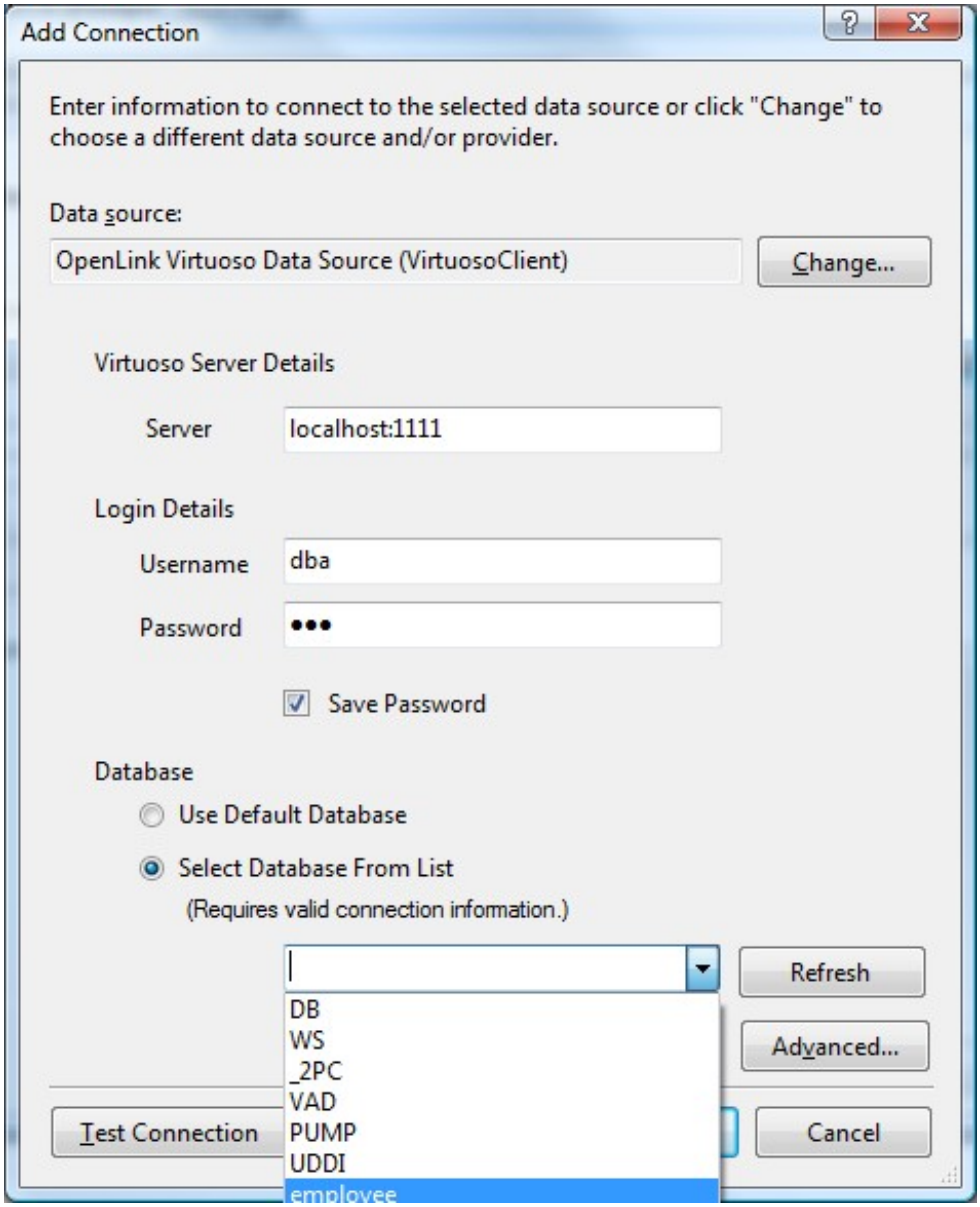
    Select Database From List

    radio button and choose the

    employee

    database from the drop down list.

    Figure 8.636. Add connection

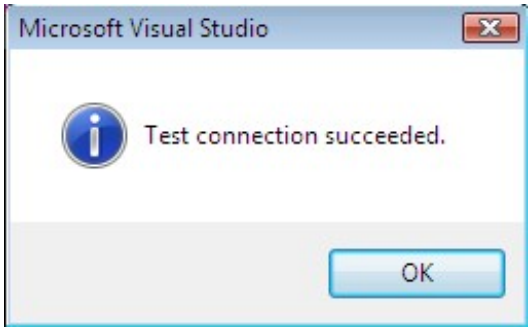


15. Click OK to add the connection.
16. Press the

    Test Connection

    dialog to verify that the database is accessible.

    Figure 8.637. Test Connection



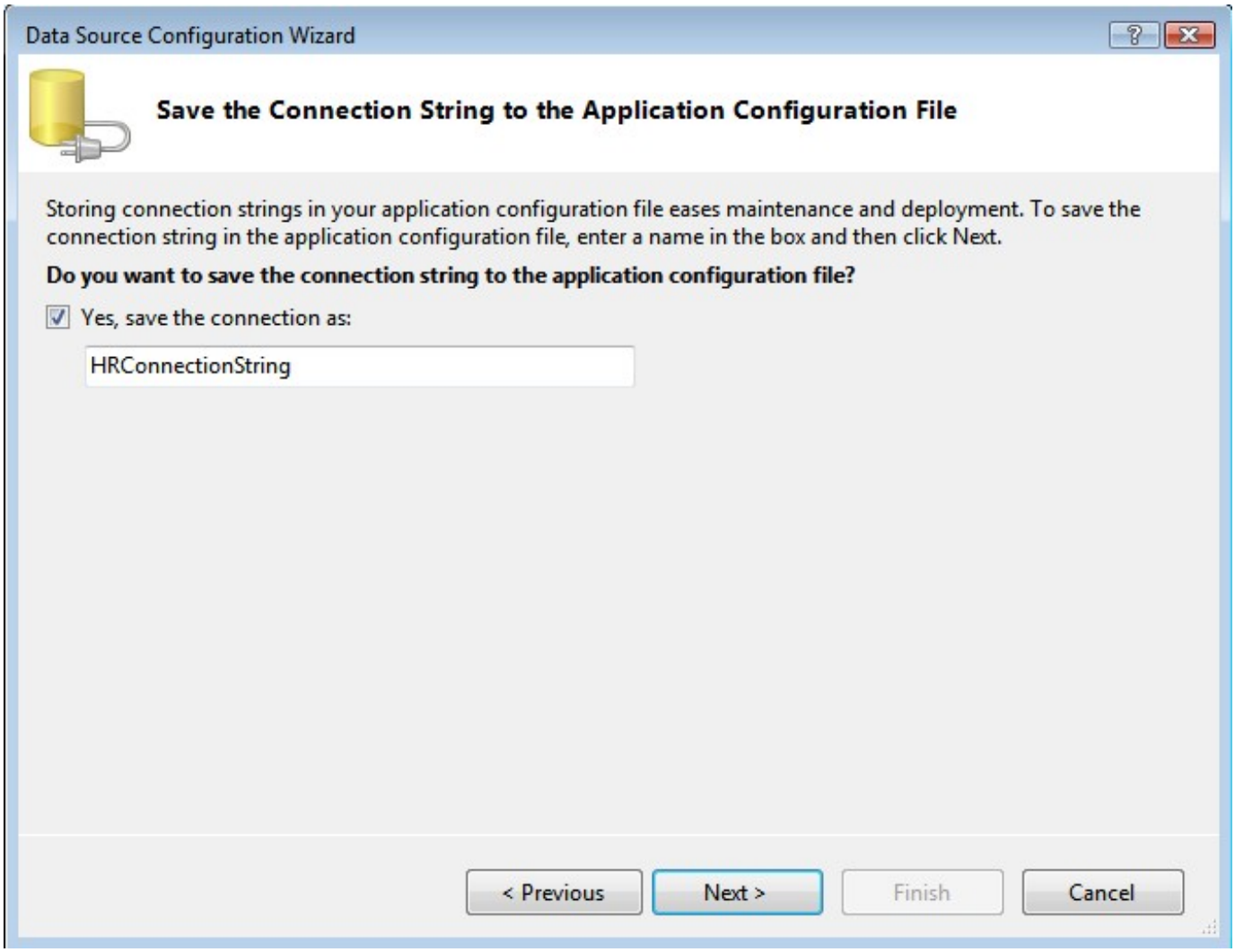
17. Leave the default connect string

    employeeConnectionString

and click

Next

Figure 8.638. employeeConnectionString



18. From the list of available tables returned for the employee database, select the

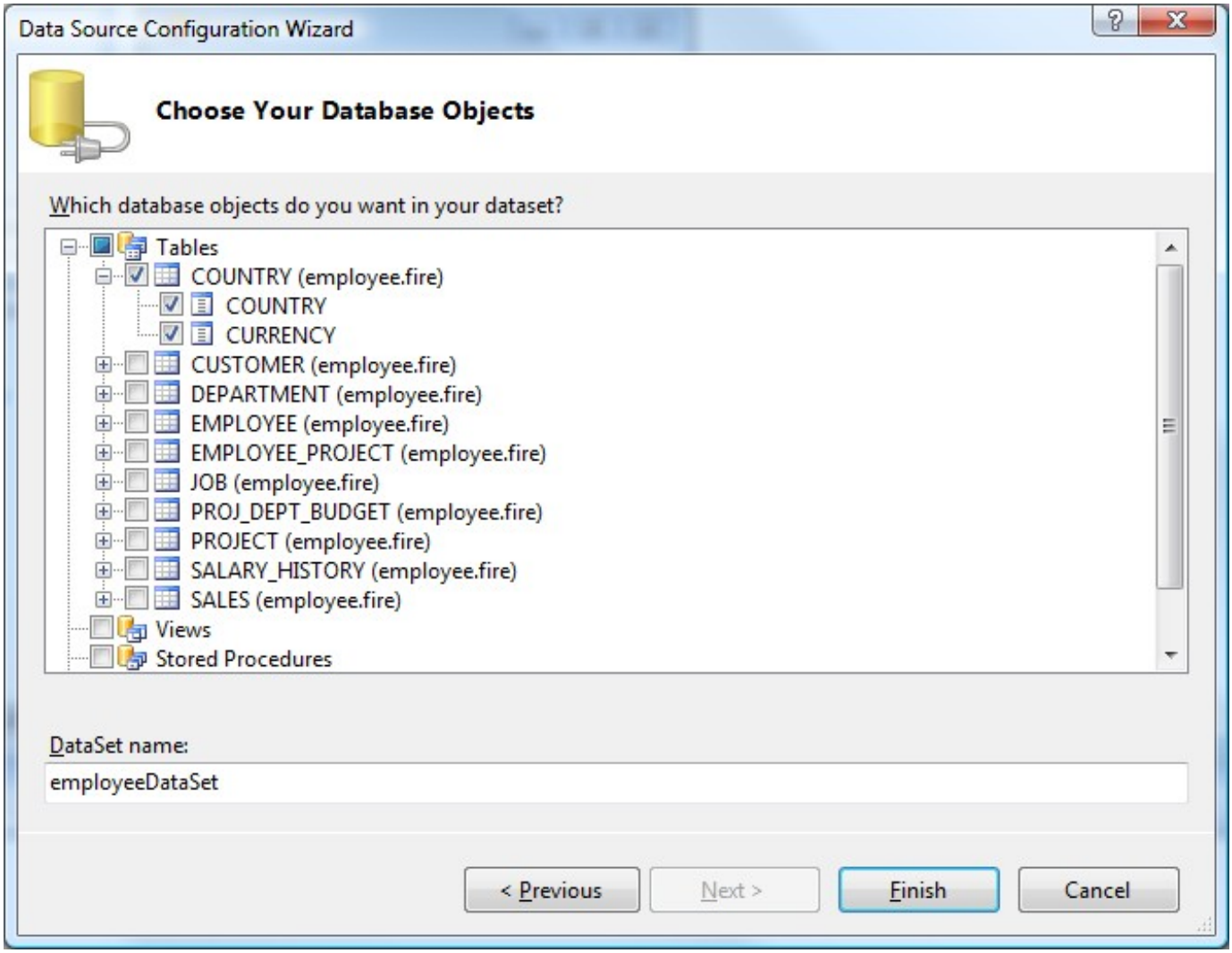
    COUNTRY

    table to be associated with the

    DataGridView

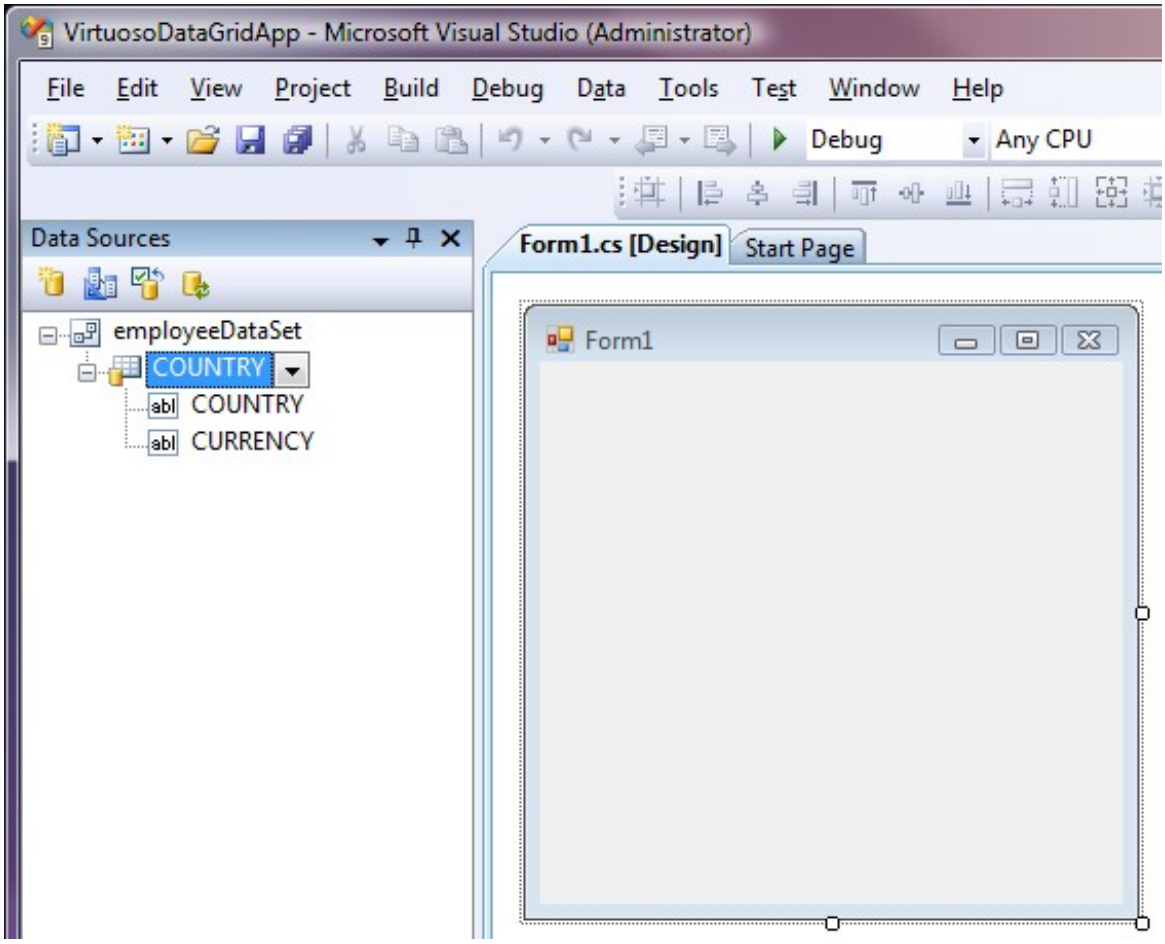
    control.

    Figure 8.639. employee database



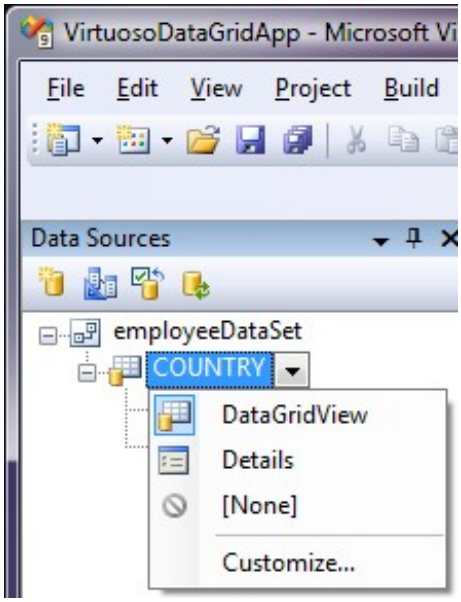
19. A DataSet for the employee database object is created

Figure 8.640. DataSet



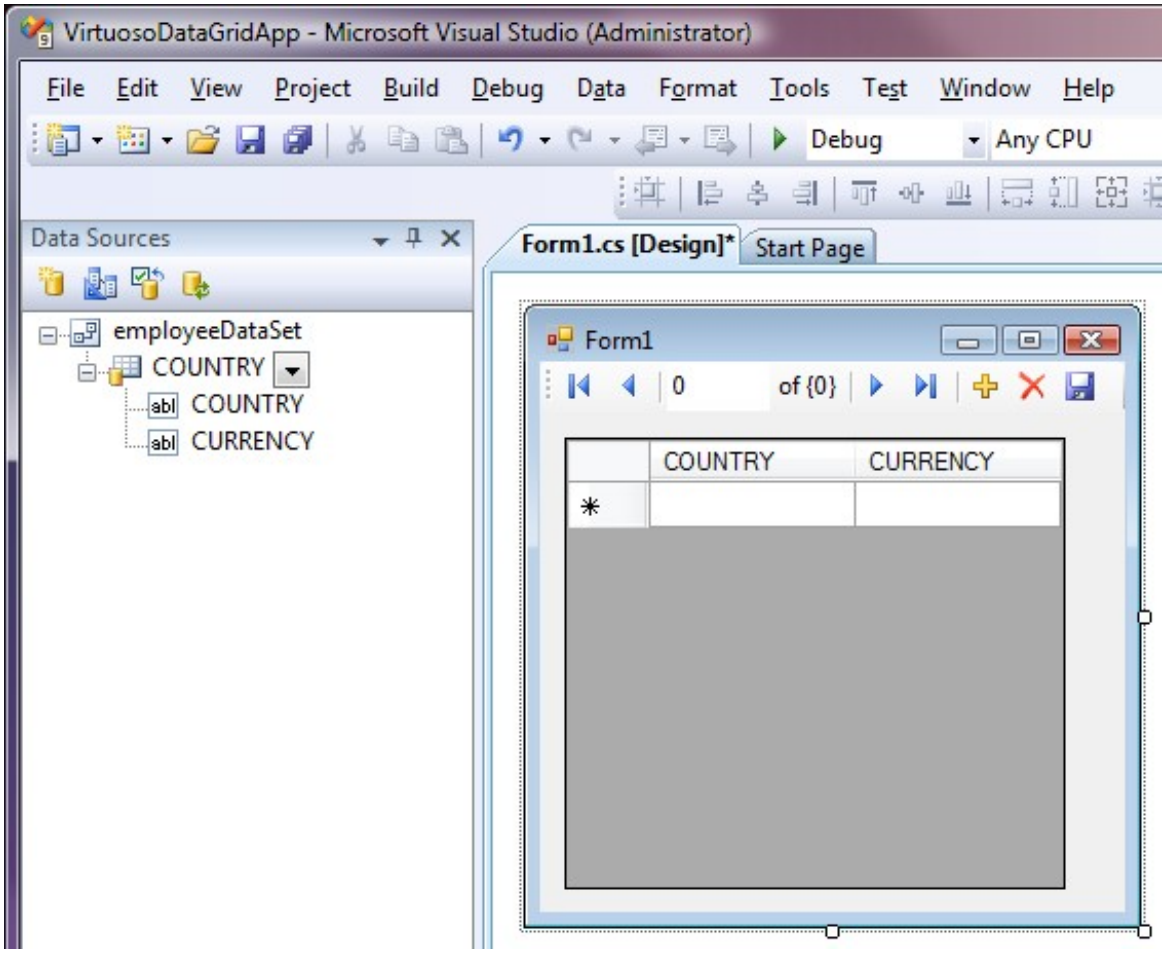
20. From the drop down list box next to the COUNTRY table ensure the DataGridView item is selected

Figure 8.641. DataGridView



21. Drag the COUNTRY DataSet item onto the Form to create a scrollable and editable association of the COUNTRY table object with the Data Grid View automatically.

Figure 8.642. association



22. To test the application, simply hit

    Ctrl+F5

    within Visual Studio or select

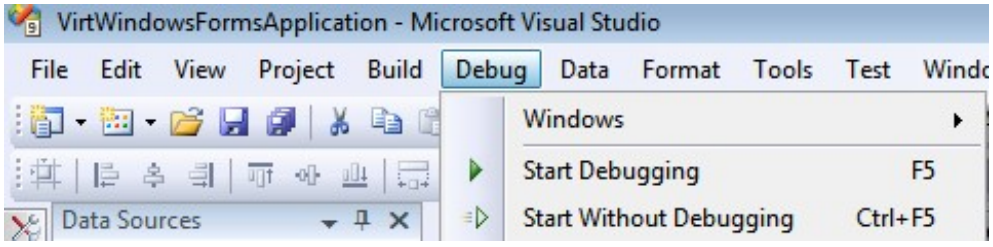
    Start Debugging

    from the

    Debug

    menu.

    Figure 8.643. Start Debugging



23. The data from the

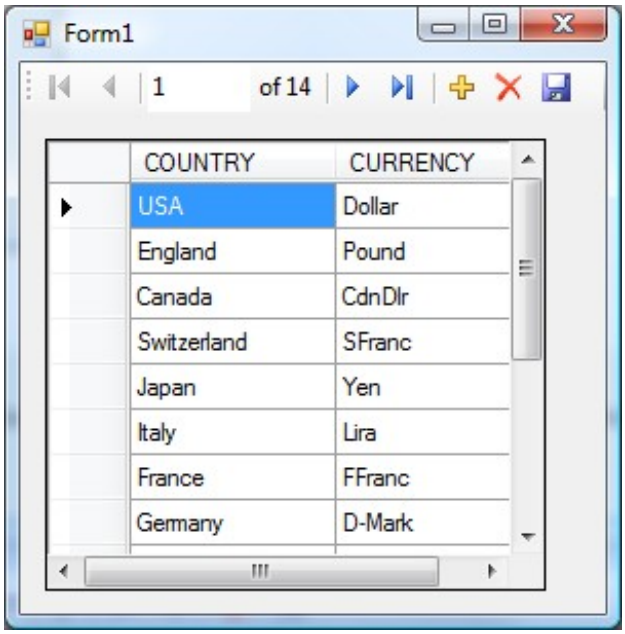
    COUNTRY

    table will be displayed in the

DataGrid

.

Figure 8.644. DataGrid

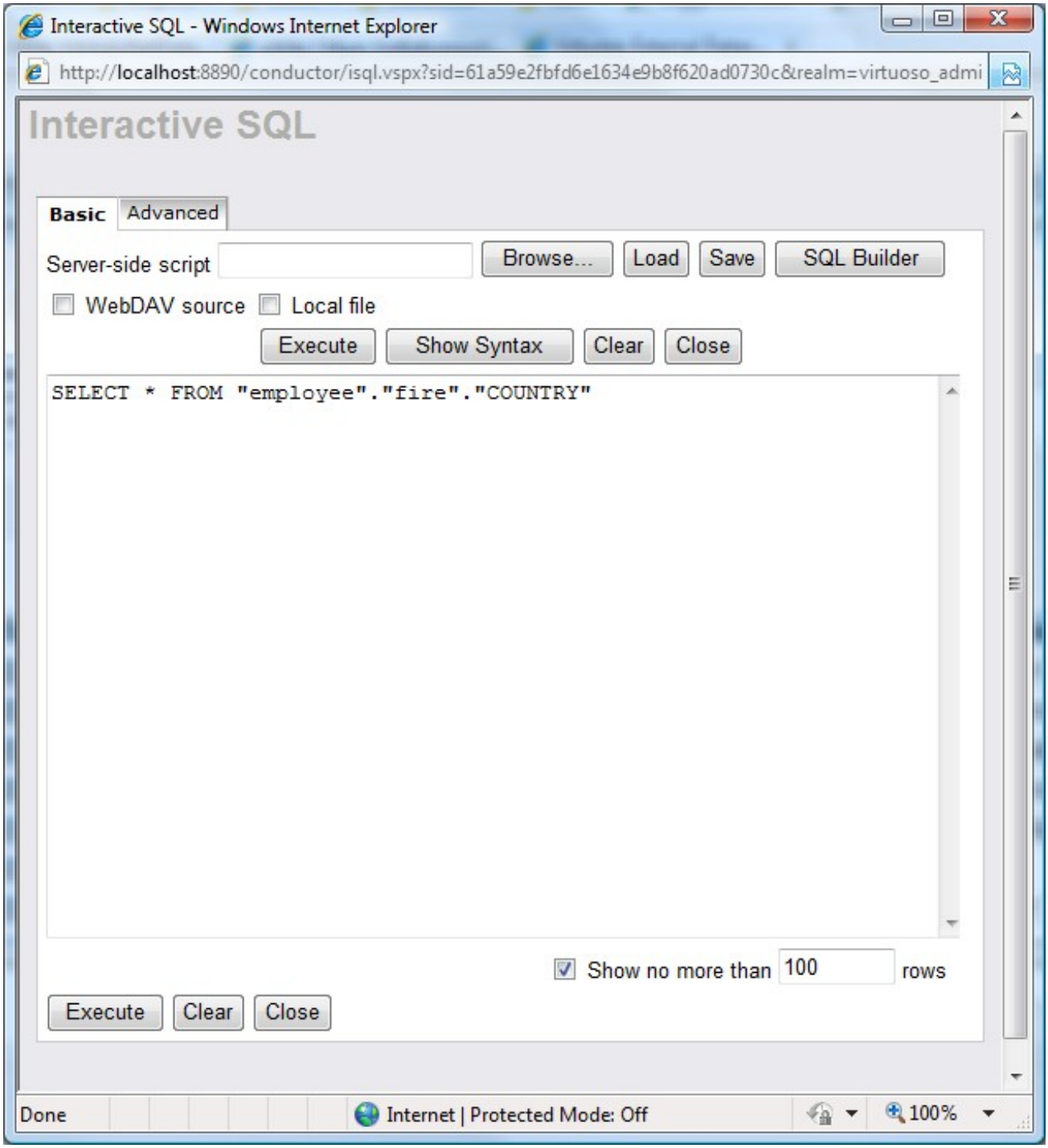


24. A new row can be inserted (updated or deleted) as indicated for the new "1111" record inserted below and the Save button clicked to save the change to the database.

Figure 8.645. new row

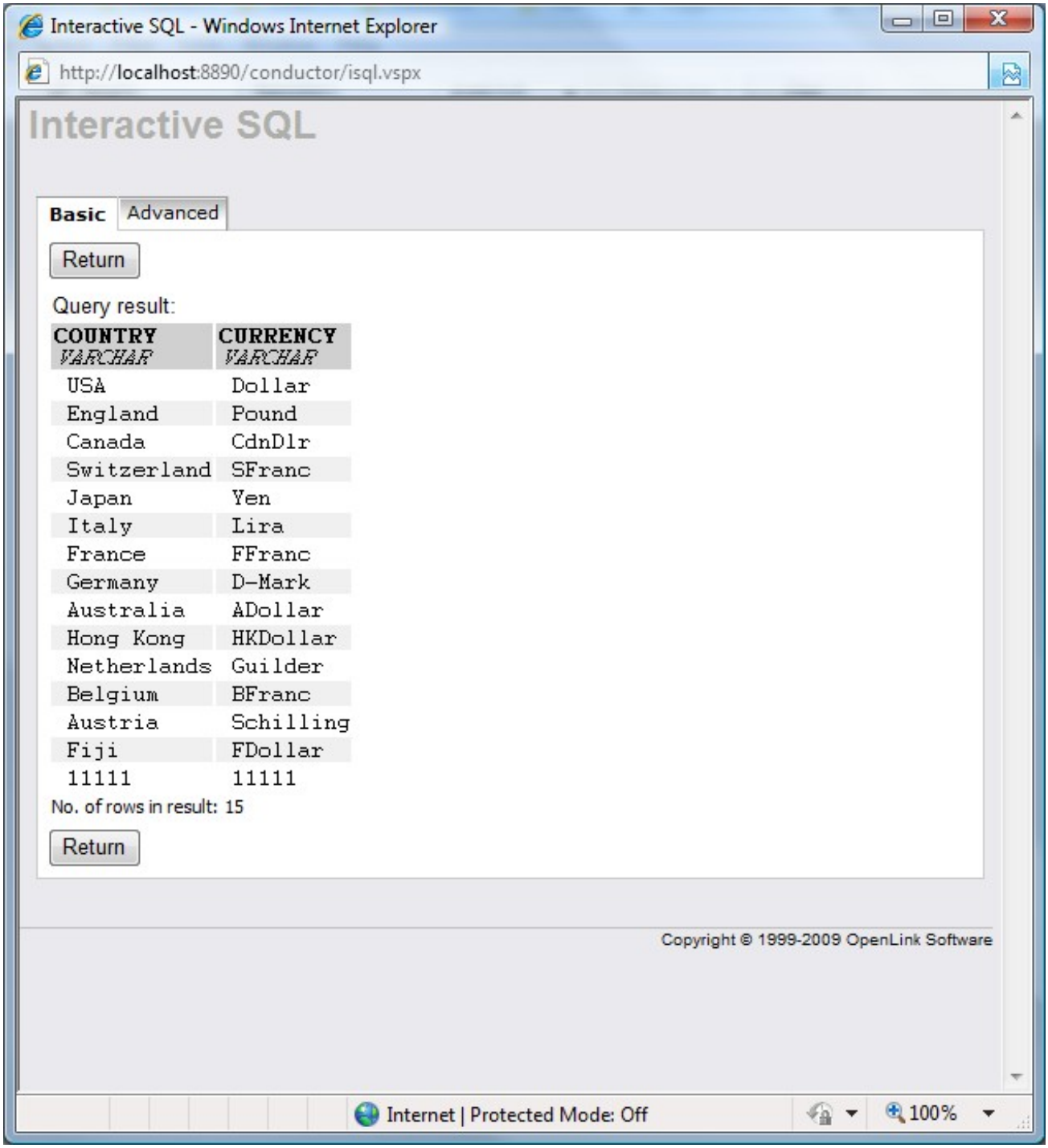
25. The Virtuoso Interactive SQL tab of the Conductor can be used to run the query select * from "employee"."fire"."COUNTRY"

Figure 8.646. Interactive SQL



26. To verify the change has been successfully made in the database.

Figure 8.647. verify



The task is now complete.

8.13. Using Microsoft Entity Frameworks to Access Microsoft SQL Server Schema Objects with Virtuoso

Abstract

This section details the steps required to provide Microsoft Entity Framework access to Microsoft SQL Server Schema Objects using the OpenLink Virtuoso Universal Server. This is achieved by linking the required Microsoft SQL Server Schema objects into Virtuoso using its built in Virtual Database engine, and then using the Virtuoso ADO.NET Entity Framework provider to query the remote Microsoft SQL Server Schema objects linked into the Virtuoso Server.

Prerequisites. The following prerequisites must be in place for this solution to be possible.

Microsoft SQL Server DBMS. A Microsoft SQL Server DBMS hosting the required Schema Objects needs to be available. In this document the *Microsoft SQL Server Northwind* sample database will be used to demonstrate the process.

A Microsoft SQL Server DBMS hosting the required Schema Objects needs to be available. In this document the *Microsoft SQL Server Northwind* sample database will be used to demonstrate the process.

ODBC Driver for Microsoft SQL Server. An Microsoft SQL Server ODBC Driver is required for Linking the Microsoft SQL Server Schema Objects into the Virtuoso Server. The OpenLink *ODBC Driver for Microsoft SQL Server* will be used in this section, for which a functional ODBC Data source name of "sql2k5ma" will be assumed to exist on the machine hosting the Virtuoso Server.

An Microsoft SQL Server ODBC Driver is required for Linking the Microsoft SQL Server Schema Objects into the Virtuoso Server. The OpenLink *ODBC Driver for Microsoft SQL Server* will be used in this section, for which a functional ODBC Data source name of "sql2k5ma" will be assumed to exist on the machine hosting the Virtuoso Server.

Virtuoso Universal Server. An Virtuoso installation including the Virtuoso Universal Server and ADO.NET Entity Framework Provider is required. The Virtuoso components used must be Release 5.10.x or above, this being the minimum version containing support for Microsoft Entity Frameworks

An Virtuoso installation including the Virtuoso Universal Server and ADO.NET Entity Framework Provider is required. The Virtuoso components used must be Release 5.10.x or above, this being the minimum version containing support for Microsoft Entity Frameworks

Microsoft Visual Studio 2008 SP1. Microsoft Visual Studio 2008 Service Pack 1 is required, this being the only version containing the necessary Entity Framework support available at the time of writing.

Microsoft Visual Studio 2008 Service Pack 1 is required, this being the only version containing the necessary Entity Framework support available at the time of writing.

Tasks

Ensure Microsoft SQL Server Primary Keys (PKs) are not nullable. The Visual Studio 2008 Entity Data Model (EDM) requires that all primary keys are *NOT* Nullable, and will fail to generate an EDM if any are. Thus ensure any tables to be used are defined as not nullable in the Microsoft SQL Server database schema before attempting to generate an EDM. In the case of the Microsoft SQL Server Northwind database all tables are not nullable, thus this should not be an issue in this case.

The Visual Studio 2008 Entity Data Model (EDM) requires that all primary keys are *NOT* Nullable, and will fail to generate an EDM if any are. Thus ensure any tables to be used are defined as not nullable in the Microsoft SQL Server database schema before attempting to generate an EDM. In the case of the Microsoft SQL Server Northwind database all tables are not nullable, thus this should not be an issue in this case.

8.13.1. Install and configure OpenLink ODBC Driver for Microsoft SQL Server

The Virtuoso Virtual Database engine uses ODBC as the connectivity mechanism for linking remote database objects into its local schema. Thus a Microsoft SQL Server ODBC Driver must be available with a suitably configured DSN for connecting to the target database. The OpenLink ODBC Drivers for Microsoft SQL Server have been used in this section, although in theory any Microsoft SQL Server ODBC Driver can be used.

Installation instructions for the OpenLink ODBC Driver for Microsoft SQL Server are available from:

- Product Installation & Basic Configuration (ODBC)

8.13.2. Install and configure OpenLink Virtuoso Universal Server

- Install and configure OpenLink Virtuoso Universal Server

8.13.3. Linking Microsoft SQL Server tables into OpenLink Virtuoso

1. Start the Virtuoso Web User Interface

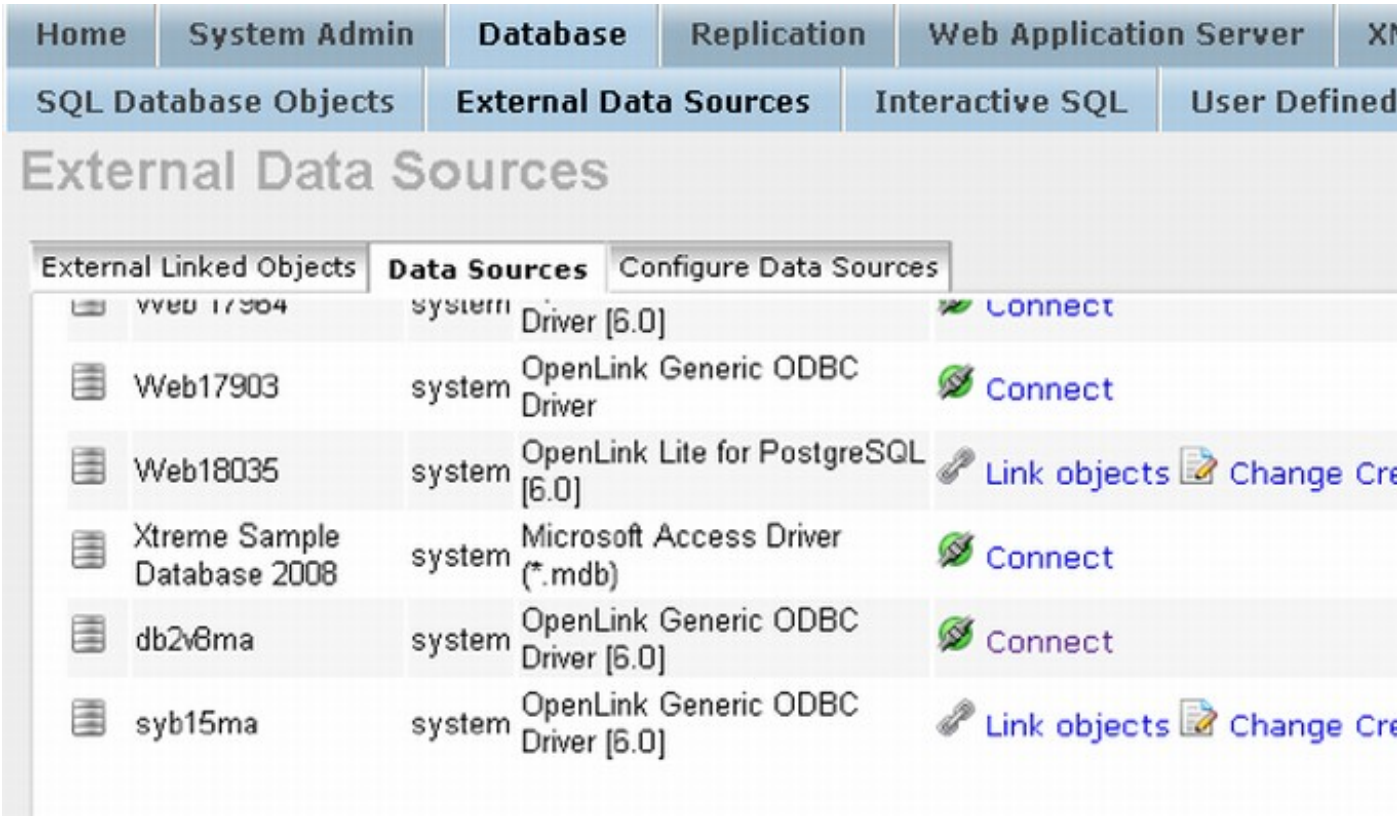
Figure 8.648. Start

2. Select the "Conductor" link in the Left Frame and Login as the "dba" user.

Figure 8.649. Conductor

3. Select the "Databases" -> "External Data Source" -> "Data Sources" tab menu items

Figure 8.650. Databases

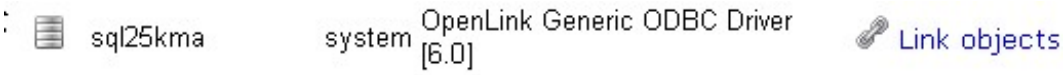


4. Select the "Connect" button for the "sql2k5ma" Microsoft SQL Server DSN.

Figure 8.651. Connect

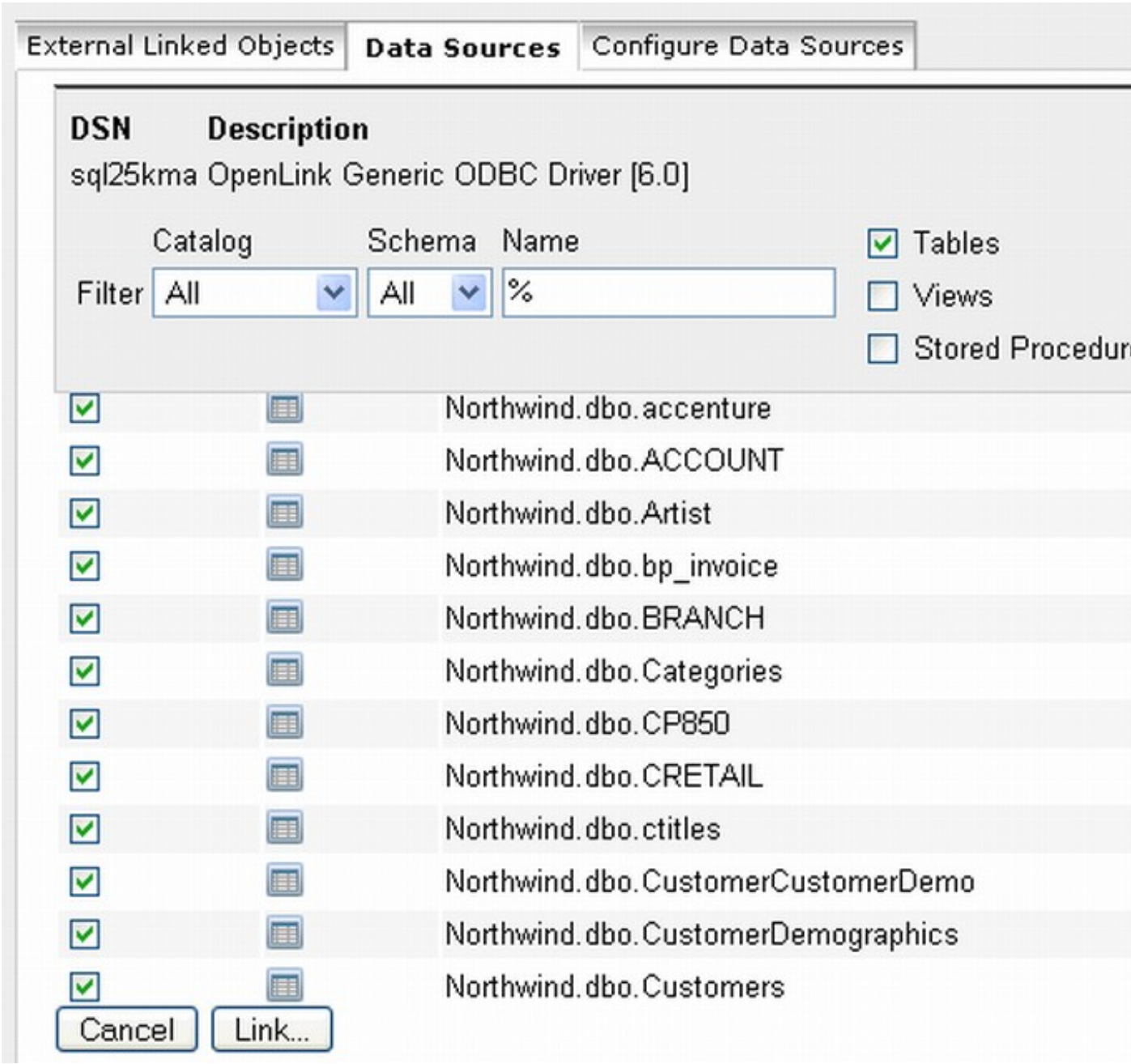
5. On successful connection Select the "Link Objects" button to obtain a list of available tables

Figure 8.652. Link Objects



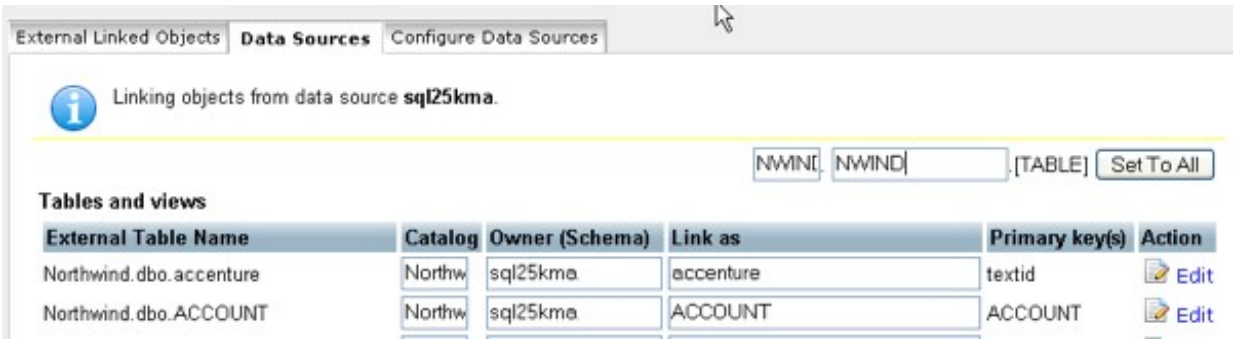
6. Select all the tables that are part of the "Northwind" catalog.

Figure 8.653. Select all tables



7. Change the Catalog for each table to be "Northwind" using the "Set All" button.

Figure 8.654. Catalog



8. All the catalog names are changed to be "NWIND"

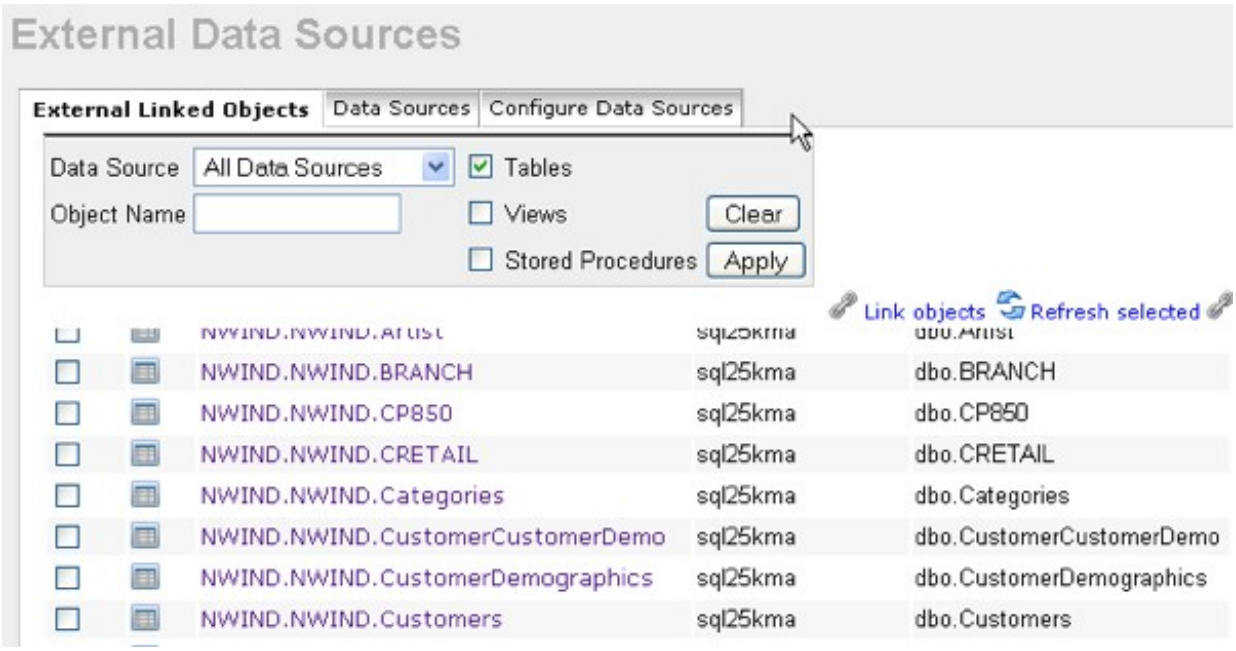
Figure 8.655. catalog names

9. Select the "Link" button to link the selected tables into Virtuoso

Figure 8.656. "Link" button

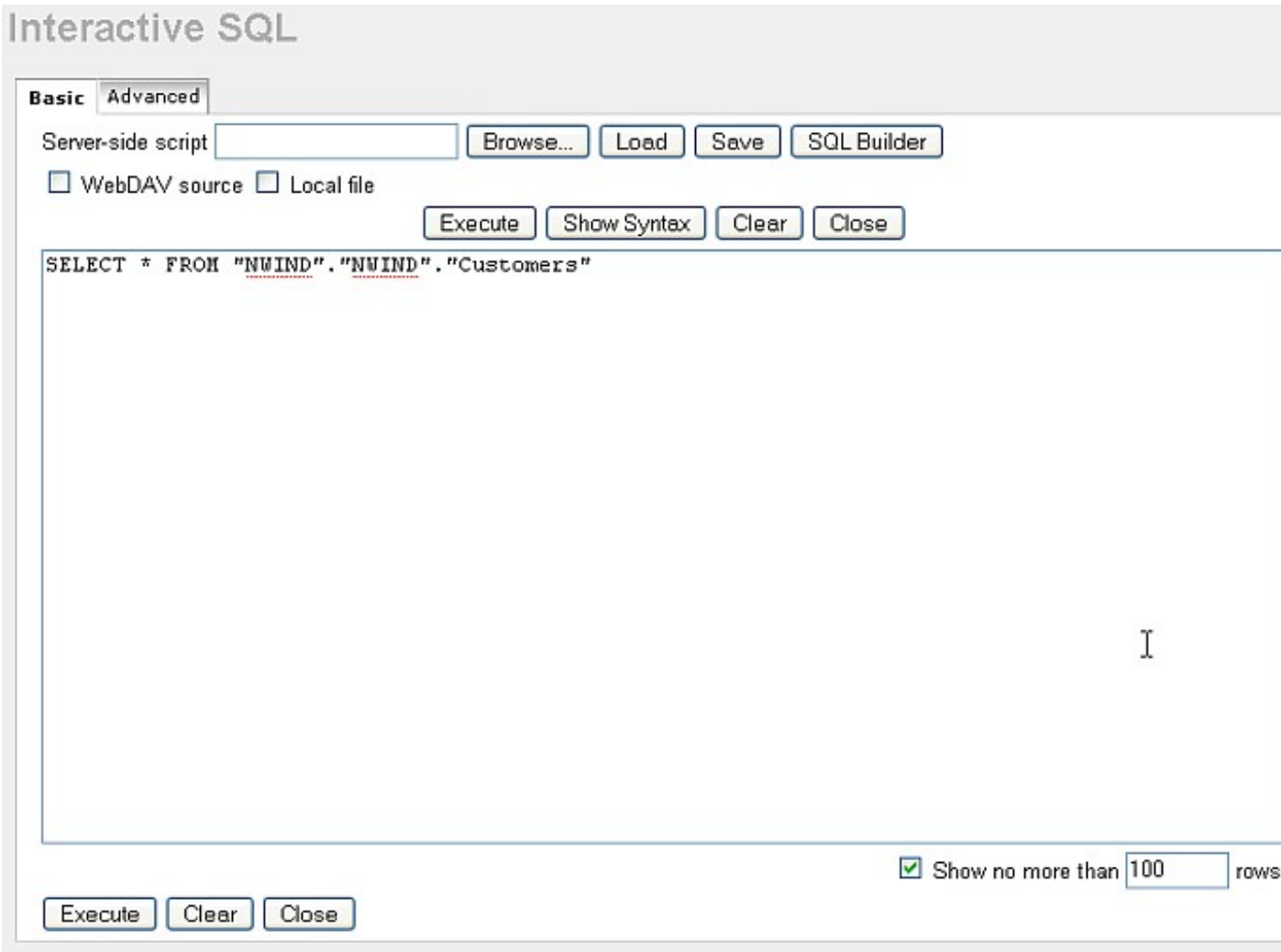
10. On completion of the Link process the tables will be displayed in the "External Linked Objects" tab.

Figure 8.657. Completion



11. At this point, you can test the remotely linked tables by clicking on the link that accompanies each table, e.g. NWIND.NWIND.Customer. This will display the Interactive ISQL interface which will have been already populated with a suitable SQL statement.

Figure 8.658. Querying

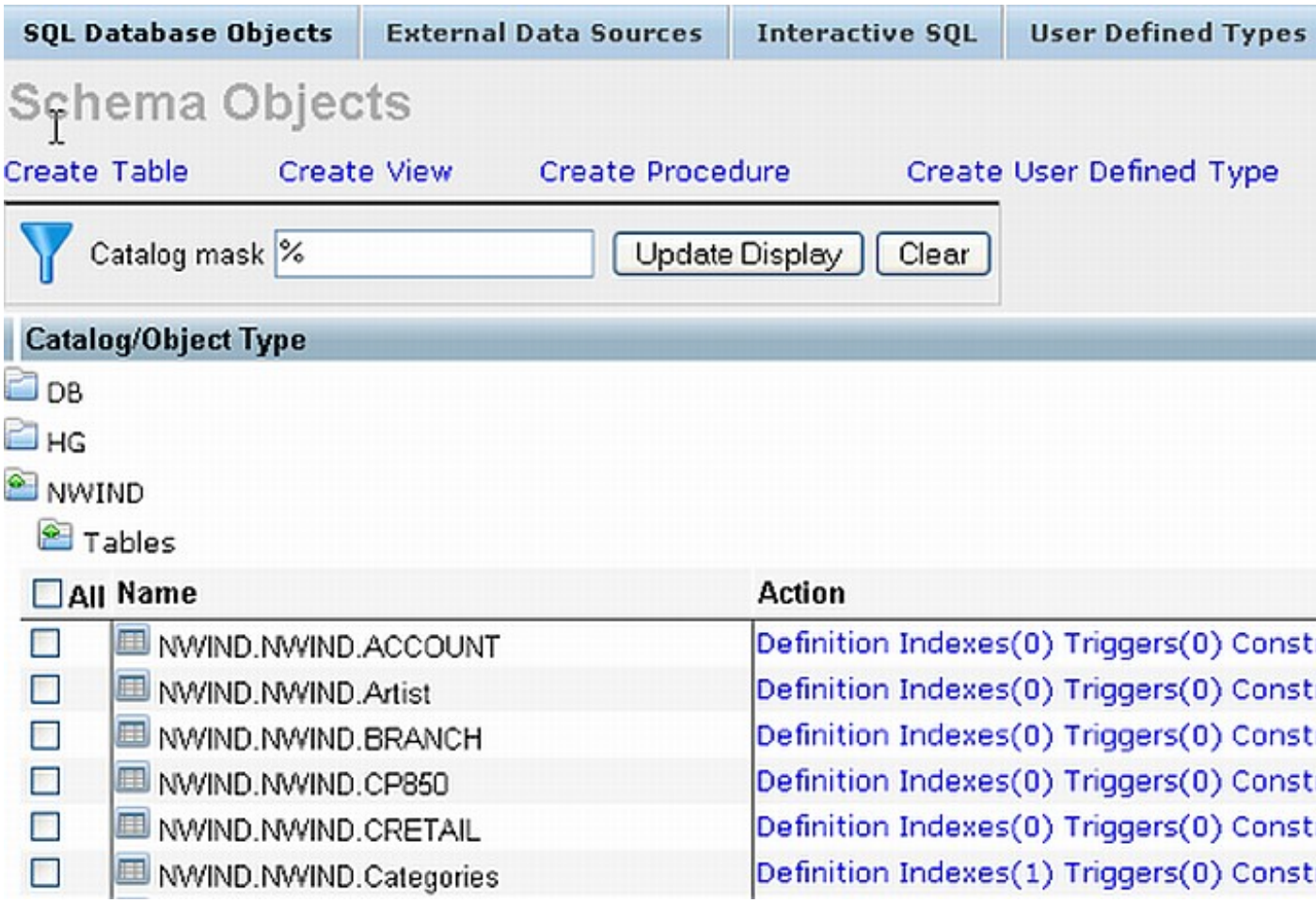


12. Select Execute to see data from the remotely linked table.

Figure 8.659. Execute

13. The tables can also be viewed as part of the Virtuoso "SQL Schema Objects" under the "Northwind" catalog name.

Figure 8.660. SQL Schema Objects



The Link process is now complete enabling the tables to be queried as if part of the Virtuoso Schema.

8.13.4. Creating EDM in Visual Studio 2008

The following steps can be used to create an Entity Data Model (EDM) for the Microsoft SQL Server Northwind database:

1. Launch the Visual Studio 2008 SP1 IDE.

Figure 8.661. Visual Studio 2008 SP1 IDE

2. Create a

   Web Application

   project by going to the

   File

   menu in Visual Studio and choosing

   New Project

   .
3. When the New Project window appears, choose either

   Visual Basic

   or

   Visual C#

   as your programming language.
4. Within the language category, click on

   Web

   and select

   ASP.NET Web Application

   from the right-hand panel.
5. Choose a name for the project, for example

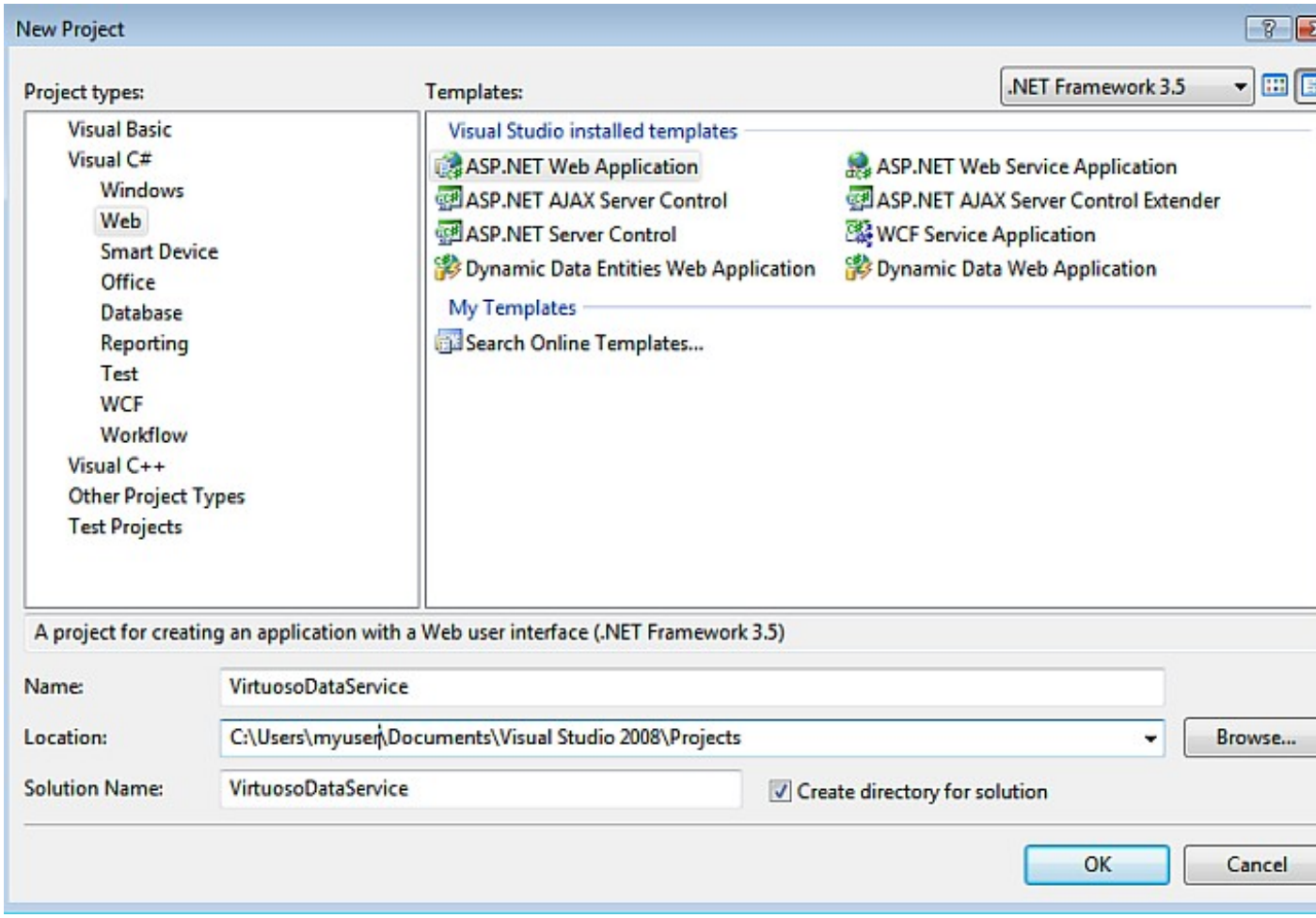
   VirtuosoDataService

   , and click

   OK

   .

Figure 8.662. name for the project

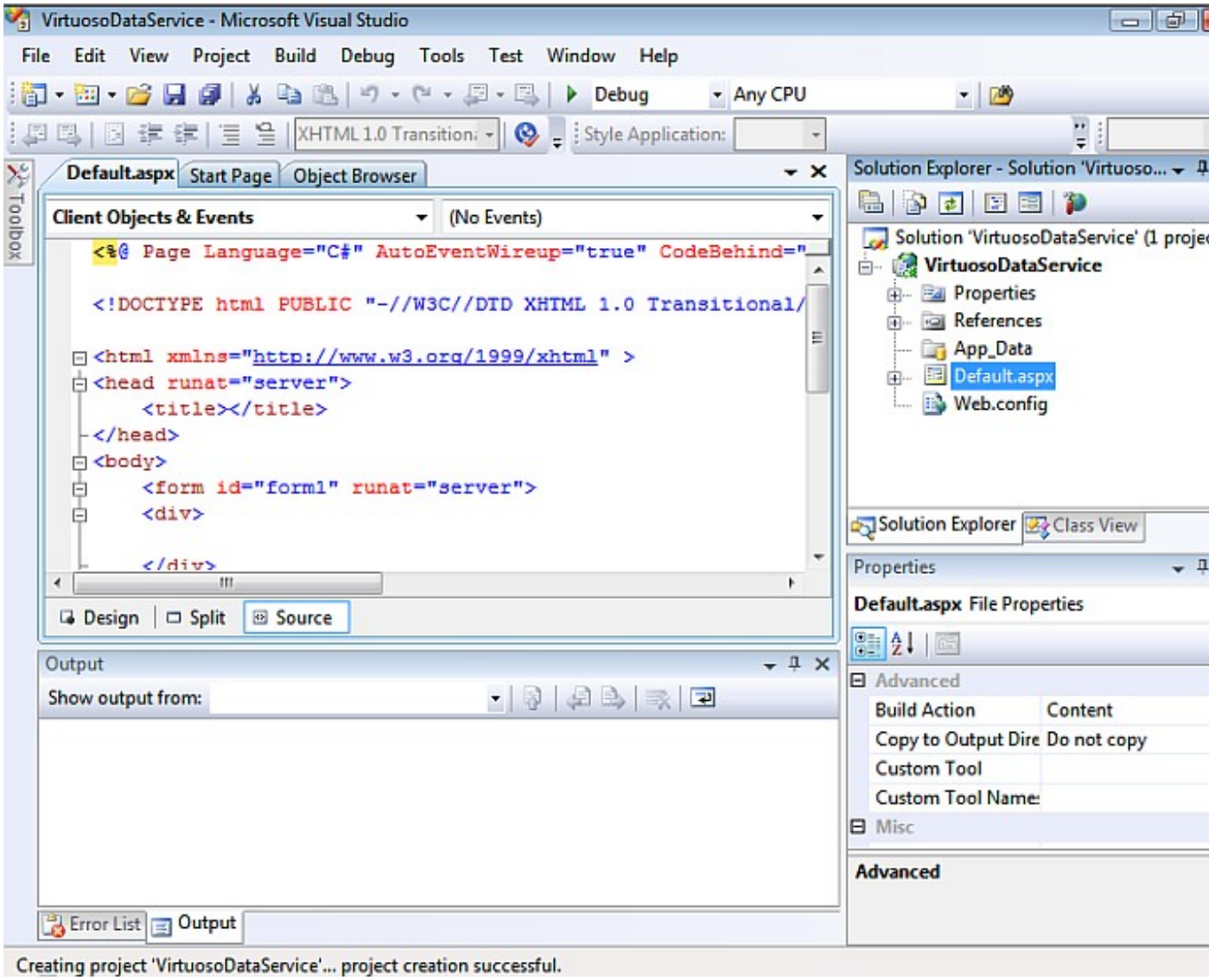


6. This will create a new project called

   VirtuosoDataService

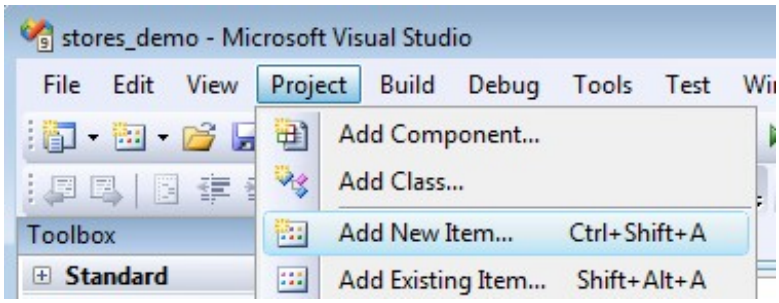
   .

Figure 8.663. create a new project



7. Right click on the VirtuosoDataService project name of the Solution Explorer pane, then select the Add -> New Item menu options.

Figure 8.664. VirtuosoDataService



8. The

Add New Item

dialog will appear. Choose the

ADO.NET Entity Data Model

template. Give it the name

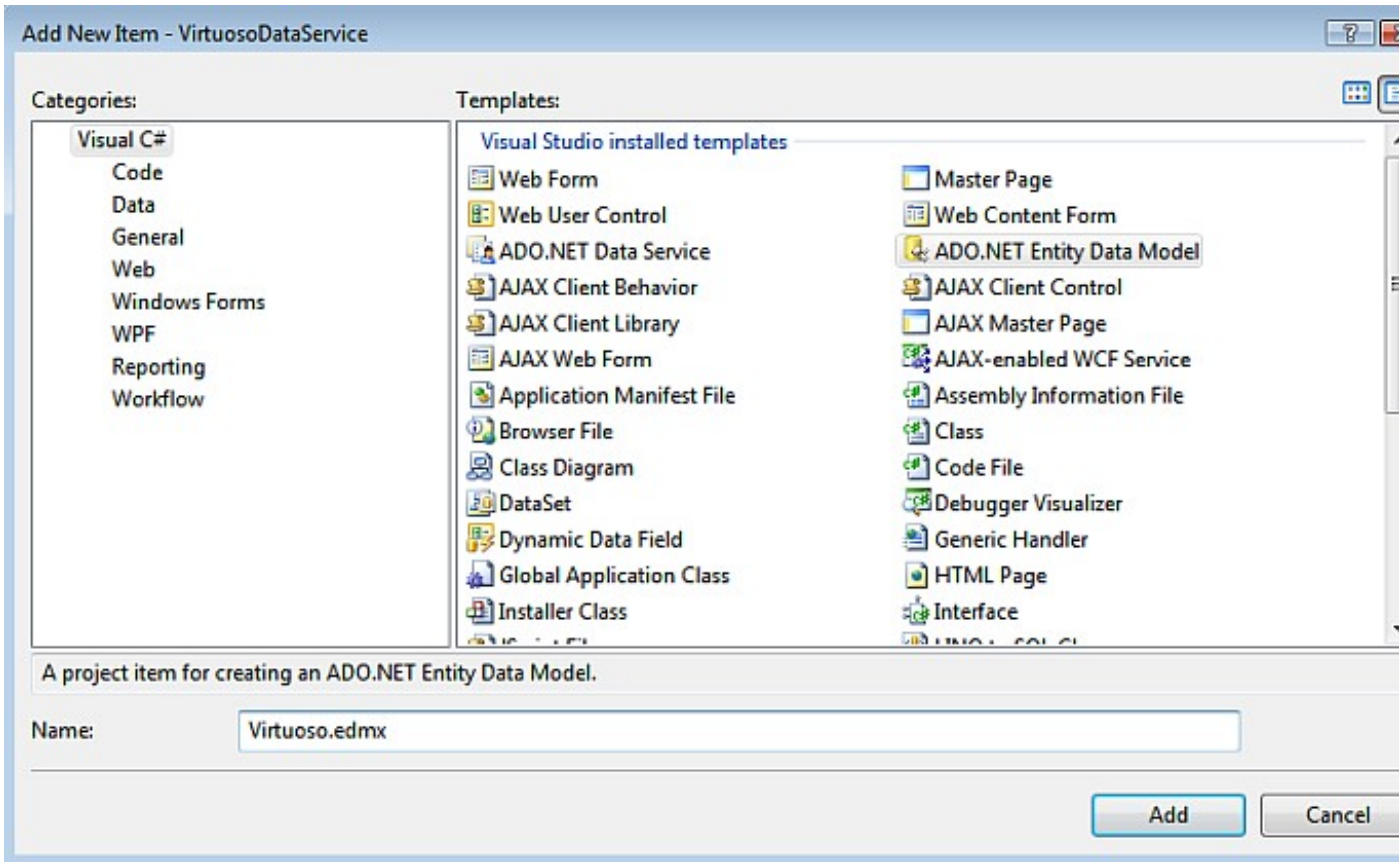
Virtuoso.edmx

and click

Add

to start the creation of the ADO.Net Entity Data Model.

Figure 8.665. Add New Item



9. In the

Entity Data Model Wizard

dialog

Choose Model Contents

page select the

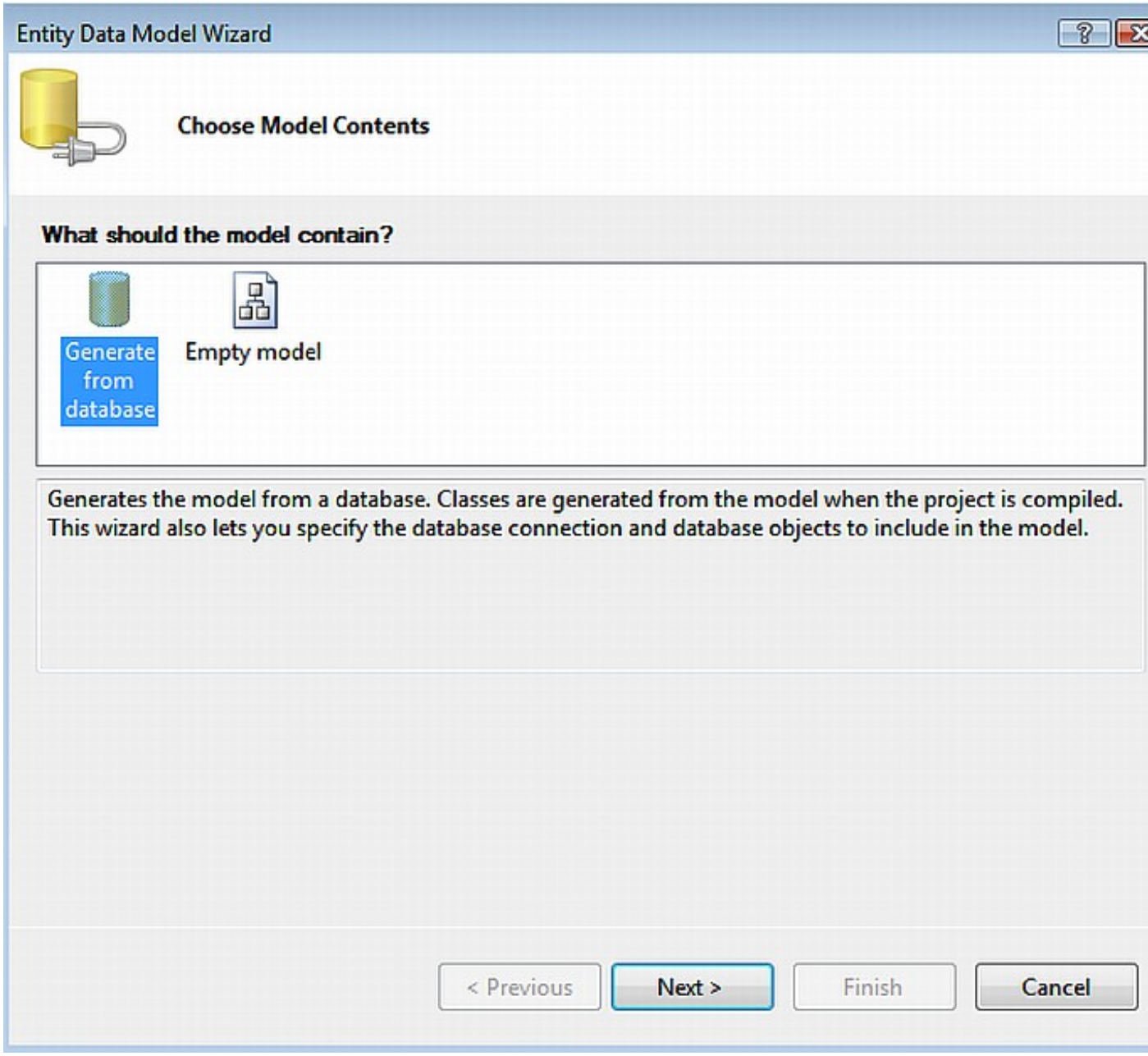
Generate from Database

model type and click

Next

.

Figure 8.666. Choose Model Contents



10. In the

    Entity Data Model Wizard

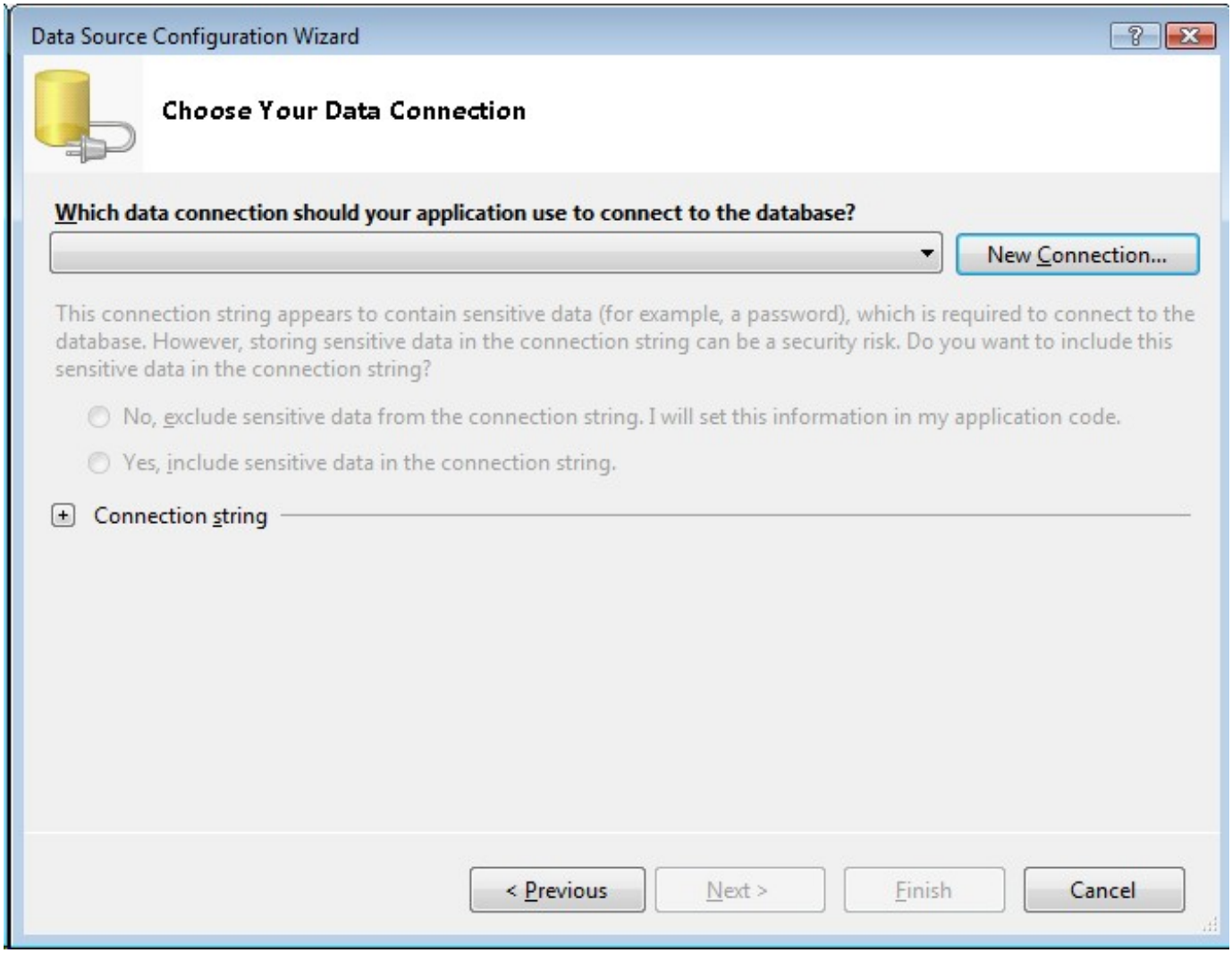
    dialog

    Choose your Data Connection

    page, select the

    New Connection button

Figure 8.667. Entity Data Model Wizard



11. In the

    Choose Data Source

    dialog, select the OpenLink

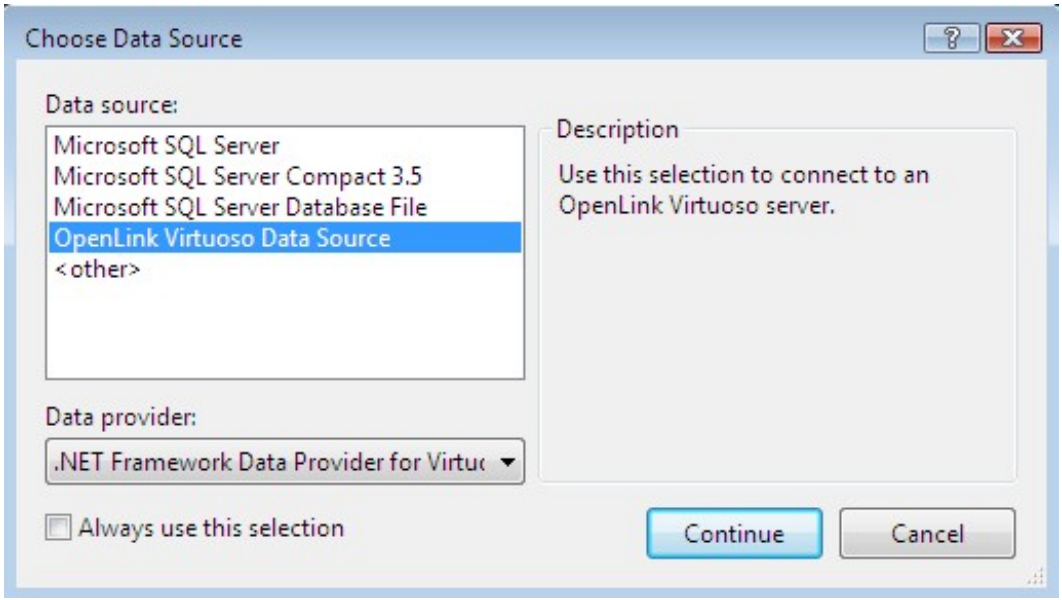
    Virtuoso Data Source

    from the list and click

    Continue

    .

Figure 8.668. Choose Data Source



12. In the

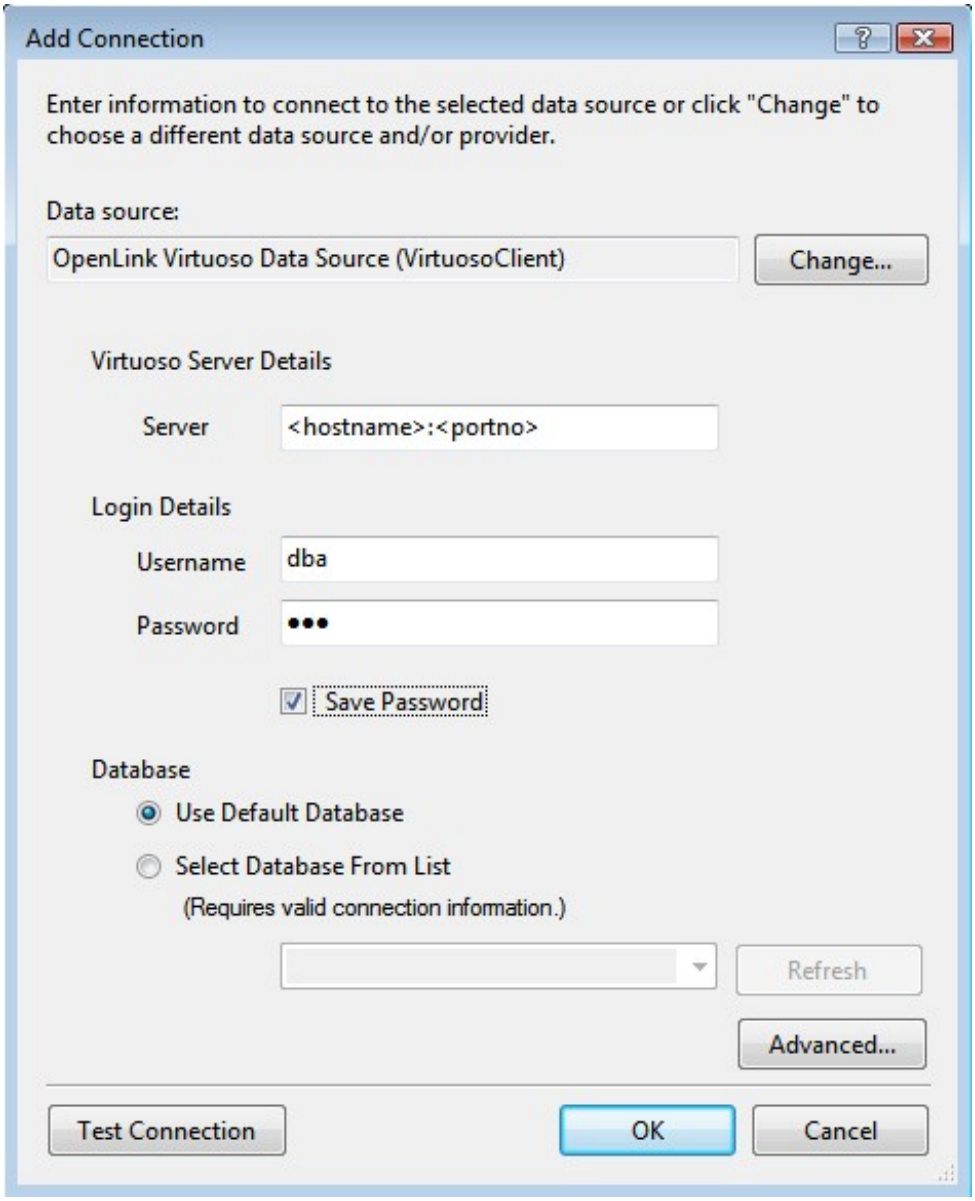
    Add Connection

    dialog, specify the

    hostname, portno, username and password

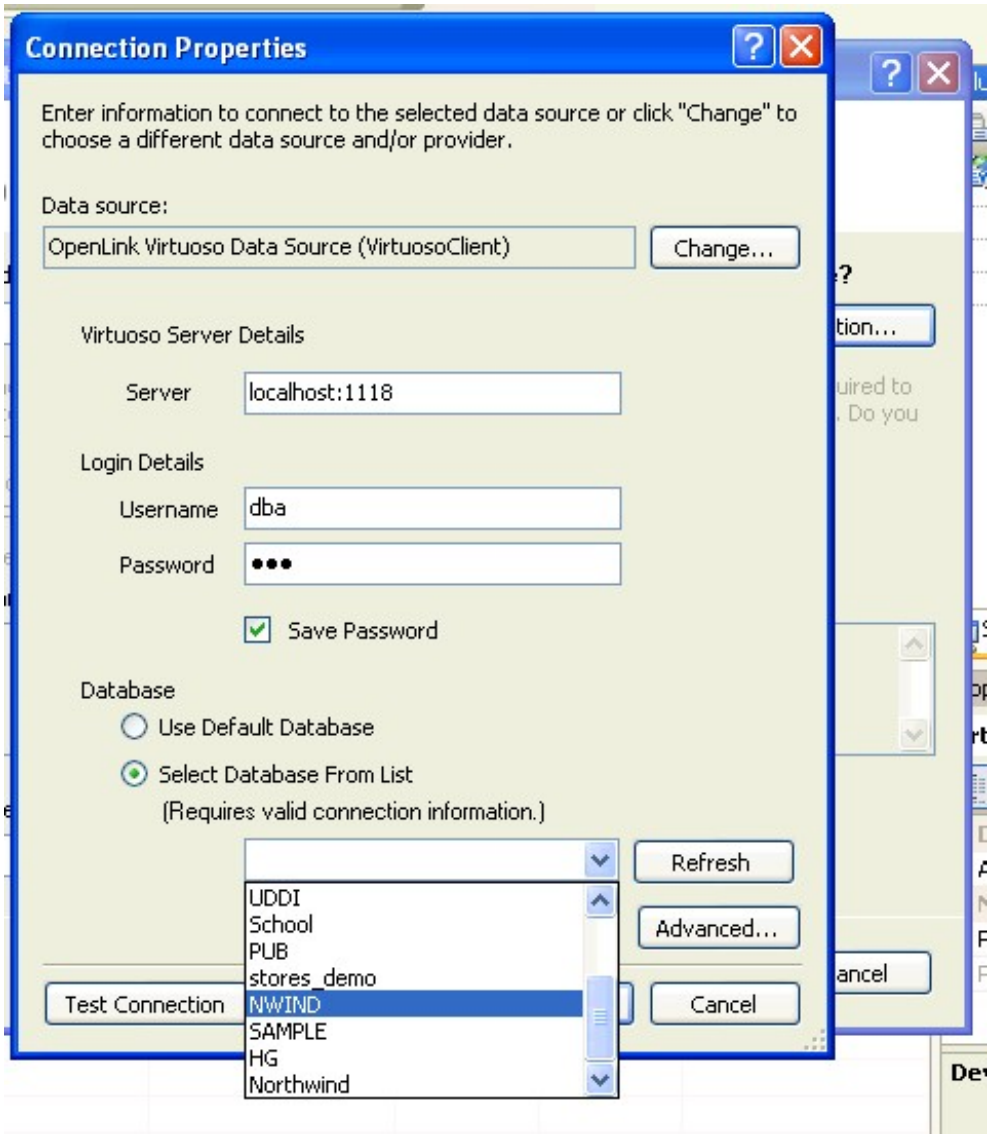
    for the target Virtuoso Server and click the Advanced button.

    Figure 8.669. Connection Properties



13. Use the Select Database From List dialog to select the NWIND catalog.

Figure 8.670. Add connection

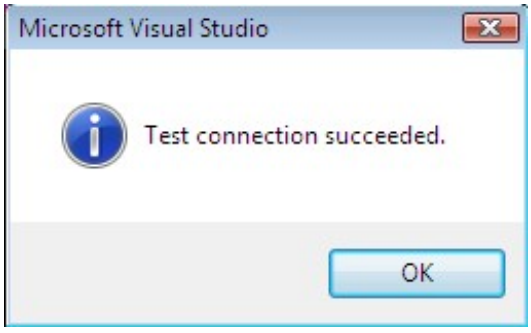


14. Press the

    Test Connection

    dialog to verify that the database is accessible. Click "OK" to persist the connection attributes, after a successful connection is verified.

    Figure 8.671. Test Connection



15. Set the

    entity connect string

    name to

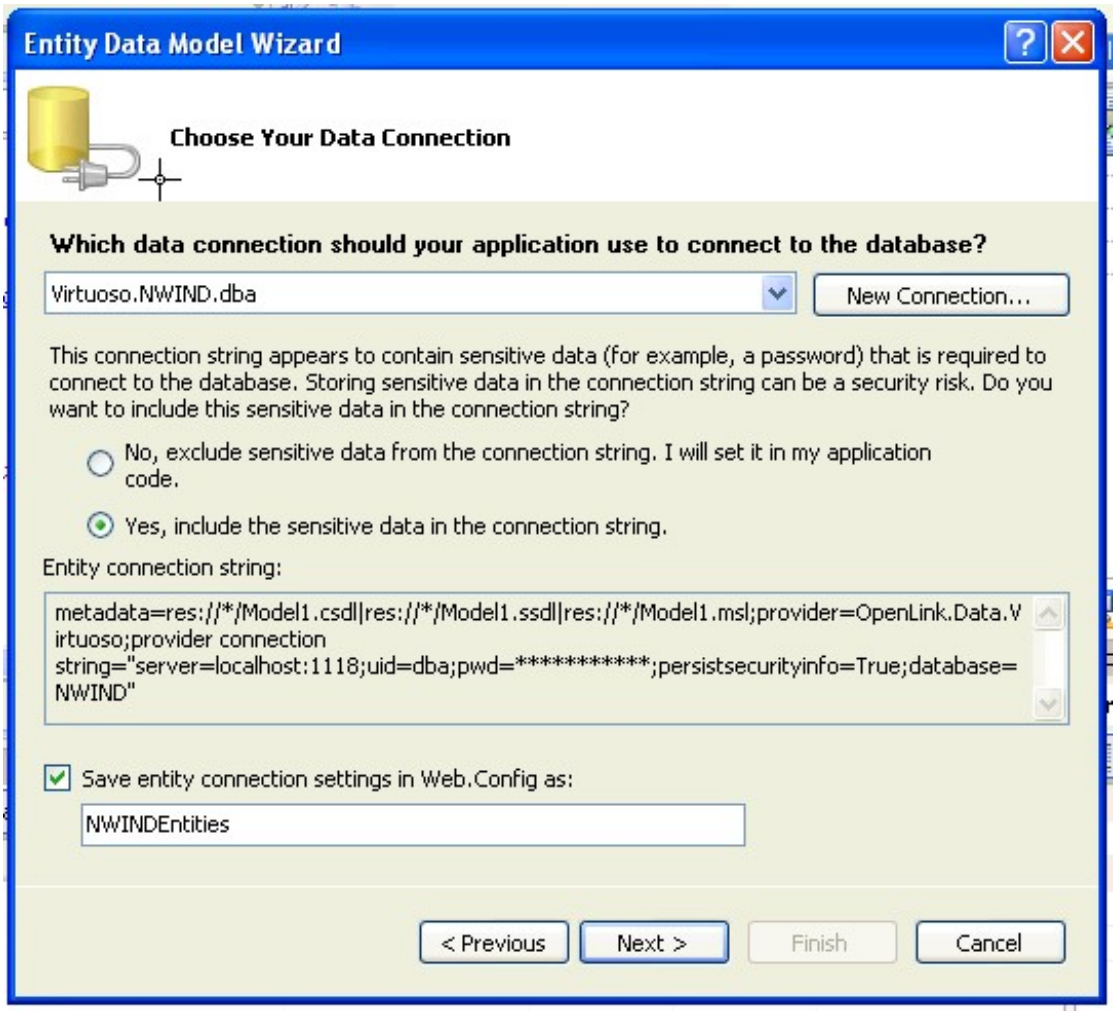
VirtuosoEntities

(note this name as it is required in the section on creating and ADO.Net Data Service) and click

Next

.

Figure 8.672. entity connect string



16. In the

    Choose your Database Objects

    page select the

    Tables

    check box to select all tables in the NWIND catalog for addition to the Entity Data Model. Set the

    Model Namespace

    to

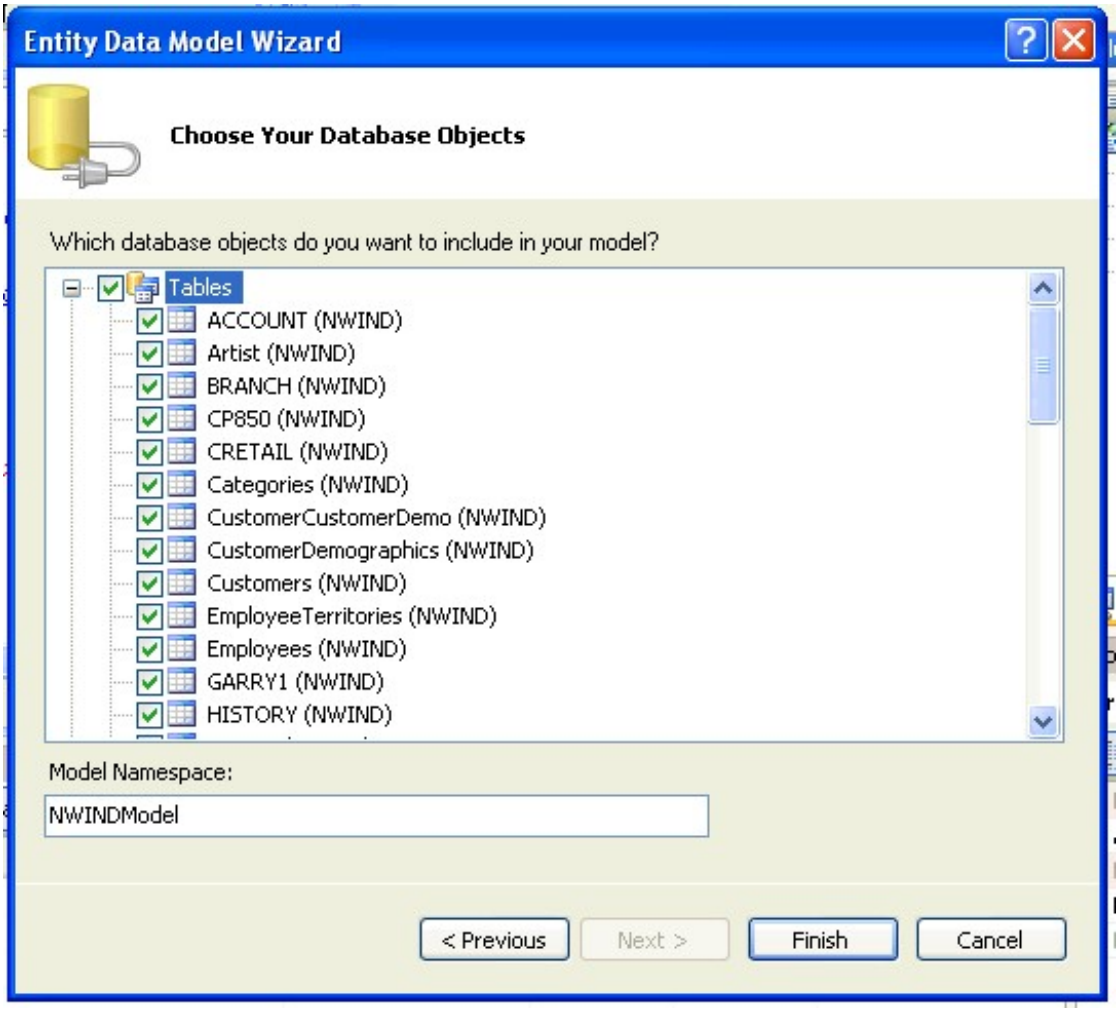
    VirtuosoModel

    and click

    Finish

.

Figure 8.673. Database Objects

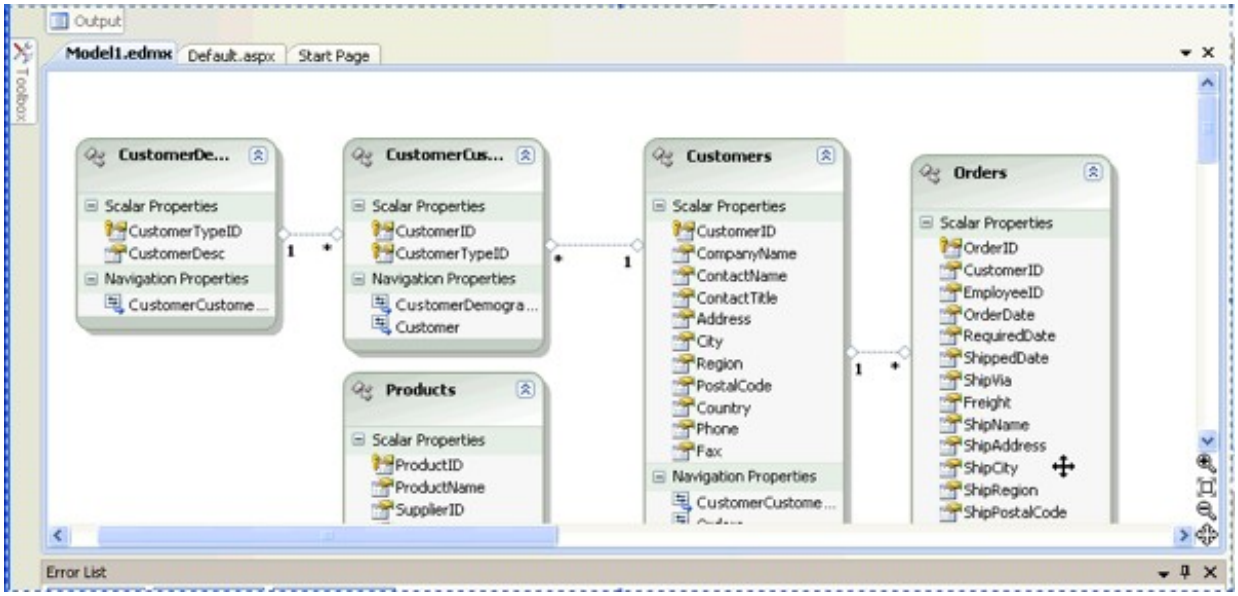


17. The

    Virtuoso.edmx

    EDM will be created with the tables and relationships displayed in the Visual Studio IDE

    Figure 8.674. Virtuoso.edmx



Creation for the Entity Data Model for the Microsoft SQL Server Northwind database is now complete.

8.13.5. Using EDM to create Entity Framework based applications

Now that a Microsoft Entity Data Model has been created for the Microsoft SQL Server Northwind database, Entity Framework applications can be created to make use of it.

Entity Frameworks based ADO.NET Data Service

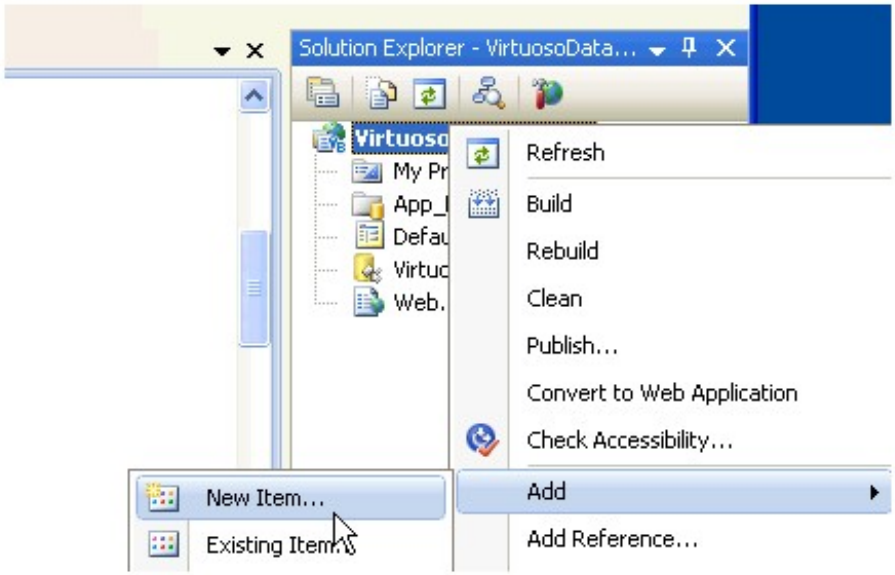
An ADO.Net Data Service for the Microsoft SQL Server tables can be created using the Entity Data Model created in the Creating EDM in Visual Studio 2008 section .

1. Open the

   VirtuosoDataService

   project created in the Creating EDM in Visual Studio 2008 section .
2. Select the Project -> Add New Item menu option.

Figure 8.675. VirtuosoDataService



3. The

   Add New Item

   dialog will appear. Choose the

   ADO.NET Data Service template

   . Give it the name

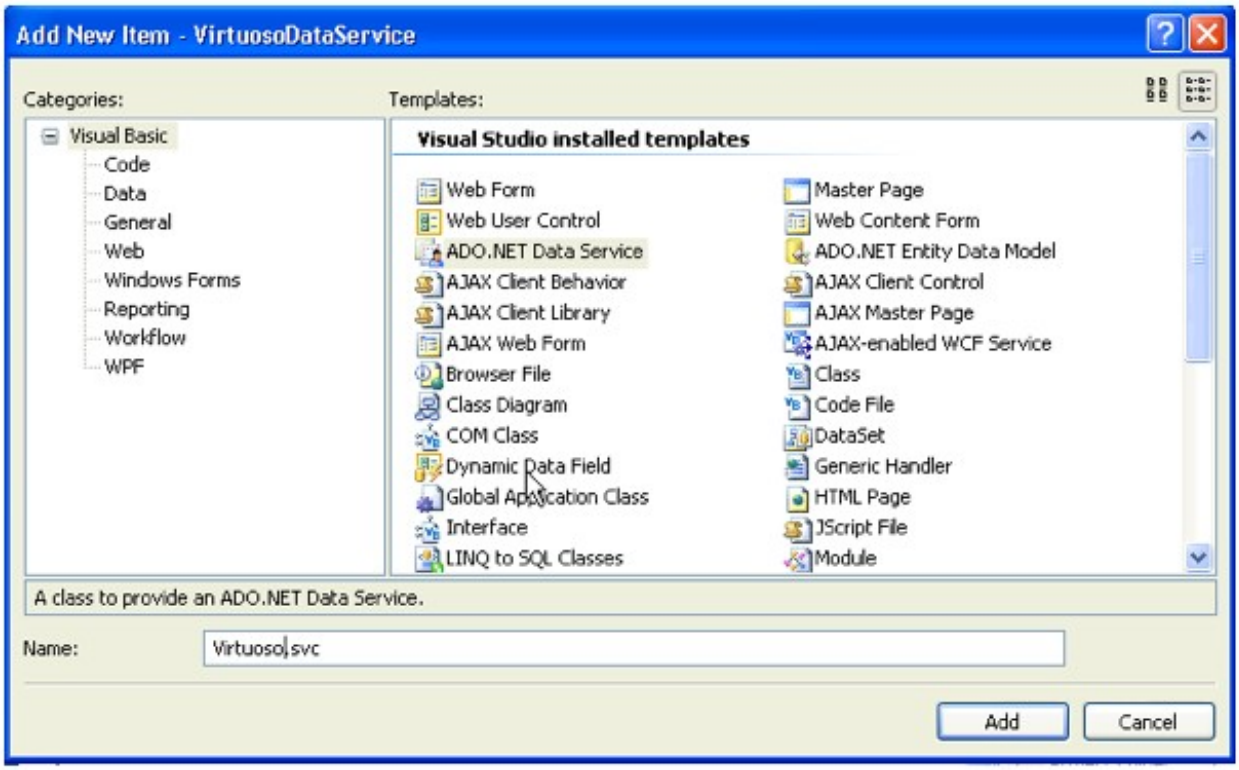
   Virtuoso.svc

   , and click

   Add

   to create the ADO.Net Data Service.

   Figure 8.676. Add New Item



4. In the newly created

   Virtuoso.svc.cs

   Data Service file, add the data source class name of

   VirtuosoEntities

   (note this is the name set in the Creating EDM in Visual Studio 2008 section) as the

   DataService

   name. Enable the access to the Data Service by adding the entry

   config.SetEntitySetAccessRule("", EntitySetRights.All);*

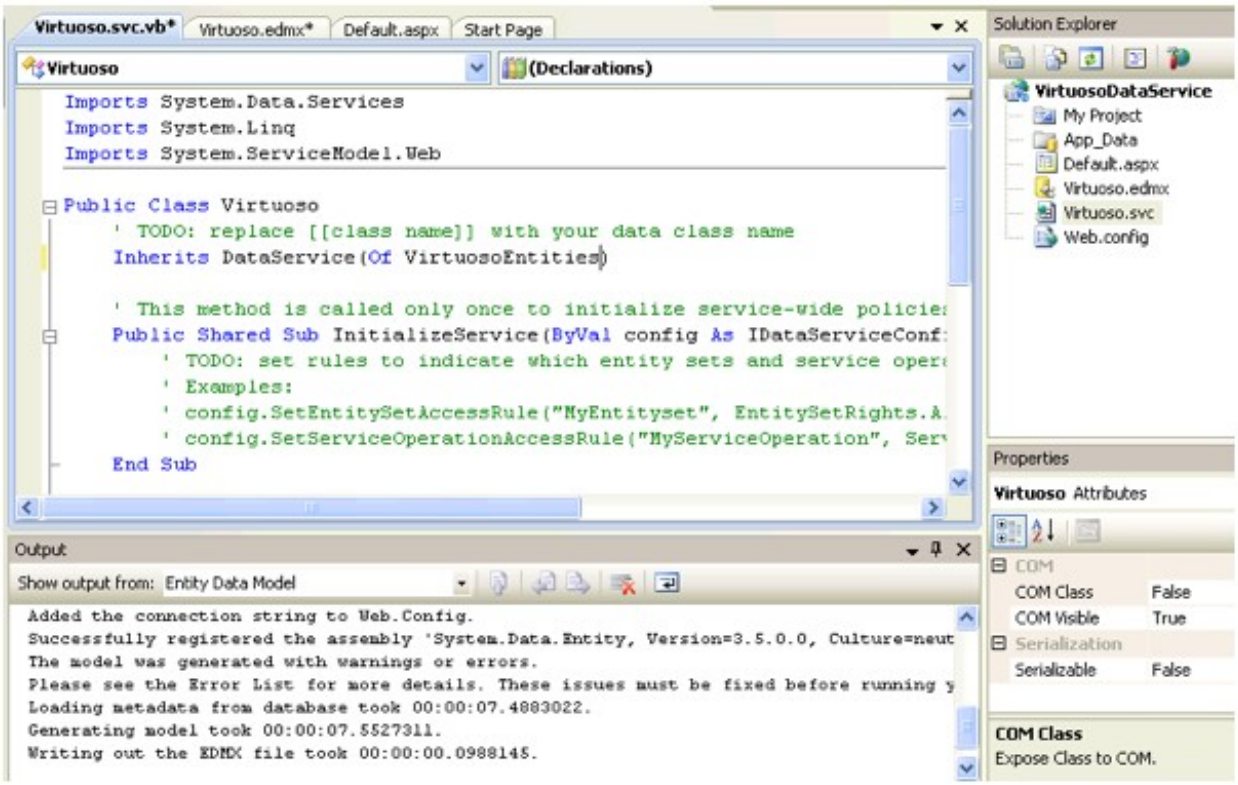
in the

InitializeService

method.

```
// C#

using System;
using System.Web;
using System.Collections.Generic;
using System.ServiceModel.Web;
using System.Linq;
using System.Data.Services;

namespace SimpleDataService
{
    public class Northwind : DataService<VirtuosoDemoEntities>
    {
        public static void InitializeService(IDataServiceConfiguration  config)
        {
            config.SetEntitySetAccessRule("*", EntitySetRights.All);
        }
    }
}
```

Figure 8.677. Virtuoso.svc.cs

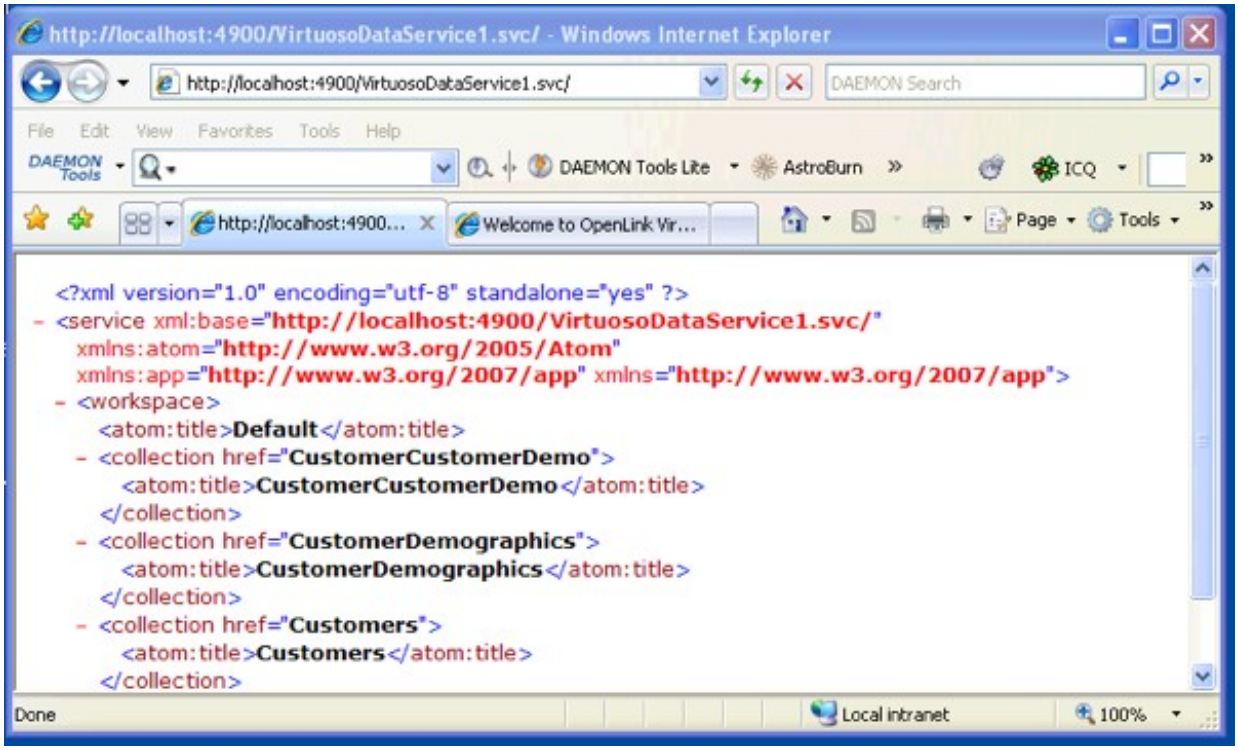


5. To test the Data Service, simply hit

   Ctrl+F5

   within Visual Studio. This will start the development web server, run the Data Services server inside and load a Web browser page displaying the list of available tables/entities for the Northwind database catalog.

Figure 8.678. Data Service test

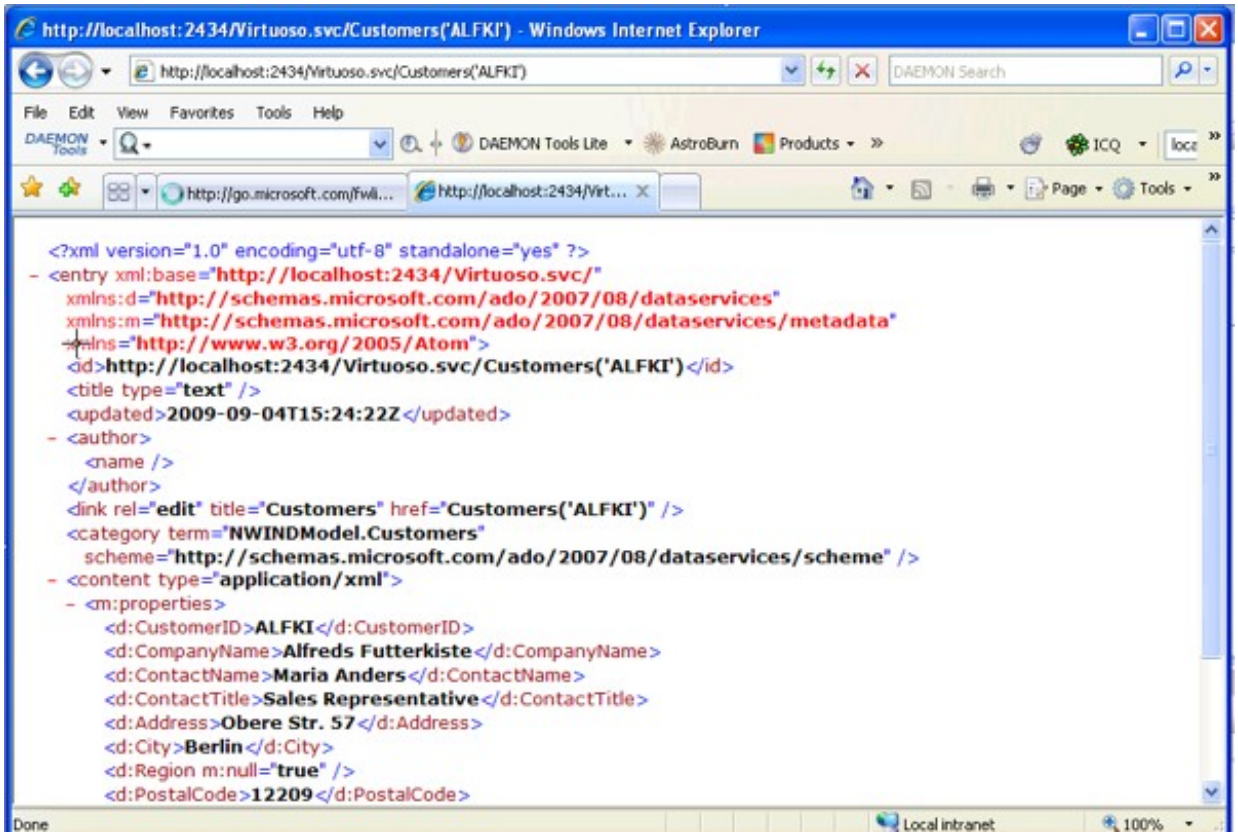


6. To access a specific entity instance like the

   Customers

   table ALFKI record, use this convention http://host/vdir/Virtuoso.svc/Customers('ALFKI').

   Figure 8.679. EMPLOYEES



Notes:

1. *Important*

   - To view

   Atom

   (the default format returned by an ADO.NET Data Service) in Internet Explorer, you must first ensure that

   Feed Reading View

   is turned

   off

   . This can be done on the

   Content tab

   of

   Tools in Internet Options

   .
2. If a Data Services entity instance URI page fails to load you can turn

   Verbose

   errors on by adding

   config.UseVerboseErrors = true;

   in the

   virtuoso.svc.cs InitializeService

   method to obtain more detailed information from the server as to why the page failed to load:

```
public static void InitializeService(IDataServiceConfiguration config)
{
config.UseVerboseErrors = true;
config.SetEntitySetAccessRule("*", EntitySetRights.All);
}
```

8.14. Parallel Operations and Bulk Data Transfer with Remote Tables

Especially when bulk copying data from a remote database into Virtuoso, parallel operation is essential for performance.

From version 7.00.3206, automatic query parallelization applies to selected remote table operations.

If read committed isolation is sufficient for remote table access, the remote table access is automatically parallelized if there is an integer key in the remote table that can be used for partitioning.

When the table is attached or when its statistics are updated with sys_stat_analyze, the remote schema is read and if there is a numeric first part of primary key, its minimum and maximum values are read. Samples of rows are read from the table at different ranges between the minimum and maximum and the values are used for filling in column value statistics in sys_col_stat.

If a remote table operation has no condition on the first part of primary key, a range condition is automatically added if the scan of the remote table is the first in a sequence of nested joins. This is the case for example when scanning the table inside a group by,

order by, hash join build side or other operation that does not immediately return rows to the client. The number of parallel ranges is given by ThreadsPerQuery in the ini file, accessed as enable_qp with sys_stat and __dbf_set. In a cluster setting, the same DSN is expected to be defined on all processes. The cluster splitting will scan a range per each slice of the elastic cluster.

The lowest range does not specify a low bound and the highest does not specify an upper bound, thus the min and max of the range column do not have to be exactly up to date.

It may be that the primary key of the remote table does not start with a numeric column. If this is the case but there still exists a column usable for splitting a scan, one can create the table in Virtuoso and declare the splitting column as first in the primary key. One can then use vd_remote_table to declare the modified table as remote and use sys_stat_analyze for getting the statistics. After this, parallel scans with the selected column used for setting ranges can be done.

Use the explain function to verify that a scan is actually parallel. One expects to see the text of the remote select followed by "split by range of <column name>" below the text of the remote statement.

In a parallel scan each range is scanned over a different connection. The transaction contexts are independent and the isolation cannot be higher than read committed. Setting a higher isolation level or selecting for update disables the parallel scan feature.

The below snippet copies a table from another Virtuoso.

```
use R1;
ATTACH TABLE lineitem
        FROM '1208'
        USER 'dba'
    PASSWORD 'dba';
```

This makes a a remote table R1.1208.LINEITEM. This is the lineitem table from the DSN "1208".

Now make a local table:

```
use DB;
CREATE TABLE LINEITEM (
    L_ORDERKEY        INTEGER NOT NULL,
    L_PARTKEY         INTEGER NOT NULL,
    L_SUPPKEY         INTEGER NOT NULL,
    L_LINENUMBER      INTEGER NOT NULL,
    L_QUANTITY        double precision NOT NULL,
    L_EXTENDEDPRICE   double precision NOT NULL,
    L_DISCOUNT        double precision NOT NULL,
    L_TAX             double precision NOT NULL,
    L_RETURNFLAG      CHAR(1) NOT NULL,
    L_LINESTATUS      CHAR(1) NOT NULL,
    L_SHIPDATE        DATE NOT NULL,
    L_COMMITDATE      DATE NOT NULL,
    L_RECEIPTDATE     DATE NOT NULL,
    L_SHIPINSTRUCT    CHAR(25) NOT NULL,
    L_SHIPMODE        CHAR(10) NOT NULL,
    L_COMMENT         VARCHAR(44) NOT NULL,
    PRIMARY KEY       (L_ORDERKEY, L_LINENUMBER) column );
```

Note the column modifier after the primary key. This ensures that the table is created in the column store mode. When copying large tables over VDB, this is nearly always done for warehousing, not for OLTP, so the column store mode is essentially always best. Insert is faster and the table takes much less space.

To transfer the data from the remote table into the local one:

```
log_enable (2);
INSERT INTO lineietm
     SELECT *
       FROM r1..llineiten;
checkpoint;
```

log_enabel (2) turns off logging an transactions and allows auto committing insert. The explicit checkpoint makes the changes durable. Killing the database server during the transfer would start without any of the effects of the transfer since there is no logging. The data reading and insertion is automatically parallelized.

Most such data transfer operations are in fact network bound since the local insert rate is well in excess of 100MB/s on a commodity server and a 1gbE connection may only transfer around 80MB/s.

The database is normally online during the transfer and the progress may be tracked by periodically counting the target table.

Chapter 9. SQL Reference

Abstract

SQL Reference.

9.1. Datatypes

CHARACTER
: CHARACTER

  VARCHAR

  VARCHAR'('INTNUM')'

  NVARCHAR

  NVARCHAR'('INTNUM')'

  CHAR'('INTNUM')'

NUMERIC
: NUMERIC

  NUMERIC'('INTNUM')'

  NUMERIC'('INTNUM','INTNUM')'

DECIMAL
: DECIMAL

  DECIMAL'('INTNUM')'

  DECIMAL'('INTNUM','INTNUM')'

INTEGER
: INT

  INTEGER

  SMALLINT

FLOAT
: FLOAT

  FLOAT'('INTNUM')'

REAL
: REAL

  DOUBLE PRECISION

LONG VARCHAR
: BLOB data

VARBINARY [(precision)]
: Binary data

LONG VARBINARY
: Binary BLOB data

TIMESTAMP

TIMESTAMP

DATETIME

TIME

DATE

<UDT>
: User Defined Type with varbinary-like size properties.

LONG <UDT>
: User Defined Type with LONG varbinary-like size properties.

LONG XML
: LONG XML BLOB-like data type.

Note:
User Defined Types can be created from native or external types, composites or classes from any hosted language such as Java or .Net. Any User Defined Type can be used to define a column in a CREATE TABLE statement.

9.1.1. Date Literals

Virtuoso does not support date literals or the DATE reserved keyword. Literal dates should be enclosed in a conversion function such as `stringdate()` , as in this example:

```
select * from demo.demo.orders o
  where o.orderdate between stringdate('1994-01-01') And stringdate('1997-12-31')
```

Alternatively type casts can be used to explicitly instruct Virtuoso to assume a string as a date, see below.

9.1.2. Casting

Blob types can be cast to varchars. This will produce a string of up to 16 MB in length and an error for longer blobs.

Nothing can be cast to a blob type. Blobs only come from selecting blob columns and are created by assigning values to such columns.

Converting non-integer values to integers rounds towards 0.

Any data type can be converted to binary with the VARBINARY target data type. The result may or may not be meaningful. Specifically datetime can be converted to binary and back.

```
cast_expression :
                CAST '(' scalar_exp AS data_type [COLLATE collation_name ] ')'

collation_name :
        identifier
        | owner '.' identifier
        | qualifier '.' owner '.' identifier
```

The CAST expression converts the type of the scalar_exp into the data_type, if possible. If the conversion is not meaningful, as from a float to a date, an error is signalled.

CAST is the recommended way of converting between data types, including any conversion between strings, different number types and datetime types.

Example 9.1. Examples:

```
select cast ('2000-1-3' as date);

select cast (cast ('2000-1-3' as date) as varchar);
        = 2000-01-03 00-00-00 000000
```

9.1.3. Time & Date Manipulation

The SQL92 standard functions for time and date queries are available. These are mapped to Virtuoso internal functions as follows:

CURRENT_DATE - `curdate()`
CURRENT_TIME - `curtime()`
CURRENT_TIMESTAMP - `curdatetime()`
The results of the above functions can also be obtained using the `now()` function and casting to the appropriate target type.

Dates and times should be input or compared as literals in a standard format. The following table describes the proper methods available:

Table 9.1. Time & date syntax

Datatype	ODBC Syntax	SQL92 Casting
Date	{d 'yyyy-mm-dd'}	cast('yyyy-mm-dd' as date)
Time	{t 'hh:mm:ss'}	cast('hh:mm:ss' as time)
Datetime/timestamp	{ts 'yyyy-mm-dd hh:mm:ss[.f...]'}	cast('yyyy-mm-dd hh:mm:ss[.f...]' as datetime)

Example 9.2. Example: comparing dates

```
select datecol from table
  where datecol > cast('1900-01-01' as date)
    and datecol < cast(now() as date);
```

now() is cast for explicit compatibility although not required.

9.1.4. Declaring Collations of Expressions

A collation can be declared with CAST for string expressions. Suppose insensitive_ascii were a collation name specifying case insensitive comparison of ASCII strings, the predicate:

```
CAST 'foo' as varchar collate insensitive_ascii) = CAST 'FOO as varchar collate insensitive_ascii)
```

would be true.

This will alter the semantic of string comparison and can be used to define a collation where the collation would otherwise be default, for instance when comparing results of expressions or constants. This can also alter the default collation implied by the collation declaration in column options in CREATE TABLE.

9.2. User Defined Types

A user-defined type is a schema object, identified by a user-defined type name. The definition of a user-defined type specifies a number of components, including in particular a list of attribute definitions. The representation of a user-defined type is expressed as a list of attribute definitions.

The definition of a user-defined type may include a method specification list consisting of one or more method specifications. A method specification is either an original method specification or an overriding method specification. Each original method specification specifies the method name, the SQL parameter declaration list, the returns data type, the <language clause>, the language (if the language is not SQL), and whether it is a STATIC or CONSTRUCTOR method.

Each overriding method specification specifies the method name, the SQL parameter declaration list and the RETURNS data type. For each overriding method specification, there must be an original method specification with the same method name and SQL parameter declaration list in some proper supertype of the user-defined type. Every SQL-invoked method in a schema must correspond to exactly one original method specification or overriding method specification associated with some user-defined type existing in that schema. A method `M` that corresponds to an original method specification in the definition of a structured type `T1` is an original method of `T1` . A method `M` that corresponds to an overriding method specification in the definition of `T1` is an overriding method of `T1` . A method `M` is a method of type `T1` if one of the following holds:

M is an original method of T1

M is an overriding method of T1

There is a proper supertype T2 of T1 such that M is an original or overriding method of T2 and such that there is no method M3 such that M3 has the same method name and SQL parameter declaration list as M and M3 is an original method or overriding method of a type T3 such that T2 is a proper supertype of T3 and T3 is a supertype of T1 .

A user defined type can be a direct subtype of one (and only one) user defined type. The user defined type cannot be a subtype of itself.

A type Ta is a direct subtype of a type Tb if Ta is a proper subtype of Tb and there does not exist a type Tc such that Tc is a proper subtype of Tb and a proper supertype of Ta .

A type Ta is a subtype of type Tb if one of the following pertains:

Ta is a direct subtype of Tb ; or

Ta is a subtype of some type Tc and Tc is a direct subtype of Tb .

By the same token, Tb is a supertype of Ta and is a direct supertype of Ta in the particular case where Ta is a direct subtype of Tb . If Ta is a subtype of Tb , then Ta is proper subtype of Tb and Tb is a proper supertype of Ta . A type cannot be a proper supertype of itself. A type with no proper supertypes is a maximal supertype. A type with no proper subtypes is a leaf type.

Let Ta be a maximal supertype and let T be a subtype of Ta . The set of all subtypes of Ta (which includes Ta itself) is called a subtype family of T or (equivalently) of Ta . A subtype family is not permitted to have more than one maximal supertype. Every value in a type T is a value in every supertype of T . A value V in type T has exactly one most specific type MST such that MST is a subtype of T and V is not a value in any proper subtype of MST . The most specific type of value need not be a leaf type. For example, a type structure might consist of a type PERSON that has STUDENT and EMPLOYEE as its two subtypes, while STUDENT has two direct subtypes UG_STUDENT and PG_STUDENT . The invocation STUDENT( ) of the constructor function for STUDENT returns a value whose most specific type is STUDENT , which is not a leaf type. If Ta is a subtype of Tb , then a value in Ta can be used wherever a value in Tb is expected. In particular, a value in Ta can be stored in a column of type Tb , can be substituted as an argument for an input SQL parameter of data type Tb , and can be the value of an invocation of an SQL-invoked function whose result data type is Tb . A type T is said to be the minimal common supertype of a set of types S if T is a supertype of every type in S and a subtype of every type that is a supertype of every type in S .

Note:

Because a subtype family has exactly one maximal supertype, if two types have a common subtype, they must also have a minimal common supertype. Thus, for every set of types drawn from the same subtype family, there is some member of that family that is the minimal common supertype of all of the types in that set.

A user-defined type is declared by a user-defined type CREATE TYPE statement .

9.2.1. CREATE TYPE Statement

```
CREATE TYPE type_name
   [ UNDER type_name ]
   [ LANGUAGE language_name [ EXTERNAL NAME literal ] ]
   [ AS (type_member, ...) ]
   [ type_option [type option ] .... ]
   [ method_specification, ... ]

type_name :
  [ [ identifier  .] [ identifier ] . ] identifier

type_member :
  identifier data_type [ DEFAULT literal ] [ EXTERNAL NAME string ] [ EXTERNAL TYPE string ]
    [ __SOAP_TYPE literal ] [ __SOAP_NAME literal ]

type_option : SELF AS REF | TEMPORARY | SOAP_TYPE literal

method_specification : original_method_specification | overriding_method_specification

original_method_specification :
  [ STATIC | INSTANCE ] METHOD identifier ( [ decl_parameter, ... ] )
    RETURNS data_type [ method_characteristics ]
  |
  CONSTRUCTOR METHOD identifier ( [ decl_parameter, ... ] )  [ method_characteristics ]
```

```
overriding_method_specification :
        OVERRIDING [ INSTANCE ] METHOD identifier ( [ decl_parameter, ... ] )
    RETURNS data_type

method_characteristics :
  [ EXTERNAL TYPE literal ] [ EXTERNAL NAME string | EXTERNAL VARIABLE NAME string ]

language_name : SQL | CLR | JAVA
```

The CREATE TYPE statements declares a user defined type. Generally speaking the user defined types can be in two states: forward-referenced, declared and instantiable.

A type is in forward reference state if it's name is quoted in some other CREATE TYPE statement (as a supertype, member type or a method parameter type or return type). When a type is in forward reference state it's instances can be copied, passed as parameter values and returned by functions, but it cannot be instantiated, no type members can be accessed and no type methods can be called. Forward references are temporary objects and they disappear at server shutdown.

A type moves to the declared state when a CREATE TYPE is executed for it. In that state type methods can be called, type members can be accessed, but the type cannot be instantiated.

A type goes into instantiable state from declared state when it has no supertype or it's supertype is also in instantiable state. The server tries to move the defined types to instantiable state on every CREATE TYPE statement.

Normally the type definitions are stored into the DB.DBA.SYS_USER_TYPES system table.

This has the following layout:

```
CREATE TABLE SYS_USER_TYPES
(
 UT_NAME VARCHAR,
 UT_PARSE_TREE LONG VARCHAR,
 UT_ID integer identity,
 UT_MIGRATE_TO integer,
 primary key (UT_NAME));
```

UT_NAME - the fully qualified user defined type name.

UT_PARSE_TREE - the user defined type definition (in machine readable form).

UT_ID - the ID of the type (used in persisting type instances to/from network/storage).

UT_MIGRATE_TO - reserved for future use.

If a TEMPORARY type_option is specified, the CREATE TYPE does not write the type definition into the table - it declares the type only in server's memory. TEMPORARY types are not persistable. They disappear when the server is restarted. A TEMPORARY type cannot be a supertype or a subtype of a non-TEMPORARY type.

The `SELF AS REF` option directs the server to return a reference to the type's instance when instantiating the type, as opposed to returning the instance itself. The references are explained in more detail in the `NEW` operator.

Note:

The CREATE TYPE is an autocommitting statement.

Example 9.3. Creating User Defined Types

This example creates a SQL implemented user defined type UDT_TEST with no supertype. It has two members : A and B, two constructor methods, a static method _ADD, an ADDIT method taking either zero or two arguments and an instance method SUB_IT.

```
create type UDT_TEST
  as (A integer default 1, B integer default 2)
  CONSTRUCTOR METHOD UDT_TEST(_a integer, _b integer),
  CONSTRUCTOR METHOD UDT_TEST(),
  STATIC METHOD _ADD(_xx integer, _yy integer) returns integer specific DB.DBA.static_add,
  METHOD ADDIT() returns integer,
  METHOD ADDIT(c integer) returns integer,
  METHOD SUB_IT () returns integer;
```

This creates a subtype of UDT_TEST named UDT_TEST_SUB. UDT_TEST_SUB extends the static method _ADD of UDT_TEST so it can also take 4 arguments, overrides the method ADDIT from UDT_TEST and defines a new instance method MULTIPLY_IT.

```
create type UDT_TEST_SUB under UDT_TEST
  as (C integer default 12, _D integer default 32)
  STATIC METHOD _ADD(_xx integer, _yy integer, _zz integer, _qq integer) returns integer,
  OVERRIDING METHOD ADDIT() returns integer,
  METHOD MULTIPLY_IT () returns integer;
```

This is a SQL wrapper for a public Java type testsuite_base (see testsuite_base.java).

```
create type testsuite_base language java external name 'testsuite_base'
as (
    protected_I integer external name 'protected_I' external type 'I',
    private_I integer external name 'private_I' external type 'I',
    sZ smallint external name 'Z' external type 'Z',
    sfalseZ smallint external name 'falseZ' external type 'Z',
    sB smallint external name 'B' external type 'B',
    sC smallint external name 'C' external type 'C',
    sS smallint external name 'S' external type 'S',
    sI int external name 'I' external type 'I',
    sJ int external name 'J' external type 'J',
    sF real external name 'F' external type 'F',
    sD double precision external name 'D' external type 'D',
    sL any external name 'L' external type 'Ljava/lang/Short;',
    sAI any external name 'AI' external type '[I',
    sAL any external name 'AL' external type '[Ljava/lang/Short;',
    sstr nvarchar external name 'str' external type 'Ljava/lang/String;',
    sdat datetime external name 'dat' external type 'Ljava/util/Date;',

    tF real external name 'F',
    "F" real,

    non_existant_var integer external name 'non_existant_var' external type 'I'
   )
    static method get_static_ro_I ()
        returns integer external type 'I' external variable name 'static_ro_I',
    static method get_static_I ()
        returns integer external type 'I' external variable name 'static_I',
    static method get_protected_static_I ()
        returns integer external type 'I' external variable name 'protected_static_I',
    static method get_private_static_I ()
        returns integer external type 'I' external variable name 'private_static_I',

    static method test_bool (x integer external type 'I')
        returns smallint external type 'Z' external name 'test_bool',

    constructor method testsuite_base (),
    constructor method testsuite_base (i integer external type 'I'),

    static method echoDouble (a double precision external type 'D')
        returns any external type 'Ljava/lang/Double;' external name 'echoDouble',
    static method getObjectType (a any external type 'Ljava/lang/Object;')
        returns varchar external type 'Ljava/lang/String;' external name 'getObjectType',
    static method echoThis (a testsuite_base external type 'Ltestsuite_base;')
        returns integer external type 'I' external name 'echoThis',
    static method static_echoInt (a integer external type 'I')
        returns integer external type 'I' external name 'static_echoInt',

    static method change_it (a testsuite_base)
        returns integer external type 'I' external name 'change_it',

    method "overload_method" (i integer external type 'I')
        returns integer external type 'I',

    method echoInt (a integer external type 'I')
        returns integer external type 'I' external name 'echoInt',

    method echoInt (a double precision external type 'D')
        returns integer external type 'I' external name 'echoInt',
```

```
    method protected_echo_int (a integer external type 'I')
        returns integer external type 'I' external name 'protected_echo_int',

    method private_echo_int (a integer external type 'I')
        returns integer external type 'I' external name 'private_echo_int',

    method "echoDbl" (a double precision)
        returns double precision,

    method non_existant_method (a integer external type 'I')
        returns integer external type 'I' external name 'non_existant_method',

    static method non_existant_static_var (a integer external type 'I')
        returns integer external type 'I' external variable name 'non_existant_static_var';
```

9.2.2. ALTER TYPE Statement

```
ALTER TYPE type_name
 (  ADD ATTRIBUTE type_member
 | DROP ATTRIBUTE identifier [ CASCADE | RESTRICT ]
 | ADD method_specification
 | DROP original_method_specification [ CASCADE | RESTRICT ]

type_name :
  [ [ identifier  .] [ identifier ] . ] identifier

type_member :
  identifier data_type [ DEFAULT literal ] [ EXTERNAL NAME string ] [ EXTERNAL TYPE string ]
    [ __SOAP_TYPE literal ] [ __SOAP_NAME literal ]

type_option : SELF AS REF | TEMPORARY | SOAP_TYPE literal

method_specification : original_method_specification | overriding_method_specification

original_method_specification :
  [ STATIC | INSTANCE ] METHOD identifier ( [ decl_parameter, ... ] )
    RETURNS data_type [ method_characteristics ]
  |
  CONSTRUCTOR METHOD identifier ( [ decl_parameter, ... ] )  [ method_characteristics ]

overriding_method_specification :
        OVERRIDING [ INSTANCE ] METHOD identifier ( [ decl_parameter, ... ] )
    RETURNS data_type

method_characteristics :
  [ EXTERNAL TYPE literal ] [ EXTERNAL NAME string | EXTERNAL VARIABLE NAME string ]

language_name : SQL | CLR | JAVA
```

The ALTER TYPE statements modifies a user defined type. It can be used for adding or dropping methods and members of user defined types.

Note:
The ALTER TYPE is an autocommitting statement.

Example 9.4. Altering User Defined Types

This example uses a SQL implemented user defined type UDT_ALTER_TYPE with no supertype defined as follows:

```
create type UDT_ALTER_TYPE as (A integer default 1)
method m1 (I integer) returns integer;
create method M1 (in I integer) returns integer for UDT_ALTER_TYPE
{
 return I;
};
```

Then it adds an attribute B to it :

```
alter type UDT_ALTER_TYPE Add attribute B integer default 2;
```

Then drops the original A attribute :

```
alter type udt_ALTER_TYPE drop attribute A;
```

Now let's add a new method M2 to the type :

```
alter type UDT_ALTER_TYPE Add method M2 (ID integer) returns integer;
create method M2 (in ID integer) returns integer for UDT_ALTER_TYPE
{
  return ID + 100;
};
```

And drop the M1 method :

```
alter type UDT_ALTER_TYPE drop method M1 (ID integer) returns integer;
```

9.2.3. DROP TYPE Statement

```
DROP TYPE type_name
```

This statement reverses the effect of CREATE TYPE statement. If the type has methods defined they are deleted as well. Note that forward references cannot be dropped by a DROP TYPE.

The DROP TYPE statement can be used only for dropping types that are not referenced in another type's UNDER statement.

Example 9.5. Dropping a user-defined type

Dropping the subtype from the previous section.

```
drop type UDT_TEST_SUB;
```

9.2.4. CREATE METHOD Statement

```
CREATE [ INSTANCE | STATIC | CONSTRUCTOR ] METHOD identifier
     ( parameter, .... ) [ RETURNS data_type ] FOR type_name
     {
       .....
     }

  parameter : { IN | OUT | INOUT } identifier data_type
```

For the SQL user defined types every method should be defined in order to be callable. It is an error to call CREATE METHOD for a non-SQL type's methods (as the methods are implemented in some other non-SQL language).

There is no DROP METHOD as each CREATE METHOD will override the current method definition (if any). The method name, parameter types and the return type should match exactly the method declaration (method_specification in CREATE TYPE).

The method name for the constructors is the name of the type (without the qualifier and the owner).

For the constructor and instance methods there is a variable named `SELF` (in scope for the compound statement defining the method) representing the current type instance.

The method members and other methods are not in scope in the method's compound statement. They should be accessed through the `SELF` variable.

Example 9.6. Simple method demonstration

This example defines the two parameter constructor for the UDT_TEST type. It sets the values for the type members A and B to the values of the constructor parameters. This example uses the SQL200n syntax for method mutators (see below).

```
create constructor method UDT_TEST (in _a integer, in _b integer)
  for UDT_TEST
```

```
{
  SELF := A(SELF, _a);
  SELF := B(SELF, _b);
  return SELF;
};
```

This defines the static method _ADD for the type UDT_TEST. Note that it does not use SELF - it would be a syntax error to do so.

```
create static method _ADD (in a1 integer, in a2 integer)
  for UDT_TEST
{
  return a1 + a2;
};
```

9.2.5. Type Instances

Every user defined type can have zero or more instances. Every instance knows it's type. The instances are SQL values. They are copyable. Instances for SQL types contain placeholders for the type's members. Instances for non-SQL types contain a reference to the real instance in type's hosted environment. (Java VM for JAVA and the CLR virtual machine for CLR). So copying an instance is different for SQL and non-SQL types. When a SQL type's instance is copied a new set of members placeholders is created and all members values are copied. This way the copy does not hold any link to it's original and they can be considered as two different instances. This means that changing members' value in the copy will not affect the original.

The non-SQL types instances hold only a reference to the real instance. So copying such an instance is equivalent of making another reference to the foreign object. Thus changing the member's value in the copy WILL affect the original. Usually the foreign virtual machines have a means of explicitly copying an instance, but they are not used by the Virtuoso server when copying the SQL values.

As with the other SQL values, an instance gets destroyed when it is no longer referenced.

9.2.6. Instance References

Because the SQL types instances cannot be referenced by more than one variable/type member Virtuoso PL implements instance references. The references are created for the types marked with `SELF AS REF type_option` (see CREATE TYPE). For such types the constructor(s) make a SQL value of reference type in addition to making the instance itself. The engine places the instances in a connection specific cache of instances and returns a SQL value of type reference which points to that instance. Copying the reference value will not cause copying the instance into the cache, so a `SELF AS REF` type will behave as a hosted class with respect to changing values in the copy. A new instance in the cache will be created only when the constructor for the type is called again. The server will accept a SQL reference value in every place where an instance value is expected. When a reference is serialized, as in when storing into a column of a table, the server will serialize the instance data, not the reference itself.

The connection's instance cache is cleared after the completion of the current server operation, i.e. completion of the top level state, statement invoked by a client or the completion of processing an HTTP request. The cache will thus survive multiple transactions if these are transacted within a single client initiated operation.

9.2.7. NEW Operator

```
[ NEW ] type_name ( [ parameter_value, .... ] )
```

This returns an instance (or reference to an instance) of the user defined type type_name. It will try to match a constructor by calculating the parameter types at compile time and matching the so produced signature to the closest constructor signature (see below: finding methods). The SQL types have an implicit constructor with no parameters that assigns the DEFAULT values to the type members (if any, otherwise NULL). When a SQL constructor is called it will have the `SELF` set-up to the result of calling the implicit constructor. The NEW operator is a scalar expression and can be used wherever the SQL syntax allows scalar expressions.

9.2.8. Finding Methods - Method Signatures Generation & Comparison

A method of a user defined type is identified uniquely by the combination of the following:

the method name
number of the method's parameters
the method's parameter types
the method's return type
This combination of a method attributes is called the method's signature.

The Virtuoso Server must know the method's types of the parameter values and the return type at compile time to calculate the method signature and find exactly the method to call in the types method table. This is different from the current practice in calling stored procedures, because the compile types are not used to find the procedure.

The majority of the system functions are known at compile time to return values of a certain SQL type (e.g.: LEFT is known to return VARCHAR, ATOI returns INTEGER etc). But there are some (e.g.: AREF) that may return values of more than one type. The Virtuoso server does type arithmetic for scalar expressions at compile time already (to be able to supply columns types of a result set to ODBC clients for example), but so far the calculated type has only informative value and was not used anywhere during the compilation. All of the type checks are done at runtime. The method/constructor invocation breaks that practice by using the calculated compile time types for the scalar expressions.

Example 9.7. Method Signatures

Consider a method `m1` of type `t1` taking an INTEGER parameter and returning an integer value:

```
CREATE METHOD m1 (in x integer)
  for t1 returns integer
{ return x + 10; };
```

Now consider calling the method as follows (in a Virtuoso/PL procedure):

```
...
declare p float;
declare ret integer;
declare t1i t1;

t1i := new t1();
p := 1;

ret := t1i.m1(p);
....
```

This will yield a compilation error explaining that there is no method `m1` of user defined type `t1` . It will do that because p has a compile time type of FLOAT.

The following will also fail to compile:

```
...
declare p integer;
declare ret float;
declare t1i t1;

t1i := new t1();
p := 1;

ret := t1i.m1(p);
....
```

This time the `ret` has a declared type of FLOAT and there is no method in t1 taking 1 INTEGER parameter and returning FLOAT.

The most consistent way of specifying the compile time type of a scalar expression is to enclose it in a CAST statement, as follows:

```
...
declare p float;
declare ret integer;
declare t1i t1;

t1i := new t1();
p := 1;

ret := t1i.m1(CAST (p as integer));
....
```

This will compile and execute correctly.

9.2.9. Getting & Setting Member Values of Type Instances (member observers & mutators)

Let `T` be a user defined type that has a member `A` of type `AT` . Let `IT` be an instance of type `T` .

Member Observers (Getting Values)

There are two alternative syntaxes (both scalar expressions):

```
SQL200n :  A(<scalar_exp>)
Virtuoso extension :  <scalar_exp>.A
```

Both of the above will return a copy of the member's value of the instance in <scalar_exp> when the scalar expression <scalar_exp> has a compile time type of `T` . If the compile time type is not determined to be a user defined type `T` the first one will be compiled as a call to the SQL function `A` and the second will either generate a syntax error or the server will consider it as reference to a scope variable (depending on the type of <scalar_exp>).

These are also scalar expressions and have a compile time type `AT` .

To specify an explicit type of the scalar expression there is a third syntax:

```
(<scalar_exp> as T).A
```

This will force the server to compile a reference to the member `A` in user defined type `T` . Whether the <scalar_exp> is indeed of type `T` will be checked at runtime.

Member Mutators (Setting Values)

There are two alternative syntaxes (both scalar expressions):

```
SQL200n :  A(<scalar_exp>, <new_value_scalar_exp>)
Virtuoso extension :  <scalar_exp>.A := <new_value_scalar_exp>
```

Both of the above will set the member's value of the instance in <scalar_exp> to a copy of <new_value_scalar_exp> when the scalar expression <scalar_exp> has a compile time type of `T` . If the compile time type is not determined to be a user defined type `T` the first one will be compiled as a call to the SQL function `A` and the second will either generate a syntax error or the server will consider it as reference to a scope variable (depending on the type of <scalar_exp>).

These are also scalar expressions and have a compile time type `T` and return a copy of the <scalar_exp>. To specify an explicit type of the scalar exp there is a third syntax:

```
(<scalar_exp> as T).A := <new_value_scalar_exp>
```

This will force the server to compile a reference to the member `A` in user defined type `T` . Whether the <scalar_exp> is indeed of type `T` will be checked at runtime.

Example 9.8. Member Construction

This will make a new object of type `UDT_FR_BASE` by calling it's two int parameters constructor and will return the member `B` value of the instance stored in member `UDT_M` of `UDT_FR_BASE` .

```
select new UDT_FR_BASE (1, 2).UDT_M.B;
```

9.2.10. Calling Static Methods

Let `T` be a user defined type that has a static method `SM` .

```
T::SM ( [ parameter, .... ] )
```

This will call the static method of `SM` of `T` and will return whatever the static method returns.

Example 9.9. Calling A Static Method

```
select UDT_TEST::_ADD (1, 2);
```

9.2.11. Calling Instance Methods

Let `T` be a user defined type that has an instance method `IM` . Let `IT` be a scalar expression having a compile time type of `T` .

```
IT.IM ( [ parameter, .... ] )
```

This will call the instance method `IM` of `T` and will return whatever the `IM` returns.

Similarly to member observers/mutators the compile time type of `IT` can be specified explicitly:

```
(IT as T).IM ( [ parameter, .... ] )
```

This syntax however has an additional property in that it will call the method of the type regardless of whether it is overloaded in a subtype or not. Let `ST` be a subtype of `T` and `ST` that has the method `IM` overloaded. Let `IST` be a scalar expression that represents an instance of `ST` .

Then:

```
(IST as T).IM ( [ parameter, ... ] )
```

will call the method `IM` as defined in `T` , whereas

```
IST.IM ( [ parameter, ... ] )
```

will call the method `IM` as defined in `ST` .

Example 9.10. Calling Overloaded Instance Methods

```
CREATE TYPE UDT_BASE
  method A () returns integer;

CREATE TYPE UDT_SUB under UDT_BASE
  OVERRIDING method A () returns integer;

create method A () returns integer for UDT_BASE
{
  return 1;
}

create method A () returns integer for UDT_SUB
{
  return 2;
}

select new UDT_SUB ().A() as IMPLICIT,
  (new UDT_SUB() as UDT_BASE).A() as EXPLICIT;
```

This will return:

```
IMPLICIT                EXPLICIT
-------------------------
2                       1
```

This is done so the overloaded methods can call the base type methods.

9.2.12. Serializing & Deserializing Type Instances

Virtuoso allows serializing and deserializing of non TEMPORARY type instances. This means that the instances can be saved as a column value and can be used with the serialize/deserialize SQL functions.

Example 9.11. Storing User Defined Types

This creates a type SER_UDT, a table UDT_TABLE with a DATA column capable of storing SER_UDT instances, stores an instance of SER_UDT into the table and demonstrates some selects using the stored instance.

```
create type SER_UDT as (A integer default 12)
  method NEGATE () returns integer;

create method NEGATE () returns integer for SER_UDT
{
  return SELF.A * -1;
}

create table UDT_TABLE (ID integer primary key, DATA SER_UDT);

insert into UDT_TABLE (ID, DATA) values (1, new SER_UDT ());

select C.DATA.A from UDT_TABLE C where C.ID = 1;

select C.ID from UDT_TABLE C where C.DATA.A > 10;

select C.ID from UDT_TABLE C where C.DATA.NEGATE() < -10;
```

Note that the table alias is mandatory here.

```
select ID from UDT_TABLE where DATA.A > 10;
```

and

```
select ID from UDT_TABLE where DATA.NEGATE() < -10;
```

will both yield a syntax error.

The columns of a certain type allow storing subtype instances as well. The subtype instances will not be converted to the their supertype when stored.

If we define the type `SER_UDT_SUB` as:

```
create type SER_UDT_SUB under SER_UDT
  as (B integer default 13);
```

then we can do:

```
insert into UDT_TABLE (ID, DATA) values (2, new SER_UDT_SUB ());

select (C.DATA as SER_UDT_SUB).B from UDT_TABLE C where C.ID = 2;
```

Type instances can be stored into an ANY column:

```
create table ANY_TABLE (ID integer primary key, DATA any);

insert into ANY_TABLE (ID, DATA) values (1, new SER_UDT());
```

```
select (C.DATA as SER_UDT).A from ANY_TABLE C where C.ID = 1;
```

Example 9.12. SERIALIZE/DESERIALIZE VSEs example

```
select (DESERIALIZE (SERIALIZE (new SER_UDT ())) as SER_UDT).A;
```

The SERIALIZE VSE can be used to store larger type instances into LONG VARCHAR columns. For example:

```
create table LOB_TABLE (ID integer primary key, LOB_DATA LONG VARCHAR);

insert into LOB_TABLE (ID, LOB_DATA) values (1, SERIALIZE (new SER_UDT()));

select (DESERIALIZE (BLOB_TO_STRING (LOB_DATA)) as SER_UDT).A
  from LOB_TABLE where ID = 1;
```

The serialization/deserialization for the non-SQL type instances is done by the means of the hosted language (Java Object serialization API and CLR Binary serialization API). So to be serialized/deserialized correctly the Java classes must implement the java.io.Serializable interface and the CLR classes should have the [Serializable] attribute set. For details refer to the respective API documentation.

9.2.13. User Defined Types Utility Functions

Virtuoso implements the following user defined types utility functions:

`udt_instance_of()`
`udt_defines_field()`
`udt_implements_method()`
`udt_get()`
`udt_set()`

9.2.14. Hosted Foreign Objects in Virtuoso

Java VM Hosted Objects

A special build of Virtuoso hosts a Java VM and allows manipulation of Java classes through the SQL user defined types.

In order to access the Java class instances they have to be defined as Virtuoso types using CREATE TYPE and specifying LANGUAGE JAVA. Java classes have to be in the CLASSPATH of the hosted Java VM.

Example 9.13. Hosted Java Objects

Java (Point.java):

```
public class Point implements java.io.Serializable
{
  public double x = 0;
  public double y = 0;

  public Point (double new_x, double new_y)
  {
    x = new_x;
    y = new_y;
  }

  public double distance (Point p)
  {
    return Math.sqrt ((p.x - this.x) * (p.x - this.x) + (p.y - this.y) * (p.y - this.y));
  }
}
```

This Java class should be compiled and the corresponding Point.class should be placed in the hosted VM's classpath. Then a

Virtuoso user defined type should be created as follows:

```
create type Point language java external name 'Point'
 as (
  x double precision external name 'x',
        y double precision external name 'y'
)

constructor method Point (new_x double precision, new_y double precision),
method distance (Point p) returns double precision external name 'distance';
```

From now on the SQL Point type can be used to create instances of the Java Point class, access it's members, call it's methods and store it into tables (since the Java Point class implements the `java.io.Serializable` interface).

For the hosted Java objects a LANGUAGE JAVA should be specified. The format of EXTERNAL NAME is:

the full name of the Java class for classes (ex. 'java.lang.Class')

the name of the methods/instance members

Since Java has static members and the Virtuoso SQL types do not, Virtuoso allows read-only access to static members through static observer functions with EXTERNAL VARIABLE NAME instead of EXTERNAL NAME.

Example 9.14. Static Member Access

```
java (stat.java) :
public class stat
{
  static stat_m double;
}
```

Virtuoso SQL:

```
create type stat language java external name 'stat'
static method observe_stat_m ()
  returns double precision external variable name 'stat_m';
```

Virtuoso does automatic mapping between the Virtuoso SQL data types and the Java data types. Since Java data types are much more primitive than Virtuoso types it is safe to explicitly specify the Java type of an instance member, method parameter or method return value. This is done by using the Type Signatures format described in the Java Native Interface Specification (chapter 3 : JNI Types and Data Structures : Table 3.2). The signatures are supplied as string values to EXTERNAL TYPE clause.

To facilitate the creation of the wrapper SQL types Virtuoso uses the Java Reflection API to get the description of the class in XML form. This XML is then transformed using an XSL stylesheet to makes the CREATE TYPE statements required automatically. In the process it preserves the superclass/subclass relationships of the specified Java classes and represents them as a supertypes/subtypes in SQL. The `jvm_ref_import()` procedure is used to create the XML by calling the Java Reflection API.

The function `import_jar()` takes the same parameters as `jvm_ref_import()` but will automatically create and execute the create type statements within the Virtuoso server.

Table 9.2. Java Type to Virtuoso Type Conversions

Java Type/Class	Virtuoso Internal Type
boolean	smallint
byte	smallint
char	smallint
short	integer
int	integer

Java Type/Class	Virtuoso Internal Type
long	integer
float	real
double	double precision
byte[]	binary
java.lang.String	NVARCHAR
java.util.Date	DATETIME
[]	vector

Table 9.3. Virtuoso Type to Java Type Conversions

Virtuoso Internal Type	Java Type/Class
smallint	short
integer	integer
real	float
double precision	double
varchar	java.lang.String
nvarchar	java.lang.String
datetime	java.util.Date
timestamp	java.util.Date
binary	byte[]

For all the other types encountered in the signatures of the Java methods/members it makes a forward reference to a Virtuoso/PL user defined type based on the java class name, replacing the dot ('.') with the underscore ('_') character.

For example:

`'java.lang.System'` becomes `'java_lang_System'`

In order to correctly map a java superclass/subclass relationship between class A and class B when importing, it is necessary to include A, B and all the intermediate classes in the superclass/subclass chain in a single `import_jar()` call.

To implement serialization/deserialization for Java object the Virtuoso needs the __virt_helper Java class. This class contains utility functions implementing serialization/deserialization. This class must be in the CLASSPATH.

The Java VM hosted inside the Virtuoso binary is not started at startup, but when first needed. It's startup is marked by a message in the Virtuoso log file. An application can control the initialization of the Java VM by explicitly initializing the Java VM (preferably on server startup) by calling the VSE: `java_vm_attach()`

CLR Hosted Objects

A special virtuoso build is available to allow SQL types integration with the CLR (Common Language Runtime) on Windows. This is achieved by providing COM server in C# (virtclr.dll) that is called from the native code through COM.

The virtclr.dll library should be registered into the CLR's Global assembly cache.

The semantics of CLR hosted objects are largely the same as those described for Java hosted objects. As before, native objects need SQL Type wrappers, but with LANGUAGE CLR clause specified.

To automatically create the SQL Type wrappers based on the CLR Reflection API the Virtuoso CLR binary has a system stored procedure: `import_clr()`

There are three forms for specifying the EXTERNAL NAME of a CLR class:

- a) `<Assembly public name>/<namespace-prefixed-class-name>` Here the Virtuoso CLR host issues

Assembly.Load with `<Assembly public name>` to find the assembly. After finding it, it looks for `<namespace-prefixed-class-name>` in it.

- b) `<namespace-prefixed-class-name>` Here the Virtuoso CLR host issues Assembly.Load with `<namespace-prefixed-class-name>` . After finding it, it looks for `<namespace-prefixed-class-name>` in it.
- c) `<path-to-the-assembly-binary>/<namespace-prefixed-class-name>` Here the Virtuoso CLR host issues Assembly.LoadFrom with `<path-to-the-assembly-binary>` . After finding it, it looks for `<namespace-prefixed-class-name>` in it.

The Virtuoso CLR host does the above when creating an instance of the type, accessing static methods or properties.

However when it deserializes an serialized CLR instance it calls the CLR deserialization class BinaryFormatter. The BinaryFormatter.Deserialize calls internally Assembly.Load to find the serialized class description. So although the classes defined with EXTERNAL name as in c) above are otherwise accessible (and serializable) they will possibly not deserialize correctly (as the assembly binary may not be findable through the Assembly.Load). To avoid that CLR limitation it is advisable to use the EXTERNAL NAME forms a) and b) wherever possible.

The Assembly.Load process of finding Assemblies is very well documented on the MSDN . Note that if an assembly was loaded through Assembly.LoadFrom it is not considered as "already loaded" by the Assembly.Load. The Virtuoso CLR is a CLR runtime host. As such it can use the normal CLR configuration files. It also is able of using private assemblies.

See Also:

The Create Assembly Syntax

ASPX Hosting Using the Hosted CLR

Virtuoso CLR hosting allows ASPX pages to be executed through the Virtuoso HTTP server inside the hosted CLR Virtual machine. To enable this support an additional library (virt_http.dll) needs to be registered with the Global Assembly cache. Having achieve this and copying ASPX project files under the Virtuoso HTTP server's root allows direct execution of the ASPX page. See the sample ASPX pages in the Virtuoso distribution.

If the ASPX project files reside in a WebDAV directory they are copied into a temporary file system directory under a special temporary directory (configurable by the TempASPXDir INI parameter in [HTTPServer] section of virtuoso ini file) before executed. In order to be execute correctly from WebDAV the ASPX files should have Execute WebDAV permission set them. The execution of ASPX is also controlled by the EnableDavVSP INI parameter in the [HTTPServer] as with any active content within WebDAV.

See Also:

Deploying ASP.Net Web Applications

Runtime Hosting

Expressing Hosted Language Supertype/Subtype Dependencies With Virtuoso/PL User-Defined-Types

It is also possible to represent the tree or in-part of Java or the CLR's superclass/subclass hierarchy with Virtuoso user defined type mappings.

Consider the following sample Java code:

```
class g1 { public int g1_value; public int mtd_g1 (int x) { return g1_value; } };
class g2 extends g1 { public int g2_value; public int mtd_g2 (int x) { return g2_value; } };
class g3 extends g2 { public int g3_value; };
class g2_sib extends g1 { public int g2_sib_value; };

class uses_types {
  public static g3 mtd (int x) { return new g3 (); }
}
```

One can create SQL user defined types for `g1` , `g2` and `g3` to represent the `g1` /`g2` / `g3` Java class hierarchy if calling `mtd_g1` and `mtd_g2` is needed:

```
create type sql_g1 language java external name 'g1' as (
    g1_value int)
method mtd_g1 (x integer) returns integer;
```

```
create type sql_g2 under sql_g1 language java external name 'g2' as (
    g2_value int)
method mtd_g2 (x integer) returns integer;

create type sql_g3 under sql_g2 language java external name 'g3' as (
    g3_value int)
method mtd_g3 (x integer) returns integer;

create type uses_types language java external name 'uses_types'
 static method mtd (x integer) returns sql_g3;
```

provided with the above, one can call `uses_types.g3 ()` method and call `mtd_g1 ()` on the returned instance in Virtuoso/PL as follows:

```
select uses_types::mtd (12).mtd_g1 (10);
```

Sometimes it is not desirable or necessary to mirror the full supertype/subtype hierarchy from Java to Virtuoso/PL.

For the above example only `sql_g2` and `sql_g3` can be defined if the goal was to call `mtd_g2()` instead of `mtd_g1()` .

When creating instances of the Virtuoso/PL user defined types to represent the data returned by the hosted code, Virtuoso tries to find the closest common ancestor of the hosted instance's class and the ones defined in Virtuoso as user defined types.

For example if in the above example a Java function returns an instance of `g3` and there is a `sql_g3` defined inside virtuoso the `g3` Java instance will be wrapped into an `sql_g3` Virtuoso/PL instance. Note that that will not depend on the presence or absence of `sql_g1` and `sql_g2` definitions - i.e. Virtuoso will favor the exact match.

If, however `sql_g3` is not defined, but `sql_g2` and `sql_g1` are, then the `g3` instance will be wrapped up in an `sql_g2` instance when returned.

Similarly, if an instance of the `g2_sib` is to be returned in Virtuoso/PL and `sql_g1` to `sql_g3` are defined, Virtuoso will wrap the `g2_sib` Java instance into an `sql_g1` SQL instance.

9.2.15. Using User Defined Types to Represent SOAP Structures

The Virtuoso SOAP server is capable of using user defined types (both native and hosted) to represent structures in SOAP requests/responses. Normally a SOAP exposed procedure would have references to defined schema types (__SOAP_TYPE for it's return type and for it's argument types). When such a schema type represents a structure (see the SOAP RPC encoding) the Virtuoso SOAP server will map the structure to an array of name/value pairs for it's members (the type of value returned by the soap_box_structure VSE). The user defined types however are better suited for representing such data. In order for a user defined type to be usable in SOAP, it must have a default constructor (no arguments). For native types that is always true (since they have the implicit constructor setting up the member's values to the respective DEFAULT values from the user defined type declaration). The Virtuoso SOAP implementation supports two ways of specifying how a SOAP value XML fragment should be materialized as a user defined type instance, as follows.

Using Schema Fragments

The `SOAP_DT_DEFINE()` function is used to map a particular schema fragment describing a composite schema type for SOAP usage. This now takes an additional optional argument to establish a link to a user defined type name:

```
create type SO_S_30
  as (
      "varString" nvarchar,
      "varInt" integer,
      "varFloat" real,
      "processingResult" nvarchar,
      "vmVersion" nvarchar)
  constructor method SO_S_30 (),
  method process_data () returns nvarchar;

soap_dt_define ('',
'<complexType name="SOAPStruct"
   xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
```

```
    xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="services.wsdl"
    xmlns:tns="services.wsdl">
   <all>
     <element name="varString" type="string" nillable="true"/>
     <element name="varInt" type="int" nillable="true"/>
     <element name="varFloat" type="float" nillable="true"/>
     <element name="processingResult" type="string" nillable="true"/>
     <element name="vmVersion" type="string" nillable="true"/>
   </all>
</complexType>', 'SO_S_30');
```

The CREATE TYPE statement defines the user defined type SO_S_30 as having 5 data members, a no-parameters constructor and a processing method. The `soap_dt_define()` call maps the SO_S_30 to a schema type SOAPStruct which also has 5 data members and makes that schema type available to SOAP. Now we create a SOAP exposed stored procedure:

```
create procedure echoSOAPStructSch (
    in sst SO_S_30 __soap_type 'services.wsdl:SOAPStruct')
returns SO_S_30 __soap_type 'services.wsdl:SOAPStruct'
{
  declare processingResult nvarchar;
  processingResult := sst.process_data ();
  return sst;
};
```

When processing the SOAP request for calling that stored procedure, the SOAP server will call the default constructor for SO_S_30 (require to create the empty instance) and will fill up the values from the incoming XML fragment for the sst parameter to the members of the newly created SO_S_30 instance. Then it will pass that instance as a value for the sst parameter of the echoSOAPStructSch function. As a result echoSOAPStructSch will correctly execute the member function process_data. Then it will return the (possibly) modified SO_S_30 instance to the SOAP server. The SOAP server will make the XML fragment for the return value based on the SOAPStruct schema fragment using the values from the SO_S_30 members.

This approach allows easy migration for the existing SOAP services using structures. To upgrade a SOAP service procedure to use user defined types one should define the types and add the additional argument to `SOAP_DT_DEFINE()` .

For developing new SOAP services, however, it is redundant to create the schema fragment in addition to creating the user defined type to hold the SOAP structure.

For this reason, Virtuoso offers a second approach in using user defined types in SOAP.

Using the User Defined Type Definition

Consider the altered definition of SO_S_30 as follows:

```
create type SO_S_30
  as (
      varString nvarchar __soap_type 'string' __soap_name 'varString',
      "varInt" integer __soap_type 'int',
      "varFloat" real __soap_type 'float',
      "processingResult" nvarchar __soap_type 'string',
      "vmVersion" nvarchar)
  __soap_type 'services.wsdl:SOAPStruct'
  constructor method SO_S_30 (),
  method process_data () returns nvarchar;
```

and the procedure echoSOAPStructSch as:

```
create procedure echoSOAPStructSch (in sst SO_S_30) returns SO_S_30
{
  declare processingResult nvarchar;
  processingResult := sst.process_data ();
  return sst;
};
```

Now all we have to do is expose the echoSOAPStructSch in a SOAP service. The SOAP server will take into account the fact that the sst type and the return type are user defined types and automatically make the WSDL description (including the schema fragments) and will correctly process the incoming XML.

The SOAP names and data type names inside the user defined type definition are optional and default to the SQL member's name for names and employ a straight mapping of the PL types to the SOAP types for data types.

9.2.16. Consuming Third-Party SOAP Services via User Defined Types

Virtuoso provides function for acting as a SOAP client called `SOAP_CLIENT()` . Embedding SOAP Web Service methods in Virtuoso/PL procedures using this function, however, is not suitable in most cases, especially if the services contained a large number of parameters of different type, the procedure could become very complex, the encoding may also vary. Thus direct SOAP_CLIENT() invocation in some cases in non-trivial and may lead to errors that are hard to debug.

To aid development of Web based applications written in PL that use the SOAP protocol, Virtuoso introduces two new approaches for consuming a Web service:

generate SOAP proxy wrappers encapsulated in a PL Module

generate a User Defined Type (UDT) for SOAP proxy encapsulation.

Please note that both methods of making a SOAP proxy-wrapper requires a WSDL description. If some SOAP service does not have a corresponding WSDL, neither of these methods can be used.

The first approach can be achieved using the SOAP_WSDL_IMPORT() function. This will make a PL module utilizing the SOAP_CLIENT() function that will import the complex types and pass appropriate parameters. But it has one significant problem in that it will return the result as a parsed XML entity. The result should then be processed in the application code, which requires prior knowledge of the return parameters.

The second approach consists of creating a UDT encapsulation of the SOAP wrappers using the `WSDL_IMPORT_UDT()` .

See Also

The `SOAP_WSDL_IMPORT()` function.

The `WSDL_IMPORT_UDT()` function.

The import function `WSDL_IMPORT_UDT()` has two phases:

retrieve and expand the WSDL file (expansion will be done if import is specified)

compile the result and make SQL script with UDT definition

The following points will hold true for this method of SOAP encapsulation:

- Any XML Schema types, required for calling the target SOAP service will be imported in database.
- A UDT will be created for each service defined within the WSDL description.
- The UDT will have members: url, request, response, and debug.
    - the 'url' member designate the endpoint for SOAP invocation
    - debug is a flag to manage wrappers to return wire dumps if needed
    - request and response members will contain wire dumps if 'debug' is equal to 'true' (integer 1).
- for each method defined in the WSDL description there will be UDT's method.
- each method of UDT will contain:
    - a number IN/OUT/INOUT parameters (depending on the target method)
    - no return value, SOAP does not define explicitly return values.
    - call to the SOAP_CLIENT () function with appropriate arguments
    - XPATH over the result and transform with SOAP validation functions to ensure value of OUT and INOUT parameters.

See Also

The Virtuoso Administration Interface provides a web based interface for importing WSDL definitions and creating UDTs and procedures. This can be found in the Visual Server Administration Interface Chapter.

9.2.17. UDT Security

Security of UDTs is maintained through normal SQL GRANT and REVOKE statements via a simple extension. You can define the level of access to both native and externally hosted UDTs.

Grants for persistent user defined types are persisted into the SYS_GRANTS table. Grants on temporary user defined types are in-memory only and are lost (together with the temporary user defined type definition) when the server is restarted.

There are two GRANT/REVOKE types for UDTs as follows:

EXECUTE - all methods and members of a class are accessible to the grantee.

UNDER - the grantee can create subclasses of the class.

```
GRANT/REVOKE EXECUTE on <user_defined_type>
GRANT/REVOKE UNDER on <user_defined_type>
```

Note:

SQL modules, user defined types and SQL stored procedures are exposed to GRANT/REVOKE in the same namespace, therefore care must be taken avoid inadvertently granting to multiple objects at the same time.

9.3. XML Column Type

Virtuoso allows for native XML storage in a database table column using the `LONG XML` type. This data type is a variation of `LONG VARCHAR` that can have plain text or XML entities, persistent or non-persistent values, but will always return an XML entity when selected.

Since ODBC does not support an XML entity type this column will appear as a LONG VARCHAR when selected from ODBC based clients.

Example 9.15. Using LONG XML Columns

```
CREATE TABLE xml_col_test (
  id INTEGER,
  txt VARCHAR,
  xmltxt LONG XML
  )
  ;

  INSERT INTO xml_col_test (id, txt, xmltxt)
    VALUES (1, 'test', '<xml><test>test</test><test>test2</test></xml>');

  INSERT INTO xml_col_test (id, txt, xmltxt)
    VALUES (2, 'test', xml_tree_doc('<xml><test>test</test><test>test2</test></xml>'));

  select * from xml_col_test;
id          txt               xmltxt
INTEGER     VARCHAR           LONG VARCHAR
_______________________________________________________________________________

1           test              <xml><test>test</test><test>test2</test></xml>
2           test              <xml><test>test</test><test>test2</test></xml>
```

See Also:

The `xml_tree_doc()` also returns an XML entity and describes other functions that work with it.

9.4. Identifier Case & Quoting

Virtuoso can operate with different identifier case conventions. The CaseMode setting in the virtuoso.ini file controls this, see the virtuoso.ini configuration section of the documentation.

The default files supplied with Virtuoso specify a CaseMode of 2, which is a case insensitive mode that preserves the declaration case of identifiers.

A CaseMode of 1 specifies the upper case mode, which is most commonly used in SQL databases, e.g. Oracle. In the upper case mode, all unquoted identifiers are converted to upper case by the SQL parser. If identifiers are not quoted, the case in which they are entered is irrelevant.

The identifier quote character is the double quote ("). Quoted identifiers are processed in the case they are written in and are thus case sensitive.

SQL reserved words are case insensitive in all case modes.

If CaseMode is 0 or absent, identifiers will be treated as case sensitive in all situations, whether quoted or not.

If an identifier's name is equal to a SQL reserved word, e.g. TABLE, it must be quoted ("TABLE") in order to be used as an identifier.

If an identifier contains non-alphanumeric characters, e.g. space, '-' etc. it must be quoted regardless of CaseMode.

Although CaseMode can be changed at any time it should only be set at database creation. Changing the CaseMode may result in view or procedure code becoming invalid if it relies on specific case conventions.

9.5. Wide Character Identifiers

All Virtuoso schema columns are confined to 8-bit character fields. This will remain for backwards compatibility and performance reasons, however, there are two options available for support of non-ASCII identifier names as follows:

- Maintain an 8-bit system. Pass all 8-bit codes that enter the system and read them back according to the current database character set. This has the convenience of a 1-to-1 correspondence between the character lengths of an identifier and their representation, so it's a subject to like single character wildcards etc.

  This works well only for languages that do have single bit encodings (like western-european languages and cyrillic). But this does not work at all for the far-east languages. It also depends on the database character set and does not allow identifiers to be composed from multiple character sets.
- Store all identifiers as UTF-8 encoded unicode strings. This would allow seamless storage and retrieval of ANY character within the unicode character space. This, however, has the disadvantage of the varying character length representation which should be taken into account when comparing identifier names with LIKE.

Virtuoso supports the above cases which are switchable through the "SQL_UTF8_EXECS" = 1/0 flag in the [Client] section of the Virtuoso INI file. Setting SQL_UTF8_EXECS = 1 enables UTF-8 identifier storage and retrieval, whereas setting SQL_UTF8_EXECS = 0 disables it. The default setting is 0: disabled for backwards compatible option.

Note:

Once a non-ASCII identifier gets stored using a particular setting for the "SQL_UTF8_EXECS" flag and the flag is subsequently changed this will make the stored identifiers unreadable by normal means (but can be read by special means).

When an SQL statement comes into the driver(s) it is expanded into unicode (using either the current database character set if it is a narrow string like in SQLExecDirect, or taking it verbatim as in SQLExecDirectW). The unicode string is then encoded into UTF-8 passed to the SQL parser. The SQL parser knows that it will receive UTF-8 so it takes that into account when parsing the national character literals (N'<literal>') and the "normal" literals ('<literal>'). It will however return identifier names in UTF-8, these will then get stored into the DBMS system tables or compared against them depending on the type of statement.

All returned identifiers will be translated from UTF-8 to Unicode when returned to the client, so the client should never actually see the UTF-8 encoding of the identifiers.

Representing a string in UTF-8 will not change the identifier parsing rules or the SQL applications logic since the SQL special characters - like dot, quote, space etc - are ASCII symbols and they will get represented as a single byte sequence in UTF-8.

The upper/lower functions should be used with care when applied to identifiers: they will get narrow strings in UTF-8, so applying an upper/lower to them may cause damage to the UTF-8 encoding. That is why the identifiers should be converted explicitly to wide strings using the charset_recode function, changed to upper or lower case and then translated back to UTF-8 using the charset_recode function again.

Using single character LIKE patterns against identifiers stored as narrow strings in system tables will generally not work, as a single character may be represented with up to 6 bytes in UTF-8. An exception to that is when using single character pattern to match an ASCII character.

9.5.1. UTF-8 Implementation Notes For ODBC

All wide functions which do return an identifier, like SQLDescribeColW and friends, will return the correct wide literal. For their narrow counterparts, such as SQLDescribeCol, the UTF-8 string will first be converted to a wide string and then to a narrow string using the current database character set. However, an extension to the ODBC standard has been implemented instructing all result set returning meta-data functions, such as SQLTables and SQLTablesW, to return SQL_NVARCHAR instead of SQL_VARCHAR columns. This is not a problem for most applications since all they do is to map the result to SQL_C_CHAR on retrieval which will convert the wide string to the appropriate narrow string inside the driver using the current database character set. This will cause problems with narrow applications like MS Query, trying to get identifiers not representable in the current narrow character set, because all they will get is the "untranslatable char" mark (currently a question mark).

9.5.2. UTF-8 Implementation Notes In JDBC

Since JAVA is all unicode there are no unavoidable deviations from the JDBC standard. However when printing the Java strings to the screen or a file or getting their byte representation, the usual JAVA conversion rules apply. The types of the meta data result set columns in JDBC are somewhat debatable, but since they are usually retrieved with Resultset.getString() the Virtuoso JDBC driver will return the raw wide string instead of trying to make it VARCHAR before returning it to the application.

9.6. Qualified Names

9.6.1. Qualifiers and Owners

Virtuoso supports multiple namespaces for tables and procedures. A table or procedure is uniquely identified by a three part name consisting of qualifier, owner and name separated by dots.

Each connection has a current qualifier which is used as the default qualifier for if a name does not specify a qualifier. The owner can be omitted from a name if the qualifier and name uniquely identify the object, e.g. DB..SYS_KEYS.

Initially, all system objects have the qualifier DB and owner DBA.

The default current qualifier of all connections is DB unless otherwise specified using db.dba.user_set_qualifier.

A user can be assigned a default qualifier set as current qualifier upon login. This is done with the stored procedure db.dba.user_set_qualifier.

```
user_set_qualifier  (in user varchar, in qualifier varchar)
```

9.6.2. Default Qualifiers

The default qualifier of the user is set to be the qualifier. The names are strings and thus case sensitive.

Example:

```
db..user_set_qualifier ('U1', 'U1DATA');
```

9.6.3. USE Statement, USE identifier

This sets the default qualifier for the connection in question. The identifier cannot be an expression. The identifier is subject to whatever case conversions may be in effect.

See Also:

CaseMode Configuration parameter .

A quoted identifier will always pass in the case it is entered. An unquoted identifier will be converted to upper case if CaseMode so specifies.

9.7. Literals, Brace Escapes

9.7.1. Strings

String literals are delimited with single quotes. A double single quote is an escape notation for a single quote character inside a string literal. Additionally, standard C language escapes are supported. Support of C escapes can be turned off for compatibility with other SQL databases by using the SQL_NO_CHAR_C_ESCAPE option in the configuration file or as an ODBC connection option.

```
Literal                 meaning
''                      <empty>
''''                    '
''''''                  ''
'\t\r\n\\'              tab, carriage return, newline, backslash
'\012'                  Character 012 octal, i.e. newline
```

9.7.2. Numbers

An integer constant consist of an optional minus sign followed by decimal digits. Integer literals are of the C type long, 32 bit.

Numeric literals with a decimal point literal are internally of the DECIMAL SQL type, a variable length decimal floating point type. The Following are examples of decimal literals:

```
123.456
-16.0
```

Numeric literals which specify an exponent, e.g. 1.2e11 or 2e-3 are read as C doubles, (64 bit IEEE binary floating point). This is potentially less precise than the DECIMAL SQL type.

Integer literals outside of the 32-bit range are interpreted as DECIMAL.

9.7.3. ODBC Brace Escapes

The Virtuoso SQL parser supports the following ODBC brace escape notations:

```
{fn function (argument, ..) }           fm (arguments .)
call procedure a1, ... }

{d 'yyyy.mm.dd'}

{t 'hh:mm.ss' }

{ts 'yyyy.mm.dd hh:mm.s fraction' }

{oj  }          -- outer join
```

9.7.4. Hexadecimal Literals

Hexadecimal values can be specified literally in two ways, prefixing the plain value with '0x' or enclosed with single quotes prefixed with 'X'. The case is not important. Hex characters should always be pairs, representing a single byte, and should be at least on pair. Here are some examples:

```
X'beef'   - valid
0xbeef    - valid

X'abeef'  - not valid
X'0abeef' - valid
X''   - not valid
```

`X'<value>'` is equivalent to `0x<value>`

9.7.5. Binary Literals

Binary strings can be specified as literals prefixed with 'B' and enclosed with single quotes. The string should not be empty and should contain only 1's or 0's. Binary strings are read from the end to beginning forming bytes on each 8-th bit:

```
B'1'    = 0x01
B'1111'   = 0x0F
B'111111111'  = 0x01FF
B'100000001'  = 0x0101
B'', X'' and 0x return binary literals.
```

9.8. CREATE TABLE Statement

9.8.1. Syntax

```
    base_table_def: CREATE TABLE new_table_name '(' base_table_element_commalist ')'
                  | CREATE TABLE new_table_name as query_exp opt_with_data

base_table_element_commalist: base_table_element
                            | base_table_element_commalist ',' base_table_element

base_table_element: column_def
                  | table_constraint_def

column_def: column column_data_type column_def_opt_list

opt_referential_triggered_action: /* empty */
                                | referential_rule
                                | referential_rule referential_rule

referential_rule: ON UPDATE referential_action
                | delete_referential_rule

delete_referential_rule: ON DELETE_L referential_action

opt_on_delete_referential_rule: /* empty */
                              | delete_referential_rule

referential_action: CASCADE
                  | SET NULLX
                  | SET DEFAULT

references: REFERENCES q_table_name opt_column_commalist opt_referential_triggered_action

column_def_opt_list: /* empty */
                   | column_def_opt_list column_def_opt

identity_opt: START_L WITH signed_literal
            | INCREMENT_L BY INTNUM

identity_opt_list: identity_opt
                 | identity_opt_list ',' identity_opt

column_def_opt: NOT NULLX
              | NULLX
              | IDENTITY
              | IDENTITY '(' identity_opt_list ')'
              | PRIMARY KEY
              | DEFAULT signed_literal
              | COLLATE q_table_name
              | references
              | IDENTIFIED BY column
              | CHECK '(' search_condition ')'
              | WITH SCHEMA column_xml_schema_def
              | UNIQUE

table_constraint_def: UNDER q_table_name
                    | opt_constraint_name PRIMARY KEY '(' index_column_commalist ')' opt_index_option_lis
                    | opt_constraint_name FOREIGN KEY '(' column_commalist ')' references
                    | opt_constraint_name CHECK '(' search_condition ')'
                    | opt_constraint_name UNIQUE '(' column_commalist ')'

opt_constraint_name: /* empty */
                   | CONSTRAINT identifier

column_commalist: column
                | column_commalist ',' column
```

```
index_column_commalist: column opt_asc_desc
                      | index_column_commalist ',' column opt_asc_desc

index_option: CLUSTERED
            | UNIQUE
            | OBJECT_ID

index_option_list: index_option
                 | index_option_list index_option

opt_index_option_list: /* empty */
                     | index_option_list

column_xml_schema_def
        : '(' STRING ',' STRING ')'
        | '(' STRING ',' STRING ',' STRING ')'

opt_with_data
: /* empty */
| WITH DATA
| WITHOUT DATA
```

The CREATE TABLE statement creates a new table. The table is identified by a new_table_name, which consists of an optional qualifier, an optional owner and the name. If the qualifier is not supplied then it defaults to the current qualifier, and likewise if the table owner is not specified then this will default to the login name of the user issuing the statement.

Following the new_table_name is a list of table elements that are either column definitions or table constraints. A column must have a unique name within the table and possible super tables. The only necessary attribute of a column is a data type.

The UNDER q_table_name constraint specifies an optional supertable. The table being created will inherit all columns, indices, constraints from the supertable, specifically including the primary key. Hence the under and primary key constraints cannot be specified together. If the table being created will be owned by a non-dba user, the supertable must be owned by the same user.

The AS query_exp opt_with_data causes the table to be created based on the compile time types of columns of the SELECT expression query_exp. If WITH DATA is specified then the resultset returned by the query_exp is fed into the new table. Otherwise (if WITHOUT DATA or not specified) only the table is created.

9.8.2. NOT NULL

Optionally a column can be declared NOT NULL. Any attempts to insert NULL into column declared NOT NULL will result in an error and the insert rejected.

9.8.3. IDENTITY (Auto Increment)

The IDENTITY keyword causes the columns to be automatically incremental, meaning that each successive insert into the table will automatically assign a value greater than any previous row of the table. These columns are often referred to as "autoincrement columns". The assigned values are normally consecutive integers. There may be gaps in the sequence if some transactions that reserve a number fail to commit.

An autoincrement column may or may not be part of the primary key or of any index.

The `identity_value()` function returns the identity column value most recently assigned on the current connection. This can be used inside procedures or from clients. See also the ODBC statement option SQL_GETLASTSERIAL.

The `set_identity_column()` function allows you to change the identity column sequence value.

The IDENTITY keyword supports the SQL 3 optional clause that allows you to specify a starting value and/or step value in the table creation statement. The syntax is:

```
IDENTITY (START WITH <N>, INCREMENT BY <Y>)
```

Example 9.16. Using the IDENTITY declaration

```
CREATE TABLE DB.DBA.AI (
        AI INTEGER IDENTITY,
        XX VARCHAR,
        PRIMARY KEY (AI)
  )
;
```

Creates a table with an autoincrement primary key column AI.

```
set_identity_column("DB"."DBA"."AI", "AI", 11);
```

will cause the next row to be inserted into AI to have the AI column equal to 11.

Example 9.17. Using the IDENTITY (START WITH) declaration

```
CREATE TABLE DB.DBA.AI (
        AI INTEGER IDENTITY (START WITH 100),
        XX VARCHAR,
        PRIMARY KEY (AI)
  )
;
```

Creates a table with an autoincrement primary key column AI, whose first value will be 100 and will auto-increment from there upwards.

See Also

Sequence Objects

9.8.4. DEFAULT

This option specifies a constant default value given to the column by an INSERT that does not specify a value for the column. The constant must be compatible with the column's data type. An IDENTITY or TIMESTAMP column cannot have a default value.

9.8.5. PRIMARY KEY Constraint

This declares a column combination that will uniquely identify each row in the table. This automatically defines an index on these columns and clusters the physical rows of the table together with the index entry for this primary key index. Always specify a primary key if there is a uniquely identifying column combination on the table. This is the case for any normalized database. Hence virtually all tables should have the primary key constraint. This is substantially more efficient than defining the primary key as a unique index with CREATE INDEX. The primary key constraint exists for the purpose of guaranteeing uniqueness of a row and hence should be respected. A unique index is not a primary key and should never substitute one.

Example 9.18. Primary Key using Constraint

This example shows how to create a table with a primary key defined in full as a named primary key table constraint

```
CREATE TABLE demo_table (
  id INTEGER NOT NULL,
  txtdata VARCHAR(20),
  CONSTRAINT demo_table_pk PRIMARY KEY (id)
  )
;
```

PRIMARY KEY is a shorthand for the PRIMARY KEY (column) table constraint which is specified in the column definition. SQL-89 required that you specify NOT NULL PRIMARY KEY but SQL-92 does not because primary keys columns do not permit NULL values. This means that no members of a combination of columns that constitute a primary key can have a NULL value.

Example 9.19. Primary Key shorthand

This example shows how to create a table with a primary key defined using shorthand:

```
CREATE TABLE demo_table (
  id INTEGER NOT NULL PRIMARY KEY,
  txtdata VARCHAR(20),
  )
;
```

Or shorter still:

```
CREATE TABLE demo_table (
  id INTEGER PRIMARY KEY,
  txtdata VARCHAR(20),
  )
;
```

If a table definition has no PRIMARY KEY clause, Virtuoso will generate a default primary key column called "_IDN".

Important

Always declare a primary key using the primary key table constraint. This is always more efficient than the default primary key.

You could allow Virtuoso to generate the primary key if you simply want an IDENTITY primary key column, however this is considered lazy and, of course, not portable. If you already have a column or combination of columns that could be a candidate for a primary key, taking the default primary key from Virtuoso will reduce the overall efficiency because an extra column will be used per row that would be redundant to the data itself.

Example 9.20. Default Virtuoso Primary Key

In the absence of a PRIMARY KEY definition:

```
CREATE TABLE SAMPLE (
  THING VARCHAR
  )
;
```

will be expanded into:

```
CREATE TABLE SAMPLE (
  THING VARCHAR,
  _IDN INTEGER IDENTITY,
        PRIMARY KEY (_IDN)
  )
;
```

Tables with generated default primary keys will appear as if they have no primary key defined. The default primary key (_IDN) column will not appear in the ODBC catalog calls `SQLColumns()`, `SQLStatistics()`, `SQLPrimaryKeys()`, `SQLColumnPrivileges()`. The column can be explicitly referenced in SQL statements just as any other. The SQL "SELECT * ..." statement will omit the _IDN column. The "INSERT INTO TABLE VALUES (.)" statement does not expect a value for the _IDN column. `SQLSpecialColumns()` with the *`SQL_BEST_ROWID`* option is the only catalog call that will return the _IDN column.

The PRIMARY KEY option may not coexist with UNDER in a CREATE TABLE statement because a subtable always inherits the supertable's primary key.

See Also:

CREATE INDEX for the index options.

9.8.6. UNDER

This allows the user to create a table as a sub-table of an existing table. A sub-table will inherit all columns and constraints of its super-table, most importantly the primary key. Primary keys shall be unique in the set of rows belonging to any direct or indirect sub-table of one super-table. A select from a super-table will see all rows, including those of any sub-tables, but not vice versa. Rows inserted directly into the super-table will not be visible to sub-tables.

The primary key option may not coexist with UNDER, since a subtable always inherits the supertable's primary key.

Example 9.21. Creating sub-tables using UNDER

Here a subtable will be created for the demo_table defined earlier. Both definitions are listed for convenience. Notice that the select over the sub-table (demo_sub_table) lists all columns whereas the select from super-table does not, however rows inserted into the super-table, demo_table, will not be seen in a select from the sub-table, but rows inserted into the sub-table will be shown in a select from the super-table.

```
CREATE TABLE demo_table (
  id INTEGER PRIMARY KEY,
  txtdata VARCHAR(20),
  )
;

CREATE TABLE demo_sub_table (
  new_col VARCHAR(10),
  UNDER demo_table
  )
;

SQL> select * from demo_table;
id                txtdata
INTEGER NOT NULL  VARCHAR
_______________________________________________________________________

0 Rows. -- 20 msec.

SQL> select * from demo_sub_table;
id                txtdata               new_col
INTEGER NOT NULL  VARCHAR               VARCHAR
_______________________________________________________________________

0 Rows. -- 10 msec.
```

9.8.7. FOREIGN KEY Constraint

A FOREIGN KEY is a column of combination of columns used to retrieve a related row from a related table. These reference constraints are automatically enforced, unless explicitly disabled. This declaration can be accessed by client applications using the `SQLForeignKeys()` ODBC function. This is used by some tools to display dependencies between tables. Foreign keys are there to maintain referential integrity within the database. This constraint will ensure that rows in one table will have corresponding rows in another table, so that your orders are not made for products that do not exist, for example.

Forward references are not permitted in FOREIGN KEY. Also a table referenced in a foreign key constraint of another table cannot be dropped. It is usual to list the columns in the referenced table corresponding to the columns in the referencing table, however, this list can be omitted, in which case the set of primary key columns of the referenced table is used automatically by default. The referenced and referencing column lists must be of equal length. To declare two mutually referencing tables, ALTER TABLE must be used after both tables are defined.

Foreign key constraints are enforced by default. It is sometimes practical to disable constraint checks for performance or for convenience when importing data or making bulk updates. To disable or enable checking for foreign key constraints you can use the `fk_check_input_values()` function. This function changes the foreign key checking behavior globally across the database so it is very important to remember to enable foreign key constraint checking again afterwards.

Columns can be defined as having foreign keys and a default value, however, the default value must not violate the constraint at the time of insert or update as this will be rejected in the normal way.

Example 9.22. Creating tables with foreign key constraints

First the Primary key table must be defined before it can be referenced:

```
CREATE TABLE T1 (
  Id INTEGER NOT NULL PRIMARY KEY,
  Dt VARCHAR
  )
;
```

Now the foreign key table can be created referencing the table above:

```
CREATE TABLE T2 (
  Act INTEGER NOT NULL,  -- will refer to Id in T1
  Retr DATETIME,
  Description VARCHAR,
  CONSTRAINT FK1 FOREIGN KEY (Act) REFERENCES T1 (Id) -- the fk constraint
  )
;
```

The statement above creates the foreign key constraint in separate line of the create table statement. This can be also be written in short form with the column definition it applies to:

```
CREATE TABLE T2 (
  Act INTEGER NOT NULL REFERENCES T1 (Id),  -- refer to Id in T1
  Retr DATETIME,
  Description VARCHAR
  )
;
```

Example 9.23. Assumed Foreign Key Columns

This CREATE TABLE statement was used for creating part of Demo database. This statement does not describe the columns to be used as foreign keys on the referred tables. Since the referred tables in each case have a single Primary Key (Orders.OrderID and Products.ProductID) they need not be mentioned and will be assumed when this statement is processed.

```
CREATE TABLE Order_Details(
  OrderID INTEGER,
  ProductID INTEGER,
  UnitPrice DOUBLE PRECISION,
  Quantity SMALLINT,
  Discount REAL,
  PRIMARY KEY (OrderID, ProductID),
  FOREIGN KEY (OrderID) REFERENCES Orders,
  FOREIGN KEY (ProductID) REFERENCES Products
  )
;
```

Foreign Key Constraint Actions

Foreign key constraint actions allow the user to define what action data should take when the data they are related to by way of a foreign key is updated or deleted. The two activities that can be programmed are:

ON DELETE

ON UPDATE

The actions available for both types of activity are:

CASCADE - automatically perform the same action on the referenced data.

SET NULL - set the referenced data to NULL.

SET DEFAULT - sets the referenced data to whatever was defined as its default value.

NO ACTION - to not perform any action at all.

Example 9.24. Foreign Key Constraint Actions

Now, the foreign key table defined again, but this time with referential rules which control how data is managed when rows are updated or deleted in the primary key table:

```
CREATE TABLE T3 (
  Act INTEGER NOT NULL DEFAULT 99,  -- will refer to Id in T1
  Retr DATETIME,
  Description VARCHAR,
  CONSTRAINT Fk1 FOREIGN KEY (Act) REFERENCES T1 (Id)
    ON UPDATE CASCADE ON DELETE SET DEFAULT
  )
;
```

9.8.8. The CHECK Constraint

The CHECK constraint allows you specify and wide range of rules that will dictate whether an insert of update will be permitted. The syntax is as follows:

```
CHECK (search-condition)
```

The search condition can be simple and comparative, or quite complicated involving regular expressions.

Example 9.25. Creating a table with the CHECK constraint

Here a simple table will be created with two CHECK constraints. One the check constraints is a simple comparison ensuring participants are over 18, the other complicated constraint verifies that the email address is correct using a regular expression. Samples SQL statements follow that will demonstrate the effectiveness of the check constraints.

```
CREATE TABLE test_check (
  name VARCHAR,
  age INTEGER
    CHECK (age > 18),
  email VARCHAR
    CHECK (regexp_like(email, '^([a-zA-Z0-9_.-])+@(([a-zA-Z0-9-])+.)+([a-zA-Z0-9]{2,4})+\$'))
  )
  ;

INSERT INTO test_check (name, age, email) VALUES ('Jack', 18, 'jack@foo.bar');
  -- will cause: *** Error 22023: ... SR363: CHECK constraint violated

INSERT INTO test_check (name, age, email) VALUES ('Jill', 19, 'up@thehill.com');
  -- will be insert correctly.

INSERT INTO test_check (name, age, email) VALUES ('Jack and Jill', 37, 'ouch/!^^!!@@');
  -- will cause: *** Error 22023: ... SR363: CHECK constraint violated, also.
```

See Also:

`regexp_like()`

9.8.9. The WITH SCHEMA Constraint

The WITH SCHEMA constraint allows you force values of an XML column to match a particular schema. The syntax is as follows:

```
WITH SCHEMA (namespace-uri, top-element-name [, optional-dtd-configuration])
```

To use this feature, you should make load an XMLSchema to the server by a CREATE XML SCHEMA statement. As soon as schema is loaded, its target namespace URI can be used in WITH SCHEMA constraint to validate every new value of the column against this schema. In addition, the constraint ensures that the document is a well-formed XML document (not a well-formed generic XML entity) and checks if the name of the top level element of the document is equal to one specified in the constraint.

Depending on document size and the complexity of the schema, schema validation may be a time- and memory- consuming operation. An application can win a lot if every stored value is validated only once and avoid repeating validations on every read of the stored data, but it also may loose if the validation is actually redundant (e.g. if the data always comes from sources which produce valid content).

The parameter "optional-dtd-configuration" lets an application to specify how strict the validation should be. In real life, documents may match the schema in general, but not in minor details that are not important for a particular application. If specified, the parameter must be a string in the format described in Configuration Options of the DTD Validator . The default value is suitable for most of applications and forces the validator to check well-formedness of the document, nesting of elements, presence of all "required" attributes, syntax of values of typed elements and attributes; it also will check referential integrity of ID and IDREF attributes.

Example 9.26. Creating a table with the WITH SCHEMA constraint

Here a simple table will be created with a WITH SCHEMA constraint. Sample SQL statements follow that will demonstrate the effectiveness of the check constraints.

```
CREATE XML SCHEMA '<?xml version="1.0" encoding="utf-8" ?>
<xs:schema targetNamespace="http://www.example.com/parts"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:p="http://www.example.com/parts">
   <xs:element name="Part">
      <xs:complexType>
         <xs:choice minOccurs="0" maxOccurs="unbounded">
           <xs:element ref="p:Part"/>
         </xs:choice>
         <xs:attribute name="ID" type="xs:string" use="required" />
         <xs:attribute name="Count" type="xs:int" use="optional" />
         <xs:attribute name="Type" type="xs:string" use="optional" />
      </xs:complexType>
   </xs:element>
</xs:schema>';

create table SPARE_PARTS (
  PACK_ID integer primary key,
  CONTENT XMLType with schema ('http://www.example.com/parts', 'Part')
  );

insert into SPARE_PARTS values (1, '
<p:Part xmlns:p="http://www.example.com/parts"
xml:id="keyboard">
  <p:Part Count="101"/>
  <p:Part
xml:id="body"/>
</p:Part>')

*** Error 42000: [Virtuoso Driver][Virtuoso Server]XML parser detected an error:
        ERROR  : Only 0 out of 1 required attributes are defined for element <p:Part>, e.g. the element h
at line 3 column 25 of '(value to be placed in column DB.DBA.SPARE_PARTS of CONTENT)'
  <p:Part Count="101"/>
------------------------^

insert into SPARE_PARTS values (2, '
<p:Part xmlns:p="http://www.example.com/parts"
xml:id="keyboard_01">
  <p:Part
xml:id="key" Count="101"/>
  <p:Part
xml:id="body_01"/>
</p:Part>')

select * from SPARE_PARTS
PACK_ID           CONTENT
INTEGER NOT NULL  LONG VARCHAR
_______________________________________________________________________________

2
<n0:Part xmlns:n0="http://www.example.com/parts"
xml:id="keyboard_02">
  <n0:Part
```

```
xml:id="key_01" Count="101" />
  <n0:Part
xml:id="body_02" />
</n0:Part>

1 Rows. -- 00000 msec.
```

ALTER TABLE ... MODIFY COLUMN statement does not support changes of WITH SCHEMA constraint. Double check your XMLSchema and carefully test it on real data used by an application before using this constraint. If you can't test your schema this way then calling of `xml_validate_schema()` in triggers may be safer than using the constraint: such triggers will be slower than the constraint but you can drop triggers without re-creating the table.

See Also:
CREATE XML SCHEMA Statement

`xml_validate_schema()`

9.9. DROP TABLE Statement

```
drop_table
        : DROP TABLE q_table_name
        ;
```

This statement drops a table. This requires dba privileges or ownership of the table. Any subtables are automatically dropped. Supertables are not affected.

9.10. CREATE INDEX Statement

```
index_column_commalist
        : column opt_asc_desc
        | index_column_commalist ',' column opt_asc_desc
        ;

index_option
        : CLUSTERED
        | UNIQUE
        ;

index_option_list
        : index_option
        | index_option_list index_option
        ;

opt_index_option_list
        : /* empty */
        | index_option_list
        ;

create_index_def
        : CREATE opt_index_option_list INDEX index
                ON new_table_name '(' index_column_commalist ')'
        ;
```

This creates an index on a table. Index names must be unique across a qualifier. The ascending / descending column attributes are recognized for compatibility but do not have any effect. The index is defined and populated at the execution of the statement. Pre-existing stored procedures and prepared statements will make use of the new index when appropriate.

The UNIQUE attribute enforces uniqueness of the specified columns across the table and subtables where the index is visible.

The CLUSTERED attribute is not recommended. It will cause keys to be unprefixed by key id, thus causing the key entries to be intermixed with entries of other CLUSTERED indices with adjacent values of key parts.

See:
See the Free Text section on creating free text indices.

9.11. DROP INDEX Statement

```
drop_index
        : DROP INDEX NAME opt_table
        ;

opt_table
        : /* empty */
        | q_table_name
        ;
```

This will drop an index, dba privileges or table ownership are required. A table's primary key which has the same name as the table can not be dropped.

Optionally, a table name can be given if the index name is not unique. The table name may be qualified.

9.12. ALTER TABLE Statement

```
add_column:
            ALTER TABLE q_table_name ADD opt_col_add_column add_col_column_def_list
          | ALTER TABLE q_table_name DROP opt_col_add_column add_col_column_list
          | ALTER TABLE q_table_name MODIFY opt_col_add_column column_def

add_col_column_def_list: column_def
                       | add_col_column_def_list &apos;,&apos; column_def

add_col_column_list: column
                   | add_col_column_list &apos;,&apos; column

table_rename:
            ALTER TABLE q_table_name RENAME new_table_name

alter_constraint:
            ALTER TABLE q_table_name constraint_op opt_table_constraint_def

constraint_op: ADD
             | DROP
             | MODIFY

opt_drop_behavior: /* empty */
                 | CASCADE
                 | RESTRICT

opt_table_constraint_def: CONSTRAINT identifier opt_drop_behavior
                        | table_constraint_def

view_query_spec: query_exp
               | query_no_from_spec

alter_table
        : ALTER TABLE q_table_name ADD opt_col_add_column add_col_column_def_list

        | ALTER TABLE q_table_name DROP opt_col_add_column add_col_column_list

        | ALTER TABLE q_table_name MODIFY opt_col_add_column column_def

        | ALTER TABLE q_table_name RENAME new_table_name

        | ALTER TABLE <table> ADD | DROP | MODIFY <constraint> [<constraint-name>]

<constraint> ::=
        : PRIMARY KEY '(' <column list> ')'
        | FOREIGN KEY '(' <column list> ')' <references>
        | CHECK ( search-condition )

<references> ::=
   REFERENCES <table> ['(' <column list> ')' ] <OPTIONS>

<OPTIONS>  ::=
   [ON UPDATE OPT_ACTION] [ON DELETE OPT_ACTION]
```

The ALTER TABLE statement adds or drops columns and renames tables.

Adding or dropping a column of a table will create a new version of the table's schema entry. The rows of the altered table will be changed to conform to the new definition when next updated. All newly inserted rows will be in the new row layout. This means that ALTER TABLE itself executes in fixed time without locking the table. The time to update the actual data will be spread over subsequent updates.

An added column will have a NULL value on those rows where the new column has not been set. A possible default will only apply to newly inserted rows.

When dropping a column one can execute a statement of the form UPDATE <table> SET <key> = <key> to force the physical change, causing space associated with the dropped column to be freed.

The column_def in the ADD clause may specify any column options, except PRIMARY KEY.

If the table name is not fully qualified it is completed as in any table reference. The new name in the RENAME clause is defaulted to the current qualifier and user account, as in a CREATE TABLE.

A primary key may only be modified, never dropped or added. Tables always have exactly one primary key.

The first unique index of an empty table becomes the primary key. This may thereafter not be dropped.

The primary key of an attached table may not be changed.

Foreign keys can be added. Dropping a foreign key will drop all foreign keys between the foreign key and primary key tables.

ALTER TABLE cannot be applied to an attached table.

Example 9.27. ALTER TABLE Examples

```
ALTER TABLE DISTRICT
    add D_SALES_MGR integer not null references EMPLOYEES (E_ID);

ALTER TABLE idt MODIFY PRIMARY KEY (K1, K2);

ALTER TABLE idt ADD FOREIGN KEY (d) REFERENCES idt (d);

ALTER TABLE idt ADD FOREIGN KEY (k2, k1) REFERENCES idt;

ALTER TABLE idt DROP FOREIGN KEY (d) REFERENCES idt (d);
```

9.12.1. Adding a CHECK Constraint

A CHECK constraint can be added to a table after it has been created and populated providing that none of the tables contents would violate the constraint.

Example 9.28. Adding a CHECK constraint to an existing table.

```
CREATE TABLE test_add_check (
  name VARCHAR,
  age INTEGER
  )
;

ALTER TABLE test_add_check ADD CONSTRAINT chkage CHECK (age > 18);
```

See Also:

The CHECK constraint

9.13. CREATE VIEW Statement

```
<view definition> ::=
    CREATE VIEW <table name> [ <left paren> <view column list>
                                <right paren> ]
      AS <query expression>
```

9.14. CREATE XML SCHEMA Statement

```
<xml schema definition> ::=
    CREATE XML SCHEMA <text of schema>
```

Virtuoso supports registering XML Schemas for use in WITH SCHEMA constraint for column values. The statement contains the whole text of the schema as a string constant, i.e. enclosed in single quotes. This is not the best possible syntax, because single quotes inside the text of schema should be carefully quoted, but this is de-facto standard. If the schema contains number of single quotes (e.g. attributes are in single quotes instead of typically used double quotes), try a system stored procedure

```
DB.DBA.SYS_CREATE_XML_SCHEMA (text_of_schema);
```

that will have the same effect, but is not portable.

In principle, you can register any valid XMLSchema, but some features can cause prohibitive loss of performance. It is strongly advised to compose the schema as a "standalone" document that has no references to external DTDs or external generic entities. It is also strongly advised to avoid xs:include and xs:import directives. The only sort of external references that does not affect performance is xs:include or xs:import of a registered "sibling" schema. They say that a schema X is a "sibling" of schema Y if their target namespace URIs have identical protocol names and host names and differs only in local path, and schema X imports Y using relative (not absolute!) URI that contain only relative path, (no protocol and no host).

Example 9.29. Registering Sibling Schemas

Two sibling schemas are registered here. First statement registers an XMLSchema for "http://www.example.com/parts" target namespace. Second statement registers an XMLSchema for "http://www.example.com/jobs" target namespace that imports the first schema using relative URI. The rest of statements demonstrate a single WITH SCHEMA constraint that inspect elements of these two target namespaces.

```
CREATE XML SCHEMA '<?xml version="1.0" encoding="utf-8" ?>
<xs:schema targetNamespace="http://www.example.com/parts"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:p="http://www.example.com/parts">
   <xs:element name="Part">
      <xs:complexType>
         <xs:choice minOccurs="0" maxOccurs="unbounded">
           <xs:element ref="p:Part"/>
         </xs:choice>
         <xs:attribute name="ID" type="xs:string" use="required" />
         <xs:attribute name="Count" type="xs:int" use="optional" />
         <xs:attribute name="Type" type="xs:string" use="optional" />
      </xs:complexType>
   </xs:element>
</xs:schema>'

CREATE XML SCHEMA '<?xml version="1.0" encoding="utf-8" ?>
<xs:schema targetNamespace="http://www.example.com/jobs"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:p="http://www.example.com/parts"
  xmlns:j="http://www.example.com/jobs">
   <xs:import schemaLocation="parts" />
   <xs:element name="Job">
      <xs:complexType>
         <xs:sequence>
           <xs:element ref="p:Part"/>
         </xs:sequence>
         <xs:attribute name="JobID" type="xs:string" use="required" />
      </xs:complexType>
   </xs:element>
</xs:schema>'
```

```
create table JOBS (
  PACK_ID integer primary key,
  CONTENT XMLType with schema ('http://www.example.com/jobs', 'Job')
  )

insert into JOBS values (1, '
<j:Job xmlns:j="http://www.example.com/jobs" xmlns:p="http://www.example.com/parts" JobID="asmkeyboard">
  <p:Part xmlns:p="http://www.example.com/parts"
xml:id="keyboard_03">
    <p:Part Count="101"/>
    <p:Part
xml:id="body_03"/>
  </p:Part>
</j:Job>')

*** Error 42000: [Virtuoso Driver][Virtuoso Server]XML parser detected an error:
        ERROR  : Only 0 out of 1 required attributes are defined for element <p:Part>, e.g. the element h
at line 4 column 27 of '(value to be placed in column DB.DBA.JOBS of CONTENT)'
    <p:Part Count="101"/>
-------------------------^

insert into JOBS values (2, '
<j:Job xmlns:j="http://www.example.com/jobs" xmlns:p="http://www.example.com/parts" JobID="asmkeyboard">
  <p:Part xmlns:p="http://www.example.com/parts"
xml:id="keyboard_04">
    <p:Part
xml:id="key_02" Count="101"/>
    <p:Part
xml:id="body_04"/>
  </p:Part>
</j:Job>');
```

Note:

There is no way to change a registered schema if it is used in some WITH SCHEMA constraint. Double-check any schema before using it, because it may be hard to fix the error later.

9.15. DROP XML SCHEMA Statement

```
<xml schema removal> ::=
    DROP XML SCHEMA <target URI>
```

This reverts the effect of CREATE XML SCHEMA .

The <target URI> should be a string constant that is equal to the value of "targetNamespace" attribute of "xs:schema" element of a previously declared XML schema.

The statement signals an error if the XMLSchema to be dropped is used in some WITH SCHEMA constraint.

9.16. Sequence Objects

Virtuoso supports sequence objects. These can be used to generate sequential numbers which can be used as unique identifiers. A sequence object is guaranteed never to give the same number twice. Each sequence has a name and a state. The state of a sequence is stored in the database at checkpoint time. Between checkpoints sequence states are logged so that a possible roll forward recovery will not lose information.

The SQL functions `sequence_next` () and `sequence_set` () are used to access and set the state of sequences. These take the name of the sequence as argument. This is a server/wide unique string. There are no restrictions on the length or character set of the sequence

Sequences do not have to be separately created. A sequence object will automatically be generated when first referenced by `sequence_next` () or sequence_set.

```
sequence_next (in name varchar) returns integer

sequence_set (in name varchar, in state integer, in mode integer)
```

```
                returns integer
```

Function `sequence_next` () returns the current state of the specified sequence and atomically increments it by one. The next call will thus return a number one greater than the previous. The sequence is shared between all connections and all transactions. Using a sequence never involves locking.

Function `sequence_set` () sets and returns the state of a sequence object. The next call to `sequence_next` () will return this same number. If mode equals 0, the state is set regardless of the previous state. If mode is non-zero, the state is set only if the new state is greater then the previous state. Calling sequence_set ('sequence" , 0, 1) will always return the sequence's state without changing it.

Each autoincrement column corresponds to an internal sequence object. The name of the sequence object is'DB.DBA.' plus the concatenation of the table's qualifier, owner, table name and column name, e.g. 'DB.DBA.db.dba.my_table.ai_column'. The user does not normally need to know about the sequence associated with an autoincrement column unless he or she wishes to change the sequence values using the `sequence_set()` function and the sequence objects name.

See the section on identity columns under create table and the function identity_value and the related ODBC statement option SQL_GETLASTSERIAL for more.

9.17. INSERT Statement

```
insert_statement
        : INSERT insert_mode table priv_opt_column_commalist values_or_query_spec
        ;

insert_mode
        : INTO
        | REPLACING
        | SOFT

priv_opt_column_commalist
        : /* empty */
        | '(' column_commalist ')'

        ;

values_or_query_spec
        : VALUES '(' insert_atom_commalist ')'
        | query_spec
        ;

insert_atom_commalist
        : insert_atom
        | insert_atom_commalist ',' insert_atom
        ;

insert_atom
        : scalar_exp
        ;

column_commalist
        : column
        | column_commalist ',' column
        ;

query_spec
        : SELECT opt_all_distinct selection table_exp
        ; ... See the SELECT statement next.
```

New rows (or records) are entered into a database using the INSERT statement.

If you have to enter a NULL you can simply use the keyword NULL, as you would a normal value. Since NULL is a special keyword you do not need to enclose it in single quotes.

You can specify the columns that you are inserting values into in the insert statement. One should always specify the columns that you are inserting into, in case the order of columns in the database are not as expected, or you are not inserting values into every column.

If a value is not specified for a column on insert, then the default value will be used for that column. If no default value has been specified either by a CREATE or MODIFY TABLE statement then NULL will be used.

9.17.1. INSERT SOFT

INSERT SOFT can be used in place of *INSERT INTO* if you are unsure whether the value to be inserted into a primary key column will violate that constraint. If the row with this primary key already exists, then the new row is not inserted.

```
SQL> create table insert_test(id integer primary key, txt varchar);
Done. -- 90 msec.
SQL> insert into insert_test(id, txt) values(1, 'test');
Done. -- 0 msec.
SQL> insert into insert_test(id, txt) values(1, 'test');

*** Error 23000: [Virtuoso ODBC Driver][Virtuoso Server]SR197:
  Non unique primary key on DB.DBA.insert_test. at line 4 (4) of Top-Level:
  insert into insert_test(id, txt) values(1, 'test')

SQL> insert soft insert_test(id, txt) values(1, 'testsoft');

Done. -- 0 msec.
SQL> select * from insert_test;
id                txt
INTEGER NOT NULL  VARCHAR
_______________________________________________________________________________

1                 test

1 Rows. -- 60 msec.
```

9.17.2. INSERT REPLACING

INSERT REPLACING can be used in place of *INSERT INTO* if you are unsure whether the value to be inserted into a primary key column will violate that constraint. If the row with this primary key already exists, then the new row will be inserted replacing the old values.

```
SQL> create table insert_test(id integer primary key, txt varchar);
Done. -- 90 msec.
SQL> insert into insert_test(id, txt) values(1, 'test');
Done. -- 0 msec.
SQL> insert into insert_test(id, txt) values(1, 'test');

*** Error 23000: [Virtuoso ODBC Driver][Virtuoso Server]SR197:
  Non unique primary key on DB.DBA.insert_test. at line 4 (4) of Top-Level:
  insert into insert_test(id, txt) values(1, 'test')

SQL> insert replacing insert_test(id, txt) values(1, 'testreplacing');

Done. -- 0 msec.
SQL> select * from insert_test;
id                txt
INTEGER NOT NULL  VARCHAR
_______________________________________________________________________________

1                 testreplacing

1 Rows. -- 0 msec.
```

9.18. UPDATE Statement

Existing rows (or records) are changed in the database using the UPDATE statement.

NULL values can be utilized using the `NULL` keyword without any quotes. Since NULL is a special keyword you do not need to enclose it in single quotes, doing so will cause it be read as a string-literal.

The update statement is made up by selecting the table to update, the search condition that identifies which rows you want to update, and the `column=value` of each column you wish to change.

Example 9.30. Basic Update Statement

A table can be updated using:

```
update demo.dba.employees e
  set username = 'thing'
  where email_address = 'thing@thingdom.com'
  ;
```

Example 9.31. Update from Select

It is possible to update one table based on a select from another table. Ensure that the selection is properly conditioned to update.

```
update demo.dba.employees e
        set username = (select U_NAME from DB.DBA.SYS_USERS u where u.U_EMAIL = e.email_address)
        ;
```

9.19. SELECT Statement

9.19.1. Syntax

```
< select statement: single row > ::= SELECT [ < set quantifier > ] < select list >

      INTO < select target list >

      < table expression >

< select target list > ::=
      < target specification > [ { < comma > < target specification > }... ]

< query expression > ::=
      < non-join query expression >

    | < joined table >

< non-join query expression > ::=
      < non-join query term >

    | < query expression > UNION  [ ALL ]
          [ < corresponding spec > ] < query term >

    | < query expression > EXCEPT [ ALL ]
          [ < corresponding spec > ] < query term >

< non-join query term > ::=
      < non-join query primary >

    | < query term > INTERSECT [ ALL ]
          [ < corresponding spec > ] < query primary >

< non-join query primary > ::=
      < simple table >

    | < left parent > < non-join query expression > < right parent >

< simple table > ::=
      < query specification >
    | < table value constructor >
    | < explicit table >

< query specification > ::=
    SELECT [ < set quantifier > ] < select list > < table expression >

< select list > ::=
      < asterisk >
    | < select sublist > [ { < comma > < select sublist > }... ]
```

```
< select sublist > ::=
      < derived column >
    | < qualifier > < period > < asterisk >

< derived column > ::= < value expression > [ < as clause > ]

< as clause > ::= [ AS ] < column name >

< table expression > ::=
    < from clause >
    [ < where clause > ]
    [ < group by clause > ]
    [ < having clause > ]

< from clause > ::= FROM < table reference >
    [ { < comma > < table reference > }... ]

< table reference > ::=
      < table name > [ [ AS ] < correlation name >
          [ < left parent > < derived column list > < right parent > ] ]
    | < derived table > [ AS ] < correlation name >
          [ < left parent > < derived column list > < right parent > ]
    | < joined table >

< derived column list > ::= < column name list >

< derived table > ::= < table subquery >

< table subquery > ::= < subquery >

< joined table > ::=
      < cross join >
    | < qualified join >
    | < left parent > < joined table > < right parent >

< cross join > ::=
    < table reference > CROSS JOIN < table reference >

< qualified join > ::=
    < table reference > [ NATURAL ] [ < join type > ] JOIN
      < table reference > [ < join specification > ]

< join type > ::=
      INNER
    | < outer join type > [ OUTER ]
    | UNION

< outer join type > ::=
      LEFT
    | RIGHT
    | FULL

< join specification > ::=

      < join condition >
    | < named columns join >

< join condition > ::= ON < search condition >

< named columns join > ::=
    USING < left parent > < join column list > < right parent >

< join column list > ::= < column name list >

< where clause > ::= WHERE < search condition >

< group by clause > ::=
    GROUP BY < grouping column reference list >

< grouping column reference list > ::=

    < grouping column reference >
```

```
        [ { < comma > < grouping column reference > }... ]

< grouping column reference > ::=

    < column reference > [ < collate clause > ]
```

9.19.2. Description

The SELECT statement is the principal means of information retrieval in SQL. A SELECT can retrieve information from one or more tables with arbitrary search criteria. SELECT's can also be nested to have the output of one serve as an input or search condition for another. Several SELECT's can be combined into one query with the UNION, INTERSECTION and EXCEPT operators.

The SELECT syntax consists of the following parts:

```
SELECT [DISTINCT] scalar_exp {, scalar_exp}
        FROM table {, table}
        WHERE < search condition >
        GROUP BY < column list >
        HAVING < search condition >
        ORDER BY < ordering spec list >
        FOR UPDATE
```

All parts are optional. If one or more of the clauses appear they must appear in the above order. All parts do not need to be specified, e.g., SELECT A FROM T FOR UPDATE is valid but SELECT A FROM T ORDER BY a WHERE < < 10 is not.

Note:
A select without a FROM clause is allowed.

This is useful for returning values of expressions to the client. Such a select always returns one row, with the values listed as columns. Typically only useful from interactive SQL.

Example:

select 1 + 2 as three;

A table reference in the FROM clause can either be a simple table name, another SELECT expression of the form described above or a join expression. A SELECT inside a FROM is called a derived table. This means that the rows selected by the derived table expression are treated as if they constituted a table. This is similar to a VIEW reference and a derived table can be thought of as an unnamed in-line VIEW declaration.

A join expression combines table references, which are either simple, derived or joined tables themselves into different joins.

A join is an operation that retrieves for each row of one table zero or more rows from another table. The join condition specifies how the rows are matched. The result of a join is a set of rows containing selected columns from both joined tables. Joins are by default so called INNER joins, which means that for a row to be in the result there must be a row matching the left table in the right table as specified by the join condition. An OUTER join is a join that will produce a result row even if there is no row in the right table for the row in the left table. The columns that would have come from the right table are then just set to NULL's.

```
table_ref ::=
        < table name > [< correlation name >]
  |     / query expression ) < correlation name >
  |     < table ref > < [NATURAL] join > < table ref > < join condition >

join ::=
        < empty >
  |     CROSS
  |     INNER
  |     LEFT [OUTER]
  |     RIGHT [OUTER]

join condition ::=
        < empty >
  |     ON < search condition >
  |     USING '(' < column > {, column} ')'
```

The < correlation name > is an identifier that is used to identify the table in a column reference if the same table appears many times in the query expression, e.g., is joined with itself.

The CROSS join has no join condition. This means that for each row in the left table all rows in the right table are included in the result.

9.19.3. Column Aliasing - AS Declaration

Virtuoso supports the AS operator in the selection list of a SELECT statement. This notation allows declaring a name and optionally a type and precision for a statement's output column.

The syntax is:

```
as_exp:
Scalar_exp AS NAME opt_data_type
```

For instance, one can write:

```
SELECT COUNT (*) AS NUMBEROFROWS FROM XX;
SELECT COUNT (*) AS NUMBEROFROWS INTEGER (2) FROM XX;
SELECT CONCATENATE (COL1, COL2) AS RESULTSTRING VARCHAR (50) FROM XX;
```

The AS expression tells the client to return specified values in the SQLDescribeCol, SQLColAttribute or equivalent calls. The semantics of statements is not otherwise affected. An AS expression can appear anywhere a scalar expression can but the only place where it has an effect is the selection list of a SELECT statement.

If a data type is given and contains a precision, that precision is returned to the client as the precision of the column in question.

9.19.4. Join examples

The following three statements produce an identical result.

```
select Orders.OrderID, ProductID
    from Orders natural join Order_Details using (OrderID)
select Orders.OrderID, ProductID
    from Orders join Order_Details on Orders.OrderID = Order_Details.OrderID
select Orders.OrderID, ProductID
    from Orders,  Order_Details where Orders.OrderID = Order_Details.OrderID
```

Note:

In all these cases if there exists no Order_Details row matching the Orders row there will no no result row corresponding to the Orders row. An outer join can can be used to also retrieve left table records for which there is no matching right table record.

```
select Orders.OrderID, ProductID
    from Orders natural left outer join Order_Details using (OrderID)
```

will produce a result identical to the above sample if for each Orders row there is at least one Order_Details row. If there is none however, the OrderID column from Orders will appear together with a NULL ProductID from the non-existent Order_Details.

A right outer join is like a left outer join with the left and right tables reversed.

9.19.5. Ordering and Grouping

The result rows of a query can be ordered based on their column values. The ORDER BY phrase allows specifying an ascending or descending sort order for a any column. The SQL interpreter will use an index if there is an index whose order reflects the order in the ORDER BY clause. If there is no appropriate index or if ascending and descending order is combined for columns of the same table the SQL interpreter will first evaluate the query and then sort the results before returning them.

See:

Optimizations below for more information.

```
select * from Employees order by BirthDate;
```

will list all employees, oldest first, in ascending order of birth date.

The GROUP BY clause allows computing functions over repeating groups. Without the GROUP by clause set functions (AVG, MIN, MAX, SUM, COUNT) may not be mixed with normal columns in a selection list. If set functions and columns are mixed, all the columns must appear in the GROUP BY section. Such a query will produce as many rows as there are distinct value combinations of the grouping columns. The set functions will be computed for each distinct column combination.

```
select OrderID, sum (UnitPrice * Quantity)
   from Order_Details group by OrderID
   having sum (UnitPrice * Quantity) > 5000  order by 2 desc;
```

Produces the OrderID and total value of the order in decreasing order of order value. The HAVING clause specifies that only orders with a value > 5000 will be counted. Note that the sum expression in having must be written identically to the same expression in the SELECT left.

The 2 in the order by refers to the second column of the select, which has no name, it being a function reference.

CUBE and ROLLUP

Virtuoso database offers a way to increase efficiency of SQL summary queries and simplify the operations. By using the options `ROLLUP` and `CUBE` in `GROUP BY` , can be executed more comprehensive summary operations. The produced result sets could be also produced without using these option but rahter with additional coding and queries.

The options `ROLLUP` and `CUBE` extend the result set of `GROUP BY` :

- `ROLLUP` : builds a consequence of subtotal aggregates on every queried level including the grand total.
- `CUBE` : this option is an extension of

  `ROLLUP` . It builds all possible subtotal aggregates combinations for given

  `GROUP BY` .

For example:

```
  SELECT j, grouping (j), k, grouping (k), t, grouping (t), SUM (i)
    FROM MyDemo
GROUP BY ROLLUP (j,k,t);

  SELECT j, grouping (j), k, grouping (k), t, grouping (t), SUM (i)
    FROM MyDemoCube
GROUP BY CUBE (j,k,t);
```

The `grouping(param)` procedure returns "1" if the column "param" is not in the dynamic GROUP BY set, and returns "0" otherwise.

The result set of:

```
  SELECT j, k, t, SUM (i)
    FROM MyDemo
GROUP BY ROLLUP (j,k,t) ;
```

is equivalent of the accumulated result sets of:

```
  SELECT j, k, t, SUM (i)
    FROM MyDemo
GROUP BY j,k,t;

  SELECT NULL, k, t, SUM (i)
    FROM MyDemo
GROUP BY k,t;

  SELECT NULL, NULL, t, SUM (i)
    FROM MyDemo
GROUP BY t;
```

```
  SELECT NULL, NULL, NULL, SUM (i)
    FROM MyDemo;
```

The result set of:

```
SELECT t,s, SUM (i) FROM MyDemo GROUP BY CUBE (t,s);
```

is equivalent of the accumulated result set of:

```
  SELECT t,s, SUM (i)
    FROM MyDemo
   GROUP BY t,s;

  SELECT t,NULL, SUM (i)
    FROM MyDemo
GROUP BY t;

  SELECT s,NULL, SUM (i)
    FROM MyDemo
GROUP BY s;

  SELECT NULL,NULL, SUM (i)
    FROM MyDemo;
```

9.19.6. Grouping Sets

The grouping sets variant of group by allows specifying exactly which combinations of groupings are needed. `CUBE` and `ROLLUP` are shorthands that expand into a grouping sets specification. This is useful for applications like faceted search where there are separate group by's on properties of interest.

The grouping function behaves identically to `CUBE` and `ROLLUP` .

The syntax is:

```
GROUP BY grouping sets (<grouping set>[,...])
```

The grouping set is a list of grouping columns in parentheses or `()` to denote an aggregate without grouping.

For example:

```
  SELECT key_table, key_is_main, COUNT (*)
    FROM SYS_KEYS
GROUP BY grouping sets ((key_table), (key_is_main), ());
```

This shows for each table the number of indices, then the number of primary key and non-primary key indices across all tables and finally the total number of indices in the schema. A `CUBE` would produce these same groupings but it would further add the `key_table` , `key_is_main` grouping.

As an extension to SQL 99 OLAP extensions, the Virtuoso grouping sets specification also supports an `ORDER BY` with optional top. In the place of a grouping set one can write:

```
ORDER BY [top k] (<column> [,...])
```

In this way a single query can produce a set of ordered result rows and different grouped aggregates on the columns in the result set.

9.19.7. Derived Tables

A SELECT expression may be used in the place of a table in a FROM clause. This provides control over where DISTINCT and ORDER BY operations are evaluated.

```
select ProductName, UnitsInStock
    from (select distinct ProductID from Order_Details) O,
        Products where Products.ProductID = O.ProductID;
```

This retrieves the name and quantity of products that have been ordered.

An equivalent phrasing would be

```
select distinct ProductName, UnitsInStock
   from Order_Details O, Products where Products.ProductID = O.ProductID;
```

The difference is that the latter retrieves a Products row for each order line whereas as the first retrieves a products row for each distinct product in the order lines. The first is therefore faster to evaluate. Also note that the rows in the DISTINCT buffer in the first example only consist of the product id whereas they are much longer in the second example.

Note that a correlation name is required for derived tables since the derived table is as such anonymous.

9.19.8. Query Expressions

```
< non-join query expression > ::=
      < non-join query term >
    | < query expression > UNION  [ ALL ]
          [ < corresponding spec > ] < query term >
    | < query expression > EXCEPT [ ALL ]
          [ < corresponding spec > ] < query term >

< corresponding spec > ::=
    CORRESPONDING [ BY < left parent >
        < corresponding column list > < right parent > ]
```

Queries can be combined by set operators `UNION`, `INTERSECTION`, and `EXCEPT` (set difference). The `ALL` keyword will allow duplicate rows in the result set. The `CORRESPONDING BY` clause allows specifying which columns will be used to determine the equality of rows from the left and right operands.

```
 select OrderID from Orders except
    corresponding by (OrderID) select OrderID from Order_Details
```

will produce the `OrderID` of orders that have no `Order_Details`. This is equivalent to:

```
select OrderID from Orders a
    where not exists (select 1 from Order_Details b
                      where a.OrderID = b.OrderID)
```

Note that the queries are executed quite differently, although to a similar effect. There may be significant difference in performance.

9.19.9. LIKE Predicate & Search Patterns

The *`LIKE`* predicate expects a pattern to be applied to a `varchar` or `nvarchar` column to qualify the results to be returned from a query.

If the pattern does not begin with an at-sign (`@`) or with two asterisks (`**`), then we test the equality of the string and pattern with ordinary wildcard matching, which behaves approximately like the filename pattern matching in the Unix shell (but not like the regular expression matching in utilities like `grep` and `sed`).

The following characters have special significance in the pattern:

`?` - Matches any single character.

`*` - Matches zero or more of any characters.

`[ ]` - (Called a group-expression here; detailed below.

The group expression `[ ]` matches any one of the enclosed characters, unless the first character following the opening `[` is `^`, then matches only if the character (in the datum string) is not any one of those specified after the `^` (i.e., the `^` negates the meaning of this expression.)

You can use character ranges like `0-9` (shorthand for `0123456789`) inside the brackets, in which case the character in the datum string must be lexically within the inclusive range of that pair. In other words, the lexical (ASCII) value of the character at

the left side of hyphen must be less than the lexical (ASCII) value of the character at the right side.

The hyphen (–) can be included in the character set by putting it as the first or last character. The right bracket (]) can be included by putting it as the first character in the expression, i.e., immediately after the opening bracket ([) or the caret (^) immediately following the opening bracket.

That is, the hyphen indicates a range between characters, unless it is the first or the last character in the group expression, in which case it matches just to itself.

Example 9.32. Examples:

```
[abc]          Matches any of the letters a, b, or c.
[^0123456789]  Matches anything, except digits. (Same as [^0-9].)
[[]            Matches [
[]]            Matches ]
[][]           Matches ] and [
[^]]           Matches anything except ]
[A-Za-z0-9]    Matches all the alphanumeric characters.
[-*+/]         Matches the four basic arithmetic operators.
[-]            Matches to single hyphen.
[]-]           Matches to ] or -
[-[] or [[-]   Matches to - or [
```

@ - Matches the character last matched to ? or group-expression. For example, ?*@ matches to all strings which begin with the same character as they end with. However, if there is neither ? nor [] expression at the left side of @ in the pattern, then @ matches just to itself (e.g., *@* should match to all e-mail addresses).

Any other characters match ONLY to themselves, that is, not even to the upper- or lower-case variants of the same letter. Use expressions like [Wo][Oo][Rr][Dd] if you want to find any mixed-case variant of the word "word", or use the substring search explained below.

However, if the pattern begins with an at-sign (@), then we compare the rest of pattern to string with the fuzzy matching, allowing differences of few characters in quality and quantity (length). If there is more than one @ at the beginning of pattern, they are all skipped, and many additional liberties are taken for the match function. The more @ signs there are at the beginning, the more fuzzy (liberal) is the search. For example, pattern "@Johnson" will match string "Jonsson", and pattern "@@Johnson" will also match "Jansson".

If the pattern begins with two asterisks (**), then we do diacritic- and case-insensitive substring search, trying to find the string given in the rest of pattern from the datum string.

Example 9.33. Example:

"**escort" will match to "Ford Escort vm. 1975".

If there are any ISO8859.1 diacritic letters (e.g., vowels with accents or umlauts, or letters like the Spanish n with ~ (tilde)) present in the datum string, then the plain unaccented (7-bit ASCII) variant of the same letter in the pattern string will match to it. However, if there is any diacritic letter specified in the pattern string, then it will match only to the upper- or lower-case variant of exactly the same diacritic letter.

The rationale behind this is that the people entering the information to the database can use the exact spelling for the word - for example, writing the word "Citroen" with the umlaut-e (e with two dots above it, ë), as it is actually written in French - and the people who search for "Citroën" can still find it without having to remember the exact orthography of the French, by just giving the word "citroen". This also allows people who have just plain 7-bit ASCII keyboards to search for words like Ra"a"kkyla" (a place in Finland, a" means umlaut-a, i.e., a with two dots above it, ä), just by entering the word raakkyla.

So the following hold for substring searches:

1) Any non-alphabetic character in the pattern matches just to itself in the datum string (e.g., ? to ? and 3 to 3).

2) Any 7-bit ASCII letter (`A-Z` and `a-z` without any diacritic signs) in the pattern matches to any diacritic variant of the same letter (as well as to the same 7-bit ASCII letter) in the datum string, either in upper- or lower-case.

3) Any diacritic letter (8-bit ISO8859.1 letter) in the pattern matches only to the same letter (in upper- or lower-case) in the datum string.

Note:

Because the internal matching functions use macros which also consider characters like `@`, `[`, `\`, `]`, and `^` to be letters, they will match against characters `` ` ``, `{`, `|`, `}`, and ~, respectively, because in some older implementations of European character sets, those characters mark the uppercase and lowercase variants of certain diacritic letters.

It is generally better to match too liberally, and so maybe sometimes give something entirely off the wall to the user, than to miss something important because of too-strict criteria.

Of course, when searching from the data which contains text in some wide-character format (like certain coding systems for Japanese and Chinese, where one character is coded with two bytes), neither fuzzy matching function nor `nc_strstr` function presented here should be used, as they would often match on entirely spurious cases.

9.19.10. The TOP SELECT Option

```
query_term :  SELECT opt_top selection ....

opt_top :  opt_all_distinct [ TOP INTNUM ]
        |  opt_all_distinct [ TOP SKIPINTNUM, INTNUM ]
        |  opt_all_distinct [ TOP (num_scalar_exp) ]
        |  opt_all_distinct [ TOP (skip_num_scalar_exp, num_scalar_exp) ]
opt_all_distinct : [ ALL | DISTINCT ]
```

The TOP n phrase can follow an optional ALL or DISTINCT keyword in a SELECT, either at top level or inside a query term of an expression or subquery. The query with the TOP option will generate at most the indicated number of result rows.

The scope of TOP is the query in question. For example

```
select top 3 row_no from t1 best union select top 3 row_no + 1 from t1;
```

Will always return 4 rows assuming there are at least 3 rows in T1.

The optional SKIPINTNUM parameter lets you offset the selection by SKIPINTNUM number of rows. If you have a ten-row table and `select top 2 from this_table` you get the first two rows, `select top 2, 2 from this_table` will return the third and fourth rows only, instead.

9.19.11. CASE, NULLIF, COALESCE, CAST Value Expressions

The CASE Expression

There are many situations where you might find it useful to alter the the data returned by a SQL query based on a few rules. For example, you may want to display Customers gender as 'Male' or 'Female' based on whether their title is 'Mr' or one of 'Miss', 'Mrs' or 'Ms'. The CASE expression can easily accommodate this.

The Syntax of CASE is:

```
CASE
  WHEN <search-condition> THEN <output>
  WHEN <search-condition> THEN <output>
  ...
  ELSE <output>
END
```

When a <search-condition> is met the corresponding <output> is returned. If no conditions are met then the <output> after is ELSE is returned as a default value.

Example 9.34. Using the CASE expression

```
SELECT Title, CustomerName,
  CASE
    WHEN Title = 'Mr' THEN 'Male'
```

```
    WHEN Title = 'Mrs' THEN 'Female'
    WHEN Title = 'Miss' THEN Female'
    WHEN Title = 'Ms' THEN 'Female'
    ELSE 'Unknown'
  END as Gender,
  Company
  FROM Customers
```

May return values such as:

```
Title   Gender  CustomerName        Company
VARCHAR VARCHAR VARCHAR             VARCHAR
_______________________________________________________

Mr      Male    Thomas Hardy        Around the Horn
Miss    Female  Christina Berglund  Berglunds shop
Mrs     Female  Hanna Moos          Blauer See Delikatessen
Mr      Male    Laurence Lebihan    Bon app
```

There is also a short hand notation for the CASE expression as follows:

```
CASE <search-parameter>
  WHEN <search-value> THEN <output>
  WHEN <search-value> THEN <output>
  ...
  ELSE <output>
END
```

This short hand is best demonstrated by the rewrite of the above example as follows:

Example 9.35. Using the CASE short-hand expression

```
SELECT Title, CustomerName,
  CASE Title
    WHEN 'Mr' THEN 'Male'
    WHEN 'Mrs' THEN 'Female'
    WHEN 'Miss' THEN Female'
    WHEN 'Ms' THEN 'Female'
    ELSE 'Unknown'
  END as Gender,
  Company
  FROM Customers
```

In both cases the ELSE keyword is optional. If ELSE is unspecified then ELSE NULL is implicit.

The NULLIF Expression

The NULLIF expression is a short hand implementation of a special case of the CASE expression for a popular demand. Consider the following CASE expression:

```
CASE col1
  WHEN 'something' THEN NULL
  ELSE col1
END
```

This is replaced by the NULLIF expression which achieves the same result using the following, much shorter expression:

```
NULLIF (col1, 'something')
```

This is often useful in situations where you have a code to denote a value as unspecified for whatever reason, but in many applications you would rather this was NULL.

The COALESCE Expression

The COALESCE expression is another application of the CASE expression to suit another frequent requirement. The syntax of COALESCE is as follows:

```
COALESCE (value-1, value-2, ..., value-n)
```

COALESCE returns the first non-NULL parameter. This is equivalent to

```
CASE
  WHEN value-1 IS NOT NULL THEN value-1
  WHEN value-2 IS NOT NULL THEN value-2
  ...
  ELSE value-n
END
```

The CAST Expression

SQL has always been considered a strongly typed language, meaning that you cannot have expressions that contain arbitrary data types. Casting is invaluable for comparing values that are obviously compatible but their data types are not, such as `1 = '1'`. This attempts compares an integer with a char which would not work unless one of the values was cast as follows:

```
cast('1' as integer) = 1
```

See Also:

The CASTING section for more information.

9.19.12. SELECT BREAKUP

Virtuoso extends the select statement with a breakup option. This option allows a single row selected by a derived table to be seen as multiple rows by the enclosing query.

This is specially useful when translating relation tables to RDF. Using breakup, one can do a single pass over a table and generate multiple triples, each presented as a separate result row.

The syntax is:

```
SELECT BREAKUP breakup_term [, ...] FROM ....
breakup_term ::=

scalar_exp [, scalar_exp...] [WHERE search_condition]
```

Each breakup term is a list of comma separated expressions with an optional search condition at the end. Each list is treated as a select list in a union, i.e., they must be of equal length and the leftmost list must provide a name for each column. This means that an AS declaration is needed if the expression is not a column.

If a breakup tern has the optional WHERE clause, the condition is evaluated in the scope of the select, i.e., all that is defined by the FROM. If the condition is true, the row represented by the breakup term is added to the result set of the breakup select, otherwise it is ignored.

A breakup select is only allowed in a derived table or a union or other set operator term inside a derived table. A top level breakup select is not allowed. To have a breakup select as the topmost select, simply write:

```
select * from (select breakup .... from ...) f;
```

Breakup cannot be mixed with distinct, top, group by or order by. Again, to combine these, use nested derived tables.

Breakup operates equally well on local and remote tables. Breakup is never passed on to a remote but the FROM of a breakup select can consist of tables from any source.

Example 9.36. Examples:

```
select * from (select breakup (a.row_no, b.fi2) (b.row_no, a.fi3 where a.fi3 is not null)
```

```
from r1..t1 a, r1..t1 b
where a.row_no < 10 and b.row_no = a.row_no)f;
```

This produces 2 rows for each result of the join, except if fi3 is null, in which case only the first term of the breakup is returned in the result set.

9.20. COMMIT WORK, ROLLBACK WORK Statement

These statements reset the current transaction. COMMIT WORK leaves all the changes made by the current transaction in effect whereas ROLLBACK work reverses them.

In both cases, the transaction will be in a fresh state, having no locks and no changes that could be rolled back. The rollback operation always succeeds, as any change is always reversible until committed. COMMIT WORK may fail if the transaction had been marked to be canceled before the COMMIT WORK operation started. A failed commit has the effect of a rollback but it will signal a SQL STATE descriptive of the error, e.g. 40001 (deadlock).

These operations are typically not needed, since the SQLTransact ODBC call and the ODBC autocommit mode are used instead for transaction control. The only use for these statements is within stored procedures, where it may be practical to break a long sequence of operations into several transactions to reduce lock contention.

These can also be used together with the WHENEVER declaration to automate retry upon deadlock inside stored procedures.

Triggers should not normally use these statements. The exception is the case where a trigger detects a state violating application consistency rules and decides to abort the transaction. This can be done by ROLLBACK WORK, typically followed by a call to the signal function for notifying the application.

Example 9.37. Examples:

```
create procedure retry (in x integer)
{
  whenever sql state '40001' goto deal;
 again:
  -- action
  return;
 deadl:
   rollback work;
   goto again;
}

create trigger sal_negative on "Employee" after update ("Salary")
{
  if ("Salary" < 0) {
    rollback work;
    signal ('A0001', 'Salary cannot be negative');
  }
}
```

See Also:

txn_error , txn_killall , signal

9.21. CHECKPOINT, SHUTDOWN Statement

```
admin_statement
        : SHUTDOWN opt_log
        | CHECKPOINT opt_log
        | BACKUP opt_log
        ;
```

The checkpoint is a point in the history of a database where all the state is written on disk as a single, consistent image that contains all the state committed so far and no uncommitted state. A transaction log starts after a checkpoint and contains the information to allow the recreation of the effect of transactions committed since the checkpoint. The checkpoint state and the

transaction log together allow recovering the database up to the last committed transaction.

The CHECKPOINT statement forces a checkpoint to be made. Making the checkpoint allows starting a new transaction log. If no new log name is specified the old log is truncated to length 0 and reused for logging transactions. If the CheckpointAuditTrail option is enabled in virtuoso.ini a new log will be started even if no new log is specified in the checkpoint or shutdown statement.

The SHUTDOWN statement performs a CHECKPOINT, and terminates the server upon completion.

BACKUP is an alternate notation for backup().

Example 9.38. Examples:

```
checkpoint 'new.log';
backup 'bak.log';
shutdown 'new2.log';
```

The above sequence of commands makes a checkpoint and starts logging subsequent transactions into new.log. The backup statement makes bak.log, which represents the state prior to starting new.log. The shutdown statement makes a new checkpoint and marks new2.log as the log file to be used for logging transactions after the database restarts. The database server exits at the completion of the SHUTDOWN statement.

```
replay ('bak.log');
replay ('new.log');
```

These statements executed on an empty database will recreate the state in effect after the last transaction to commit before the SHUTDOWN statement of the previous example.

See Also

The Backup section for more backup and recovery information.

Example 9.39. Example for control the transaction logging:

```
create procedure log_test ()
{
  -- disable the transaction logging
  log_enable (0);

  -- action code, for ex.:
  delete from TAG_REL_INX;
  insert into TAG_REL_INX (TR_T1, TR_T2, TR_COUNT) select TR_T1, TR_T2, TR_COUNT from TAG_REL;

  exec('checkpoint');

  -- enable the transaction logging
  log_enable (1);
};
```

9.21.1. Checkpoint & Page Remapping

In concept, making a checkpoint consists of writing the last committed state of a page on the read-only pre-checkpoint state. This is in case the page existed before the previous checkpoint. If the page was created after the last checkpoint, making the checkpoint consists of just writing it to disk. Now if the number of pages modified between checkpoints largely exceeds the amount of RAM, the checkpoint will be a disk-to-disk copy process which takes a time proportional to the number of modified pages. This can take a long time. Virtuoso offers a mechanism called checkpoint remap. This allows making a committed state persistent, i.e. safe, without copying all the delta collected since the last checkpoint over the pre-checkpoint state.

The checkpoint remap mechanism means that a page, say number 12 get remapped to 15 when updated. Suppose now that the page were written to disk. Now we have the pre-checkpoint page on 12 and the committed post checkpoint state on 15. If a checkpoint now takes place, instead of copying 15 over 12, which may involve reading the page, we just write that 12 is actually on 15. This speeds up the checkpoint but causes one page to take the space of 2. Now suppose 12 were modified again, now we

would see that the pre checkpoint page is 15 and that the original 12 is free. The page now gets mapped to 12. The next checkpoint now will mark that 12 is 12, which was the original condition and 15 is free.

The mechanism is actually more complex but this is the basic idea. By allowing a page to occupy two pages in some situations we dramatically cut down on atomic checkpoint time and improve availability.

Now we may argue that this destroys locality since pages that were created contiguous are no longer that after an update. This is why there is also a mechanism called unremap. and there is an upper limit to checkpoint remaps. If every page were updated between checkpoints, they would alternately be in their original place or in a remote place from which the next update-checkpoint combination would return them to the original. Now this is not always the case. Also there is not always enough space to keep up to three copies of each page. Therefore there is an unremap mechanism and an upper limit to checkpoint remaps.

The *MaxCheckpointRemap* parameter in the virtuoso.ini file controls how many pages may be stored on a page other than their logical page. The *UnremapQuota* parameter in the virtuoso.ini file controls how many previously remapped pages are unremapped each time a checkpoint is made.

Having an unlimited MaxCheckpointRemap and zero UnremapQuota will cause a checkpoint to consist exclusively of a flush of dirty buffers. The downside is that a page can take the space of two for an unlimited period of time and that original locality is not maintained. Setting the maximum checkpoint remap to zero in such a situation causes the next checkpoint to unremap all checkpoint remapped pages, restoring physical sequence and freeing disk space.

See Also:
The TPC C Benchmark chapter for examples of checkpoint remapping in use on test environments.

9.22. Stored Procedures as Views & Derived Tables

Virtuoso allows using a stored procedure result set in place of a table. A view may also be defined as a stored procedure. This provides smooth integration to external procedural logic in queries.

When a procedure appears as a table, the procedure is called and its result set is inserted into a temporary space. Processing continues from that point on as if the data came from a table.

Queries involving procedure views or derived tables are subject to normal join order selection. For this purpose it is possible to associate a cost to a procedure used in a procedure view or derived table. If the option (order) clause is given at the end of the select, joins are done left to right. If a procedure is in the leftmost position in the from it will be called once for the query, if it is in the second position it will be called once for every row of the leftmost table that passes selection criteria applicable to it and so on.

Procedures used as tables can get parameters from the query. These parameters are expressed in the containing select's where clause as column = expression, where column is a parameter name of the procedure table.

A procedure derived table is of the form:

```
q_table_name '(' column_commalist ')' '(' column_def_commalist ')' alias
```

The first column commalist is the parameter list. The second column_def_list is a description of the result set, as in a CREATE TABLE statement. The correlation name alias is required if the procedure occurs as a derived table, with no view definition.

A procedure view is declared as follows:

```
CREATE procedure VIEW new_table_name
        AS q_table_name '(' column_commalist ')' '(' column_def_commalist ')'
```

The columns in the column definition list should correspond to the procedure's result set columns. The columns are explicit in the view so as to be able to interpret the view definition and to be able to compile procedures and queries using the view before the procedure itself is defined. Thus the procedure need be defined only at time of execution, not at time of definition.

The meta-data returned by ODBC catalog calls for a procedure view will show the columns as they were declared, just like a regular view. Procedure views are never updatable.

9.22.1. Procedure Table Parameters

If there is a condition that is in the top level set of AND'ed conditions in the table expression's WHERE clause and if it is an equality condition and if it references a parameter of a procedure table and if the other side of the equality does not reference the procedure table or a table to the right of it, then this condition is considered a parameter. This means that the condition is not actually evaluated but rather that the other side of the equality is evaluated before calling the procedure and that the value returned is passed to the procedure as an input parameter in the position indicated by the name in the parameter list of the view or derived table.

If the procedure table is referenced in an explicit join, as in outer or inner join syntax, only equalities in the join condition (ON clause) are eligible to specify a parameter. Equalities in the enclosing query's WHERE phrase will be considered tests on the results, not parameters.

The parameter names in the procedure view's or derived table's parameter list do not have to be names in the output columns, although this will often be the case. The number of parameters in the parameter list in the view or derived table must match that in the procedure definition but the names do not have to be the same. Only input parameters are supported.

If a parameter is specified but no applicable predicate is found, a NULL value is passed.

9.22.2. Procedure Table Result Sets

The result set is declared in the derived table or view. This should match the result_names in the procedure but the former will take precedence on the latter. If an actual result row is shorter than the declared set, the missing columns will default to NULL.

9.22.3. Procedure Tables & Security

Accessing a procedure as a table requires execute privileges on the procedure. Privileges declared on the view are not checked.

9.22.4. Procedure Table Cost and Join Order

The __cost declaration in a procedure definition can associate a cost to a procedure. This declaration is then used for assessing different join orders. Note that depending on the join order, some parameters of a procedure table may or may not be available. It is of the form __cost (n1, n2,...), where each n is a literal number. The declaration can figure anywhere in the procedure's body as a regular statement. At least one number is required. The first number is the one-time cost of calling the procedure. The second number is the number of result rows that will be produced by the call, defaulting to 1. The numbers from third onwards correspond to the parameters of the procedure from left to right. If a parameter is NOT given or is NOT known because of join order, then the cost of the single call and the result count will be multiplied by the number corresponding to the parameter.

Consider the declaration __cost (10, 3, 5);. The procedure will take 10 units of time per call and produce 3 rows if the first parameter is known. Otherwise it will take 50 units and produce 15 rows. This is a rough way to specify the selectivity of specifying a parameter versus not specifying it. One may liken this to specifying or not specifying conditions of a table's columns.

The unit of cost is an internal abstract unit. For purposes of scaling, selecting a single row from a table of 1000 on an exact match of an integer key is 3 units. The costs are shown by the explain function with a second argument of -5.

9.22.5. Limitations

There is no limitation to the number of rows in a procedure result set. The temporary storage takes place in the database similarly to a sorted ORDER BY. Number of columns is limited to the maximum number of columns in a real table. Total row size limit for tables and ORDER BY intermediate results applies. Blobs are allowed and do not count towards the row length limit.

9.22.6. Procedure Table Examples

```
create procedure n_range (in first integer, in  last integer)
{
  declare n, n2 integer;
  n := first;
  result_names (n, n2);
  while (n < last){
    result (n, 2 * n);
    n := n + 1;
```

```
  }
}

select n, n2 from n_range (first, last) (n int, n2 int) n
        where first = 2 and last = 12;
```

This returns a set of numbers from 2 to 11 and from 4 to 22.

```
select a.n, b.n from n_range (first, last) (n int, n2 int) a,
    n_range (f2, l2) (n int, n2 int) b
  where first = 2 and last = 12 and f2 = a.n - 2 and l2 = a.n + 2;
```

Here we join the second call to the procedure to the first, effectively passing the output of the left call as parameters to the right call.

```
create procedure view n_range as n_range (first, last) (n1 int, n2 decimal);
```

This defines the procedure as a view, so that it can be referenced like a table.

```
select * from n_range a, n_range b where a.first = 1 and a.last = 11
                and b.last = a.n1 + 2 and b.first = a.n1 - 2;
```

This is the previous join but now using the view.

9.23. GRANT, REVOKE Statement

```
privilege_def
        : GRANT ALL PRIVILEGES TO grantee
        | GRANT privileges ON table TO grantee_commalist opt_with_grant_option
        | GRANT grantee_commalist TO grantee_commalist opt_with_admin_option
        ;

privilege_revoke
        : REVOKE ALL PRIVILEGES FROM grantee_commalist
        | REVOKE privileges ON table FROM grantee_commalist
        | REVOKE grantee_commalist FROM grantee_commalist
        ;

opt_with_grant_option
        : /* empty */
        | WITH GRANT OPTION
        ;

opt_with_admin_option
        : /* empty */
        | WITH ADMIN OPTION
        ;

privileges
        : ALL PRIVILEGES
        | ALL
        | operation_commalist
        ;

operation_commalist
        : operation
        | operation_commalist ',' operation
        ;

operation
        : SELECT priv_opt_column_commalist
        | INSERT
        | DELETE
        | UPDATE priv_opt_column_commalist
        | REFERENCES opt_column_commalist
        | EXECUTE
        | REXECUTE
        | role_name
        ;
```

```
grantee_commalist
        : grantee
        | grantee_commalist ',' grantee

        ;

grantee
        : PUBLIC
        | user
        ;

user
        : IDENTIFIER

role_name
        : IDENTIFIER
```

The GRANT and REVOKE statements are used to define privileges on resources to users and user groups (roles). A resource is a table, view or stored procedure. A grantee can be PUBLIC, meaning any present or future user accounts or a user name. Granting a privilege to a user name means that this user AND any users which have this user as their user group have the privilege.

Only a granted privilege can be revoked. The sequence:

```
grant select on employee to public;
revoke select (e_review) from joe;
```

Is invalid because the privilege being revoked was not granted, instead it was implied by the select on all column to public.

Any role name created by the CREATE ROLE statement is a valid grantee and a valid grantable operation.

The term 'effective privilege of a user' means the set of privileges given to a user by virtue of 1. granting them to public 2. granting them to a role which is granted to the user or to a role either directly or indirectly granted to the user or 3. granting them, to the specific user. The dba user and all users whose group is 'dba' have all privileges.

The phrase

```
GRANT ALL PRIVILEGES TO user;
```

is synonymous with setting the user's group to 'dba'.

The effective privileges inside a trigger are those of the owner of the table whose trigger is executing. The privilege of executing a trigger is derived from the privilege of performing the trigger action, e.g. update of a specific column.

The effective privilege inside a stored procedure is that of the owner of the procedure. The privilege to execute a given procedure is granted with the EXECUTE clause.

Note:
The grantee names are identifiers. This means that their case can be converted to upper case unless they are quoted. The identifier case conversion depends on the global CaseMode setting.

A user may grant or revoke privileges on resources of which he/she is the owner. A user with effective dba privileges may manipulate any privileges.

Example 9.40. Examples:

```
grant update ("Salary") on "Employees" to "Manager";

grant execute on "SalesReport" to "Accounting";
```

GRANT REFERENCES is a privilege required by a user on a table so that this user can create new tables referencing such tables in foreign keys where he/she would otherwise be restricted.

Remote SQL Data Sources. To provide further consistent security to remote data, only the DBA group is permitted to use the `rexecute()` , unless explicitly granted. Caution is required here since any user granted use of `rexecute()` has full control of the remote data source set-up by the DBA, however limited to the overall abilities of the remote user on the remote data source.

Users can be granted and denied access to `rexecute()` using the following syntax:

```
GRANT REXECUTE ON '<attached_dsn_name>' TO <user_name>
REVOKE REXECUTE ON '<attached_dsn_name>' FROM <user_name>
```

UDTs. Security of UDTs is maintained through normal SQL GRANT and REVOKE statements via a simple extension. You can define the level of access to both native and externally hosted UDTs. Grants for persistent user defined types are persisted into the SYS_GRANTS table. Grants on temporary user defined types are in-memory only and are lost (together with the temporary user defined type definition) when the server is restarted.

There are two GRANT/REVOKE types for UDTs as follows:

EXECUTE - all methods and members of a class are accessible to the grantee.

UNDER - the grantee can create subclasses of the class.

```
GRANT/REVOKE EXECUTE on <user_defined_type>
GRANT/REVOKE UNDER on <user_defined_type>
```

Note:

SQL modules, user defined types and SQL stored procedures are exposed to GRANT/REVOKE in the same namespace, therefore care must be taken avoid inadvertently granting to multiple objects at the same time.

See Also:

Virtual Database Procedures & Functions

9.24. SET Statement

9.24.1. ISOLATION

This allows setting a transaction isolation in a stored procedure or trigger body. The values are:

uncommitted

committed

repeatable

serializable

These are case insensitive strings.

This takes effect dynamically until replaced by another SET ISOLATION setting. The effect never persists over the return of the containing procedure or trigger. The effect does extend into procedures or triggers called from after executing the SET ISOLATION statement.

Example 9.41. Setting the isolation level

```
set isolation='serializable';
```

The initial isolation comes from the SQL_TXN_ISOLATION statement option in the ODBC API (SQLSetConnectOption). The default isolation is repeatable read.

9.24.2. LOCK_ESCALATION_PCT

This controls the escalation from row locking to page locking. A set of row locks can be converted into one page lock if: (a) All the row locks on the page belong to the same transaction, or, (b) No other transaction waits for any of these locks. The value of this parameter is the percentage of rows on a page that must be held by the transaction before the locking goes to page level. The default is 50, meaning that for a page of 120 rows the 61st row lock will escalate the lock if all the previous locks belong to the same transaction and there is no wait pending on any. A value of -1 means that locking is always at page level if there is more

than one lock on the page. A value in excess of 100 causes lock escalation to be turned off. The effect of this setting is global and persists until the server is restarted. This setting does not affect the semantic of locking.

9.24.3. transaction_timeout

This allows setting a timeout for the current transaction. The value must be an integer count of milliseconds from the beginning of the transaction. The transaction is deemed to begin when the first client statement executes inside it or, if the transaction starts from a commit work or rollback work statement in PL, from the time of this statement. If the transaction does not commit or rollback within so many milliseconds of its beginning, it is liable to get terminated, which is signalled to any code running inside the transaction as a SQL state S1T00. When this is signalled the transaction is set into an uncommittable state and must be rolled back. When the timeoutg elapses, all locks and uncommitted state belonging to the transaction are freed.

This setting remains in effect until the transaction terminates. Any timeout for a next transaction must be set separately. This feature can be used for enforcing maximum running times on operations. The timeout will take effect also if the transaction holds no locks. This setting corresponds to the Virtuoso ODBC extension SQL_TXN_TIMEOUT statement option.o

9.24.4. PARAM_BATCH

This sets the batch size used by the virtual database array parameter optimization batch size. This causes several consecutive executes of the same statement to be grouped as a single ODBC operation with array parameters. This optimizes joins of tables on different servers and searched updates, inserts and deletes on attached tables. Most ODBC drivers do not support array parameters. A value of 1 or 0 disables the optimization. This should be done if there is a driver which falsely claims to support array parameters. If a given driver returns an error when setting array parameters the VDB will detect this and will not try to use them.

The effect of this setting is global and persists until the server is restarted. The default value comes from the ArrayParameters configuration parameter.

9.25. Anytime Queries

Starting with version 6, Virtuoso offers a partial query evaluation feature that guarantees answers to arbitrary queries within a fixed time. This is intended for use in publicly available SPARQL or SQL end points on large databases. This enforces a finite duration to all queries and will strive to return meaningful partial results. Thus this provides the same security as a transaction timeout but will be more user friendly since results will generally be returned, also for aggregate queries. Outside of a public query service, this may also be handy when exploring a large data set with unknown properties.

The feature is activated with the statement

```
set result_timeout == <expression>;
```

The expression should be a count of milliseconds. The scope of the setting is the connection, thus it remains in effect past the return of the calling procedure. Setting the timeout to 0 returns to the default state of having no limit on query run time and always returning complete results. The initial value of the timeout is 0. On a web server thread, the timeout is reset to 0 at the start of processing each new request header.

After a query or cursor has exceeded the running time, it is reset. In the event of the reset occurring inside an aggregation, the part of the query that produces rows for the aggregation is terminated, the timeout is reset and the query continues by using the aggregated results that were obtained before the first timeout. If there is again a timeout, the present aggregation is reset and the query gets another lease on life for processing the results accumulated so far. Thus, a query which has a select, a group by and an order by can consume at most three timeouts. One for producing the aggregated rows, another for sorting them and finally a third for returning them to a client or iterating over them in a stored procedure.

If the timeout occurs outside an aggregation, the result set is simply truncated.

The fact of a query returning partial results is indicated by the result set ending with a S1TAT SQL state. If the result set is read to end without this state being signalled, the results are complete and the query was not interrupted. If the result set is not consumed to end, the client cannot be sure of its completeness.

The error message associated with the S1TAT state contains the human readable version of the output of db_activity () for the query. If result_timeout is non-zero, the opening of a cursor or execution of a query on a client connection or with the exec ()

function will reset the client resource consumption counters automatically. This is done so as to have the resource consumption statistics for the error message scoped to the query. To get the exact counts, the application may call db_activity (1) in the same procedure. If this is called on a client server connection with the timeout on, the counts will be reset before db_activity () is called.

The resolution of the timeout is fairly low, timeouts are checked every two seconds with default settings.

A query cannot set a timeout from inside itself. Only a stored procedure or a top level statement on a client connection can do this.

A request to SPARQL web service endpoint may specify the timeout as an additional *&timeout=<milliseconds>* parameter. The parameter value should be equal to or greater than 1000, otherwise it will be ignored. If *SPARQL* Virtuoso configuration file contains *MaxQueryExecutionTime* parameter and its value is greater than or equal to 1000 then the actual "anytime" timeout is the minimum of the requested value and the value in the configuration file.

The current standard of SPARQL protocol does not provide any support for partial results. When a SPARQL query ends with S1TAT state, the returned HTTP header contains four additional rows, *X-SQL-State:* , *X-SQL-Message:* , *X-Exec-Milliseconds:* and *X-Exec-DB-Activity:* ; the returned document is formatted according to the requested MIME type as if the result is complete.

9.26. Best Effort Union

Virtuoso offers a SQL extension for an error tolerant UNION operation. The idea is that when querying multiple remote data sources in a single union construct some of the participating data sources may be allowed to fail while still returning a result for the successfully queried data sources.

The construct is introduced by the BEST keyword before UNION or UNION ALL. If a query expression of multiple unions has a single BEST keyword the entire union chain is considered as a best effort union. It is however recommended to have the BEST keyword in FROM of all the UNION keywords.

When a run time error occurs during the evaluation of a term in a best effort union the evaluation of the term is interrupted and the union continues with the next term. The partial result set that may have been generated by the failed term is considered when making the result.

Aliasing constant columns selected in the terms of the union by the names __SQLSTATE, __MESSAGE and __SET_NO retrieve individual error messages. If these are present and a union term encounters an error an extra row is generated for the term with all NULLs and the __SQLSTATE, __MESSAGE and __SET_NO columns set to the SQL state, SQL message and the union term number respectively. If neither of these is specified and a union term fails without producing any result rows the error will not be visible.

Note

No error encountered during a best effort union will be signalled in the normal fashion.

If a term of a best effort union meets the criteria for a pass through query on a specific remote database and a transaction error occurs when evaluating it, the transaction on the VDB is not aborted as would normally happen as a result of a VDB transaction error.

Thus if a pass through term dies of deadlock on its data source the query continues normally for other data sources referenced in the best effort union. The VDB will however get the transaction error when attempting to commit the transaction where the best effort union took place since the remote transaction branch will still be deadlocked.

Example 9.42. Examples

```
select 2222, 1 / 0 from sys_users best union all select key_id, 1 / (1000 - key_id)  from sys_keys;
```

The first term will immediately hit the /0 error and will produce no rows. The second term will produce a few rows for system tables but will hit /0 when getting to key_id 1001.

To see the errors one can write:

```
select '00000' as __sqlstate, '' as __message, 2222, 1 / 0 from sys_users best union
all select '00000' as __sqlstate, '', key_id, 1 / (1001 - key_id)  from sys_keys;
```

Note

The columns are named by the first term, hence the AS declaration in the second term is optional.

The BEST keyword does not affect the ALL or CORRESPONDING BY options of UNION.

9.27. Standard and User-Defined Aggregate Functions

In addition to whole set of standard SQL aggregate functions, Virtuoso provides a method to create application-specific aggregate functions to create complex data structures inside SQL queries.

The most evident way to calculate some value that depend on number of rows of a table is to write a stored procedure that opens a cursor, then fetches row after row in a loop and repeatedly modifies some intermediate values ("accumulators") inside the loop. When the fetch operation signals that there are no more data, a final result is calculated from values of "accumulators"

E.g. to find an average of all values in a table's column COL, one may open a cursor to fetch values from the COL, then set two "accumulators" TOTAL and CNT to zero, then fetch row after row adding 1 to CNT and adding current value of COL to TOTAL. At the end of loop, an error should be signalled if CNT is zero, otherwise the result is TOTAL divided by CNT.

Obviously, a competent programmer will use built-in AVG() aggregate function inside a single SELECT statement: the code is much more readable and the use of aggregate reduces the overhead in filling in result-set and fetching from it.

More important advantage of aggregate functions is the ability to process many groups of records in parallel, performing one table scan instead of many. SQL optimizer may use sophisticated heuristics to find the fastest way of doing a complex query with aggregates but it cannot optimize the code of a stored procedure.

9.27.1. Create Aggregate Statement

```
CREATE AGGREGATE aggregate_name (parameter, parameter ...) [ RETURNS data_type ]
FROM
    init_procedure,
    acc_procedure,
    final_function
    [, merge_procedure ]

parameter : parameter_type name data_type

parameter_type : IN | INOUT
```

The create aggregate statement forces the SQL compiler to treat any call of the declared aggregate_name as special routine similar to one used for built-in aggregate functions, so the call produces a single value for resultset instead of a separate value for every row of the resultset.

Every occurrence of the aggregate_name in a select statement creates an invisible "accumulator" column in resultset. The value of "accumulator" is calculated by init_procedure, then every row of data causes a call of acc_procedure that may modify the "accumulator" then the call of final_function should get data from "accumulator" and return the total result.

The init_procedure must have one inout parameter and return nothing. It should set the parameter to the required initial value of the "accumulator". (The SQL query compiler will use the declared data type of the parameter as the data type of the "accumulator" so no casting problem may occur) equal to

The acc_procedure must have at least one inout parameter, one or more parameters of any sort and must return nothing. It should get the previously calculated value of the "accumulator" from its first parameter, calculate a new value and set the result back to the first parameter.

The final_function must have one in or inout parameter and return a value. It should get the value of the "accumulator" from its parameter and return the calculated result.

The optional merge_procedure must have two inout parameters that accept values of two "accumulators". If defined, it may be used by SQL optimizer to split the table scan into two or more scans through parts of the table. When every 'children' scan iterates some part of the table, a separate "accumulators" is used for every part. merge_procedure gets two accumulators and adds the content of the second accumulator to the content of the first. If merge_procedure is not defined, no such tricky optimization is used. This optimization is never tried if the database resides on less than four disks, if the number of CPUs is less than four and if the query contains any call of user-defined aggregate with no merge_procedure defined; thus you will probably never need to

write such a function.

The declaration of aggregate is persistent. If create aggregate statement is successful the text is stored into the SYS_PROCEDURES table. This table is read at startup so user-defined aggregates are thus always available for use and need be defined only once.

9.27.2. Drop Aggregate Statement

```
DROP AGGREGATE aggregate_name
```

The drop aggregate statement removes the declaration of aggregate_name from SYS_PROCEDURES table but does not drop or change any declarations of functions previously listed in create aggregate statement.

The syntax of the 'call' operator is the same for 'plain' and 'aggregate' functions. If an aggregate name is used in the text of stored procedure before the aggregate is actually created then it is compiled as plain function call and causes run-time errors. In order to prevent such confusion, keyword AGGREGATE may be placed right before the name of aggregate function.

9.27.3. Examples of User-Defined Aggregates

Aggregate for Composing URIs

Consider a table USER_ATTRIBUTES that contains configuration data for users of some remote web-service. Every user have a number of parameters that should be passed to that service as a part of URI when HTTP PUT is used. Every row of USER_ATTRIBUTES contain a name and a value of some parameter for some type of users.

```
create table USER_ATTRIBUTES    (
        ID              integer not null primary key,
        ATTR_NAME       varchar,
        USER_TYPE       char(4),
        ATTR_VALUE      varchar );

insert into USER_ATTRIBUTES (ID, ATTR_NAME, USER_TYPE, ATTR_VALUE) values (1, 'key1', 't1', 'val1');
insert into USER_ATTRIBUTES (ID, ATTR_NAME, USER_TYPE, ATTR_VALUE) values (2, 'key3', 't1', 'val3');
insert into USER_ATTRIBUTES (ID, ATTR_NAME, USER_TYPE, ATTR_VALUE) values (3, 'key4', 't1', 'val4');
insert into USER_ATTRIBUTES (ID, ATTR_NAME, USER_TYPE, ATTR_VALUE) values (4, 'key5', 't2', 'val5');
insert into USER_ATTRIBUTES (ID, ATTR_NAME, USER_TYPE, ATTR_VALUE) values (5, 'key6', 't2', 'val6');
insert into USER_ATTRIBUTES (ID, ATTR_NAME, USER_TYPE, ATTR_VALUE) values (6, 'key2', 't1', 'val2');
insert into USER_ATTRIBUTES (ID, ATTR_NAME, USER_TYPE, ATTR_VALUE) values (7, 'key8', 't3', 'val8');
insert into USER_ATTRIBUTES (ID, ATTR_NAME, USER_TYPE, ATTR_VALUE) values (8, 'key7', 't2', 'val7');
insert into USER_ATTRIBUTES (ID, ATTR_NAME, USER_TYPE, ATTR_VALUE) values (9, 'key9', 't4', 'val9');
```

The application contains number of places where the resulting URI for a user should be calculated. The URI is either an empty string (if no attributes are set) or a string that starts from '?' char and consists of tokens like "attribute_name=attribute_value" delimited by "&" character.

```
create function ATTR_URI_INIT (inout _env any)
{
  _env := string_output(); -- The "accumulator" is a string session. Initially it is empty.
}

create function ATTR_URI_ACC (
  inout _env any,               -- The first parameter is used for passing "accumulator" value.
  in _attrname varchar,         -- Second parameter gets the value passed by first parameter of aggregate
  in _attrvalue varchar )       -- Third parameter gets the value passed by second parameter of aggregate
{
  if (_attrname is null)        -- Attributes with NULL names should not affect the result.
    return;
  -- The non-NULL name of attribute is appended to "accumulator"
  http ('&', _env);
  http_url (_attrname, null, _env);
  http ('=', _env);
  if (_attrvalue is null)       -- NULL values of attributes are not printed.
    return;
  -- The non-null value of attribute is appended to "accumulator"
  http_url (_attrvalue, null, _env);
}
```

```
create function ATTR_URI_FINAL (inout _env any) returns varchar
{
  declare _res varchar;
-- For empty groups in the resultset,  the initialization may be bypassed
-- so the value of "accumulator" remains NULL.
-- Finalization function should always handle such case
-- by signalling an error (like built-in AVG function) or
-- by returning some default value (like this function)
  if (_env is null)
    return '';
  _res := string_output_string (_env);
  if (_res = '')
    return '';
-- In nonempty lists of URI attributes, the first '&' should be replace with '?'
  aset (_res, 0, 63);
  return _res;
}

create aggregate ATTR_URI (in _key varchar, in _val varchar) returns varchar
  from ATTR_URI_INIT, ATTR_URI_ACC, ATTR_URI_FINAL;
```

Now we are ready to run some tests. Let's try the typical case:

```
select ATTR_URI (ATTR_NAME, ATTR_VALUE) as URI from USER_ATTRIBUTES where USER_TYPE='t1';
URI
VARCHAR
_______________________________________________________________________________
?key1=val1&key2=val2&key3=val3&key4=val4
1 Rows. -- 0 msec.
```

Now let's try the case of empty set of rows to aggregate:

```
select ATTR_URI (ATTR_NAME, ATTR_VALUE) as URI from USER_ATTRIBUTES where USER_TYPE='nosuchtype';
URI
VARCHAR
_______________________________________________________________________________

1 Rows. -- 0 msec.
```

OK, we've got an empty string. Now let's try the final case:

```
select ATTR_URI (ATTR_NAME, NULL) as URI from USER_ATTRIBUTES;
URI
VARCHAR
_______________________________________________________________________________
?key1=&key2=&key3=&key4=&key5=&key6=&key7=&key8=&key9=
1 Rows. -- 0 msec.
```

The created aggregate is very fast but unusable if intermediate results should be saved in a temporary table. It may happen if the SQL statement contains clauses such as DISTINCT, ORDER BY, GROUP BY, especially if the grouping columns are not the first columns of the primary key, or if SQL optimizer thinks that it is the fastest way to calculate a join. The following version produces the same results and may work with grouping, but may be a bit slower because concatenation of long strings is slower than writing to a string session:

```
create procedure ATTR_URI2_INIT (inout _env varchar)
{
  _env := ''; -- The "accumulator" is a string. Initially it is empty.
}

create procedure ATTR_URI2_ACC (
  inout _env varchar,  in _attrname varchar,  in _attrvalue varchar ){
  if (_attrname is null)-- Attributes with NULL names should not affect the result.
    return;
  if (_attrvalue is null)-- NULL values of attributes are not printed.
    _env := concat (_env, sprintf('&amp;%U', _attrname));
  else -- The non-null value of attribute is appended to "accumulator"
    _env := concat (_env, sprintf('&amp;%U=%U', _attrname, _attrvalue));
}
```

```
create function ATTR_URI2_FINAL (inout _env any) returns varchar
{
-- For empty groups in the result set,  the initialization may be bypassed
-- so the value of "accumulator" remains NULL.
-- Finalization function should always handle such case
-- by signaling an error (like built-in AVG function) or
-- by returning some default value (like this function)
  if (_env is null)
    return '';
  if (_env = '')
    return '';
-- In nonempty lists of URI attributes, the first '&amp;' should be replace with '?'
  aset (_env, 0, 63);
  return _env;
}

create aggregate ATTR_URI2 (in _key varchar, in _val varchar) returns varchar
  from ATTR_URI2_INIT, ATTR_URI2_ACC, ATTR_URI2_FINAL;

select ATTR_URI2 (ATTR_NAME, ATTR_VALUE) as URI from USER_ATTRIBUTES where USER_TYPE='t1'
URI
VARCHAR
_______________________________________________________________________________
?key1=val1&key3=val3&key4=val4&key2=val2
1 Rows. -- 0 msec.

select USER_TYPE, ATTR_URI2 (ATTR_NAME, ATTR_VALUE) as URI from USER_ATTRIBUTES group by USER_TYPE
USER_TYPE  URI
VARCHAR  VARCHAR
_______________________________________________________________________________
t2       ?key5=val5&key6=val6&key7=val7
t1       ?key1=val1&key3=val3&key4=val4&key2=val2
t4       ?key9=val9
t3       ?key8=val8
4 Rows. -- 0 msec.
```

Aggregate for Composing XMLs

Virtuoso offers variety of ways to place relational data into XML documents, and two best of them are select statements with FOR XML clause and XML views. But in some rare cases you may prefer to compose XML as an output of plain select statement, e.g. to get a few percent faster code.

Among other things, the CREATE XML VIEW statement creates set of Virtuoso/PL functions that return various XML trees. Those functions use special 'node building' vectors to store intermediate results. NODEBLD (which stands for 'Node Building vector') stores a partially built XML tree. A xte_nodebld() function returns a new empty NODEBLD that may be converted into an element of XML tree in the future. xte_nodebld_acc() appends new children to an existing NODEBLD. xte_node_from_nodebld() converts NODEBLD into ready-to-use XML tree. xte_nodebld_final() may work either as xte_node_from_nodebld() or as a finalization function of user defined aggregate.

```
create aggregate XTE_NODEBLD (in _child any) returns any
  from xte_nodebld_init, xte_nodebld_acc, xte_nodebld_final;
```

The use of this aggregate is not as easy as creating XML views, but it avoids creating persistent objects:

```
select
  xml_tree_doc (
    xte_node_from_nodebld (
      xte_head ('UserAttributes'),
      XTE_NODEBLD (
        xte_node (xte_head ('Row', 'Id', ID, 'UserType', USER_TYPE),
          xte_node (xte_head ('Attr', 'Name', ATTR_NAME),
            ATTR_VALUE ) )
      ) ) )
  from USER_ATTRIBUTES;
callret
VARCHAR
_______________________________________________________________________________
<UserAttributes>
<Row
xml:id="1" UserType="t1"><Attr Name="key1">val1</Attr></Row>
```

```
<Row
xml:id="2" UserType="t1"><Attr Name="key2">val2</Attr></Row>
<Row
xml:id="3" UserType="t1"><Attr Name="key3">val3</Attr></Row>
<Row
xml:id="4" UserType="t1"><Attr Name="key4">val4</Attr></Row>
<Row
xml:id="5" UserType="t2"><Attr Name="key5">val5</Attr></Row>
<Row
xml:id="6" UserType="t2"><Attr Name="key6">val6</Attr></Row>
<Row
xml:id="7" UserType="t2"><Attr Name="key7">val7</Attr></Row>
<Row
xml:id="8" UserType="t3"><Attr Name="key8">val8</Attr></Row>
<Row
xml:id="9" UserType="t4"><Attr Name="key9">val9</Attr></Row>
</UserAttributes>
1 Rows. -- 0 msec.
```

The function xml_tree_doc() is used because there is no standard way to print an XML tree without making a complete XML entity whose underlying XML document consists of the given XML tree.

```
select
  xml_tree_doc (
    xte_node_from_nodebld (
      xte_head ('AttrStatistics'),
      XTE_NODEBLD (
        xte_node (xte_head ('Row', 'UserType', groups.type, 'AttrCount', groups.cnt))
  ) ) )
  from
    (select USER_TYPE as type, COUNT (*) as cnt from USER_ATTRIBUTES group by USER_TYPE) groups;
callret
VARCHAR
_______________________________________________________________________________

<AttrStatistics>
<Row UserType="t2" AttrCount="3" />
<Row UserType="t1" AttrCount="4" />
<Row UserType="t4" AttrCount="1" />
<Row UserType="t3" AttrCount="1" />
</AttrStatistics>
1 Rows. -- 0 msec.
```

9.28. Virtuoso SQL Optimization

Virtuoso provides a cost based SQL optimizer which performs the following types of query transformation:

Join Order

Loop Invariants

Opening derived tables

Migrating enclosing predicates into derived tables or unions

Dropping unreferenced columns or results

Detection of identically false predicates

Index selection

Grouping of co-located remote tables into single remote statements

Selection of join algorithm

Virtuoso evaluates various permutations of joined tables against its cost model and determines the best fit, from which it generates a query graph. This query graph can be returned as a result set by the `explain()` SQL function. The cost model is based on table row counts, defined indices and uniqueness constraints, and column cardinalities, i.e. counts of distinct values in columns. Additionally, histograms can be made for value distribution of individual columns.

Virtuoso automatically maintains statistics about tables in the local database. When tables are attached from known types of remote DBMS's, Virtuoso also attempts to retrieve statistics information if available. The sys_stat_analyze or sys_db_stat stored procedures can be used to force an update of statistics, also recompiling all SQL statements or procedures depending on these statistics. Once this is done, this overrides the automatic statistics. The values of automatic statistics can be seen in the

SYS_COL_AUTO_STAT table.

The stored procedure:

```
DB.DBA.SYS_STAT_ANALYZE (in full_table_name varchar, in prec integer, )
```

constructs the basic table statistics and feeds it into the DB.DBA.SYS_COL_STAT system table. The DB.DBA.SYS_DB_STAT stored procedure performs this operation on the entire database.

The stored procedure:

```
DB.DBA.SYS_STAT_HISTOGRAM (in full_table_name varchar, in full_column_name varchar, in
n_buckets integer, in prec integer, )
```

constructs table column histogram and feeds it into the DB.DBA.SYS_COL_HIST system table. The default value of prec, in both cases, is 5, which implies that a five percent sample of the table will be used. A percentage of 0 means that the whole table will be read.

Example 9.43. Demonstration of the STAT_ANALYSE & STAT_HISTOGRAM Procedures

The following script is intended for use with the ISQL program as the user dba, in the DB qualifier. The foreach statement is a special feature of the ISQL utility.

```
create table "DB"."DBA"."DTTEST" ("ID" integer primary key);
foreach integer between 1 10 insert into "DB"."DBA"."DTTEST" ("ID") values (?);
sys_stat_analyze ('DB.DBA.DTTEST');
select * from DB.DBA.SYS_COL_STAT;
sys_stat_histogram ('DB.DBA.DTTEST', 'ID', 2);
select * from DB.DBA.SYS_COL_HIST;
```

That yields:

```
Resultset 1 (from DB.DBA.SYS_COL_STAT)
---------------------------------------
CS_TABLE         CS_COL           CS_N_DISTINCT CS_MIN  CS_MAX  CS_N_VALUES CS_N_ROWS
VARCHAR NOT NULL VARCHAR NOT NULL INTEGER       VARCHAR VARCHAR INTEGER     INTEGER
_______________________________________________________________________________________

DB.DBA.DTTEST    ID               10            1       10      10          10

Resultset 2 (from DB.DBA.SYS_COL_HIST)
---------------------------------------
CH_TABLE          CH_COL            CH_NTH_SAMPLE     CH_VALUE
VARCHAR NOT NULL  VARCHAR NOT NULL  INTEGER NOT NULL  VARCHAR
_______________________________________________________________________________

DB.DBA.DTTEST     id                0                 1
DB.DBA.DTTEST     id                5                 6
DB.DBA.DTTEST     id                10                0
```

9.28.1. Optimization Techniques

Join Order

The SQL compiler constructs permutations of tables in the FROM clause of a query. The qualified inner join (x inner join y on p syntax) does not dictate a join order but the outer join syntax does. Otherwise join order is subject to change by the compiler's decision. This can be disabled with the OPTION (ORDER) global query option, see sections below.

The first order tried by the optimizer is the initial left to right order of the FROM clause. The compiler remembers its best result so far attained in trying various compilations, so it will not explore branches of the table permutation tree which, while incomplete, exceed the cost of the best complete result so far. Thus a good guess of initial table order by the programmer may save compilation time.

Loop Invariants

The compiler will evaluate all expressions and predicates as early as possible.

Consider the query:

```
select count (*) from item where i_price > (select avg (i_price) from item);
```

The compiler will detect that the sub query is data-independent from the main from and thus will compute the average price before starting the select for the count. Thus the above query executes in time linear to the item table count instead of quadratic time, as it would without the removal of the invariant.

Opening Derived Tables & Views

Views are initially opened into equivalent derived tables. If a derived table in a FROM clause is sufficiently simple, i.e. has no distinct, top, group by, and is not a multiple query expression such as a union, it can be in-lined.

Example 9.44. Derived Tables & Views

```
create view i as select * from item;

select i_id from i;
```

becomes

```
select i_id from (select * from item) xx;
```

which becomes

```
select i_id from item;
```

In joins involving views representing joins, the in-lining of a derived table will add degrees of freedom to join order selection, since the tables joined in the view will not have to be laid out contiguously.

Migrating Enclosing Predicates

When there is a derived table or view expansion which cannot be in-lined, i.e. it has a group by or is a union or such, predicates of the enclosing WHERE phrase are migrated into the derived table itself. The set of predicates that may be thus migrated will be a function of the join order, thus different combinations will be tried.

Example 9.45. Example of Migrating Predicates

```
select i_id from (select * from item1 union all select * from item2) f where i_id = 11;
```

becomes

```
select i_id from (select i_id from item1
  where i_id = 11 union all select i_id from item2 where i_id = 11) f;
```

Dropping Unreferenced Columns or Results

It may happen as a result of view expansion that columns get introduced into derived table selection which are nowhere referenced in the actual query. The compiler will detect this and will not compile code accessing these. (see above for example)

Detection of Identically False Predicates

It is possible that view expansions introduce predicates that are never simultaneously true with predicates of the enclosing query. This is recognized and can result in an empty query being produced or in union terms being dropped.

The rules for combining predicates are as follows:

```
- a < b and a < c -> a < min (a, b)
```

And similarly for <=, >, and >=.

```
a in (constant1) and a in (constant2)
```

becomes identically false if the intersection of the constant lists is empty.

```
- a = b and a = c
```

becomes identically false if b and c are different constants.

This can lead to transformations such as

```
select * from (select * from item where i_type = 2 union all
  select * from item where i_type = 3) xx where i_type = 2;
```

becomes

```
select * from (select * from item where i_type = 2 ) xx where i_type = 2;
```

Index Selection

Once a join order is hypothesized, the compiler picks various indices to use for accessing tables. The predicates applicable to the table in the specific join order being considered, as well as a possible ORDER BY clause affect the index choice. If there is an ORDER BY and there is an index which can be used to directly satisfy this, the compiler tries this index as well as the index which seems best in the light of the available predicates. Thus having an applicable ORDER BY does not always imply index selection. If there is a restrictive index for row selection and an order by which could be done by following an index for which there are no predicates, the restrictive index plus sorting will be preferred.

The general rules for index selection based on predicates are:

Primary key is preferred if 2 equally good choices exist.

Equality conditions are preferred over all others.

Uniqueness of an index is a plus.

Grouping Co-Located Tables

In queries involving distributed data the principal cost factor is often the RPC latency between the participating databases. In a local area network environment with no other load we estimate one RPC to be about 5 times longer than the selection of a single row on primary key from a table of 1 million, not counting disk I/O. In wide area contexts and with effective load the difference is still more marked. Thus when considering a loop join between a local and a remote table, it becomes obvious that the remote table should be the outer loop, unless there are extremely restrictive criteria on the local table. The cost model takes such considerations into account.

The best case of remote statement compilation is when the statement can be passed through as is to the remote. This is first checked for the whole query and subsequently for each sub query or derived table. The pass-through is not possible if:

The expression involves tables from multiple remotes or local and remote ones.

The expression involves predicates or functions or SQL constructs which do not exist on the remote.

Even if a derived table or sub query can be passed through the normal SQL optimization applies, thus Virtuoso will import predicates and suggest a join order for these cases, so that the query gets rewritten with the suggested join order left to right in the text.

If one join has multiple tables from the same remote but also other tables, the optimizer will attempt to group the co-located tables together.

```
select * from r1 join t1 on r1.k = t1.k join r2 on r1.k = r2.k;
```

will preferentially be compiled as

```
select * from (select r1.k as r1_k, ... from r1, r2
```

```
where r1.k = r2.k) xx, t1 where r1_k = t1.k;
```

Note that above the join order is meant left to right and that the derived table of r1, r2 is passed through as a unit. This becomes the leading loop in the join because fewer RPC's are likely to be needed then due to row prefetch. As for placing function calls, the general rule is to place them where the arguments are, thus passed through to the remote if they are defined there. On the other hand, if these are invariant they are always computed locally. The explain function can be used to see the details of the compilation. If a function is described as being defined on the particular remote database, this information is taken into account when compiling.

Join Type Selection

Starting with Virtuoso 3.0, hash joins are used where appropriate. A hash join is about three to four times faster for a local table than with a loop join using an exact match of primary key, thus the gain is substantial.

A hash join works by scanning the shorter of two joined tables and making a hash table from the join columns. This is only possible for joins involving equality, but most joins tend to have equalities in practice.

Note:

A hash join does not need indices to be defined and the hash table constructed for one hash join may be reused by another if the table is not modified in-between.

A hash join will be preferred if there is a sufficient number of estimated accesses to the joined table in the query. If a table is expected to be accessed only once, there is less point in making the temporary hash table.

The hash join temporary structures are disk based and reside in the temporary side of the database. They do not survive a database restart but will be ad hoc reused if such are left over from earlier queries and there is no change in the base table between the construction of the hash and its reuse.

A hash join may be specially beneficial if a remote table is on the inner loop of a loop join. The n random accesses can then be replaced with a single sequential scan of the remote table.

The selected join type can be seen in the explain output. The WITH keyword can be used to explicitly specify a join type for a table, see query options below.

9.28.2. Query Options

These are effective from Virtuoso 3.0 onwards.

```
<sql_option>:
        TABLE OPTION (<table option>,...)

<table option>:
        HASH | LOOP | INDEX index_name | INDEX PRIMARY KEY | INDEX TEXT KEY

<option clause>:
        OPTION (<query_option>,...)

<query_option>:
        ORDER
```

The TABLE OPTION clause can follow a table's correlation name in a FROM clause. The OPTION clause can appear at the end of the query, after all ORDER BY and FOR specifications.

The INDEX table option is used to control which index is chosen by the optimized SQL compiler for a given table (when you want to override the one chosen automatically).

INDEX index_name - selects a named table index (unprefixed - same name as in create index).

INDEX PRIMARY KEY - selects the primary key

INDEX TEXT KEY - selects the free-text index as a driving one if there's a free-text condition over that table in the statement.

Example 9.46. Use of the OPTION & TABLE OPTION clauses

```
select count (*) from t1 a, t1 b table option  (hash) where .row_no = b.row_no option(order)
```

This forces the a, b join order and specifies that the join will be a hash join.

Example 9.47. IN Predicate and Joins

Sometimes statements of the form

```
select col from t1 where t1.key in (select key from t2)
```

get optimized into a loop over t2 and a nested lookup on t1, as in

```
select .... from (select distinct key from t2) f, t1 where t1.key = f.key.
```

This optimization is mostly done when there is an index on t1.key and the IN subquery selects fewer rows than the other conditions on t1 would select.

In some cases the developer may wish to explicitly force this optimization to be made or not to be made. This is done with the LOOP EXISTS query option, as follows:

```
select row_no from t1 where row_no in (select row_no from t1 where fi2 = 2) option (loop exists)
```

will force the IN subquery to become a joined derived table.

```
select row_no from t1 where row_no in (select row_no from t1 where fi2 = 2) option (do not loop exists)

select row_no from t1 where row_no in (select row_no from t1 where fi2 = 2 option (do not loop exists))
```

will prevent this optimization. Note that the option (do not loop exists) can be either in the subquery or in the enclosing query.

The same logic applies to IN predicates in searched update or delete statements.

9.28.3. Query Optimization Diagnostics

Queries involving a large number of possible plans may run out of memory during optimization. There are a number of settings that influence query optimization memory utilization.

These are set in the virtuoso.ini file or can be altered on a running system with `__dbf_set` function:

Table 9.4.

INI section	INI parameter name	__dbf_set function name	__dbf_set function description
Parameters	MaxMemPoolSize	*sqlo_max_mp_size*	Controlls the size limit in bytes for transient memory consumption. Increasing this may help. The given value should be over 10M, increasing this over 100M is seldom useful but can be tried.
Parameters	MaxOptimizeLayouts	*sqlo_max_layouts*	Decreasing will reduce the number of plans tried, hence save memory. Reasonable values are 0 for no limit or somewhere in excess of 500 for a limit.
Parameters	StopCompilerWhenXOverRunTime	*sqlo_compiler_exceeds_run_factor*	Setting to 1 will stop optimization once the best plan is expected to take less time and the amount of time spent optimizing so far.
Flag	enable_joins_only	*enable_joins_only*	When set, will cause the optimizer to only consider next plan candidates that are connected by a join to the existing partial plan. In other words, no cartesian products

INI section	INI parameter name	__dbf_set function name	__dbf_set function description
			will be considered. This may save some space and time.

When reporting issues with query optimization it will be useful to include statistics from the database in order to facilitate reproducing the effect. The function `stat_export()` produces a statistics summary that can be read back into another database with the `stat_import()` function.

To export statistics:

```
string_to_file('stat.dv', serialize(stat_export()), -2);
```

To load exported statistics into a database:

```
stat_import (deserialize (file_to_string ('stat.dv')));
```

When exporting statistics as part of bug reporting, make sure to first run the queries exhibiting the problem then only export the stats. The queries drive statistics gathering.

9.28.4. ANY ORDER

When applied to a select with no aggregation or order by, this causes the select to produce results in an order that may vary between consecutive executions and may not correspond to the order of any index. In a cluster situation, running a query in this manner may be up to several times more efficient. This is not the default since SQL and SPARQL require that two consecutive executions of a query return the same results in the same order even if no order by is specified. Selects that contain aggregation or order by evaluate the part which generates input to the aggregation or order in this manner automatically. This option affects the select at the end of which it occurs and all selects inside it.

example:

```
select a.row_no from t1 a, t1 b wherea.row_no = 1 + b.row_no option (any order);
```

9.28.5. VDB Statistics for the SQL Compiler Collection

In order to correctly estimate the cost of the VDB operations overhead the optimized SQL compiler should have information for the time it takes to make a "round trip" - send message and receive response from a given remote database.

Virtuoso will automatically collect such information when you first attach a table from the remote data source. The information will be cached in the `DS_CONN_STR` field of the `SYS_DATA_SOURCE` system table.

Sometimes it may be desirable to update that values manually, for example when you change the network connection etc. Virtuoso provides, for the purpose, the stored procedure:

```
vd_statistics (in _dsn varchar := '%',in vd_table_mask varchar := '%')
```

The *_dsn* parameter is a LIKE-mask for the name of the DSN as stored in DS_DSN column of SYS_DATA_SOURCE system table. Its default value is '%' having the effect to update the "round trip" times for all the remote data sources.

The *vd_table_mask* parameter is a LIKE mask for the name of the table , as stored in RT_NAME column of SYS_DATA_SOURCE system table. Its default value of '%' means update all tables.

The current round trip time can be recalculated using the function:

```
vdd_measure_rpc_time (in _dsn varchar)
```

This will return the estimated round-trip time in milliseconds. Calling this function will not alter the cached statistics for the datasource.

Before collecting statistics data by querying the (potentially very large) remote tables the Virtuoso VDB will try to collect the statistics in various less resource extensive ways.

Generally speaking some of the statistics the SQL compiler needs in order to do cost based optimization are available through one of the following channels (in order of preference):

by querying the remote DBMS statistics system tables: This will provide a varying amount of information depending on the remote and the mapping function.

through SQLStatistics() ODBC call CARDINALITY column : A platform/remote independent way. This will provide information for the row count only.

by querying the remote table to collect various aggregate functions values (like COUNT (*), AVG() etc) : most complete, but may be very slow for large remote tables.

Unfortunately the first method is DBMS dependent. Therefore some extra setup is to be done before this can return meaningful data. This is supported off the shelf for Oracle and Virtuoso.

The `SYS_STAT_ANALYZE()` Virtuoso function will do a lookup of the remote DBMS name and version (as returned by SQL_DBMS_NAME and SQL_DBMS_VER SQLGetInfo () ODBC call) in the system table DB.DBA.SYS_STAT_VDB_MAPPERS to find a Virtuoso/PL stored procedure to call for that DBMS to retrieve the statistics data from the remote DBMS'es system tables in DBMS dependent way.

The DB.DBA.SYS_STAT_VDB_MAPPERS has the following layout:

```
create table SYS_STAT_VDB_MAPPERS (
   SVDM_TYPE varchar,
   SVDM_PROC varchar not null,
   SVDM_DBMS_NAME_MASK varchar not null,
   SVDM_DBMS_VER_MASK varchar not null,
   primary key (SVDM_TYPE, SVDM_DBMS_NAME_MASK, SVDM_DBMS_VER_MASK))
```

SVDM_TYPE is the type of the statistics returned by the procedure. Currently only 'SYS_COL_STAT' is supported.

SVDM_PROC is the Fully-qualified-name of the Virtuoso/PL stored procedure to be called.

SVDM_DBMS_NAME_MASK is the LIKE kind of mask used to check the SQL_DBMS_NAME SQLGetInfo() returned string (in uppercase).

SQL_DBMS_VER_MASK is the LIKE kind of mask use to check the SQL_DBMS_VER SQLGetInfo() returned string (in uppercase)

The `SYS_STAT_ANALYZE()` will use the first matching row in the order of the primary key.

The procedure that collects the remote DBMS statistics (usually by doing rexecute() against the remote) has the following signature:

```
create procedure DB.DBA.__VIRTUOSO_SYS_COL_STAT (
    in DSN varchar,
    in RT_NAME varchar,
    in RT_REMOTE_NAME varchar
)
```

Here:

DSN is the Virtuoso dsn name.

RT_NAME is the local Virtuoso table name of the attached table (as in the RT_NAME of SYS_REMOTE_TABLE system table).

RT_REMOTE_NAME is the name of the attached table in the remote DBMS (as in RT_REMOTE_NAME of SYS_REMOTE_TABLE system table).

The procedure may return a resultset that will be written into the SYS_COL_STAT system table. The resultset has to have a row for each attached table column and the following layout:

CS_COL - this is the column name

CS_N_DISTINCT - the number of distinct non-null values in that column

CS_MIN - minimum value

CS_MAX - maximum value

CS_AVG_LEN - average data length

CS_N_VALUES - number of non-null values

CS_N_ROWS - total number of rows

Virtuoso predefines two such procedures - one for another virtuoso as a remote and the second for an Oracle DBMS. Other DBMSes can be freely added.

```
create procedure DB.DBA.__ORACLE_SYS_COL_STAT (in DSN varchar, in RT_NAME
varchar, in RT_REMOTE_NAME varchar)
returns ANY
{
  declare _meta, _res any;

  rexecute (DSN,
    'select c.COLUMN_NAME, c.NUM_DISTINCT, NULL, NULL, c.AVG_COL_LEN, t.NUM_ROWS - c.NUM_NULLS, t.NUM_ROW
    ' from ALL_TABLES t, ALL_TAB_COLUMNS c where t.TABLE_NAME = c.TABLE_NAME and t.OWNER = c.OWNER and '
    '  t.OWNER = ? and t.TABLE_NAME = ?',
    NULL, NULL, vector (name_part (RT_REMOTE_NAME, 1, NULL), name_part(RT_REMOTE_NAME, 2, NULL)), NULL, _

  if (isarray (_res) and length (_res) > 0 and isarray (_res[0]) and isarray(_meta) and isarray (_meta[0]
    {
      declare _inx, _len integer;
      _inx := 0;
      _len := length (_res);
      exec_result_names (_meta[0]);
      while (_inx < _len)
        {
          exec_result (_res[_inx]);
          _inx := _inx + 1;
        }
    }
  return NULL;
};

create procedure DB.DBA.__VIRTUOSO_SYS_COL_STAT (in DSN varchar, in RT_NAME
varchar, in RT_REMOTE_NAME varchar)
returns ANY
{
  declare _meta, _res any;

  rexecute (DSN,
    'select CS_COL, CS_N_DISTINCT, encode_base64 (serialize (CS_MIN)), encode_base64 (serialize (CS_MAX))
    ' CS_AVG_LEN, CS_N_VALUES, CS_N_ROWS from ALL_COL_STAT where CS_TABLE = complete_table_name (?, 1)',
    NULL, NULL, vector (RT_REMOTE_NAME), NULL, _meta, _res);

  if (isarray (_res) and length (_res) > 0 and isarray (_res[0]) and isarray(_meta) and isarray (_meta[0]
    {
      declare _inx, _len integer;
      _inx := 0;
      _len := length (_res);
      exec_result_names (_meta[0]);
      while (_inx < _len)
        {
          declare _res_row any;
          _res_row := _res[_inx];
          _res_row[2] := deserialize (decode_base64 (_res_row[2]));
          _res_row[3] := deserialize (decode_base64 (_res_row[3]));
          exec_result (_res_row);
          _inx := _inx + 1;
        }
    }
  return NULL;
};
```

And here are the respective registration INSERT statements for the above procedures:

```
insert soft SYS_STAT_VDB_MAPPERS (SVDM_TYPE, SVDM_PROC, SVDM_DBMS_NAME_MASK, SVDM_DBMS_VER_MASK)
 values ('SYS_COL_STAT', 'DB.DBA.__ORACLE_SYS_COL_STAT', '%ORACLE%', '%');

insert soft SYS_STAT_VDB_MAPPERS (SVDM_TYPE, SVDM_PROC, SVDM_DBMS_NAME_MASK, SVDM_DBMS_VER_MASK)
```

```
values ('SYS_COL_STAT', 'DB.DBA.__VIRTUOSO_SYS_COL_STAT', '%VIRTUOSO%', '%');
```

In order to facilitate the access to statistics in Virtuoso the following four views are created with SELECT granted to PUBLIC:

ALL_COL_STAT - subset of SYS_COL_STAT (same row layout) but only returning data for the tables the user has access to.
USER_COL_STAT - subset of SYS_COL_STAT (same row layout) but returns rows only for the user-owned tables
ALL_COL_HIST - subset of SYS_COL_HIST (same row layout) but only returning data for the tables the user has access to.
USER_COL_HIST - subset of SYS_COL_HIST (same row layout) but returns rows only for the user-owned tables

9.29. SQL Inverse Functions

Many virtual database application scenarios require performing various conversions on data in legacy tables. An example of this is currency conversions, conversions between British and metric units, replacing complex composite keys with single numeric row id's and so forth.

SQL views are the normal way of implementing such conversions. However, when making queries with selection criteria referencing the results of these conversions we often need to first convert and then test, possibly having to bring data from the source system to the virtual database. Further, we will not be able to use indices that may exist on the legacy system if we first have to convert and only then test the value of a column.

Virtuoso introduces an SQL extension for dealing with these issues. Virtuoso allows the definition of two SQL functions that are inverses of each other, for example `dollar_to_euro` and `euro_to_dollar` would be examples of inverse functions.

Thus, when we define a view like:

```
create view euro_item as Select I_id, dollar_to_euro (I_price) as I_price, I_name from item;
```

We can further declare:

```
db..sinv_create_inverse ('euro_to_dollar', 'dollar_to_euro', 1);
```

and perform the query:

```
select * from euro_item where I_price > 100;
```

Normally, this would internally perform the query:

```
select * from item where dollar_to_euro (I_price) > 100;
```

Now however it will make use of the inverse function information and we get:

```
select * from item where I_price > euro_to_dollar (100)
```

The second form is substantially more efficient since only one function call needs to be done for the whole query, not to mention the fact that if the currency conversion function were only defined on the vdb and the item table were linked from another server, all data would have to be brought over to the vdb for the conversion.

Now if the conversion function were defined only on the vdb, the amount of euros would be converted to dollars and then passed as a parameter to

```
Select * from item where I_price > ?
```

Many simple conversions can be efficiently handled by this method. For most unit and data type conversions, the conversion preserves collation order. Amount a will be greater than amount b whether a and b be expressed in euros or dollars, as long as both a and b are in the same currency. Some other conversions such as compression, encryption, hashes and mappings of arbitrary unique ids to another set of ids do not preserve collation. Whether a particular mapping preserves collation is indicated by the third argument of `sinv_create_inverse` . In the above example we see a value of 1, meaning that collation is preserved. Zero specifies that collation is not preserved.

Sometimes it will be desirable to map multi-part keys into a single key for convenience of querying or for compatibility with another database schema.

We will use the orders and customer tables from the TPC C benchmark schema to illustrate the feature. Both tables have a primary key consisting of three parts, a warehouse id, a district id and then a customer or order id. The order references the customer with a foreign key consisting of its own warehouse and district id and of a customer id.

Now we may wish to simplify the situation and make a one-column order id and a one column foreign key reference to customer for presenting the system to an outside reporting application. We will do this completely without touching the original database, all logic taking place on the virtual database.

In this case, the original keys consist of three numbers: n digits of warehouse id, 2 digits of district id and 9 digits of order or customer id. We can simply concatenate these numbers into a single decimal. To do this, we define the following functions:

```
create function num_truncate (in n numeric)
{
  return cast (cast (n as numeric) - 0.5 as numeric (40, 0));
}

create function num_mod (in x numeric, in y numeric)
{
  return (x - num_truncate (cast (x as numeric) / y) * y);
}

/* This function takes the three key parts and returns a numeric where each occupies a fixed range of dig

create function o_iid (in w_id int, in d_id int, in o_id int) returns numeric
{
  return (w_id * 100000000000 + d_id * 1000000000 + o_id);
}

/* The below three functions retrieve each of the fields encoded in the numeric produced by o_iid */

create function o_iid_w_id (in iid numeric) returns int
{
  return cast (num_truncate (iid / 100000000000) as int);
}

create function o_iid_d_id (in iid numeric) returns int
{
  return cast (num_mod (num_truncate (iid / 1000000000), 100) as int);
}

create function o_iid_o_id (in iid numeric)
{
  return cast (num_mod (iid, 1000000000) as int);
}

db..sinv_create_inverse ('O_IID', VECTOR ('O_IID_W_ID', 'O_IID_D_ID', 'O_IID_O_ID'), 0);
```

The `sinv_create_inverse` now defines an accessory for each of the arguments of o_iid. The first function in the vector retrieves the first argument of a call to o_iid, the second the second and so forth. Since we are talking about mapping a set of values into one value it does not make sense to speak of preserving a collation, thus the last argument is 0. When making this declaration we also assert that for each distinct combination of arguments of o_iid, we get a distinct value from which all the original arguments can be retrieved using the functions mentioned.

Now we define the same functions for constructing a synthetic one-part customer id. The functions are exactly the same as for the order id.

```
create function c_iid (in w_id int, in d_id int, in c_id int) returns numeric
{
  return (w_id * 100000000000 + d_id * 1000000000 + c_id);
}

create function c_iid_w_id (in iid numeric) returns int
{
  return cast (num_truncate (iid / 100000000000) as int);
}

create function c_iid_d_id (in iid numeric) returns int
{
  return cast (num_mod (num_truncate (iid / 1000000000), 100) as int);
```

```
}

create function c_iid_c_id (in iid numeric)
{
  return cast (num_mod (iid, 1000000000) as int);
}

db..sinv_create_inverse ('C_IID', VECTOR ('C_IID_W_ID', 'C_IID_D_ID', 'C_IID_C_ID'), 0);
```

Now we have defined functions for producing the synthetic keys we wish to use. Now we bind these functions to the original database using views. We assume that the original orders and customer tables are attached from a remote database under the names orders and customer.

```
create view  orders_2 as select o_iid (o_w_id, o_d_id, o_id) as o_iid numeric, c_iid (o_w_id, o_d_id, o_c

create view customer_2 as select c_iid (c_w_id, c_d_id, c_id) as c_iid numeric, * from customer;
```

As a simple test, let us get the synthetic o_c_iid from an order using the synthetic o_iid key.

```
SQL> select o_c_iid from orders_2 where o_iid = 102000000002;

returns_________

102000000935
```

The order #2 in district #2 of warehouse #1 refers to customer #935 or district #2 of warehouse #1.

```
SQL> explain ('select o_c_iid from orders_2 where o_iid = 102000000002');

returns

{

Precode:
      0: $25 "callret" := Call O_IID_O_ID (<constant (102000000002)>)
      7: $26 "callret" := Call O_IID_D_ID (<constant (102000000002)>)
      14: $27 "callret" := Call O_IID_W_ID (<constant (102000000002)>)
      21: BReturn 0
Remote  SELECT "t2"."O_C_ID", "t2"."O_D_ID", "t2"."O_W_ID" FROM "DBA"."ORDERS" "t2"  where "t2"."O_ID" =
Params ($25 "callret", $26 "callret", $27 "callret")
Output ($29 "t1.O_C_ID", $30 "t1.O_D_ID", $31 "t1.O_W_ID")

After code:
      0: $40 "O_C_IID" := Call C_IID ($31 "t1.O_W_ID", $30 "t1.O_D_ID", $29 "t1.O_C_ID")
      7: BReturn 0
Select ($40 "O_C_IID")
}
```

In this query execution plan we see that the virtual database first decomposes the synthetic o_iid into its component parts, passes these to a query run on the remote database, retrieves the components of the synthetic o_c_iid and finally assembles this and returns it to the client.

We may also use this mapping in more complex queries:

```
select count (*) from orders_2, customer_2 where o_c_iid = c_iid;
```

This uses the synthetic foreign and primary keys for joining. Normally, since the functions for making these keys are only defined on the vdb side, each order would have to be retrieved and then the corresponding customer fetched. Now however we can take advantage of the inverse declaration and decompose the comparison of the synthetic keys into comparisons of each of their arguments:

```
SQL> explain ('select count (*) from orders_2, customer_2 where o_c_iid = c_iid');

returns

{
Remote  SELECT COUNT ( *) FROM "DBA"."ORDERS" "t2" , "DBA"."CUSTOMER" "t4"  where "t4"."C_ID" = "t2"."O_C
Output ($26 "aggregate")
Select ($26 "aggregate")
```

```
}
```

We see that the SQL compiler decomposed the comparison of the c_iid's into a comparison of the original arguments of the function. This can be done because we earlier asserted that each distinct argument combination would produce a unique result of the function.

The examples covered thus far are simple in that there is a clear way of mapping the data back and forth using SQL functions. This is not always possible, for example if we wish to generate unique numeric id's from keys consisting of arbitrary strings.

To this effect, Virtuoso provides a mechanism for automatically generating mapping functions, which take a combination of n arguments of an arbitrary searchable data type and generate a unique id for each distinct combination. The ids are assigned as needed and the mapping is persisted in an automatically generated table.

We could have generated the o_iid, o_iid_w_id, o_iid_d_id and o_iid_o_id functions with the following single call:

```
db..sinv_create_key_mapping ('O_IID', vector ('W_ID', 'int', 'D_ID', 'int', 'O_ID', 'int'));
```

The first argument is the name of the mapping function to generate. The following vector lists the arguments of this function and their data types. The name of each argument is appended to the name of the mapping function, separated by an underscore for forming the inverse functions, one for each argument. Thus the functions are named exactly as in the previous example where we defined them manually.

Internally, this generates a table for keeping the w_id, d_id and o_id for each allocated o_iid. The o_iid's come from a sequence object. The functions manage this table automatically, without requiring developer intervention. The name of the sequence is the same as that of the mapping, thus sequence_set can be used for setting the sequence counter. Note that sequence names are case sensitive.

When a new combination of w_id, d_id, o_id is seen by o_iid, it inserts a row in the mapping table and assigns this a new unique number. This update of the mapping table is committed at the same time the enclosing transaction commits. The table is normally read with read committed isolation and updated with serializable isolation. This guarantees that when a single w_id, d_id, o_id combination is seen at the same time on two threads the result will be the same id being given on both threads.

Any value returned by the mapping function may at all times be used for the inverse functions. The inverse functions will return NULL if given a value that was at no time returned by the mapping function.

The table created for the mapping is named MAP_<name of the mapping function>. The system does not define an automatic cleanup function for removing established mappings because the efficient way of implementing this is quite application dependent. Such a function can however be easily written by the developer if one is needed.

Because comparison of mapping functions is reduced into pair-wise comparison of their arguments, the query

```
select count (*) from orders, customer where o_c_iid = c_iid
```

Will work even if all orders or customers have not been assigned a mapping on the vdb side. If a virtual database application wishes to retrieve rows from a remote database using a synthetic key made by functions defined with `sinv_create_key_mapping` , any value returned by the mapping function in a transaction that was successfully committed is valid and will retrieve the data intended.

9.29.1. Updating through Inverses

A view that selects calls to functions, which have inverses, is updateable. This means that the SQL compiler will use the inverse function on the value being assigned before assigning the column. This is done for both insert and update operations. No special declaration is needed. If the function with inverse has multiple arguments, all of which are columns, then assigning the function will assign all the argument columns, having called the appropriate inverse for getting each of the separate argument values.

For example:

```
SQL> insert into orders_2 (o_iid) values (1102000000003);

Done. -- 1 msec.
SQL> select * from orders_2 where o_w_id = 11;
O_IID            O_C_IID          O_ID             O_D_ID           O_W_ID           O_C_ID       O_ENT
```

```
DECIMAL          DECIMAL          INTEGER NOT NULL  INTEGER NOT NULL  INTEGER NOT NULL  INTEGER     DATE
_____________________________________________________________________________________________

1102000000003    NULL             3                 2                 11                NULL        NULL

1 Rows. -- 1 msec.
SQL> update item_euro set i_price = 120 where i_id = 1234;
```

Conclusions

The examples in this document used tables attached from remote databases because the features discussed here are most likely to be useful in such contexts. Also the query execution plans clearly show which operations take place where. The inverse function mechanism is however in no way limited to virtual database applications. All the examples will work equally well with local tables.

9.30. SQL Grammar

```
sql_list
        : sql ';'

        | sql_list sql ';'

        ;

sql
        : schema_element_list
        | view_def
        ;

schema_element_list
        : schema_element
        | schema_element_list schema_element
        ;

schema_element
        : base_table_def
        | create_index_def
        | drop_table
        | drop_index
        | add_column
        | table_rename
        | privilege_def
        | privilege_revoke
        | create_user_statement
        | delete_user_statement
        | set_pass
        | set_group_stmt
        ;

base_table_def
        : CREATE TABLE new_table_name '(' base_table_element_commalist ')'

        ;

base_table_element_commalist
        : base_table_element
        | base_table_element_commalist ',' base_table_element
        ;

base_table_element
        : column_def
        | table_constraint_def
        ;

column_def
        : column data_type column_def_opt_list
        ;

references
        : REFERENCES q_table_name opt_column_commalist
        ;

column_def_opt_list
```

```
        : /* empty */
        | column_def_opt_list column_def_opt
        ;

column_def_opt
        : NOT NULLX
        | IDENTITY
        | NOT NULLX PRIMARY KEY opt_index_option_list
        | DEFAULT literal
        | references
        ;

table_constraint_def
        : UNDER q_table_name
        | PRIMARY KEY '(' index_column_commalist ')' opt_index_option_list
        | FOREIGN KEY '(' column_commalist ')' references
        ;

column_commalist
        : column
        | column_commalist ',' column
        ;

index_column_commalist
        : column opt_asc_desc
        | index_column_commalist ',' column opt_asc_desc
        ;

index_option
        : CLUSTERED
        | UNIQUE
        ;

index_option_list
        : index_option
        | index_option_list index_option
        ;

opt_index_option_list
        : /* empty */
        | index_option_list
        ;

create_index_def
        : CREATE opt_index_option_list INDEX index
                ON new_table_name '(' index_column_commalist ')'

        ;

drop_index
        : DROP INDEX NAME opt_table
        ;

opt_table
        : /* empty */
        | q_table_name
        ;

drop_table
        : DROP TABLE q_table_name
        ;

add_col_column_def_list
        : column_def
        | add_col_column_def_list ',' column_def
        ;

add_col_column_list
        : column
        | add_col_column_list ',' column
        ;

add_column
```

```
        : ALTER TABLE q_table_name ADD opt_col_add_column add_col_column_def_list
        | ALTER TABLE q_table_name DROP opt_col_add_column add_col_column_list
        | ALTER TABLE q_table_name MODIFY opt_col_add_column column_def
        ;

table_rename
        : ALTER TABLE q_table_name RENAME new_table_name
        ;

view_def
        : CREATE VIEW new_table_name opt_column_commalist
              AS query_exp opt_with_check_option
        ;

opt_with_check_option
        : /* empty */
        | WITH CHECK OPTION
        ;

opt_column_commalist
        : /* empty */
        | '(' column_commalist ')'

        ;

priv_opt_column_commalist
        : /* empty */
        | '(' column_commalist ')'

        ;

privilege_def
        : GRANT ALL PRIVILEGES TO grantee
        | GRANT privileges ON table TO grantee_commalist opt_with_grant_option
        | GRANT grantee_commalist TO grantee_commalist opt_with_admin_option
        ;

privilege_revoke
        : REVOKE ALL PRIVILEGES FROM grantee_commalist
        | REVOKE privileges ON table FROM grantee_commalist
        | REVOKE grantee_commalist FROM grantee_commalist
        ;

opt_with_grant_option
        : /* empty */
        | WITH GRANT OPTION
        ;

opt_with_admin_option
        : /* empty */
        | WITH ADMIN OPTION
        ;

privileges
        : ALL PRIVILEGES
        | ALL
        | operation_commalist
        ;

operation_commalist
        : operation
        | operation_commalist ',' operation
        ;

operation
        : SELECT priv_opt_column_commalist
        | INSERT
        | DELETE
        | UPDATE priv_opt_column_commalist
        | EXECUTE
        ;

grantee_commalist
```

```
        : grantee
        | grantee_commalist ',' grantee
        ;

grantee
        : PUBLIC
        | user
        ;

set_pass
        : SET PASSWORD NAME NAME
        ;

create_user_statement
        : CREATE USER user
        | CREATE ROLE user
        ;

delete_user_statement
        : DELETE USER user [CASCADE]
        | DROP ROLE user
        ;

set_group_stmt
        : SET USER GROUP user user
        ;

cursor_def
        : DECLARE NAME CURSOR FOR query_spec
        ;

opt_order_by_clause
        : /* empty */
        | ORDER BY ordering_spec_commalist
        ;

ordering_spec_commalist
        : ordering_spec
        | ordering_spec_commalist ',' ordering_spec
        ;

ordering_spec
        : INTNUM opt_asc_desc
        | column_ref opt_asc_desc
        | function_ref opt_asc_desc
        ;

opt_asc_desc
        : /* empty */
        | ASC
        | DESC
        ;

sql
        : manipulative_statement
        ;

manipulative_statement
        : query_exp
        | update_statement_positioned
        | update_statement_searched
        | insert_statement
        | delete_statement_positioned
        | delete_statement_searched
        | call_statement
        | admin_statement
        | use_statement
        ;

use_statement
        : USE NAME
        ;
```

```
close_statement
        : CLOSE cursor
        ;

delete_statement_positioned
        : DELETE FROM table WHERE CURRENT OF cursor
        ;

delete_statement_searched
        : DELETE FROM table opt_where_clause
        ;

fetch_statement
        : FETCH cursor INTO target_commalist
        ;

insert_mode
        : INTO
        | REPLACING
        | SOFT

insert_statement
        : INSERT insert_mode table priv_opt_column_commalist values_or_query_spec
        ;

values_or_query_spec
        : VALUES '(' insert_atom_commalist ')'
        | query_spec
        ;

insert_atom_commalist
        : insert_atom
        | insert_atom_commalist ',' insert_atom
        ;

insert_atom
        : scalar_exp
        ;

cursor_option
        : EXCLUSIVE
        | PREFETCH INTNUM
        ;

cursor_options_commalist
        : cursor_option
        | cursor_options_commalist ',' cursor_option
        ;

opt_cursor_options_list
        : /* empty */
        | '(' cursor_options_commalist ')'

        ;

open_statement
        : OPEN cursor opt_cursor_options_list
        ;

with_opt_cursor_options_list
        : /* empty */
        | WITH opt_cursor_options_list
        ;

select_statement
        : SELECT opt_all_distinct selection table_exp
        | SELECT opt_all_distinct selection
            INTO target_commalist table_exp with_opt_cursor_options_list
        ;

opt_all_distinct
        : /* empty */
        | ALL
```

```
        | DISTINCT
        ;

update_statement_positioned
        : UPDATE table SET assignment_commalist WHERE CURRENT OF cursor
        ;

assignment_commalist
        : /* empty */
        | assignment
        | assignment_commalist ',' assignment
        ;

assignment
        : column COMPARISON scalar_exp
        ;

update_statement_searched
        : UPDATE table SET assignment_commalist opt_where_clause
        ;

target_commalist
        : target
        | target_commalist ',' target
        ;

target
        : column_ref
        ;

opt_where_clause
        : /* empty */
        | where_clause
        ;

query_exp
        : query_term
        | query_exp UNION query_term
        | query_exp UNION ALL query_term
        ;

query_term
        : query_spec
        | '(' query_exp ')'

        ;

query_spec
        : SELECT opt_all_distinct selection table_exp
        ;

selection
        : scalar_exp_commalist
        ;

table_exp
        : from_clause opt_where_clause opt_group_by_clause opt_having_clause
              opt_order_by_clause opt_lock_mode
        ;

from_clause
        : FROM table_ref_commalist
        ;

table_ref_commalist
        : table_ref
        | table_ref_commalist ',' table_ref
        ;

table_ref
        : table
        | '(' query_exp ')' NAME
        | joined_table
```

```
        ;

table_ref_nj
        : table
        | subquery NAME
        ;

opt_outer
        : /* empty */
        | OUTER
        ;

jtype
        : LEFT
        ;

joined_table
        : table_ref jtype opt_outer JOIN table_ref_nj ON search_condition
        | BEGIN_OJ_X table_ref jtype opt_outer JOIN table_ref_nj
          ON search_condition ENDX
        ;

where_clause
        : WHERE search_condition
        ;

opt_group_by_clause
        : /* empty */
        | GROUP BY ordering_spec_commalist
        ;

opt_having_clause
        : /* empty */
        | HAVING search_condition
        ;

opt_lock_mode
        : /* empty */
        | FOR UPDATE
        ;

search_condition
        : /* empty */
        | search_condition OR search_condition
        | search_condition AND search_condition
        | NOT search_condition
        | '(' search_condition ')'

        | predicate
        ;

predicate
        : comparison_predicate
        | between_predicate
        | like_predicate
        | test_for_null
        | in_predicate
        | all_or_any_predicate
        | existence_test
        | scalar_exp_predicate
        ;

scalar_exp_predicate
        : scalar_exp
        ;

comparison_predicate
        : scalar_exp COMPARISON scalar_exp
        | scalar_exp COMPARISON subquery
        ;

between_predicate
        : scalar_exp NOT BETWEEN scalar_exp AND scalar_exp
```

```
        | scalar_exp BETWEEN scalar_exp AND scalar_exp
        ;

like_predicate
        : scalar_exp NOT LIKE scalar_exp opt_escape
        | scalar_exp LIKE scalar_exp opt_escape
        ;

opt_escape
        : /* empty */
        | ESCAPE atom
        | BEGINX ESCAPE atom ENDX
        ;

test_for_null
        : column_ref IS NOT NULLX
        | column_ref IS NULLX
        ;

in_predicate
        : scalar_exp NOT IN subquery
        | scalar_exp IN subquery
        | scalar_exp NOT IN '(' scalar_exp_commalist ')'

        | scalar_exp IN '(' scalar_exp_commalist ')'

        ;

all_or_any_predicate
        : scalar_exp COMPARISON any_all_some subquery
        ;

any_all_some
        : ANY
        | ALL
        | SOME
        ;

existence_test
        : EXISTS subquery
        ;

subquery
        : '(' SELECT opt_all_distinct selection table_exp ')'

        ;

scalar_exp
        : scalar_exp '+' scalar_exp
        | scalar_exp '-' scalar_exp
        | scalar_exp '*' scalar_exp
        | scalar_exp '/' scalar_exp
        | '+' scalar_exp %prec UMINUS
        | '-' scalar_exp %prec UMINUS
        | atom
        | column_ref
        | function_ref
        | '(' scalar_exp ')'

        | '(' scalar_exp ',' scalar_exp_commalist ')'

        | function_call
        | as_expression
        | assignment_statement
        | cvt_exp
        ;

cvt_exp
        : CONVERT '(' data_type ',' scalar_exp ')'

        ;

as_expression
```

```
        : scalar_exp AS NAME data_type
        | scalar_exp AS NAME
        ;

opt_scalar_exp_commalist
        : /* empty */
        | scalar_exp_commalist
        ;

function_call
        : q_table_name '(' opt_scalar_exp_commalist ')'

        | BEGIN_FN_X NAME '(' opt_scalar_exp_commalist ')' ENDX
        | BEGIN_FN_X USER '(' opt_scalar_exp_commalist ')' ENDX
        | BEGIN_FN_X CHARACTER '(' opt_scalar_exp_commalist ')' ENDX
        | CALL '(' scalar_exp ')' '(' opt_scalar_exp_commalist ')'

        ;

obe_literal
        : BEGINX NAME atom ENDX
        ;

scalar_exp_commalist
        : scalar_exp
        | scalar_exp_commalist ',' scalar_exp
        ;

atom
        : parameter_ref
        | literal
        | USER
        | obe_literal
        ;

parameter_ref
        : parameter
        | parameter parameter
        | parameter INDICATOR parameter
        ;

function_ref
        : AMMSC '(' '*' ')'

        | AMMSC '(' DISTINCT scalar_exp ')'

        | AMMSC '(' ALL scalar_exp ')'

        | AMMSC '(' scalar_exp ')'

        ;

literal
        : STRING
        | INTNUM
        | APPROXNUM
        | NULLX
        ;

q_table_name
        : NAME
        | NAME '.' NAME
        | NAME '.' NAME '.' NAME
        | NAME '.'  '.' NAME
        ;

new_table_name
        : NAME
        | NAME '.' NAME
        | NAME '.' NAME '.' NAME
        | NAME '.'  '.' NAME
        ;
```

```
table
        : q_table_name
        | q_table_name AS NAME
        | q_table_name NAME
        ;

column_ref
        : NAME
        | NAME '.' NAME
        | NAME '.' NAME '.' NAME
        | NAME '.' NAME '.' NAME '.' NAME
        | NAME '.' '.' NAME '.' NAME
        | '*'
        | NAME '.' '*'
        | NAME '.' NAME '.' '*'

        | NAME '.' NAME '.' NAME '.' '*'

        | NAME '.' '.' NAME '.' '*'

        ;

data_type
        : CHARACTER
        | VARCHAR
        | VARCHAR '(' INTNUM ')'

        | CHARACTER '(' INTNUM ')'

        | NUMERIC
        | NUMERIC '(' INTNUM ')'

        | NUMERIC '(' INTNUM ',' INTNUM ')'

        | DECIMAL
        | DECIMAL '(' INTNUM ')'

        | DECIMAL '(' INTNUM ',' INTNUM ')'

        | INTEGER
        | SMALLINT
        | FLOAT
        | FLOAT '(' INTNUM ')'

        | REAL
        | DOUBLE PRECISION
        | LONG VARCHAR
        | LONG VARBINARY
        | TIMESTAMP
        | DATETIME
        | TIME
        | DATE
        | OWNER
        ;

column
        : NAME
        ;

index
        : NAME
        ;

cursor
        : NAME
        ;

parameter
        : PARAMETER
        ;

user
        : NAME
```

```
        ;

opt_log
        : /* empty */
        | scalar_exp
        ;

comma_opt_log
        : /* empty */
        | ',' scalar_exp
        ;

admin_statement
        : CHECKPOINT opt_log
        | SHUTDOWN opt_log
        | SET REPLICATION atom
        | LOG OFF
        | LOG ON
        ;

sql
        : routine_declaration
        | trigger_def
        | drop_trigger
        ;

routine_declaration
        : CREATE routine_head new_table_name rout_parameter_list
            opt_return compound_statement
        ;

routine_head
        : FUNCTION
        | PROCEDURE
        ;

opt_return
        : /* empty */
        | RETURNS data_type
        ;

rout_parameter_list
        : '(' parameter_commalist ')'

        ;

parameter_commalist
        : rout_parameter
        | parameter_commalist ',' rout_parameter
        ;

rout_parameter
        : parameter_mode column_ref data_type
        ;

parameter_mode
        : IN
        | OUT
        | INOUT
        ;

routine_statement
        : select_statement
        | update_statement_positioned
        | update_statement_searched
        | insert_statement
        | delete_statement_positioned
        | delete_statement_searched
        | close_statement
        | fetch_statement
        | open_statement
        | commit_statement
*/
```

```
        | ';'

        ;

compound_statement
        : BEGINX statement_list ENDX
        ;

statement_list
        : statement_in_cs
        | statement_list statement_in_cs
        ;

statement_in_cs
        : local_declaration ';'

        | compound_statement
        | routine_statement ';'

        | control_statement
        | NAME ':' statement
        ;

statement
        : routine_statement ';'

        | control_statement
        | compound_statement
        ;

local_declaration
        : cursor_def
        | variable_declaration
        | handler_declaration
        ;

variable_declaration
        : DECLARE variable_list data_type
        ;

variable_list
        : NAME
        | variable_list ',' NAME
        ;

condition
        : NOT FOUND
        | SQLSTATE STRING
        ;

handler_declaration
        : WHENEVER condition GOTO NAME
        | WHENEVER condition GO TO NAME
        ;

control_statement
        : call_statement ';'

        | return_statement ';'

        | assignment_statement ';'

        | if_statement
        | goto_statement ';'

        | while_statement
        ;

assignment_statement
        : lvalue EQUALS scalar_exp
        ;

lvalue
```

```
        : column_ref
        ;

if_statement
        : IF '(' search_condition ')' statement opt_else
        ;

opt_else
        : /* empty */
        | ELSE statement
        ;

call_statement
        : CALL q_table_name '(' opt_scalar_exp_commalist ')'

        | function_call
        ;

goto_statement
        : GOTO NAME
        | GO TO NAME
        ;

return_statement
        : RETURN scalar_exp
        | RETURN
        ;

while_statement
        : WHILE '(' search_condition ')' statement
        ;

trigger_def
        : CREATE TRIGGER NAME action_time event ON q_table_name
                        opt_old_ref compound_statement
        ;

action_time
        : BEFORE
        | AFTER
        ;

event
        : INSERT
        | UPDATE
        | DELETE
        ;

opt_old_ref
        : /* empty */
        | REFERENCING old_commalist
        ;

old_commalist
        : old_alias
        | old_commalist ',' old_alias
        ;

old_alias
        : OLD AS NAME
        | NEW AS NAME
        ;

drop_trigger
        : DROP TRIGGER q_table_name
        ;
```

9.31. Bitmap Indices

A bitmap index is a special type of index that is tailored for being efficient for key columns with relatively few distinct values, i.e. low cardinality key columns. A bitmap index is created with the normal create index statement by putting the bitmap keyword in front of index, as follows:

```
create table customer (c_id int primary key, c_state char (2), c_gender char (1), .... );

create bitmap index c_gender on customer (c_gender);
create bitmap index c_state on customer (c_state);
```

Bitmap indices offer space savings of up to 1000 x in some cases and specially for large tables the savings in I/O can be very significant.

A bitmap index can only be used on tables with an integer primary key or in other situations where the last effective key part of the index is an integer. This is understandable since a bitmap index uses a bitmap for representing the values of the last key part, thus having one bitmap for each distinct combination of leading key parts. In the above example, the customer table has an integer c_id column as primary key and a character for the customer's gender and a 2 character field for the state where the customer is located. Thus, to count all the male customers in Massachusetts, one will take the males bitmap and the MA bitmap and perform a bitwise AND of the two. This will have a 1 bit corresponding to the c_id of each male customer in Massachusetts.

We also note that in order to do the count, the customer table itself does not even have to be referenced, as the bitmaps hold all the information. Even if the table did have to be referenced, for example for adding up the outstanding credit of all male customers in Massachusetts, the bitwise AND could be done first and only the relevant rows would have to be retrieved from the table itself.

Virtuoso's implementation of bitmap indices is designed to work efficiently even when the leading key parts have relatively high cardinality, i.e. many distinct values, causing there to be a large number of mostly empty bitmaps. Of course, if each bitmap has only one bit set, for example if every customer is in a different state, there is no benefit to bitmap indices. On the other hand, there is also almost no penalty, only 6 bytes more per index entry than for a regular index. Therefore, for any non-unique key where bitmap indices are applicable, even if there are only a few repeated values, bitmap indices are a safe choice. If there are at least 2 times more rows than distinct values of the keys, space savings are certain.

A bitmap index may have any number of key parts of any type, provided that the last effective part is an integer or and IRI id. The last key parts of an index are those primary key parts that do not occur elsewhere in the key. Thus, if the primary key is a single integer, bitmap indices are always applicable. However, supposing the primary key were an integer plus a string, it would be possible to make a bitmap index where the string were first, followed by the integer. This would make sense and save space if the string were not unique by itself. As another example, the RDF_QUAD system table, the default location of the Virtuoso RDF triple store, has the columns P, G, O and S, where all are IRI ID's, except for O which is ANY. Thus, The primary key is the concatenation of all columns, by default in the order GSPO. There is another key in the order PGOS which can be implemented as a bitmap index because S is an IRI ID, hence integer-like for purposes of bitmap indices.

9.31.1. Bitmap Indices and Transactions

The minimum locking unit is the row. In the case of a bitmap index, one row holds a bitmap which most often refers to many rows. Locking is therefore less granular than with regular indices. Thus, if multiple threads insert rows with bitmap indices, more waits may occur than if the index were not bitmapped. A single row of a bitmap index references maximally 8192 other rows, most often however the count is much less.

In all other respects, locking and transactional behavior are identical with other indices.

9.31.2. Performance Implications

The main advantage of a bitmap index is more compact size, reflected in less I/O. Inserting an entry takes on the average 10% longer than for another type of index, likewise for random lookups with exact key values. Sequential access is usually faster. Space savings and thereby improved working set behavior can produce dramatic gains for large tables.

9.31.3. Physical Structure and Overheads

Bitmap indices divide the range of signed 64 bit integer values into ranges holding 8192 (8K) values. Each such range where at least one bit is set is represented by a compression entry (CE). Multiple CE's can be on the same row. CE's having one bit set take 4 bytes, CE's with 512 or less bits set take 4 bytes plus 2 bytes per bit, CE's with over 512 bits set take 1K byte regardless of how many bits are set.

A bitmap index where the bitmap holds only one bit takes 6 bytes more than the corresponding non-bitmap index entry. A second value, if it falls in the same 8K range adds 2 bytes, 4 bytes if it does not fall within in the same 8K range. If more than 512 values fall within the same 8K range, the bits are represented as a 1K byte bitmap and adding subsequent values takes no extra space.

Virtuoso supports bitmap indices since version 4.5.2919.

Bitmap Indices

9.32. Transitivity in SQL

Virtuoso SQL supports tree and graph data structures represented with SQL relations through the use of transitive subqueries.

A derived table, i..e. a select inside a from clause, can be declared to be transitive. This is done by putting the TRANSITIVE modifier after the SELECT keyword, at the place where a DISTINCT or TOP modifier would go.

The syntax of this is:

```
transitive_decl ::= TRANSITIVE <trans_option>[, ...]

trans_opt :: =
        | T_MIN (INTNUM)
        | T_MAX (INTNUM)
        | T_DISTINCT
        | T_EXISTS
        | T_NO_CYCLES
        | T_CYCLES_ONLY
        | T_NO_ORDER
        | T_SHORTEST_ONLY
        | T_IN <position_list>
        | T_OUT ( <position_list )
        | T_END_FLAG  (INTNUM)
        | T_FINAL_AS NAME
        | T_STEP ( <position_or_path_spec> )
        | T_DIRECTION (INTNUM)

position_list ::=  INTNUM [,...]

position_or_path_spec ::= INTNUM | 'step_no' | 'path_id'
```

A transitive derived table has a selection that may consist of four different types of columns. These are input, output, step data and special columns. When a transitive derived table occurs in a query, the enclosing query must specify an equality condition for either all input, all output columns or both. The designation of input and output columns is for convenience only. The order of query execution will be generally decided by the optimizer, unless overridden with the T_DIRECTION option.

Consider a simplified social network application:

```
create table knows (p1 int, p2 int, primary key (p1, p2))
alter index knows on knows partition (p1 int);
create index knows2 on knows (p2, p1) partition (p2 int);

insert into knows values (1, 2);
insert into knows values (1, 3);
insert into knows values (2, 4);
```

All persons have single integer identifiers. There is a row in the knows table if person p1 claims to know person p2.

The most basic query is to find all the people that a given person knows either directly or indirectly.

```
select * from (select transitive t_in (1) t_out (2) t_distinct  p1, p2 from knows) k where k.p1 = 1;
```

The transitive derived table simply selects from the knows table. The enclosing top level query gives an initial value for the input column of the transitive select. This leaves the output column P2 unbound, so the query will iterate over the possible values of P2. Initially, the query loops over the people directly known by 1. In the next stage, it takes the binding of P2 and uses it as a new value of the input column P1 to look for people the first degree contact knows and so on, until no new values are found.

The basic meaning of the transitive modifier is that given initial values for input column(s), the subquery is evaluated to produce values for the output columns. Then these values are fed back as values of input columns and so forth, until some termination condition is reached. If there are equality conditions for columns designated as output but no conditions for columns designated as input, then the same process runs from output to input. The terms input and output do not imply execution order. If there are

bindings for both input and output columns in the enclosing query, then the transitive derived table looks for ways of connecting the input and output bindings. If no such way is found, the subquery is empty and causes the whole enclosing query to also have no result. A transitive derived table cannot be the right side of an outer join directly but can be wrapped in a derived table that is. In this way, an outer join usage is also possible, whether finding a path is optional.

The result set of a transitive subquery can be thought of as a set of paths. A path consists of one or more consecutive bindings for the input columns and is ordered. In our example, a path is p1=1, p1=2, p1=4. This is the path connecting persons 1 and 4. If there are columns in the select that are neither input or output, they too are recorded for each step of the path. The result set may include just the ends of a path, i.e. one row where the input columns have the beginning and the output columns the end of the path. This means there is one row per distinct path. The result set may also include a row for each step on each path.

In this example, we bind both ends of the transitive subquery and ask how person 1 and 4 are connected. Since the columns p1 and p2 have an equality condition, each row of the result set has these at values 1 and 4 respectively.

```
select * from (select transitive t_in (1) t_out (2) t_direction 3 t_distinct t_shortest_only  p1, p2, t_s

P1        P2        VIA                                                                    PATH

1         4         1                                                                      0
1         4         2                                                                      0
1         4         4                                                                      0
```

The three rightmost columns allow returning information in the intermediate steps of the transitive evaluation. t_step (1) means the value of the column at position 1 at the intermediate step. The t_step ('step_no) is the sequence number of the step returned. The t_step ('path_id') is a number identifying the connection path, since there may be many paths joining persons 1 and 4.

In this situation, the result set has one row per step, including a row for the initial and final steps. While the evaluation order may vary internally, the result set is presented as if the query were evaluated from input to output, i.e. looking for people known by 1, finding 2 and 3, then looking for people they know, finding that 2 knows 4, which is a solution, since p2 = 4 was specified in the outer select. If the outere query had p1 = 4 and p2 = 1, there would be an empty result set since there is no path from 4 to 1.

For example, if tables have multipart keys, there can be many input and output columns but there must be an equal number of both, since the engine internally feeds the output back into the input or vice versa. The transitive derived table may be arbitrarily complex.

We may have an application that returns extra information about a step. This could for example be a metric of distance. In such a case, a column which is neither designated as input nor output and is not a t_step () function call, will simply be returned as is.

The result set of a transitive subquery will either have one row for each state reached, or it may have one row for each step on the path to each state reached.

The first example returns only the ends of the paths, i.e. directly and indirectly known person id's. It does not return for each returned id how this person is known, through which set of connections. The second example returns a row for each step on each path. Steps will be returned if the selection has t_step () calls or columns that are neither input or output.

The forms of t_step are:

```
t_step (<column number>)
```

This returns the value that the column, one of the columns designated as input, has at this step. The input or output columns themselves, if there is a condition on them, look equal to the condition. This allows seeing intermediate values of input columns on a path.

t_step ('step_no')

This returns the ordinal number of the step on the path. Step -0 corresponds to the input variables being at the value seen in the enclosing query. Step 1 is one removed from this. Step numbering is assigned as if evaluating from input to output.

Consider this:

```
select * from (select transitive t_in (1) t_out (2) t_min (0) t_distinct   p1, p2, t_step (1) as via, t_s
P1        P2        VIA                                                                    PATH
```

```
1         1         1                                                                                                0
1         3         1                                                                                                1
1         3         3                                                                                                1
1         2         1                                                                                                2
1         2         2                                                                                                2
1         4         1                                                                                                3
1         4         2                                                                                                3
1         4         4                                                                                                3
```

This returns four paths, all starting at 1: the paths. The path from 1 to 1, the path from 1 to 2, the path from 1 to 3 and the path from 1 to 2 to 4. The path_id column has values from 0 to 3, distinguishing the four different paths returned. The p1 column is the start of the path, thus always 1 since this is given in the outer query. The p2 column is the end of the path. The via column is the value of p1 at the intermediate step. The step number where via is equal to p1 is 0. The next step number is 1. At the highest step number of each path, p2 and via are the same.

Now, let us do this in reverse:

```
select * from (select transitive t_in (1) t_out (2) t_min (0) t_distinct   p1, p2, t_step (1) as via, t_s
P1        P2        VIA                                                                                    PATH

4         4         4                                                                                      0
2         4         2                                                                                      1
2         4         4                                                                                      1
1         4         1                                                                                      2
1         4         2                                                                                      2
1         4         4                                                                                      2
```

We give an initial value to p2 and leave p1 free. Now we get three paths, the path from 4 to 4, from 2 to 4 and from 1 to 2 to 4. We enumerate the steps as if counting from input to output, albeit internally the evaluation order is the reverse. Again, step number 0 has the via column equal to p1 and the highest numbered step has via equal to p2.

Now we may look more formally at the meaning of the transitive options:

- T_MIN (INTNUM) - This means that paths shorter than the number are not returned. In the examples above, we had min at 0, so that a path of zero length was also returned, i.e. where the first output equals the outer conditions for the inputs.
- T_MAX (INTNUM) - This gives a maximum length of path. Paths longer than this many steps are not returned. A value of 1 means that the subquery is evaluated once, i.e. the outputs of the first evalyuation are not fed back into the inputs. Specifying a minimum of 0 and a maximum of one means an optional join. Specifying min and max both to 1 means an ordinary derived table.
- T_DISTINCT - This means that if a binding of input columns is produced more than once, only the first is used. Id est, the same point is not traversed twice even if many paths lead to it.
- T_EXISTS - Only one path is generated and returned.
- T_NO_CYCLES - If a path is found that loops over itself, i.e. a next step has the input values equal to the input values of a previous step on the path, the binding is ignored.
- T_CYCLES_ONLY - Only paths that have a cycle, i.e. input values of a subsequent step equal the input values of a previous step on the same path, are returned.
- T_SHORTEST_ONLY - If both ends of the path are given, the evaluation stops at the length of path where the first solution is found. If many paths of equal length are found, they are returned but longer paths are not sought.
- T_IN (column_positions) - This specifies which columns are called input.
- T_OUT (column_positions) - This specifies which columns are called output.
- T_DIRECTION INTNUM - A value of 0 (default) means that the SQL optimizer decides which way the transitive subquery is evaluated. 1 means from input to output, 2 from output to input, 3 from both ends. Supposing we are looking at how two points are related, it makes sense to start expanding the transitive closure at both ends. in the above example, this would be going from p1 to p2 on one side and from p2 to p1 on the other.

See Also:

Collection of Transitivity Option Demo Queries for SPARQL.

9.33. Fast Phrase Match Processor

An "annotation phrase" is a keyword or key phrase associated with some application specific data and a uniform "annotation phrase set" may map dictionary words to their descriptions or ICAO codes to airport names and co-ordinates or geographical names to maps or Wikipedia topics to links to Wikipedia pages. An application may need to scan a given text and find all occurrences of phrases from given phrase set, for various purposes, e.g., to improve the resource with additional hyperlinks, to replace technical codes with user-friendly names, or to suggest appropriate tags and keywords. Virtuoso has built-in phrase match processor that is fast enough to deal with long documents and big phrase sets in interactive applications.

9.33.1. Phrases, Phrase Sets and Phrase Classes

An annotation phrase is a pair of a key (a string that is supposed to be a word or phrase of some natural language) and a value (of any type if its serialization is shorter than 2 kilobytes or a string shorter than 10 megabytes). The key phrase will be divided into words and normalized by language-specific functions used by free text search; after normalization it should contain from one to four words. The associated value may be very long in principle, but it will add noticeable overheads related to memory allocation and copying so it is recommended to keep it short if performance is somehow important. It is usual to make the value as short as an ID in some external "detailed" table, not only for speed but to support multiple synonyms for one thing or names of a thing in different languages.

Annotation phrases are grouped in "phrase sets" and every "phrase set" belongs to some "phrase class".

Phrase classes are enumerated in a DB.DBA.SYS_ANN_PHRASE_CLASS table.

```
create table DB.DBA.SYS_ANN_PHRASE_CLASS
(
  APC_ID integer not null primary key,
  APC_NAME varchar(255) unique,         -- unique name for use in API/UI
  APC_OWNER_UID integer,                -- references SYS_USERS (U_ID), NULL if the record writable for a
  APC_READER_GID integer,               -- references SYS_USERS (U_ID), NULL if the record is readable fo
  APC_CALLBACK varchar,
  APC_APP_ENV any
  )
;
```

Phrase sets and phrase classes may contain confidential information. E.g., a phrase set may be used to check that a given outgoing document does not mention titles of confidential projects or names of persons that should act anonymously for public. So any application may specify access restrictions when a phrase class is created. The application usually sets APC_CALLBACK to name of some of its functions (but it may be any string) and APC_APP_ENV to value of any type (application may pass it to the APC_CALLBACK function as one of arguments). Phrase match processor only keeps these data in memory, unchanged, and returns to the application as a part of text processing report, so they can be used for any purpose.

Phrase class describes an access to some application-specific callback, but it does not mention any phrases at all. Individual phrases are grouped into phrase sets. Each phrase set belongs to exactly one phrase class and is restricted to one language handler. It does not necessarily mean that it should consist of phrases of one natural language, because many language handlers support mix of languages, but it may apply some restrictions.

Phrase sets are enumerated in a DB.DBA.SYS_ANN_PHRASE_SET table.

```
create table DB.DBA.SYS_ANN_PHRASE_SET
(
  APS_ID integer not null primary key,
  APS_NAME varchar(255) unique,         -- unique name for use in API/UI
  APS_OWNER_UID integer,                -- references SYS_USERS (U_ID), NULL if the record writable for a
  APS_READER_GID integer,               -- references SYS_USERS (U_ID), NULL if the record is readable fo
  APS_APC_ID integer not null,          -- references SYS_ANN_PHRASE_CLASS (APC_ID)
  APS_LANG_NAME varchar not null,       -- name of language handler that is used to split texts of phrase
  APS_APP_ENV any,
  APS_SIZE any,                         -- approximate number of phrases inserted in the set (actual or e
  APS_LOAD_AT_BOOT integer not null     -- flags whether phrases should be loaded at boot time.
  )
;
```

APS_APP_ENV can be of any type (application may pass it to the APC_CALLBACK function as one of arguments). Like APC_APP_ENV, phrase match processor only keeps it in memory and returns in a text processing report.

The APS_SIZE tweaks the amount of memory consumed by a phrase set when it's loaded. The average "price" of placing a phrase to a set is 2 bytes per phrase. Memory amounts are less than a kilobyte while APS_SIZE is below 256, less than 64 kilobytes while APS_SIZE grow up to 32 thousands of phrases and reaches its maximum of 64 megabytes when APS_SIZE reaches its practical limit of 32 million phrases. Only rough similarity to the actual number of inserts is needed, say, same order of magnitude. The exaggerated value of APS_SIZE will allocate a bit more memory and may slightly improve the speed. It is not practical to set APS_SIZE much smaller than it should be because it will cause frequent table lookups and disk buffers in use will overweight any memory savings in the phrase match processor. If there are numerous phrase sets that are used only occasionally, do not decrease APS_SIZE, instead set APS_LOAD_AT_BOOT to zero for them.

Note that if phrase set is edited intensively during server run then the number of inserted phrases is important, an effect of phrase removal is visible only after server restart. E.g. if a phrase set is frequently cleaned and refilled with new phrases then it is much better to delete an obsolete set and create a new one.

Individual annotation phrases are stored in a table DB.DBA.SYS_ANN_PHRASE, that should not be modified by applications directly by data manipulation statements. The content of the table is used to build special search structures in memory and SQL operations like INSERT and DELETE can not keep in-memory structures in sync with the content of the table. DBA may read the table but should not update; in case of occasional update all phrase sets that contain edited phrases may become unusable until server restart.

```
create table DB.DBA.SYS_ANN_PHRASE
(
  AP_APS_ID integer not null,           -- references SYS_ANN_PHRASE_SET (APS_ID),
  AP_CHKSUM integer,                    -- phrase check-sum
  AP_TEXT varchar,                      -- original text
  AP_LINK_DATA any,                     -- Associated data about links etc.
  AP_LINK_DATA_LONG long varchar,       -- Same as AP_LINK_DATA but for long content, one of two is alway
  primary key (AP_APS_ID, AP_CHKSUM, AP_TEXT)
  )
;
```

The "almost direct" way of writing to the table is via BIF `ap_add_phrases` . It gets two arguments, the integer ID of annotation phrase set and a vector of descriptions of phrases that should be edited in that phrase set. Every item of vector of descriptions is in turn vector of one or two values; first value is the text of the phrase, second value is associated application specific data, the absence of second value indicates that the phrase should be removed. If same text of phrase appears in the vector of description more than once, and associated data differ then any version of data can be stored for future use; it is the roll of dice because the vector is reordered for faster processing.

9.33.2. Phrase Set Configuration API

- `DB.DBA.ANN_PHRASE_CLASS_ADD`
- `DB.DBA.ANN_PHRASE_CLASS_DEL`
- `AP_BUILD_MATCH_LIST`: The report R is a vector of 6 elements:

    1. R[0] vector of all distinct phrase classes for phrase sets of found phrases; every pair of items represents one phrase class: first item is an integer APC_ID of a class, second item is a description of phrase class as vector of APC_NAME, APC_CALLBACK and APC_APP_ENV;
    2. R[1] vector of all distinct phrase sets of found phrases; every pair of items represents one phrase set: first item is an integer APS_ID of a class, second item is a vector of APS_NAME, APS_APC_ID, index of phrase class description in R[0] and APS_APP_ENV;
    3. R[2] vector of all distinct found phrases; every item represents a phrase as a vector of AP_APS_ID, index of phrase set description in R[1], AP_TEXT and application-specific data from AP_LINK_DATA or AP_LINK_DATA_LONG;
    4. R[3] vector of all composed arrows for the text; every item represents one place in a text, as an "arrow" described below;
    5. R[4] vector of indexes of arrows that point to words in the text; every item is an integer that is index in R[3];
    6. R[5] vector of descriptions of occurrences of annotation phrases in text; every item represents one occurrence as vector of index of first word in R[3], index of last word in R[3], index of found phrase in R[2], index of previous occurrence of same phrase in R[5].
    7. Every "arrow" A is vector of length 5 or 6, it is longer when arrow points inside occurrence of some annotation phrase.

        a. A[0] integer that indicates type of text fragment:

- 0 is for plain word (only this type occurs in reports for plain text),
- 1 is for text of opening tag,
- 2 is for text of closing tag,
- 3 is something exceptional like unrecoverable HTML syntax error

b. A[1] integer offset of the first byte of a fragment in the text
c. A[2] integer offset of the first byte after the end of a fragment
d. A[3] integer that is a bit-mask of opened but not yet closed tags
e. A[4] integer index of the arrow of the innermost tag that is opened but not yet closed where the arrow begins
f. A[5] may absent, if presents then it is a vector of indexes in R[2] of all containing phrases.

8. Bit mask of opened but not yet closed tags consists of the following bits:

```
0x00000001      PCDATA containers (such as OPTION, TEXTAREA, XBODY, XHEAD)
0x00000002      Inlined highlight tags (such as ABBR, ACRONYM, B, BDO, BIG, CITE, CODE, DFN
0x00000004      Tag A
0x00000008      Tag LABEL

0x00000010      Inlined content (such as ADDRESS, APPLET, H1-H6, LABEL, LEGEND, P, PRE, and
0x00000020      Blocks (such as BLOCKQUOTE, BUTTON, DD, DIV, DL, DT, FIELDSET, FORM, IFRAME

0x00000100      Tags of list and ordered list (MENU, OL, UL)
0x00000200      Tag LI
0x00000400      Tag DL
0x00000800      Tags DD and DT

0x00001000      Tag FORM
0x00002000      Tag SELECT
0x00004000      Tag OPTGROUP
0x00008000      Tag BUTTON

0x00010000      Tag TABLE
0x00020000      Tags inside TABLE but outside table rows (such as TBODY, TFOOT, THEAD)
0x00040000      Tag TR
0x00080000      Tags TH and TD

0x00FFFFFF      Tags XBODY and XHEAD

0x01000000      Tag HEAD
0x02000000      Tag FRAMESET
0x04000000      Tag NOFRAMES

0x10000000      Tag HTML
0x20000000      Tag BODY
0x40000000      Tags INS and DEL
0x80000000      Tag XMP
```

For long document, the report may be too long, esp. vectors R[3] and R[4]. A simple application may not need locations of every tag and every word of the document. The report_flags argument is a bitmask, and some bits control the size of the report. If bit 1 is set then closing tags are excluded from report. If bit 2 is set then only words in found phrase are placed to the report, the rest of phrases is excluded.

9.33.3. Advertisers and Advertisement Rules

Phrase sets are sufficient for many purposes but advertisement-specific applications need more built-in functionality for maximum speed. This functionality can be reused by any application that adds hyperlinks to texts depending on content of the text.

Consider advertisers that want to show links to their resources near phrases they choose as relevant, and one phrase can be chosen by many advertisers. To make the processing easier, there exists special support for phrase sets that store lists of links for phrases.

An advertiser is some very abstract "source" of annotation phrases. The exact nature of an advertiser is application-specific, but each advertiser has an identifiable account. All accounts are in one table:

```
create table DB.DBA.SYS_ANN_AD_ACCOUNT (
  AAA_ID integer not null primary key,
  AAA_NAME varchar(255) unique,         -- unique name for use in API/UI
  AAA_OWNER_UID integer,                -- references SYS_USERS (U_ID), NULL if the record writable for a
  AAA_READER_GID integer,               -- references SYS_USERS (U_ID), NULL if the record is readable fo
```

```
  AAA_DETAILS long xml,                 -- any details, e.g., in RDF
  AAA_APP_ENV any
  )
;
```

Advertisement links are stored in a separate table:

```
create table DB.DBA.SYS_ANN_LINK (
  AL_ID integer primary key,
  AL_OWNER_UID integer,                 -- references SYS_USERS (U_ID), NULL if the record writable for a
  AL_URI varchar,                       -- URI template for A HREF
  AL_TEXT varchar,                      -- text template for body of <A>
  AL_NOTE varchar,                      -- text after the link (or around it)
  AL_TAGS any,                          -- tags to add or remove
  AL_CALLBACK varchar,
  AL_APP_ENV any
  )
;
```

Advertisers, links and phrases are linked together by advertisement rules. Each rule says that if a given phrase is found in some text then a given link should be shown and a specified advertiser's account should be charged.

```
create table DB.DBA.SYS_ANN_AD_RULE (
  AAR_AAA_ID integer not null,          -- advertiser who pays for the ad
  AAR_APS_ID integer not null,          -- phrase set
  AAR_AP_CHKSUM integer not null,       -- phrase check-sum
  AAR_TEXT varchar not null,            -- original text
  AAR_AL_ID integer not null,           -- references SYS_ANN_LINK (AL_ID)
  AAR_APP_ENV any,
  primary key (AAR_AAA_ID, AAR_APS_ID, AAR_AP_CHKSUM, AAR_TEXT, AAR_AL_ID)
  )
;
```

When an annotation phrase is stored in its phrase set via advertisement API, its application-specific data is always a vector, and vector's length is divisible by three. When a new ad rule is added to the vector, it gets three new items: value of AL_ID of a link, value of advertiser's AAA_ID and AAR_APP_ENV of the ad rule.

Anyone can describe a link but can not edit other's descriptions:

```
create function DB.DBA.ANN_LINK_ADD (
  in _owner_uid integer,
  in _uri varchar,              -- value for AL_URI
  in _text varchar,             -- value for AL_TEXT
  in _note varchar,             -- value for AL_NOTE
  in _tags any,                 -- value for AL_TAGS
  in _callback varchar,
  in _app_env any) returns integer
```

So one user may create links in favor of other user even without permission, but can not edit other's links. To edit or delete, one should be an owner and pass authentication:

```
create function DB.DBA.ANN_LINK_MODIFY (
  in _id integer,
  in _owner_uid integer,
  in _uri varchar,
  in _text varchar,
  in _note varchar,
  in _tags any,
  in _callback varchar,
  in _app_env any,
  in auth_uname varchar,
  in auth_pwd varchar) returns integer

create function DB.DBA.ANN_LINK_DEL (
  in _id integer,
  in auth_uname varchar,
  in auth_pwd varchar) returns integer
```

A link can not be dropped even by its owner if it is used by some advertiser.

```
create function DB.DBA.ANN_AD_RULE_ADD (
  in aaa_name varchar,          -- Advertiser's name (or its integer AAA_ID key), to store as AAR_AAA_ID
  in aps_name varchar,          -- Name of a phrase set (or its integer APS_ID key), to store as AAR_APS_
  in _text varchar,             -- Text of the phrase, will be stored as AAR_TEXT and duplicated in an AP
  in _al_id integer,            -- ID of a link (AL_ID in DB.DBA.SYS_ANN_LINK), to store as AAR_AL_ID
  in _app_env any,              -- Application-specific data, will be stored in AAR_APP_ENV and become a
  in _lang_name varchar,        -- Language name, for verification only; an error will be signalled if th
  in auth_uname varchar,
  in auth_pwd varchar) returns integer
```

Application-specific data should be short, because for any given phrase, sum of data from all advertisers should not exceed 2 kilobytes. Fortunately, most of needed data are in DB.DBA.SYS_ANN_LINK row already so AAR_APP_ENV is frequently a NULL.

```
create function DB.DBA.ANN_AD_RULE_DEL (
  in aaa_name varchar,
  in aps_name varchar,
  in _text varchar,
  in _al_id integer,
  in _lang_name varchar,
  in auth_uname varchar,
  in auth_pwd varchar) returns integer
```

If a phrase set is edited by DB.DBA.ANN_AD_RULE_ADD / DB.DBA.ANN_AD_RULE_DEL and by ap_add_phrases() then errors are almost guaranteed (and not detected or recovered automatically in current version). The function ap_build_match_list() works fine with mix of phrase sets managed by both methods because it is not sensitive to the format of data in phrases.

9.33.4. Example

The following example demonstrates managing phrases sets and data associated with them:

```
SQL>insert replacing DB.DBA.SYS_ANN_PHRASE_CLASS (APC_ID, APC_NAME, APC_OWNER_UID, APC_READER_GID, APC_CA
values (3, 'Debug apc #3', http_dav_uid(), http_admin_gid(), 'DB.DBA.AP_DEBUG_CALLBACK', 'Debug apc #3 en
;

Done. -- 0 msec.

SQL>insert replacing DB.DBA.SYS_ANN_PHRASE_SET (APS_ID, APS_NAME, APS_OWNER_UID, APS_READER_GID, APS_APC_
values (5, 'Debug aps #5', http_dav_uid(), http_admin_gid(), 3, 'x-any', 'Debug aps #5 env', 10000, 0)
;

Done. -- 0 msec.

SQL>insert replacing DB.DBA.SYS_ANN_PHRASE_SET (APS_ID, APS_NAME, APS_OWNER_UID, APS_READER_GID, APS_APC_
values (7, 'Debug aps #7', http_dav_uid(), http_admin_gid(), 3, 'x-any', 'Debug aps #7 env', 10000, 0)
;

Done. -- 0 msec.

SQL>ap_add_phrases (5,
  vector (
    vector ('Debug5', 'Debug5 env'),
    vector ('Debug5 one', 'Debug5 one env'),
    vector ('Debug5 two', 'Debug5 two env'),
    vector ('Debug5 three', 'Debug5 three env'),
    vector ('Debug5 twenty one', 'Debug5 twenty one env'),
    vector ('Debug5 twenty two', 'Debug5 twenty two env'),
    vector ('Debug5 twenty three', 'Debug5 twenty three env')
    ) )
;
Done. -- 0 msec.

SQL>ap_add_phrases (7,
  vector (
    vector ('Debug7', 'Debug7 env'),
    vector ('Debug7 one', 'Debug7 one env'),
    vector ('Debug7 two', 'Debug7 two env'),
    vector ('Debug7 three', 'Debug7 three env'),
    vector ('Debug7 twenty one', 'Debug7 twenty one env'),
    vector ('Debug7 twenty two', 'Debug7 twenty two env'),
```

```
    vector ('Debug7 twenty three', 'Debug7 twenty three env')
    ) )
;

Done. -- 0 msec.

SQL>ap_add_phrases (5,
  vector (
    vector ('Debug5 thirty one', 'Debug5 thirty one BAD env'),
    vector ('Debug5 thirty two', 'Debug5 thirty two BAD env'),
    vector ('Debug5 thirty three', 'Debug5 thirty three BAD env'),
    vector ('Debug5 one hundred', 'Debug5 one hundred BAD env')
    ) )
;

Done. -- 0 msec.

SQL>ap_add_phrases (5,
  vector (
    vector ('Debug5', 'Debug5 UPDATED env'),
    vector ('Debug5 thirty one', 'Debug5 thirty one NEW env'),
    vector ('Debug5 thirty two', 'Debug5 thirty two NEW env'),
    vector ('Debug5 thirty three', 'Debug5 thirty three NEW env'),
    vector ('Debug5 one hundred')
    ) )
;
Done. -- 0 msec.

SQL>create function ptext_1 ()
{
  return '
    vector ''Debug5'', ''Debug5 env'',
    vector ''Debug5 one'', ''Debug5 one env'',
    vector ''Debug5 two'', ''Debug5 two env'',
    vector ''Debug5 three'', ''Debug5 three env'',
    vector ''Debug5 twenty one'', ''Debug5 twenty one env'',
    vector ''Debug5 twenty two'', ''Debug5 twenty two env'',
    vector ''Debug5 twenty three'', ''Debug5 twenty three env''
ap_add_phrases 7,
  vector
    vector ''Debug7'', ''Debug7 env'',
    vector ''Debug7 one'', ''Debug7 one env'',
    vector ''Debug7 two'', ''Debug7 two env'',
    vector ''Debug7 three'', ''Debug7 three env'',
    vector ''Debug7 twenty one'', ''Debug7 twenty one env'',
    vector ''Debug7 twenty two'', ''Debug7 twenty two env'',
    vector ''Debug7 twenty three'', ''Debug7 twenty three env''
ap_add_phrases 5,
  vector
    vector ''Debug5 thirty one'', ''Debug5 thirty one BAD env'',
    vector ''Debug5 thirty two'', ''Debug5 thirty two BAD env'',
    vector ''Debug5 thirty three'', ''Debug5 thirty three BAD env'',
    vector ''Debug5 one hundred'', ''Debug5 one hundred BAD env''
ap_add_phrases 5,
  vector
    vector ''Debug5'', ''Debug5 UPDATED env'',
    vector ''Debug5 thirty one'', ''Debug5 thirty one NEW env'',
    vector ''Debug5 thirty two'', ''Debug5 thirty two NEW env'',
    vector ''Debug5 thirty three'', ''Debug5 thirty three NEW env'',
    vector ''Debug5 one hundred''
    ';
}
;

Done. -- 20 msec.

SQL>create function ptext_2 ()
{
  return '
    vector ''Debug5 twenty one''
    vector ''Debug7 twenty one''
    vector ''Debug5 thirty one''
    vector ''Debug5 thirty one''
```

```
    vector ''Debug5 one hundred''
    ';
}
;
Done. -- 10 msec.

SQL>create function test_ptext_1()
{
  declare ses any;
  ses := string_output();
  ap_debug_langhandler (ptext_1 (), 'x-any', vector (5, 7), ses );
  return string_output_string (ses);
}
;
Done. -- 10 msec.

SQL>create procedure dump_match_list (in vect any, in path varchar := null)
{
  declare VDATA varchar;
  if (path is null)
    {
      result_names (VDATA);
      path := '';
    }
  if (vect is null)
    {
      result (path || ' NULL');
      return;
    }
  if (193 <> __tag (vect))
    {
      result (path || ' ' || (cast (vect as varchar)));
      return;
    }
  declare l, ctr integer;
  l := length (vect);
  if (0 = l)
    {
      result (path || ' (empty vector)');
      return;
    }
  if (isinteger (vect [l - 1]))
    {
      declare ses any;
      ses := string_output ();
      for (ctr := 0; ctr < l; ctr := ctr + 1)
        {
          http (sprintf (' [%d]=%s', ctr, cast (vect[ctr] as varchar)), ses);
        }
      result (path || string_output_string (ses));
      return;
    }
  for (ctr := 0; ctr < l; ctr := ctr + 1)
    {
      dump_match_list (vect[ctr], sprintf ('%s[%d]', path, ctr));
    }
}
;

Done. -- 10 msec.

SQL>select test_ptext_1();
callret
VARCHAR
_______________________________________________________________________________

    (vector) '[[Debug5 UPDATED env]Debug5]]', '[[Debug5 UPDATED env]Debug5]] (env)', (vector) '[[Debug5 U
    (vector) '[[Debug5 UPDATED env ; Debug5 two env]Debug5]] [[Debug5 two env]two]]', '[[Debug5 UPDATED e
    (vector) '[[Debug5 UPDATED env ; Debug5 twenty one env]Debug5]] [[Debug5 twenty one env]twenty]] [[De
    (vector) '[[Debug5 UPDATED env ; Debug5 twenty two env]Debug5]] [[Debug5 twenty two env]twenty]] [[De
    (vector) '[[Debug5 UPDATED env ; Debug5 twenty three env]Debug5]] [[Debug5 twenty three env]twenty]]
    (vector) '[[Debug7 env]Debug7]]', '[[Debug7 env]Debug7]] (env)', (vector) '[[Debug7 one env ; Debug7
    (vector) '[[Debug7 two env ; Debug7 env]Debug7]] [[Debug7 two env]two]]', '[[Debug7 two env ; Debug7
```

```
    (vector) '[[Debug7 three env ; Debug7 env]Debug7]] [[Debug7 three env]three]]', '[[Debug7 three env ;
    (vector) '[[Debug7 twenty one env ; Debug7 env

1 Rows. -- 50 msec.

SQL>dump_match_list (ap_build_match_list (vector (5, 7), ptext_2 (), 'x-any', 0, 0 ) );
VDATA
VARCHAR
_______________________________________________________________________________

[0][0] 3
[0][1][0] Debug apc #3
[0][1][1] DB.DBA.AP_DEBUG_CALLBACK
[0][1][2] Debug apc #3 env
[1][0] 5
[1][1][0] Debug aps #5
[1][1][1] 3
[1][1][2] 1
[1][1][3] Debug aps #5 env
[1][2] 7
.....
139 Rows. -- 231 msec.

SQL>dump_match_list (ap_build_match_list (vector (5, 7), ptext_2 (), 'x-any', 0, 3 ) );
VDATA
VARCHAR
_______________________________________________________________________________

[0][0] 3
[0][1][0] Debug apc #3
[0][1][1] DB.DBA.AP_DEBUG_CALLBACK
[0][1][2] Debug apc #3 env
[1][0] 5
[1][1][0] Debug aps #5
[1][1][1] 3
[1][1][2] 1
.....
133 Rows. -- 231 msec.

SQL>create procedure ap_make_js_menus (
  in ap_set_ids any, in source_UTF8 varchar, in lang_name varchar, in is_html integer)
{
  declare res_out, script_out, match_list any;
  declare m_apc, m_aps, m_app, m_apa, m_apa_w, m_aph any;
  declare apa_w_ctr, apa_w_count integer;
  declare app_ctr, app_count integer;
  declare prev_end integer;

  match_list := ap_build_match_list ( ap_set_ids, source_UTF8, lang_name, is_html, 3); -- 3 is to have le
  m_apc := aref_set_0 (match_list, 0);
  m_aps := aref_set_0 (match_list, 1);
  m_app := aref_set_0 (match_list, 2);
  m_apa := aref_set_0 (match_list, 3);
  m_apa_w := aref_set_0 (match_list, 4);
  m_aph := aref_set_0 (match_list, 5);
  apa_w_count := length (m_apa_w);
  app_count := length (m_app);
  if (0 = app_count)
    {
      return source_UTF8;
    }
  res_out := string_output ();
  script_out := string_output ();
  http ('<script DEFER language="javescript"><!--', script_out);
  http ('\n  var v_descs = [', script_out);
  for (apa_w_ctr := 0; apa_w_ctr < apa_w_count; apa_w_ctr := apa_w_ctr + 1)
    {
      declare apa_idx integer;
      declare apa any;
      apa_idx := m_apa_w [apa_w_ctr];
      apa := aref_set_0 (m_apa, apa_idx);
      if (5 = length (apa))
        {
```

```
          declare apa_beg, apa_end, apa_hpctr, apa_hpcount integer;
          apa_beg := apa [1];
          apa_end := apa [2];
          apa_hpcount := length (apa[4]);
          http (subseq (source_UTF8, prev_end, apa_beg), res_out);
          http (sprintf ('<span
xml:id="apjsm%d">', apa_idx), res_out);
          http (subseq (source_UTF8, apa_beg, apa_end), res_out);
          http ('</span>', res_out);
          prev_end := apa_end;
          if (apa_w_ctr > 0)
            http(',', script_out);
          http (sprintf ('\n\t["apjsm%d"\t, [', apa_idx), script_out);
          for (apa_hpctr := 0; apa_hpctr < apa_hpcount; apa_hpctr := apa_hpctr + 1)
            {
              if (apa_hpctr > 0)
                http (',', script_out);
              http (cast (apa[4][apa_hpctr] as varchar), script_out);
            }
          http (']\t, [', script_out);
        -- todo: list service indexes. Do we need this in such a form?
          http ('\t]', script_out);
        }
    }
  http (subseq (source_UTF8, prev_end), res_out);
  http (' ];', script_out);
  http ('\n  var v_descs = [', script_out);
  -- todo list services. Do we need this in such a form?
  http (' ];', script_out);
  http ('\n  var v_links = [', script_out);
  for (app_ctr := 0; app_ctr < app_count; app_ctr := app_ctr + 1)
    {
      declare app any;
      app := m_app [app_ctr];
      if (app_ctr > 0)
        http(',', script_out);
      http (sprintf ('\n\t["%s"\t, "%s"\t, "%s"\t, "%s"\t]', app[3][0], app[3][1], app[3][2], app[3][3]),
    }
  http (' ];', script_out);
  http ('\n-->

</script><script DEFER language="javascript" src="lookup.js"></script>', script_out);
  return
    replace (
      replace (
        string_output_string (res_out),
        '<body', '<body onload="makePopupDivs (v_descs, v_services, v_links)"' ),
      '</html>', string_output_string (script_out) || '</html>' );
}
;

Done. -- 0 msec.

SQL>create function ptext_3 ()
{
  return '<html>
  <head>
  <link href="gogo.css" rel="stylesheet" type="text/css" />
  </head>
  <body>
  <p>The OpenLink Virtuoso team has developers based in Bulgaria</p>
  </body>
</html>';
}
;

Done. -- 10 msec.

SQL>create function ptext_3 ()
{
  return '<html>
  <head>
  <link href="gogo.css" rel="stylesheet" type="text/css" />
```

```
  </head>
  <body>
  <p>The OpenLink <strong>Virtuoso</strong> team has developers based in Bulgaria</p>
  </body>
</html>';
}
;

Done. -- 0 msec.

SQL>insert replacing DB.DBA.SYS_ANN_PHRASE_SET (APS_ID, APS_NAME, APS_OWNER_UID, APS_READER_GID, APS_APC_
values (9, 'Gogo aps', http_dav_uid(), http_admin_gid(), 3, 'x-any', 'Gogo aps env', 10000, 0);

Done. -- 0 msec.

SQL>ap_add_phrases (9,
    vector (
      vector ('OpenLink'                    , vector ('OpenLink'                    , 'http://www.openl
      vector ('OpenLink Virtuoso'       , vector ('OpenLink Virtuoso'   , 'http://virtuoso.openlinksw.com
    ) )
;

Done. -- 0 msec.

SQL>select ap_make_js_menus (vector (9), ptext_3(), 'x-any', 1);

callret
VARCHAR
_______________________________________________________________________________

<html>
  <head>
  <link href="gogo.css" rel="stylesheet" type="text/css" />
  </head>
  <body onload="makePopupDivs (v_descs, v_services, v_links)">
  <p>The OpenLink <strong>Virtuoso</strong> team has developers based in Bulgaria</p>
  </body>
<script DEFER language="javescript"><!--
  var v_descs = [ ];
  var v_descs = [ ];
  var v_links = [
        ["OpenLink Virtuoso"    , "http://virtuoso.openlinksw.com"  , "OpenLink Virtuoso"   , " OpenLink
        ["OpenLink"     , "http://www.openlinksw.com"   , "OpenLink Software"   , " The industry leading
-->

</script><script DEFER language="javascript" src="lookup.js"></script></html>

1 Rows. -- 0 msec.
```

9.34. Geometry Data Types and Spatial Index Support

Virtuoso includes a spatial database capability from version 6.01.3126 onwards. This capability is specially geared towards geographical data in RDF but can be used with SQL also. The functions implemented follow the SQL MM spatial specification where applicable but the entire SQL MM function set is not supported.

Spatial indexing is supported by a two dimensional R tree implementation. The geometries supported may have from two to four dimensions, with a choice of WGS 84 (or similar) latitude and longitude coordinates with haversine distances or a flat 2 dimensional plane for spatial reference system.

9.34.1. Spatial References

The default reference system is WGS-84 with coordinates in degrees of longitude and latitude on Earth. The SRID number for this is 4326 and is assigned by default to all new geometries. `ST_Transform` can be used to re-calculate coordinates of a shape from one spatial reference system to another. `ST_SetSRID` can be used for altering SRID without altering coordinates.

Distances and precisions for "latitude-longitude" spatial reference systems are in kilometers, as approximately calculated with the haversine formula. For other systems they are in the same unit as the coordinates.

9.34.2. Geometric Objects

Virtuoso supports a wide variety of spatial features that can be represented in WKT or SHP: BOX, BOX2D, BOX3D, BOXM, BOXZ, BOXZM CIRCULARSTRING COMPOUNDCURVE CURVEPOLYGON EMPTY GEOMETRYCOLLECTION, GEOMETRYCOLLECTIONM, GEOMETRYCOLLECTIONZ, GEOMETRYCOLLECTIONZM LINESTRING, LINESTRINGM, LINESTRINGZ, LINESTRINGZM MULTICURVE MULTILINESTRING, MULTILINESTRINGM, MULTILINESTRINGZ, MULTILINESTRINGZM MULTIPOINT, MULTIPOINTM, MULTIPOINTZ, MULTIPOINTZM MULTIPOLYGON, MULTIPOLYGONM, MULTIPOLYGONZ, MULTIPOLYGONZM POINT, POINTM, POINTZ, POINTZM POLYGON, POLYGONM, POLYGONZ, POLYGONZM POLYLINE, POLYLINEZ RING, RINGM, RINGZ, RINGZM

Some less popular types are not yet supported: CIRCULARSTRINGM, CIRCULARSTRINGZ, CIRCULARSTRINGZM COMPOUNDCURVEM, COMPOUNDCURVEZ, COMPOUNDCURVEZM CURVE, CURVEM, CURVEZ, CURVEZM CURVEPOLYGONM, CURVEPOLYGONZ, CURVEPOLYGONZM GEOMETRY, GEOMETRYZ, GEOMETRYZM MULTICURVEM, MULTICURVEZ, MULTICURVEZM MULTIPATCH MULTISURFACE, MULTISURFACEM, MULTISURFACEZ, MULTISURFACEZM POLYHEDRALSURFACE, POLYHEDRALSURFACEM, POLYHEDRALSURFACEZ, POLYHEDRALSURFACEZM POLYLINEM SURFACE, SURFACEM, SURFACEZ, SURFACEZM TIN, TINM, TINZ, TINZM

For the sake of speed and scalability, spatial support has some intentional restrictions, that will probably persist for long time.

--- No support for circumpolar shapes. A ring with a pole inside it or on the border of it may be treated as an error or result in wrong calculations of spatial relations. E.g. the st_intersects of two triangles POLYGON((0 89, 120 89, 240 89, 0 89)) and POLYGON((0 88, 120 88, 240 88, 0 88)) may return FALSE. st_intersects of two triangles POLYGON((0 89, 60 90, 120 89, 0 89)) and POLYGON((120 88, 180 90, 240 88, 120 88)) may also return FALSE. Note that Equator is a circumpolar shape.

--- No support for long orthodromic or near-orthodromic arcs. The implementation assumes that shapes reside inside their bounding boxes so st_intersects of two orthodromic arcs, LINE (0 1, 180 1) and LINE (90 1, 270 1) may return FALSE. However big shapes made of small segments (such as a coastline of Eurasia) will be handled correctly. Shapes that are close to 180 longitude get two sets of bounding boxes, shifted 360 one from another, so relations between Alaska and Chukotka will be calculated correctly no matter what offset is used.

--- In order to deal with weird data sources, the range for longitudes is -270 to -540 and the range for latitudes is -90 to +90 for most cases and -96 to +96 for some features such as bounding boxes. If the parsing of a shape did not report range errors for co-ordinates then there's no warranty that all points are in WGS ranges.

--- Peculiarities like self-touch or self-intersection of polygons can be detected, but no warranties are given.

--- Arcstrings are poorly supported and their support will never be optimized for speed. Each arc is supposed to be no longer than 180 degrees. As a result, ESRI circles that consist of two arcs 180+180 are supported whereas ST_Intersects on "ill" circles like 270+90 may return wrong answers.

9.34.3. Precision of Geometries

The internal precision of geometry operations is 64 bit IEEE floating point. Normally, a geometry index for geography uses can be made with 32 bit floats for space efficiency, as these still give a precision of about 2 meters on the surface of the Earth. A geometry object is stored with all coordinates either as float or double depending on which data type was supplied in creating the object. Note that the object itself can have more precise coordinates than the representation of its bounding box in the R tree index.

Bounding boxes for shapes can be calculated approximately as soon as shape contain any arcs. In any case, bounding box of a shape is a bit bigger than the shape itself in order to tolerate rounding errors. Predicates for spatial relations accept an additional, non-standard, argument for tolerance. If the tolerance is greater than zero, borders of shapes are treated as wide lines with thickness is approximately equal to the specified tolerance. That is useful for dealing with inaccurate data such as amateur GPS/GLONASS tracks, coordinates rounded to minutes when published in Dbpedia and the like.

9.34.4. Predicates

```
isgeometry (in x any)
```

Returns 1 if the argument is a geometry.

See Also:

`isgeometry`

9.34.5. Querying Geometric Relations

The SQL MM predicates `st_intersects` , `st_contains` and `st_within` can be used to test whether two geometries overlap in different ways.

`st_intersects` means that the 1st and 2nd arguments have at least one shared point, `st_contains` means that the 2nd argument is fully contained in the 1st argument, `st_within` means that the 1st argument is fully contained within the 2nd argument.

These functions are extended with a third argument which gives a precision. The precision is a number that specifies a maximum distance between two points such that the points will still be considered to overlap. If both the 1st and 2nd arguments are points, then precision can be interpreted as the maximum distance between them.

These do not require the presence of a geometry index but will use one if one is present and one of the geometry arguments is a column on which there is such an index.

9.34.6. Defining a Geometry Index

The below sequence defines a table called geo and a geometry index on its geo column.

```
create table GEO (ID bigint, GEO any, primary key (ID))
 alter index geo on geo partition (id int);

CREATE TABLE GEO_INX ( X real no compress,
                       Y real no compress,
                       X2 real no compress,
                       Y2 real no compress,
                       id bigint no compress,
                       primary key (X, Y, X2, Y2, id))
                       ALTER INDEX geo_inx ON geo_inx PARTITION (id int);

INSERT INTO sys_vt_index ( vi_table,
                           vi_index,
                           vi_col,
                           vi_id_col,
                           vi_index_table,
                           vi_id_is_pk,
                           vi_options)
  VALUES ('DB.DBA.GEO',
          'GEO',
          'GEO',
          'ID',
          'DB.DBA.GEO_INX',
          1,
          'G');

-- Reload the changed schema
__ddl_changed ('DB.DBA.GEO');
```

A geometry index is a table of always five columns, the first 4 are the lower x, y and higher x, y of the object's bounding box, id is the identifier of the object, a foreign key into the table holding the geometries. In a cluster setting the geometry index must be partitioned on the id column. The columns of a geometry index must never be compressed, hence the mandatory no compress declaration.

Geometry indices are declared in the sys_vt_index table, which also declares text indices. The G in the options column declares that this is a geometry index.

A geometry index will be used when appropriate. The table holding the R tree with the bounding boxes should not normally be accessed directly. Explicit selects from a geometry index table with conditions on the columns will not generally work since the collation is not linear as in a regular index. Inserts by means other than geo_insert are not allowed.

9.34.7. Insert and Delete

```
geo_insert (in tb any, in geo any, in id int);
geo_delete (in tb any, in geo any, in id int);
```

These functions insert and delete a geometry from an R tree index. The tb must be a fully qualified name of an R tree table.

These functions will typically be used from triggers on a geometry column. The functions are transactional and make a transaction log record of the action.

See Also:

`geo_insert`

`geo_delete`

9.34.8. Using Geometries in Client Applications and SQL Procedures

In SQL procedures, a geometry is a member of the ANY data type. The `isgeometry ()` function can be used for testing if an object is a geometry. A geometry can only be stored in a column declared to be of type ANY.

If a geometry would be returned to a client application as part of a result set, it is returned as a string containing its WKT text representation. For a client application to pass geometries to the server, it must pass these either as a WKT string parameter to st_geomfromtext or as numeric coordinate values for st_point or related functions. This applies to all clients, ODBC, JDBC, .net and any RDF frameworks.

See Also:

RDF and Geometry

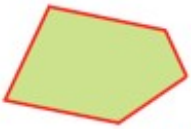
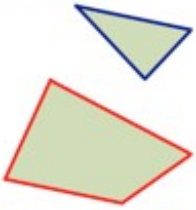
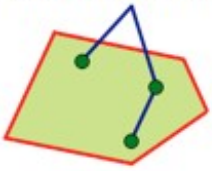
9.34.9. Virtuoso 7.1+ Geo Spatial Data type and function enhancements

As of Virtuoso 7.1+ (open source and commercial) a number of major enhancements in Geo Spatial support have been made to improve the Geometry data types and functions supported, as well as improve compliance with the emerging GeoSPARQL and OGC standards.

Virtuoso Geo Spatial Geometry data types and sample queries

The table below outlines the common WKT (Well Known Text) representations for several types of geometric objects used in RDF:

Figure 9.1. Geo Spatial Geometry Data Types

Geometry Type	WKT representation
Point 	POINT(3 7)
Multipoint 	MULTIPOINT(3 7, 4 2, 8 6)
LineString 	LINESTRING(1 2, 3 6, 9 4)
MultiLineString 	MULTILINESTRING((1 8, 4 4), (4 9, 8 5, 6 2, 1 4))
Polygon 	POLYGON((1 2, 6 1, 9 3, 8 5, 3 6, 1 2))
Polygon (*with hole*) 	POLYGON((1 2, 6 1, 9 3, 8 5, 3 6, 1 2), (3 3, 5 5, 6 2, 3 3))
MultiPolygon 	MULTIPOLYGON(((1 2, 6 1, 9 3, 3 6, 1 2)), ((4 9, 7 6, 9 8, 4 9)))
GeometryCollection 	GEOMETRYCOLLECTION(POINT(4 5), POINT(7 4), POINT(6 2), LINESTRING(4 5, 6 7, 7 4, 6 2), POLYGON((1 2, 6 1, 9 3, 8 5, 3 6, 1 2)))

The following queries "counts the number of items of each type whose coordinates fall within a bounded box shape" for the various RDF geometry data types now supported by Virtuoso. The links are to live examples of the query running against the OpenLink LOD Cloud Server instance.

BOX

```
SELECT ?f AS ?facet COUNT(?s) AS ?cnt
FROM <http://linkedgeodata.org>
WHERE
  {
   ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?f .
   ?s <http://www.w3.org/2003/01/geo/wgs84_pos#geometry> ?p .
   FILTER(bif:st_intersects(bif:st_geomfromtext("BOX(0.3412 43.5141, 9.3412 48.0141)"), ?p))
```

```
  }
GROUP BY ?f
ORDER BY DESC(?cnt)
LIMIT 10
```

- View the SPARQL Query Definition via SPARQL Protocol URL;
- View the SPARQL Query Results via SPARQL Protocol URL

POLYGON

```
SELECT ?f AS ?facet COUNT(?s) AS ?cnt
FROM <http://linkedgeodata.org>
WHERE
  {
    ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?f .
    ?s <http://www.w3.org/2003/01/geo/wgs84_pos#geometry> ?p .
    FILTER( bif:st_intersects( bif:st_geomfromtext( "POLYGON((1 2, 6 1, 9 3, 8 5, 3 6, 1 2))" ), ?p ) )
  }
GROUP BY ?f
ORDER BY DESC(?cnt)
LIMIT 10
```

- View the SPARQL Query Definition via SPARQL Protocol URL;
- View the SPARQL Query Results via SPARQL Protocol URL

POLYGON WITH HOLE

```
SELECT ?f AS ?facet COUNT(?s) AS ?cnt
FROM <http://linkedgeodata.org>
WHERE
  {
    ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?f .
    ?s <http://www.w3.org/2003/01/geo/wgs84_pos#geometry> ?p .
    FILTER( bif:st_intersects( bif:st_geomfromtext( "POLYGON((1 2, 6 1, 9 3, 8 5, 3 6, 1 2), (3 3, 5 5, 6
  }
GROUP BY ?f
ORDER BY DESC(?cnt)
LIMIT 10
```

- View the SPARQL Query Definition via SPARQL Protocol URL;
- View the SPARQL Query Results via SPARQL Protocol URL

MULTIPOLYGON

```
SELECT ?f AS ?facet COUNT(?s) AS ?cnt
FROM <http://linkedgeodata.org>
WHERE
  {
    ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?f .
    ?s <http://www.w3.org/2003/01/geo/wgs84_pos#geometry> ?p .
    FILTER( bif:st_intersects( bif:st_geomfromtext( "MULTIPOLYGON(((1 2, 6 1, 9 3, 3 6, 1 2)), ((4 9, 7 6
  }
GROUP BY ?f
ORDER BY DESC(?cnt)
LIMIT 10
```

- View the SPARQL Query Definition via SPARQL Protocol URL;
- View the SPARQL Query Results via SPARQL Protocol URL

GEOMETRY COLLECTION

```
SELECT ?f AS ?facet COUNT(?s) AS ?cnt
FROM <http://linkedgeodata.org>
WHERE
  {
    ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?f .
    ?s <http://www.w3.org/2003/01/geo/wgs84_pos#geometry> ?p .
    FILTER( bif:st_intersects( bif:st_geomfromtext( "GEOMETRYCOLLECTION( POINT(4 5), POINT(7 4), POINT(6
  }
```

```
GROUP BY ?f
ORDER BY DESC(?cnt)
LIMIT 10
```

- View the SPARQL Query Definition via SPARQL Protocol URL;
- View the SPARQL Query Results via SPARQL Protocol URL

MULTI POINT

```
SELECT ?f AS ?facet COUNT(?s) AS ?cnt
FROM <http://linkedgeodata.org>
WHERE
  {
    ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?f .
    ?s <http://www.w3.org/2003/01/geo/wgs84_pos#geometry> ?p .
    FILTER( bif:st_intersects( bif:st_geomfromtext( "MULTIPOINT(3 7, 4 2, 8 6)" ), ?p ) )
  }
GROUP BY ?f
ORDER BY DESC(?cnt)
LIMIT 10
```

- View the SPARQL Query Definition via SPARQL Protocol URL;
- View the SPARQL Query Results via SPARQL Protocol URL

LINE STRING

```
SELECT ?f AS ?facet COUNT(?s) AS ?cnt
FROM <http://linkedgeodata.org>
WHERE
  {
    ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?f .
    ?s <http://www.w3.org/2003/01/geo/wgs84_pos#geometry> ?p .
    FILTER( bif:st_intersects( bif:st_geomfromtext( "LINESTRING(1 2, 3 6, 9 4)" ), ?p ) )
  }
GROUP BY ?f
ORDER BY DESC(?cnt)
LIMIT 10
```

- View the SPARQL Query Definition via SPARQL Protocol URL;
- View the SPARQL Query Results via SPARQL Protocol URL

MULTI LINE STRING

```
SELECT ?f AS ?facet COUNT(?s) AS ?cnt
FROM <http://linkedgeodata.org>
WHERE
  {
    ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?f .
    ?s <http://www.w3.org/2003/01/geo/wgs84_pos#geometry> ?p .
    FILTER( bif:st_intersects( bif:st_geomfromtext( "MULTILINESTRING((1 8, 4 4), (4 9, 8 5, 6 2, 1 4))" )
  }
GROUP BY ?f
ORDER BY DESC(?cnt)
LIMIT 10
```

- View the SPARQL Query Definition via SPARQL Protocol URL;
- View the SPARQL Query Results via SPARQL Protocol URL

Supported types of shapes

```
BOX, BOX2D, BOX3D, BOXM, BOXZ, BOXZM
CIRCULARSTRING
COMPOUNDCURVE
CURVEPOLYGON
EMPTY
GEOMETRYCOLLECTION, GEOMETRYCOLLECTIONM, GEOMETRYCOLLECTIONZ,
GEOMETRYCOLLECTIONZM
LINESTRING, LINESTRINGM, LINESTRINGZ, LINESTRINGZM
MULTICURVE
```

```
MULTILINESTRING, MULTILINESTRINGM, MULTILINESTRINGZ, MULTILINESTRINGZM
MULTIPOINT, MULTIPOINTM, MULTIPOINTZ, MULTIPOINTZM
MULTIPOLYGON, MULTIPOLYGONM, MULTIPOLYGONZ, MULTIPOLYGONZM
POINT, POINTM, POINTZ, POINTZM
POLYGON, POLYGONM, POLYGONZ, POLYGONZM
POLYLINE, POLYLINEZ
RING, RINGM, RINGZ, RINGZM
```

Not yet supported types

```
CIRCULARSTRINGM, CIRCULARSTRINGZ, CIRCULARSTRINGZM
COMPOUNDCURVEM, COMPOUNDCURVEZ, COMPOUNDCURVEZM
CURVE, CURVEM, CURVEZ, CURVEZM
CURVEPOLYGONM, CURVEPOLYGONZ, CURVEPOLYGONZM
GEOMETRY, GEOMETRYZ, GEOMETRYZM
MULTICURVEM, MULTICURVEZ, MULTICURVEZM
MULTIPATCH
MULTISURFACE, MULTISURFACEM, MULTISURFACEZ, MULTISURFACEZM
POLYHEDRALSURFACE, POLYHEDRALSURFACEM, POLYHEDRALSURFACEZ,
POLYHEDRALSURFACEZM
POLYLINEM
SURFACE, SURFACEM, SURFACEZ, SURFACEZM
TIN, TINM, TINZ, TINZM
```

Virtuoso Geo Spatial geometry functions

The following Virtuoso Geo Spatial geometry functions are available for use in both SQL and RDF Geo Spatial queries. The listed functions are built-in SQL functions. As all built-in functions of Virtuoso, geo-specific functions can be called from SPARQL with prefix `bif:` . For example:

```
bif:earth_radius()
etc.
```

- `earth_radius()` : returns geom.mean of radius of Earth in kilometers, 6367.43568
- `haversine_deg_km (lat1, long1, lat2, long2)` : returns distance between two points on Earth sphere, adjusting radius of sphere to latitudes.
- `dist_from_point_to_line_segment (Xpoint, Ypoint, Xsegment1, Ysegment1,Xsegment2, Ysegment2)` : returns the distance between a point and a segment on a plane.
- `st_point` : returns a point with given coordinates in default SRID. z and/or m may be missing or equal to NULL, indicating the absence of co-ordinate.
- `st_linestring (item1, item2,..., itemN)` : returns a linestring in default SRID, the coordinates of vertices are specified by arguments that are points, 2-, 3- or 4-item vectors of coordinates, linestrings, arcstrings or vectors of the above mentioned values. Repeating vertices are automatically removed, except the case of repeating vertices in the middle of a linestring/arcstring argument.
- `st_x()` : returns X or longitude of a point.
- `st_y()` : returns X or longitude of a point.
- `ST_XMin (shape)` ,

  `ST_YMin (shape)` ,

  `ST_XMax (shape)` ,

  `ST_YMax (shape)` : return boundaries of a bounding box around a shape. Bounding boxes around arcs are calculated in assumption that no one arc is longer than a half of full circle.
- `st_intersects (shape1, shape2 [, proximity])` : checks whether two shapes intersect or some of its points are within the specified proximity. Current version is not complete and does not support arcs of all sorts and rings of polygons, this is fixed in the next release.
- `st_may_intersect (shape1, shape2 [, proximity])` : checks whether bounding boxes of two shapes intersect or some of its points are within the specified proximity. This is much faster than full

  `st_intersects()` check (actually that is the initial part of

  `st_intersects()` execution)
- `(haystack_shape, needle_shape [, proximity])` :

- `st_within (needle_shape, haystack_shape [, proximity])` : checks whether `haystack_shape` contains the `needle_shape` . If proximity is specified, it is treated as an extra wide border around `haystack_shape` . In current version, only a combination of bounding box and a point is supported, the functionality will be extended in the next release.
- `st_distance (shape1, shape2)` : returns the distance in units of plane or in kilometers on sphere.
- `isgeometry (v)` : returns whether the given v is a spatial object.
- `st_astext (shape)` : returns EWKT representation of a shape.
- `ST_SRID (shape)` : returns SRID of shape's spatial reference system or 0 for shape on plane.
- `ST_SetSRID (shape, new_srid)` : replaces the SRID of a shape but does not transform the shape or its coordinates from old SRID to a new one.
- `st_geomfromtext (ewkt_text)` : Parses the string and returns the corresponding geometry.
- `st_ewkt_read (ewkt_text)` : Parses the giwen text as a EWKT and returns the parsed shape.
- `http_st_ewkt (shape, ses)` : Writes a EWKT representation of a shape to the given session, a fast replacement for

  `st_astext (shape)` .
- `http_st_dxf_entity (shape, attrs, ses)` : writes a DXF (Data Exchange Format) representation of shape into the given output session.Attrs is a vector of arbitrary DXF properties in form (tag1, value1, tag2, value2...) where tags are integer codes according to DXF specification, related values are not validated and printed to the session as is. Current version does not support ARCSTRINGs, CURVEs and CURVEPOLYGONs. If shape is null, the function returns without writing anything to the shape.
- `st_get_bounding_box (shape)` : returns BOX2D that is a bounding box of a shape.
- `GeometryType (shape)` : returns EWKT type name of a shape.
- `ST_NumGeometries (shape)` : returns number of members of a MULTI... or ...COLLECTION shape, 1 for other sorts of shapes.
- `ST_GeometryN (shape, idx)` : Given a 1-based index of a member of a MULTI... or ...COLLECTION shape, returns the member.
- `ST_ExteriorRing (polygon_shape)` : Returns an external (the very first) ring of a polygon.
- `ST_NumInteriorRings (polygon_shape)` : returns number of interior rings of the given polygon, NULL if shape is not a polygon.
- `ST_InteriorRingN (polygon_shape, idx)` : Given a 1-based index of an interior ring of a polygon, returns the ring. Wrong index is not reported as an error and NULL is returned.
- `st_get_bounding_box_n (shape, idx)` : Given a 1-based index of a member of a MULTI... or ...COLLECTION shape, returns the bounding box of a member. This is a fast equivalent of

  `st_get_bounding_box (ST_GeometryN (shape,idx))` .
- `ST_Translate (shape, dX, dY [, dZ])` : returns a copy of a shape with all coordinates shifted by the provided dX, dY and dZ.
- `ST_TransScale (shape, dX, dY, Xfactor, Yfactor)` : returns a copy of a shape with all coordinates shifted by the provided dX, dY and then multiplied by Xfactor and Yfactor. In current version, differente values for Xfactor and Yfactor will result in distorted arcs.
- `st_transform_by_custom_projection (shape, algorithm_id, ...)` : Performs a custom projection of shape, using the specified algorithm and algorithm-specific arguments. Current version supports only one algorithm, `st_transform_by_custom_projection (shape, 'OLAEAPS', long_of_center, lat_of_center)` for Oblique Lampert Azimuthal Equal-Area Projection System with the specified center point.
- `ST_Transform (shape, deST_SRID, [orig_proj4_string, dest_proj4_string])` : Transforms the given shape from its current spatial reference system to one specified by `deST_SRID` . Two optional arguments are for "cheating": the SRID of the resulting shape is set to `deST_SRID` but the conversion is done by proj4 using `origin_proj4_string` for projection of original shape and `dest_proj4_string` for the result. If `orig_proj4_string` or `dest_proj4_string` argument is passed but is NULL instead of string, the projection corresponding to original or destination SRID is used. `ST_Transform` is provided by a separate plugin named v7proj4, as described below. When the plugin is loaded, functions like `st_intersects()` support pairs of arguments with different SRIDs by converting coords of second argument into the system of the first one, as required by OGC and GeoSPARQL.
- `postgis_proj_version()` : Returns the version of proj4 in use, as a string, for compatibility with PostGIS.

v7proj4 plugin

The Virtuoso v7proj4 hosted plugin module is required for performing transformation between different co-ordinates systems using the `ST_Transform()` function. The plugin is based on Frank Warmerdam's proj4 library and it's practical to have the proj4 package installed on every box of Virtuoso cluster, even if the build is performed on single machine or on different location

at all. The reason is that the plugin should load data about coordinate systems to work and the simplest way to get the right data from a high-quality source is to use the package.

Compiling open source v7proj4 plugin

The v7proj4 is currently available in the default develop/7 branch of the "v7fasttrack " git repository and can be build as follows:

```
git clone https://github.com/v7fasttrack/virtuoso-opensource.git
cd virtuoso-opensource
./autogen.sh
export CFLAGS="-msse4.2 -DSSE42"
./configure
make -j 24
make install
```

Note first the "proj.4" library must be installed on the system and can be proj.4 download area, which the configure script will detect the installation of enabling the v7proj4 plugin library to be built in ~/libsrc/plugin/.libs:

```
ls libsrc/plugin/.libs/v7proj4*
libsrc/plugin/.libs/v7proj4.a
libsrc/plugin/.libs/v7proj4.la
libsrc/plugin/.libs/v7proj4.lai
libsrc/plugin/.libs/v7proj4_la-import_gate_virtuoso.o
libsrc/plugin/.libs/v7proj4_la-sql_v7proj4.o
libsrc/plugin/.libs/v7proj4_la-v7proj4_plugin.o
libsrc/plugin/.libs/v7proj4.so
libsrc/plugin/.libs/v7proj4.ver
```

Installation and Configuration of v7proj4 plugin

When the plugin (v7proj4.so) is built, it needs to be added to the [Plugins] section of Virtuoso configuration file (virtuoso.ini or the like), also on every node of a cluster:

```
[Plugins]
LoadPath = ./plugins
Load2    = plain, v7proj4
```

If everything is fine then the virtuoso.log file will contain something like the following lines after the next startup:

```
21:30:10 { Loading plugin 1: Type `plain', file `shapefileio' in `.'
21:30:10   ShapefileIO version 0.1virt71 from OpenLink Software
21:30:10   Shapefile support based on Frank Warmerdam's Shapelib
21:30:10   SUCCESS plugin 1: loaded from ./plugins/shapefileio.so }
21:30:10 { Loading plugin 2: Type `plain', file `v7proj4' in `.'
21:30:11   plain version 3208 from OpenLink Software
21:30:11   Cartographic Projections support based on Frank Warmerdam's
proj4 library
21:30:11   SUCCESS plugin 2: loaded from ./plugins/v7proj4.so }
21:30:11 OpenLink Virtuoso Universal Server
21:30:11 Version 07.10.3208-pthreads for Linux as of Mar 31 2014
...
21:30:28 PL LOG: Initial setup of DB.DBA.SYS_V7PROJ4_SRIDS data from
files in "/usr/share/proj"
21:30:30 PL LOG: DB.DBA.SYS_V7PROJ4_SRIDS now contains 6930 spatial
reference systems
...
21:30:32 Server online at 1720 (pid 9654)
```

To store descriptions of coordinate systems, the plugin creates a table:

```
create table DB.DBA.SYS_V7PROJ4_SRIDS (
 SR_ID integer,
 SR_FAMILY varchar not null,
 SR_TAG varchar,
 SR_ORIGIN varchar not null,
 SR_IRI IRI_ID_8,
 SR_PROJ4_STRING varchar not null,
 SR_WKT varchar,
 SR_COMMENT varchar,
```

```
 SR_PROJ4_XML any,
 primary key (SR_ID, SR_FAMILY) )
;
```

and fills it with data from files epsg, esri, esri.extra, nad83 and nad27 of directory /usr/share/proj. Note these files must exist in the /usr/share/proj directory otherwise a message will be reported in the log file indicated the file could not be found. Every row of the table is identified with name of "family" of coordinate systems and an integer SRID. Different sources may assign same SRID to different reference systems, however descriptions of well-known systems match exactly or the difference is not noticeable for any practical application.

The loading process uses family names 'EPSG', 'ESRI', 'NAD83' and 'NAD27'. When the `ST_Transform()` searches for a coordinate system that corresponds to a given SRID then it returns first record found while checking the following families in the following order: 'PG', 'EPSG', 'ESRI','NAD83', 'NAD27'. Thus it is practical to put all custom definitions in 'PG' family, thus they will get the highest priority.

A sample EPSG file containing the mapping for the proj.4 EPSG:4326 coordinate system is:

```
$ cat /usr/share/proj/epsg
<4326>+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs<>

SQL> SELECT * FROM DB.DBA.SYS_V7PROJ4_SRIDS;
SR_ID             SR_FAMILY          SR_TAG    SR_ORIGIN              SR_IRI    SR_PROJ4_STRING
INTEGER NOT NULL  VARCHAR NOT NULL   VARCHAR   VARCHAR NOT NULL       VARCHAR   VARCHAR NOT NULL
_____________________________________________________________________________________________

4326              EPSG               4326      /usr/share/proj/epsg   NULL      +datum=WGS84 +ellps=WGS84

1 Rows. -- 0 msec.
SQL>
```

There are two procedures are available for loading more co-ordinate systems:

1. `DB.DBA.V7PROJ4_LOAD_SYS_SRIDS` : it is called at server startup if the v7proj4 plugin is loaded:

   ```
   DB.DBA.V7PROJ4_LOAD_SYS_SRIDS (
     in projdir varchar := '/usr/share/proj',
     in only_if_empty_table integer := 0
   )
   ```
2. `DB.DBA.V7PROJ4_LOAD_INIT_FILE` : it is lower-level procedure:

   ```
   DB.DBA.V7PROJ4_LOAD_INIT_FILE (
     in path varchar,
     in _sr_family varchar
   )
   ```

The main part of `DB.DBA.V7PROJ4_LOAD_SYS_SRIDS()` is a sequence of:

```
DB.DBA.V7PROJ4_LOAD_INIT_FILE (projdir || '/epsg', 'EPSG');
DB.DBA.V7PROJ4_LOAD_INIT_FILE (projdir || '/esri', 'ESRI');
DB.DBA.V7PROJ4_LOAD_INIT_FILE (projdir || '/esri.extra', 'ESRI');
DB.DBA.V7PROJ4_LOAD_INIT_FILE (projdir || '/nad83', 'NAD83');
DB.DBA.V7PROJ4_LOAD_INIT_FILE (projdir || '/nad27', 'NAD27');
```

Rows with same SRID but different `SR_FAMILY` may exist in the table, however only one projection per SRID is used and `SR_FAMILY` defines the priority. The internal search query for projection by SRID is:

```
SELECT COALESCE
   (
     ( SELECT SR_PROJ4_STRING FROM DB.DBA.SYS_V7PROJ4_SRIDS WHERE SR_ID=:0 AND SR_FAMILY='PG'),
     ( SELECT SR_PROJ4_STRING FROM DB.DBA.SYS_V7PROJ4_SRIDS WHERE SR_ID=:0 AND SR_FAMILY='EPSG'),
     ( SELECT SR_PROJ4_STRING FROM DB.DBA.SYS_V7PROJ4_SRIDS WHERE SR_ID=:0 AND SR_FAMILY='ESRI'),
     ( SELECT SR_PROJ4_STRING FROM DB.DBA.SYS_V7PROJ4_SRIDS WHERE SR_ID=:0 AND SR_FAMILY='NAD83'),
     ( SELECT SR_PROJ4_STRING FROM DB.DBA.SYS_V7PROJ4_SRIDS WHERE SR_ID=:0 AND SR_FAMILY='NAD27')
   )
```

so for `ST_Transform()` function 'PG' overrides everything else, EPSG is the next highest priority, then ESRI, NAD83 and NAD27. However custom queries and procedure may select whatever they please, including sr families not listed here or strings

from other tables etc., and feed projection strings directly to `ST_Transform()` .

The co-ordinate systems can also be updated by direct manipulations on `DB.DBA.SYS_V7PROJ4_SRIDS table` (the table it is readable for public and writable for DBA only). When the table is edited, call function "`Proj4 cache_reset()` " to prevent SQL runtime from using previously prepared projections that might become obsolete after changes in the table. Note that proj4 projections are for normalized data in radians whereas Virtuoso stores shapes using numbers that come from WKT, i.e. they're latitudes and longitudes in degrees for almost all cases.

The v7proj4 plugin automatically applies `RAD_TO_DEG` multiplier before conversion and/or `RAD_TO_DEG` multiplier after conversion when source and/or destination coordinate systems are latitude-longitude or geocentric. Even if this conversion is done automatically, you should remember that it happens, for the following reason. Many "how-to" instructions for spatial data sets contain paragraphs like "how to convert these data to WGS-84" and sample C/C++ code contains transformations like `{ x *= RAD_TO_DEG; y *= RAD_TO_DEG; }` . These transformations will probably be redundant in the corresponding Virtuoso/PL code whereas proj4 strings can be used unchanged and passed as 3rd and 4th arguments of `ST_Transform()` function. If degrees-to-radians conversion is made twice then the data can be calculated as if the shape is located in a totally different place of ellipsoid. If the post-transformation radians-to-degrees conversion is also made twice, the resulting shape may look like the real one but coordinates may be tens kilometers away from correct values.

ST_Transform() Example

Below are some example uses of the `ST_Transform()` function to transform some of the sample coordinate systems load into Virtuoso:

```
SQL> SELECT * FROM DB.DBA.SYS_V7PROJ4_SRIDS;
SR_ID               SR_FAMILY           SR_TAG    SR_ORIGIN               SR_IRI    SR_PROJ4_STRING
INTEGER NOT NULL    VARCHAR NOT NULL    VARCHAR   VARCHAR NOT NULL        VARCHAR   VARCHAR NOT NULL
_____________________________________________________________________________________________

2005                EPSG                2005      /usr/share/proj/epsg    NULL      +ellps=clrk80 +k=0.99950
2249                EPSG                2249      /usr/share/proj/epsg    NULL      +datum=NAD83 +ellps=GRS8
4326                EPSG                4326      /usr/share/proj/epsg    NULL      +datum=WGS84 +ellps=WGS8

3 Rows. -- 1 msec.
SQL> SELECT ST_Transform (st_geomfromtext ('POLYGON((-16 20.25,-16.1 20.35,-15.9 20.35,-16 20.25))'), 1,
unnamed
VARCHAR NOT NULL
_____________________________________________________________________________

SRID=1;POLYGON((-1495284.211473 1920596.789917,-1504629.737795 1930501.842961,-1485938.685152 1930501.842

1 Rows. -- 0 msec.
SQL> SELECT ST_AsText(ST_Transform(ST_GeomFromText('POLYGON((743238 2967416,743238 2967450, 743265 296745
wgs_geom
VARCHAR NOT NULL
_____________________________________________________________________________

POLYGON((-71.177685 42.390290,-71.177684 42.390383,-71.177584 42.390383,-71.177583 42.390289,-71.177685 4

1 Rows. -- 1 msec.
SQL>
```

9.35. SQL Bulk Load, ELT, File Tables and Zero Load Operations

9.35.1. File Tables

Virtuoso supports mapping files of comma separated values as tables from version 7.00.3206. This makes bulk load of relational data straightforward and very fast. This also makes it possible to query data in files without any loading into the database. The latter is specially convenient if data is queried just once, where loading and subsequently dropping the data would be needless overhead. Also arbitrary ETL transformations can be expressed in SQL, reading from a file table and inserting into a database table. This includes the whole range of SQL functionality, including intermediate aggregations and the like.

This feature deprecates the CSV load functions in previous versions, e.g. csv_load.

Queries can freely mix file tables and regular tables. Joins with file tables make hash join plans where the smaller file is copied into a memory resident hash table. For very large hash tables, partitioned hash join can be used, thus arbitrary joining between

files is possible, fast and convenient.

A file table is declared with a regular create table statement. If the table definition includes a primary key, the file is assumed to be sorted as declared. In most cases a primary key should not be declared when creating a file table. After the create table statement, the procedure fs_set_file declares that the table is in fact a file. In a cluster setting the file should be visible to all the server processes via a shared file system.

For example, to bulk load the TPC-H part table:

```
CREATE TABLE PART_F (
    P_PARTKEY     INTEGER NOT NULL,
    P_NAME        VARCHAR(55) NOT NULL,
    P_MFGR        CHAR(25) NOT NULL,
    P_BRAND       CHAR(10) NOT NULL,
    P_TYPE        VARCHAR(25) NOT NULL,
    P_SIZE        INTEGER NOT NULL,
    P_CONTAINER   CHAR(10) NOT NULL,
    P_RETAILPRICE double precision NOT NULL,
    P_COMMENT     VARCHAR(23) NOT NULL    );

ft_set_file ('PART_F', 'src/part.tbl', delimiter => '|');

-- turn off logging and set insert mode to auto committing, non-transactional
log_enable (2);

-- read the file and insert into the previously create part table
insert into part select * from part_f;
```

The ft_set_file procedure takes the table name as first argument and a file system path relative to the server's working directory as the second argument. Optional arguments allow specifying a delimiter, newline character and escape character.

```
create procedure ft_set_file (in tb varchar, in fname varchar, in delimiter varchar, in newline v
```

Creating a file table requires dba group privileges and the file is subject to the file system access limitations that apply to file_to_string and other file system access functions.

The newline and escape characters need to be single character strings. A newline or escape character following the escape character is added to the parsed input without its special interpretation.

Each column in the CSV file is expected to end with the delimiter character. A field of zero length is considered a SQL NULL value, i.e. if two delimiters are adjacent or if a line begins with the delimiter, the field is considered NULL. The text in the field is parsed according to the data type declared for the column whose position in the create table corresponds to the field position on the line. The parsing is as by the SQL cast function from a varchar value. If the cast fails the line is silently ignored.

When a table is declared as a file table, the file is sampled and statistics are written into the sys_col_stat table. In this way the system is capable of making correct query plans involving joins of file tables. For this reason, the file should exist and have the relevant content when the file table is created.

A file table will be read in parallel on multiple threads if a normal table would be read in parallel in the same situation. In a single server, the ThreadsPerQuery (enable_qp in sys_stat) setting controls the number of threads used. In a cluster setting each elastic slice corresponds to a fraction of the file. This is why the file is expected to be visible from all the servers.

If a table had previous content and subsequently was declared a file table, the previous content will no longer be visible. The file table may cease to be a file table by the dba deleting the corresponding row from sys_file_table and calling __ddl_changed on the file table. At this point the previous, databases resident content of the table will be visible again. However the statistics gathered from the file will still shadow the statistics of the table.

It is often the case that many CSV files have the same structure. For these cases it is sufficient to create a single table and attach it to one of the files with ft_set_file. This file will provide the statistics. In order to read a different file, one can use the FROM clause in the TABLE OPTION clause of in the FROM clause of a select statement, as follows:

```
-- get the count of well formed part rows from the part.tbl file
SELECT COUNT (*)
  FROM part_f;
```

```
-- do the same from src/part.tbl.2
SELECT COUNT (*)
  FROM part_f TABLE OPTION (FROM 'src/part.tbl.2');
```

The value of the FROM option can be an arbitrary expression, however it must be independent of values bound by the containing statement. It may depend on variables bound in an enclosing stored procedure or from parameters but it cannot join to a column of another table in the same select statement.

Arbitrary search conditions can be applied to file tables, for example:

```
SELECT p_name, p_size, p_partkey
  FROM part_f
 WHERE p_name LIKE '%green%';
```

Joins are possible, for example:

```
SELECT SUM (ps_availqty)
  FROM partsupp, part_f
 WHERE ps_partkey = p_partkey
   AND p_name LIKE '%green%';
```

This would join the partsupp table to the part file.

File tables are not cached by the DBMS, they are read from the OS every time they are needed. A single query will not read the same file multiple times, unless in the case of a partitioned (multtipass) hash join. Nested loop joins are not applied to file tables. File tables cannot be updated and no transaction control applies to them. Select permissions can be granted as for any other table.

9.35.2. Parallel Insert With File Tables and Transactions

A file table is copied into a database resident table with an INSERT... SELECT statement. Such a statement executes in parallel if the session is in auto commit mode, i.e. log_enable (2) or log_enable (3) has been previously executed on the session.

A file can be loaded inside a transaction if the connection is not in auto commit log_enable (2) or 3. This will be multithreaded if enable_mt_txn is 1 in the [Flags] section of the ini file or __dbf_set ('enable_mt_txn', 1) has been executed previously. The setting is global. Defaults vary according to server version. Use sys_stat ('enable_nt_txn') to read the value of the setting.

For long files the transaction is liable to run out of rollback space. File table operations as such do not affect the transaction context. Explicit commits may be interspersed in a select statement from a file or other tables.

For example, the history keeping dimension updates from TPC DS can be implemented as follows. The item table is a history keeping dimension that has an index on the i_item_id business key and has a primary key of item surrogate key, with a new value for each version of the item record. The item record has a start and end date (i_rec_start_date, i_rec_end_date) to mark the period of validity of the information. A null value in item_rec_end_date marks the currently applicable record. When the item data is updated, the, a new item is inserted and the previously current item record has its end date set to the current date. These operations must occur atomically. Otherwise the implementation may choose whether to update many item records in the same transaction.

In the below listing most columns have been left out for brevity:

```
CREATE PROCEDURE item_update (in i_id varchar,...)
{
  vectored;

  UPDATE item
     SET i_rec_end_date = curdate
   WHERE i_id = i_id
     AND i_rec_end_date IS NULL;

  INSERT INTO item (i_ietm_sk, i_item_id, i_rec_end_date,...)
           VALUES (sequence_next ('item_sk_seq'), i_id, NULL,... )
  not vectored { COMMIT WORK};
}

SELECT COUNT (item_update (i_item_id,....)
  FROM item_f....;
```

The select statement call the item_update procedure on a vector of item ids and other columns. The procedure marks the expired record and inserts the new record, assigning new surrogate keys from a sequence. After each batch it performs one commit. The next batch of items are updated in a separate transaction.

This should be run on a single thread. In a multithreaded transaction the threads may not issue individual commits. The code could be multithreaded by leaving out the commit from the stored procedure. Then the commit would have to be after the completion of the select statement.

The following isql script bulk loads a whole TPC-H database. We leave out the create tables and ft_set_files for brevity, they are all as in the part example above.

```
log_enable (2); INSERT INTO lineitem SELECT * FROM lineitem_f &
log_enable (2); INSERT INTO orders   SELECT * FROM orders_f &
log_enable (2); INSERT INTO customer SELECT * FROM customer_f &
log_enable (2); INSERT INTO part     SELECT * FROM part_f &
log_enable (2); INSERT INTO partsupp SELECT * FROM partsupp_f &
log_enable (2); INSERT INTO supplier SELECT * FROM supplier_f &
log_enable (2); INSERT INTO nation   SELECT * FROM nation_f &
log_enable (2); INSERT INTO region   SELECT * FROM region_f &

wait_for_children;
checkpoint;
```

A multithreaded, non-logged, non-transactional insert is started for each table-file pair as a background task. The wait_for_children isql command waits for all the background tasks to complete. The checkpoint statement makes the state durable. Killing the server in before the checkpoint would result in the server starting in a state with none of the effects of the bulk load present, since the log_enable(2) turns off logging. The database is online during the bulk load and the progress may be followed by periodically counting the tables, for example.

Chapter 10. Virtuoso Cluster Programming

Abstract

This chapter contains cluster programming aspects such as explaining the basics of how SQL queries work on clustered Virtuoso, sql options, Sequences and identity columns, Distributed Pipe constructions and many others. It also describes RDF tables partitioning and troubleshooting.

These sections apply to Virtuoso as of version 6.0.

10.1. Cluster SQL Execution Model

This section explains the basics of how SQL queries work on clustered Virtuoso.

Query optimization for cluster is similar to query optimization for a single process. The main issues of optimization have too do with join order, index choice and join type.

Still, the performance characteristics of a distributed memory cluster are radically different from a single process database. Namely, the cost of a network round trip between nodes, even if these were only different processes on a shared memory multiprocessor, is between 5 and 50 single row lookups from a big table, supposing the row being sought for is in memory. The 5x factor applies when within the same machine, the 50 times factor applies over 1Gbit ethernet.

From this follows that a single network message must process several hundred rows and several messages must be concurrently dispatched to multiple nodes for clustering to yield any benefit in terms of a single query's processing time.

Of course, a cluster can have essentially unlimited RAM, which is not the case of a single machine and a network round trip remains faster than a disk random access. For serial throughput, the disk controller is several times faster than the network, though. Of course, with large numbers of concurrent users, the latencies get absorbed and a high aggregate throughput can be attained.

Still, the first question of any distributed memory system is the aggregation of messages. This can be done in two complementary ways: By Sending a large number of simple operations in a single message, preferably asynchronously to multiple recipients and on the other hand by putting more complex operations into a message.

Also, a random access pattern can be transformed to a serial access pattern by using hash join, i.e. making a temporary copy of an index or part thereof. In this way, the query does not have to go to the data more than once even if it is used inside a tight loop in the query plan.

We will use the TPC H tables to illustrate some points below:

When performing a loop join, Virtuoso will run as many iterations as possible in parallel. Consider a join like:

```
select o_date
  from nation, customer, order
 where o_custkey = c.custkey
   and c_nationkey = n_nationkey
   and n_name = 'FRANCE';
```

We presume the indices:

```
create index c_nk on customer (c_nationkey) partition (c_nationkey int)
create index o_ck on orders (o_custkey) partition (c_custkey int);
```

This gets the order dates for order by customers in France. This is typically a loop join since this is expected to access a fraction of orders at random. First we get the n_nationkey for France. Then we get the c_custkey for French customers. This is a sequential read of an index and produces a set of c_custkey values. Given these, we join to orders using the o_ck index. This index produces a set of o_orderkey values for each o_custkey. These are then used for getting the o_date's from the actual order row.

If the batches are large enough, the query will execute in 3 network round trips: One to get the set of c_curstkey's for France, one to get the o_orderkey's from the o_custkey's and one for getting the o_date for the o_orderkey's. The nation table for getting the n_nationkey for the country name is small and can be replicated on all nodes so this does not count as a round trip.

The first round trip goes to one partition. The resulting c_custkey's will be likely spread over all partitions, so each applicable node gets a message with multiple lookups of the o_ck inde, each with a different o_custkey value. The replies are collated and this time partitioned by o_orderkey and sent as parallel batches to the nodes that hold the actual order rows.

The batch size is settable but is anyway over 1000 operations. The total CPU time for running the query is slightly longer than running the query serially in a single process but the real time is likely shorter because the load is concurrently carried by multiple processes.

Now, if we just wanted the date of the most recent order from France, we could write:

```
select max (o_date)
  from nation, customer, order
 where o_custkey = c.custkey
   and c_nationkey = n_nationkey
   and n_name = 'FRANCE';
```

This would evaluate as explained above but would not return the dates but rather just remember the latest one on each node. At the end, just the latest dates from each cluster node accessed in the query would be returned to the node running the query. Further, SQL specifies that running the same query on the same data is expected to return the same order of results on consecutive runs, even if no order by is present. This means that indices must be traversed in order, i.e. the data received from multiple partitions in indefinite order must be deterministically assembled to provide a predictable order. This is important if slicing the result set for example. When returning an aggregate this condition is relaxed and data can be processed in whatever order received, thus increasing parallelism.

Another optimization is taking advantage of co-located indices.

```
select sum (l_extendedprice)
  from lineitem, orders, customer
 where l_orderkey = o_orderkey
   and o_custkey = c_custkey
   and c_comment like '%fluffily%';
```

This time we scan customers linearly, lookinf for 'fluffily' in the comment field. Finding one, we look for the orders. But we are already in the partition of the c_custkey and the o_ck index of orders is partitioned on this same value, thus we know that the o_orderkey is in fact in the same partition, so no round trip is needed. Then we just send the o_orderkey to the appropriate partition for l_orderkey to get the sum of the extended prices of lines for this customer. At the end of the query, all nodes hosting lineitems will have their local sum and these are just added up and returned.

As long as queries are written as joins, they will run with good parallelism. If they are written as procedure loops and single row lookups from partitioned tables, performance will be an order of magnitude worse.

The join type and join order switches work identically with cluster and single process databases.

10.2. Sequences, Identity and Registry

Sequences and identity columns have a cluster-wide scope. Thus, an identity column can be used as a primary key and partitioning column and the system guarantees that there will be no duplicates.

Sequence numbers are signed 64 bit integers.

The sequence numbers are locally ascending on each node. When a cluster node first requests a sequence number, it is assigned a block of numbers from which it will assign subsequent numbers. Thus, two nodes will allocate from different ranges. The global order is not necessarily ascending but numbers stay unique.

The master node of the cluster keeps a master sequence for each sequence. This is used for allocating blocks of values. The so called:

```
__NEXT__<seq>
```

The sequence itself holds the local next value. The

```
__MAX__<seq>
```

sequence holds the end of the value range allocated for <seq> at the local node.

Allocating a batch of sequence numbers is synchronously logged on both the master node and the requesting node. Thus, upon roll forward recovery, this information survives even if the transaction requesting the numbers was never committed.

The sequence_next function has an additional optional argument for specifying how many values should be requested from the master node at a time.

```
sequence_next (sequence_name[, increment[, batch]])
```

The increment defaults to 1 and the batch to 1000.

Specifying the batch and increment to be equal will get a globally ascending sequence order but will need a round trip to the master for each number.

The registry_get and related functions operate on purely local content. As with single process databases, the registry is used for storing some schema information, sequence values and global application flags.

10.3. SQL Options

For purposes of debugging or writing stored procedures that are specifically meant to work with local data only, it is useful to disable cluster functionality.

This is done with the NO CLUSTER table option. This can be used in the table option clause of a table in a FROM or in an update or delete.

Specially when writing procedures to be called with DAQ, see below, it us necessary to ensure that the procedures will not access data outside of the host running them.

Examples:

```
select count (*) from x table option (no cluster);
update x table option (no cluster) set y = y + 1;
update x set y = 1 where current of cr option (no cluster)
insert into xx key xx option (no cluster) (c1, c2) values (1, 2);
```

Note that for positioned (where current of) deletes and updates the option is at the end. For searched ones it is after the table. Use the explain () function to see the compilation of statements to make sure about the locus of execution. If a table is to be accessed across the cluster, this is indicated in the output.

All other SQL options work as with single server databases.

10.3.1. Parallel INSERT Options

Searched updates and deletes can be parallelized as they are written. However, inserts in loops are not as obviously parallelizable. Therefore insert has an option for queueing multiple inserts into a queue for partitioned parallel execution.

The format of the statement is:

```
INSERT INTO table [KEY <key>] [OPTION (INTO <daq>) <column list> <values or select>
```

The DAQ is a distributed async queue object, see section on calling procedures in cluster. Specifying the INTO daq option partitions the insert, determining where each key should go and buffers the operation into the DAQ. When the result of the DAQ is fetched, all the buffered operations are sent in parallel, as a single message per node concerned. This is easily tens or hundreds of times more efficient than inserting row by row. The transactional behavior is controlled by the DAQ, see the section on calling procedures in cluster. Inserts can mix with other types of operations in the DAQ. If the DAQ is transactional, any failed insert, as in the case of uniqueness violation, will make the transaction uncommittable, preserving integrity.

Example

```
create procedure ct_fill_daq (in n1 int, in n2 int, in st int := 1)
{
 declare daq any;
 declare ct int;
 daq := daq (1);
 for (ct := n1; ct < n2; ct:=ct+st)
 {
 insert into CT option (into daq) (ROW_NO, DT) values (ct, cast (ct as varchar) || ' xxxxxxxxxxxxxx');
 }
 while (daq_next (daq));
}
```

10.3.2. INSERT KEY Option

When indices are partitioned on different columns, indices pertaining to a single row can be located on different nodes. In special situations, for example when exploiting co-location of keys from different tables, one may wish to insert a things index by index. This happens with the use of partitioned functions with DAQ's. Inserting some indices and not inserting others will make an inconsistent database, thus even if keys are inserted separately, all keys should be inserted within the same transaction or have some other guarantee of getting all inserted. Also, application code will have to change if the indices change. This is not considered good practice.

Consider a partition account table and a history table both partitioned on the same key.

We could make a procedure updating the balance of the account and recording the event.

```
create table acct (id int, balance numeric);
alter index acct on acct partition (a_id int);
create table hist (h_id int, ts timestamp, note varchar, primary key (h_id, ts));
alter index hist on hist partition (h_id int);

create procedure evt (in a_id int, in delta numeric, in note varchar)
{
 update acct table option (no cluster) set balance = balance + delta where id = a_id;
 insert into hist key hist option (no cluster) (h_id, note) values (a_id, note);
)
```

Such a procedure would be called on a particular partition using a DAQ In practice, the RDF store uses single key operations for atomically reserving ID's for IRI's, for example..

10.4. Calling Procedures in Cluster

Normally, all interprocess communication in the cluster is transparent. In special cases, the developer may wish to execute a given procedure on a given host of the cluster. This is typically the case when there is affinity between data and logic.

A regular stored procedure or trigger is executed on the host where it is invoked. With the distributed async queue (DAQ) system one can execute procedures on specified remote hosts.

Procedures invoked over DAQ are restricted to dealing with data that is held on the host where they execute. Generic procedures or triggers may use any data from anywhere.

The DAQ offers a mechanism corresponding to the map operation of the map-reduce paradigm. The DAQ may or may not be transactional. In the transactional mode, all the invocations are enlisted into a distributed transaction. The transaction is to be committed separately. The non-transactional DAQ will not enlist operations into a distributed transaction and will automatically

commit after each procedure that executes without error and rollback after each procedure that terminates in an error.

A DAQ is created with the SQL function daq ().

```
daq (in is_txn int) returns any
```

This returns a DAQ object. The object may be passed by value or reference but cannot be returned to a SQL client or passed to another thread or persisted in a table. The DAQ object is a handle for making remote procedure calls and getting their results.

```
daq_call ( in daq any,
           in part_table varchar,
           in part_key varchar,
           in proc varchar,
           in args any,
           in is_update int) returns any
```

This function applies proc to the vectors of args on the host determined by using the partitioning of part_key of part_table designated by args.

Let us suppose that the index part_key of part_table has its 1st and 2nd key part as partitioning keys. Then the 1st element (index 0) of args would be used as the value of the 1st key part and the 2nd element as value of the 2nd key part for determining the relevant partition.

Thus the argument list must be congruent with the layout of the partitioning key. The table or partition key is not referenced otherwise, it is just used for deciding which host gets the call.

If the partition is stored in multiple places, the is_update flag decides whether all will get the call or if only one is picked.

After the host(s) that will get the call are known, the call is queued for execution. The daq_call function returns a vector of two elements. The first is the sequence number of the call. This can be used for associating results to calls. The second element is the list of hosts that got the call.

Each call on each host will produce one or more results. These results are accessed with the daq_next function.

```
daq_next (in daq any) returns any
```

This function returns a result from one of the calls queued on the daq.

The return value can be one of the following:

- 0 - This means that all results have been returned and no more data is available.
- vector with request number, host number and response.

The response is one of:

- Vector with 1 as 0th element - The function returned a result set row. The row is a nested vector as next element.
- Vector with 2 as 0th element. The next element is the return value of the function, the function executed with no error.
- Vector with 3 as 0th element. The next element is a vector describing the SQL error that terminated the function. The error is a nested vector with the constant 3, the SQL state and the text of the message.

If the DAQ is transactional, the first error on the remote host terminates processing. This makes the distributed transaction uncommittable, thus rolling it back at the next transact.

If the DAQ is not transactional, all requests are processed and the state is committed after each function call as a local single phase commit on the processing host. if the function signals an error, the transaction is locally rolled back.

There is no guaranteed order of execution. The calls are buffered until the first call to daq_next. daq_next will return whatever data is first available. If no data from cluster peers is available, it will execute requests that were assigned to the local host. These are executed on the calling thread.

Example

```
create procedure daq_results (in daq any)
```

```
{
 declare row any;
 for (;;)
 {
 row := daq_next (daq);
 if (0 = row)
 return;
 dbg_obj_print (row);
 }
}

create procedure clexec_srv (in str varchar)
{
 exec (str);
}

create procedure clexec (in str varchar)
{
 declare d any;
 d := daq (0);
 daq_call (d, 'DB.DBA.SYS_COLS', 'SYS_COLS_BY_NAME', 'DB.DBA.CLEXEC_SRV', vector (str), 1);
 daq_results (d);
}
```

The clexec procedure will execute the given SQL string on all hosts of the cluster. It returns after all have completed.

10.5. Partition Functions

Given a key and a set of values, the partition function can determine which cluster nodes hold the value.

```
partition_list ( in table_name varchar,
                 in key_name varchar,
                 in list any,
                 in is_update int)
```

The table name is a case sensitive full name of a table as it appears in SYS_KEYS. The key_name is the case sensitive name of the index. The values are key part values in the index order. The is_update, if non-zero, specifies that if the value is stored in multiple places, all are to be returned, otherwise just one is picked at random, preferring the local if there is a local copy of the partition.

The value is a list of node numbers, corresponding to the Host<n> entries in the cluster.ini file.

Example:

```
select partition_list ('DB.DBA.CT', 'CT', vector (2), 1);
```

10.6. Distributed Pipe

A distributed pipe is a single construct that can be used for map-reduce and stream transformation. It is a further development of the DAQ.

A dpipe is an object which accepts a series of input rows and generates an equal amount of output rows. It may or may not preserve order and it may or may not be transactional. The input row of a dpipe is a tuple of values. To each element of the tuple corresponds a transformation. The transformation is expressed as a partitioned SQL function, basically a function callable by daq_call, with arguments specifying the partition where it is to be run. The output row is formed by gathering together the transformation results of each element of the input tuple.

Conceptually, this is like a map operation, like running several DAQ's, one for each column of the dpipe. A transformation function does not always need to produce a value. It may also produce a second partitioned function call with new arguments which will be partitioned and scheduled by the dpipe. Since the second function is independently partitioned, this may be used for implementing a reduce phase. This phase may then return a value and/or further functions to be called.

Ultimately, the dpipe functions are all expected to return a value. When each function of the input row has returned a value, the output row is ready and can be retrieved. Returning a value is a way for the application to synchronize with the operation of the dpipe. The dpipe can be used for mapping values or side effects or both.

The dpipe object is created with the dpipe SQL function. This takes a set of flags and a list of previously declared dpipe transformation names. A dpipe transformation is declared by specifying a SQL procedure, a partitioning key and a set of flags.

The dpipe is fed data with the dpipe_input function. This takes the dpipe object and one value per transformation specified when the dpipe was made. The values can be arrays, so that a transformation function can logically have multiple arguments. The results of the dpipe are retrieved with the dpipe_next function,. This returns a new result row at each subsequent call until one row has been returned for each input row. Depending on flags, these may be returned as they become available or in the original order. Each input is substituted with the transformation result in the output row. Some transformation can be identity so as to preserve a row id for correlating inputs and outputs if they are not processed in order.

After this, the dpipe can be reused. The dpipe is freed when there are no longer references to it.

Finally, since transformations may, in addition to producing values, also produce other functions to be called for their effects, dpipe_reuse is called to make sure that all these functions are run until no more operations are left.

10.6.1. SQL optimization and Dpipe

Calls to SQL functions in queries can be translated to dpipe operations if a dpipe equivalent is declared. Consider a select statement returning the values of a user defined function of some column selected in the query. If this function itself accesses the database, possibly specially if the data accessed is partitioned, calling the function for each row and waiting for the result before processing the next row would have a catastrophic performance impact. The dpipe mechanism allows the functions to be partitioned by the SQL run time and sent to the appropriate cluster nodes as large batches. This applies to any operation that can be expressed as a sequence of steps which will each run within a single partition.

We take a trivial example:

```
create table ct (row_no int primary key, dt varchar);
alter index CT on CT partition (ROW_NO int (255));

create procedure CT_DT_PIPE (in n int)
{
 return vector ((select dt from ct table option (no cluster) where row_no = n), 1);
}

dpipe_define ('CT_DT', 'DB.DBA.CT', 'CT', 'DB.DBA.CT_DT_PIPE', 0);
```

We can now use statements like:

```
select sum (length (ct_dt (row_no))) from ct;
```

This is equivalent to:

```
select sum (length (b.dt))
  from ct a, ct b
  where a.row_no = b.row_no;
```

The point is that the function call will be mapped into the dpipe operation of the same name defined in the dpipe_define. This means that a large batch of row_no's is retrieved and for each, the partition where to invoke the dpipe function is calculated. Then the function calls are sent in concurrent batches to each of these places, running the function in parallel, each on the node that has the data of interest. Getting the value of a column of a partitioned table is a trivial example but any complex computation might be substituted for this, including ones that updated the state on the node, with the proviso that they should not access data from outside the host on which they run.

Further, because the results are summed into an aggregate, the results can be processed in the order they come, thus improving parallelism.

10.7. Cluster and RDF

The RDF tables are partitioned by default on any fresh clustered database. Thus RDF operations are not affected by clustering.

For RDF loading, use the single-threaded load functions DB.DBA.RDF_LOAD_RDFXML and DB.DBA.TTLP. These should essentially always be run in row autocommit mode and without logging. Thus do log_enable (2) on the connection before invoking these functions.

Running these functions in the default transactional mode will load within the current transaction. This will cause widespread locking and will run out of rollback space after some millions of triples. This has a strict transactional semantic but is not generally relevant in RDF applications.

Integrity between all tables and indices is guaranteed after loading the file completes, also in non-transactional mode. After all loading is complete, do a single explicit checkpoint with cl_exec ('checkpoint');

This will guarantee that the disk based image is complete. Automatic checkpoints during non-transactional file loads may have half-files and possibly partial triples in the checkpointed state.

For all SPARUL operations, row autocommit mode is likewise recommended.

Logging is not needed if one makes a manual global checkpoint after any bulk import or update operations. Logging will be useful if one has a continuous feed of smaller files, even if transactional semantics were not needed.

For best import speed, run one or two parallel streams of load commands on each cluster node. Split the data to be loaded into approximately equal chunks and load each with a call to DB.DBA.RDF_LOAD_RDFXML or DB.DBA.TTLP. There is no point in using the _MT variants of these functions on a cluster.

A single load will process about 10000 triples with only about 5 cluster round trips. Still, more of the work is done by the node doing the parsing than by other nodes. To get best use of total throughput, divide the load commands over the cluster nodes. Lock contention will be minimal if the loads are in row autocommit mode. If they are transactional, deadlocks are quite probable due to indeterminate locking order and large transaction size. As a general rule, do not mix transactions and RDF.

10.8. Cluster, Virtual Database and Replication

Clustering has no relation to any virtual database, transactional or snapshot replication mechanism on Virtuoso.

Transactional replication is not supported with clustering. Snapshot replication will work.

Virtual database operations work identically with single process Virtuoso databases. All operations on remote tables are done by the cluster node running the SQL statement. For purposes of symmetry, it is desirable to have all the remote data sources defined for all server processes so that they can be used interchangeably.

An external transaction monitor is not supported with cluster. A Virtuoso cluster could be seen as an XA resource manager but the XA logic is not connected to the cluster transaction logic.

10.9. Troubleshooting

If an operation seems to hang, see the output of status ().

Check for the presence of the following conditions:

- The cluster line shows 0% CPU, no message traffic and an unchanging number of buffers wired, this is probably a bug. To reset, restart the cluster or the offending process if found. Restart is done by executing raw_exit (); over an SQL connection to the process in question.
- The cluster line shows many threads waiting compared to total threads. If CPU is 0 and this does not change there could be a transaction that holds locks indefinitely. To clear, execute txn_killall (1); . Do this at a node that has local threads waiting. This is seen in the Lock Status paragraph of status ('') when connected to the node in question.
- The cluster line shows a changing number in the pfs field. The system is swapping and slowed down.
- If the status () itself hangs, try another process of the cluster. See that there is no temporary atomic activity like a long checkpoint. If the situation persists there is a bug. The checkpoint can be seen by the presence of the checkpinmt_in_progress file in each server's working directory.
- To check the integrity of database files, do cl_exec() ('backup "/dev/null"'); If this returns, the databases are OK. If one is found to be corrupt the corresponding server exits.

See Also:

- Cluster Installation and Config

See Also:

- Cluster Operation
- Virtuoso Cluster Fault Tolerance.

Chapter 11. SQL Procedure Language Guide

Abstract

Stored procedures are a key component of database performance. The fewer messages are sent between the client and the server for a given transaction, the faster it will complete.

Virtuoso/PL is a simple and straightforward language for writing stored procedures and triggers in Virtuoso. Its syntax is a combination of SQL and C, making learning it as easy as possible. It offers the features commonly found in database procedure languages in a simple, efficient and concise package. This document presents the primary concepts of the language and ends with a reference section.

11.1. General Principles

A stored procedure is a named piece of Virtuoso/PL code stored in the SYS_PROCEDURES table. Stored procedures are created with the create procedure statement and are used by executing a procedure call statement through the regular SQL API.

A procedure takes zero or more arguments and optionally returns a value. Procedure arguments may be input, output or input and output. In this manner a procedure may modify a variable passed to it by its caller. If the procedure is called from a call statement executed by a client process, the client process gets back the procedure's return value and the values of output parameters.

Procedures can be called with positional or keyword parameters. A call with positional parameters will bind the first argument in the call to the first parameter in the procedure parameter list and so on. A keyword parameter call allows specifying named parameters, where the argument of a given name is bound to the parameter of the same name in the procedure's parameter list. Procedure parameters may be required or optional. The combination of optional parameters and the keyword call notation make it convenient to have procedures with large numbers of parameters of which only part are used at any one time.

Procedures have local variables and cursors that are not visible to other procedures. Procedures can call each other without limitations, including recursively.

In addition to returning a value and changing values of output parameters a procedure may yield one or more result sets. The client can receive rows in result sets just like rows returned by a select statement. A procedure calling another procedure cannot receive a result set produced by the called procedure, however. While parameters and return values work equally well between procedures as between procedure and client application, a result set always goes to the client, even if the procedure has been called by another procedure. A procedure view is a separate construct which allows a procedure to iterate over another procedure's result set. See the Procedure Views section.

A procedure consists of statements and expressions similar to those of any procedural language. In addition, procedures may contain SQL statements operating on the procedure's arguments and local variables. Writing a stored procedure is thus much like using embedded SQL in C, except that a stored procedure is typically much faster.

The elements of the procedure are:

- **Procedure Declaration.** This is a create procedure statement that names the procedure and its arguments.
- **Variable Declaration.** This declares a local variable for the procedure.
- **Cursor Declaration.** This declares a cursor, A cursor allows a procedure to iterate over the rows produced by a select statement.
- **Manipulative SQL statement.** This can be a delete or update statement, either searched or positioned, a cursor manipulation or other so called routine statement.
- **Control statement.** This is any control structure, loop, assignment or procedure call.

- **Handler declaration.** This specifies what to do in a specific exception situation. Exceptions are error conditions produced by SQL statements (e.g. deadlock) or 'not found' situations.

11.2. Scope of Declarations

A declaration can appear anywhere inside a compound statement. It affects all statements in the compound statement following the declaration statement.

11.3. Data Types

Virtuoso/PL supports the regular SQL scalar data types as well as user-defined-types (UDTs). UDTs and structures can be composed of data types or classes made from any hosted language such C# or Java. Local temporary tables are not supported by the present Virtuoso but may be added in the future.

Memory management is automatic. Parameters, cursors and intermediate results are automatically allocated and freed.

A parameter or variable can be of any data type a column can. Variables are however typed at run time, so that the declared type is mostly for documentation. The declared types also affect how interactive SQL shows certain values.

There is a special *vector* construct, declared as type *ANY* , that can be used as an array. Vectors must be instantiated using the `vector()` function, optionally containing initial elements as a comma-separated list, and can be increased in size using the `vector_concat()` . Elements of a vector are read and changed using the `aref()` and `aset()` functions respectively. A special notation can be used as a short-hand for accessing the elements of a vector one-level deep only. The notation is by using the variable with the index in square brackets. Hence, `aref(vec, 1)` is the same as `vec[1]` . The following example reveal more:

Example 11.1. Using Vectors

Simple excerpt showing how to instantiate a vector.

```
declare vec1, vec2, vec3 any;

vec1 := vector();  -- simple empty vector
vec2 := vector('a', 'b', 1, 2);  -- vector of mixed types
vec3 := vector(vector('a', 'b'), vector(1, 2));  -- vector of vectors
```

Second-level elements of vec3 cannot be referenced as vec3[1][1].

Here is the code for a simple VSP page that shows how vectors can be used.

```
<html>
 <body>
<?vsp
declare vec1 any;
declare i integer;

vec1 := vector();

http('loading up the vector using vector_concat...<br>');
i := 0;
while ( i <= 5 ) {
  vec1 := vector_concat(vec1, vector(i * 5));
  i := i + 1;
}

http('displaying the contents using aref<br>');
i := 0;
while ( i <= 5 ) {
  http_value(aref(vec1, i)); http('<br>');
  i := i + 1;
}

http('changing the values using aset<br>');
i := 0;
while ( i <= 5 ) {
```

```
  aset(vec1, i, i * 10);
  i := i + 1;
}

http('displaying the contents using [] notation<br>');
i := 0;
while ( i <= 5 ) {
  http_value(vec1[i]); http('<br>');
  i := i + 1;
}

http('changing the values using [] notation<br>');
i := 0;
while ( i <= 5 ) {
  vec1[i] := i * 15;
  i := i + 1;
}

http('displaying the contents using [] notation again<br>');
i := 0;
while ( i <= 5 ) {
  http_value(vec1[i]); http('<br>');
  i := i + 1;
}
?>
 </body>
</html>
```

Which produces the following output:

```
loading up the vector using vector_concat...
displaying the contents using aref
0
5
10
15
20
25
changing the values using aset
displaying the contents using [] notation
0
10
20
30
40
50
changing the values using [] notation
displaying the contents using [] notation again
0
15
30
45
60
75
```

See Also:

`aref()`

`aset()`

`vector_concat()`

11.4. Handling Result Sets

A single Virtuoso procedure may produce multiple result sets, each with different result columns. A normal procedure produces one empty result set, only returning a possible return value and values of output parameters to the application.

The `result_names()` predefines variables to be used in a result set to follow. The variables must be previously declared, from which the column data types are ascertained. This assigns the meta data but does not send any results. The `result()` function sends its parameters as a single row of results. These parameters should be compatible with those in the previous `result_names()` . The `end_results()` function can be used to separate multiple result sets. The `result_names()` can then be used to alter the structure of the next result set.

The `result_names()` call can be omitted if the application already knows what columns and their types are to be returned.

See Also:
`result()` , `result_names()` , `end_result()`

11.5. Result Sets and Array Parameters

A procedure may be called with array parameters, c.f. SQLParamOptions. Each call can yield multiple result sets.

The SQLMoreResults function is used to get from one result set to the next and from one procedure call to the next. One may have to call this function an indeterminate number of times before all results from a procedure with array parameters have been received.

Each procedure return is marked with SQL_SUCCESS_WITH_INFO with SQL state 'PMORE'. The next SQLFetch will retrieve the first row of the first result set of the next procedure invocation.

11.6. Exception Semantics

Exceptions are of two types: Not Found and SQLSTATE. A not found exception occurs when a select - into or open statement finds no row or when a fetch statement reads past the last row of a cursor. A SQLSTATE exception may result from any operation, typically a manipulative SQL statement. The SQLSTATE '4001', deadlock is an example of this. A user-written procedure may signal a user defined exception with the signal function.

Virtuoso/PL supports PSM 96 style exception handlers. These allow catching specific SQL states or ranges of SQL states, invoking a specific block of code when the state is signalled from within the scope of the handler. The handler may propagate the exception to an outer handler or transfer control to any appropriate point in the containing procedure.

An unhandled exception will cause the procedure where it is detected to return the exception to its caller. If the caller is another procedure that has a handler for the specified exception that procedure invokes the handler. If the caller is a call statement issued by a client, the client gets the SQLSTATE and the SQLExecute function called by the client returns SQL_ERROR and the client application may retrieve the SQLSTATE and message with the SQLError function.

A SQLSTATE is any short string used to identify an error or exception condition. The system itself generates certain predefined SQLSTATE's for error conditions. Applications may add other states.

See the DECLARE HANDLER, whenever statement and signal function for an example of exception handling.

11.7. Virtuoso/PL Syntax

11.7.1. Create Procedure Statement

```
CREATE PROCEDURE NAME (parameter , parameter...) [RETURNS data_type]
{ statement ... }

parameter: parameter_type name data_type opt_default

parameter_type: IN | OUT | INOUT

opt_default: | DEFAULT literal | := literal
```

The create procedure statement actually performs a "create or replace" type operation. The create procedure statement compiles and stores a Virtuoso/PL procedure. The procedure text is first parsed and compiled into Virtuoso virtual machine code and if the compilation is successful the text is stored into the SYS_PROCEDURES table. This table is read at startup. Stored procedures are thus always available for use and need be defined only once. New procedures created with the same name as existing procedures automatically replace their predecessor.

See Also:

CREATE PROCEDURE Syntax - External hosted procedures

```
CREATE PROCEDURE FIBO (IN X INTEGER)
{
    IF (X < 2)
        RETURN X;
    ELSE
        RETURN (FIBO (X - 1) + FIBO (X - 2));
}

CREATE PROCEDURE CFIBO (IN X INTEGER)
{
    DECLARE RES INTEGER;

    RES := FIBO (X);

    RESULT_NAMES (RES);

    RESULT (RES);
}
```

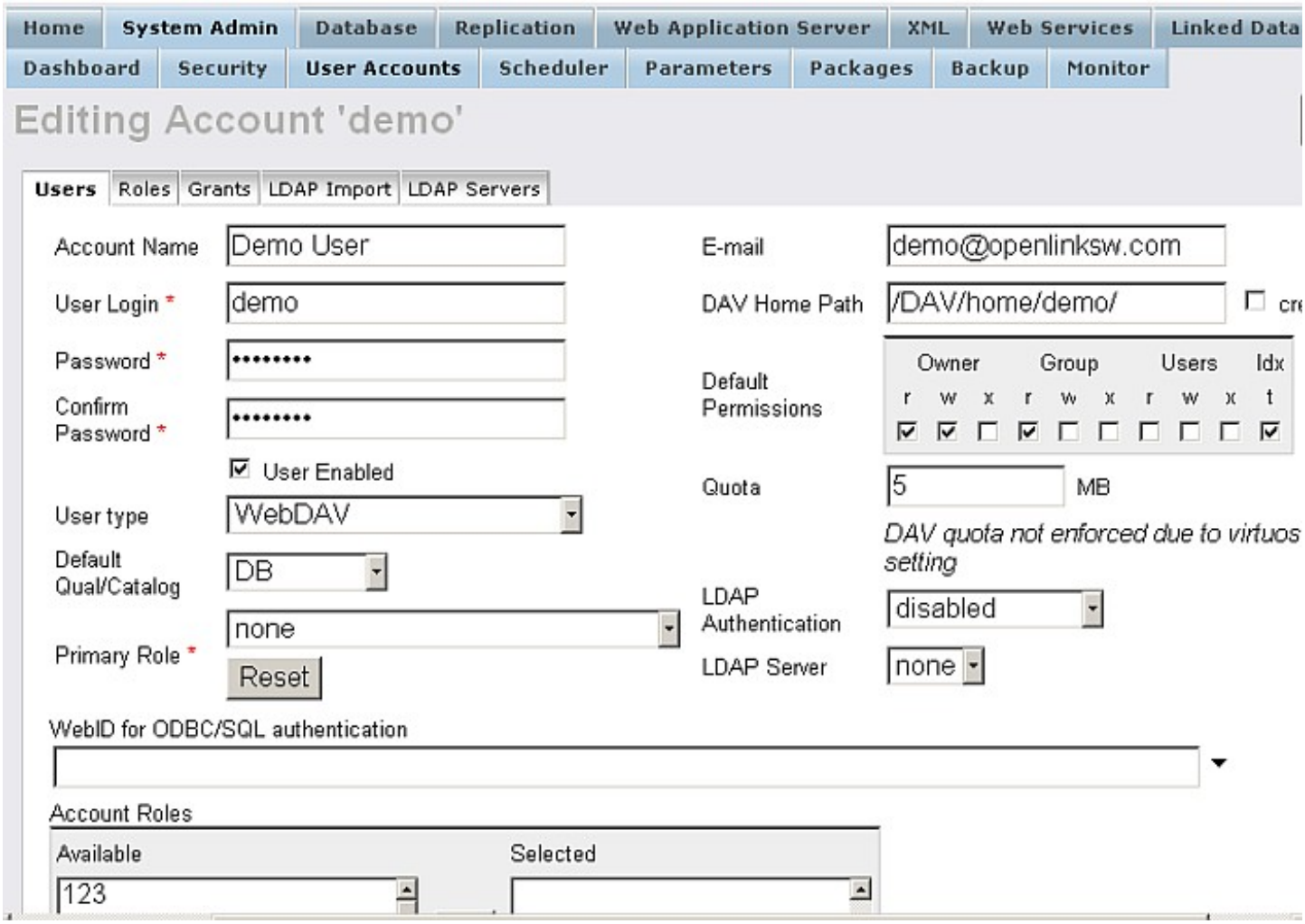
11.7.2. Grant Execute Statement

```
GRANT EXECUTE ON procedure_name TO "{USER | ROLE}" ;
```

The identifier quote character (double quotes) is important usage information since it indicates that the USER or ROLE has a literal identifier. Just as a reference (e.g., URL or URI) has the identifier quote characters "<" and ">" .

The grantee should have SQL rights in order execution of procedure to be granted to this user. The rights can be set from Conductor->System Admin->User Accounts->Account->Edit->User Type:

Figure 11.1. User Type



Example

```
SQL>create procedure DB.DBA.SimplePrint (in txt varchar)
{
 return sprintf('Output is %s', txt);
}
;

Done. -- 0 msec.

SQL>grant execute on DB.DBA.SimplePrint to "demo";

Done. -- 0 msec.

SQL>use demo;

Done. -- 0 msec.

SQL>select DB.DBA.SimplePrint('Virtuoso');

callret
VARCHAR
_______________________________________________________________________________

Output is Virtuoso

1 Rows. -- 0 msec.
```

11.7.3. Stored Procedures as Views & Derived Tables

Virtuoso allows using a stored procedure result set in place of a table. A view may also be defined as a stored procedure. This

provides smooth integration to external procedural logic in queries.

When a procedure appears as a table, the procedure is called and its result set is inserted into a temporary space. Processing continues from that point on as if the data came from a table.

See Also:
For more information about Store Procedures as Views & Derived Tables go to the SQL Reference Chapter

11.7.4. Keyword and Optional Procedure Arguments

Normally arguments in a procedure call are bound to formal parameters from left to right, as is the default behavior in any programming language. If a default value is specified for a parameter in the procedure definition this parameter is optional and the default value will be assigned to it if the caller does not specify a value. A call may consist of zero or more positional arguments followed by zero or more keyword arguments. A positional argument is any scalar expression. A keyword argument is marked with the syntax:

```
NAME => scalar_exp
```

This notation specifies that the expression is to be bound to the parameter NAME in the procedure declaration. The names are matched case-insensitively in all case modes. After all leading positional arguments have been bound to the matching formal parameters in the procedure definition, each keyword argument is bound to the parameter of the same name. After this all unbound formal parameters are assigned to their default values. If a parameter with no default remains unbound an error is signalled. OUT and INOUT parameters are always required, regardless of the mode of calling.

An expression can be passed as INOUT or OUT, but in that case the output value assigned by the procedure is not accessible in the caller. The output value is only accessible if the actual parameter is a variable or parameter.

Arguments of procedures are always evaluated left to right.

```
create procedure kwd (in k1 int := 111, inout k2 int, in k3 int := 333)
{
  result_names (k1, k2, k3);
  result (k1, k2, k3);
}

kwd (1,1+1,3);
-- results 1,2,3

kwd ();
-- error because inout parameters are always required

kwd (k2=>1);
-- error because a constant is not a suitable value for an inout parameter.

kwd (k2=>1+2);
-- result 111, 2, 333

kwd (k3=>3, k1=>1,k2=>1+1);
-- result 1, 2, 3

kwd (1, k2=>1+1);
-- result 1, 2, 333
kwd (1);
-- error, k2 is required
kwd (badkey=>2, k2=>2+1);
-- error, badkey not a parameter of the function

create procedure kwd2 (in k1 int , in k2 int, in k3 int)
{
  result_names (k1, k2, k3);
  result (k1, k2, k3);
}

kwd2 (k1=>1, k2=>2, k3=>3);
-- result 1, 2, 3
kwd2 (1,2,3);
-- result 1, 2, 3
```

11.7.5. if, while, for, foreach statements

```
if_statement
        : IF '(' search_condition ')' statement opt_else

opt_else
        : /* empty */
        | ELSE statement

while_statement
        : WHILE '(' search_condition ')' statement

for_statement
  :     FOR '(' for_init_statement_list ';' for_opt_search_cond ';' for_inc_statement_list ')' statement
  | FOREACH '(' data_type_ref identifier IN_L scalar_exp ')' DO statement
```

The IF statement executes the immediately following statement if the condition is true. If there is an else clause and the condition is false the statement immediately following the else keyword will be executed.

The while statement evaluates the search condition and executes the following statement if the condition is true. It does this as long as the condition is true. To exit from a loop, use goto. C-like break and continue statements are not available.

The for statement initiates the for_init_statement_list and executes the following statement until the search condition is true. After every execution of the statement it executes for_inc_statement_list. You can exit the loop with using goto syntax also.

The foreach statement executes the statement for each element from an array and sets a variable to the corresponding element of that array.

```
IF (A > B)
    A := A + 1;
ELSE
    B := B + 1;

WHILE (1 = 1) {
    A := A + 1;
}

FOR (declare X any, X := 1; X <= 2 ; X := X + 1){
    S := S + X;
}

FOR (declare X any, X := 1; X <= 2 ; ){
    S := S + X;
    X := X + 1;
}

FOR (declare X any, X := 1; ; X := X + 1){
    if (X > 2)
    goto exit_loop;
    S := S + X;
}
exit_loop:

declare X integer;
X := 1;
FOR (; X <= 2 ; X := X + 1){
    S := S + X;
}

ARR := vector (1,2);
FOREACH (int X in ARR) do {
    S := S + X;
}
```

11.7.6. compound statement

```
compound_statement
        : '{' statement_list '}'
        ;
```

```
statement_list
        : statement_in_cs
        | statement_list statement_in_cs
        ;

statement_in_cs
        : local_declaration ';'
        | compound_statement
        | routine_statement ';'
        | control_statement
        | label ':' statement
        ;

statement
        : routine_statement ';'
        | control_statement
        | compound_statement
        ;

local_declaration
        : cursor_def
        | variable_declaration
        | handler_declaration
        ;

variable_declaration
        : DECLARE variable_list data_type
        ;

variable_list
        : NAME
        | variable_list ',' NAME
        ;
```

The compound statement is the main building block of procedures. Statements in a compound statement are executed left to right, unless the flow of control is changed with a goto statement. The compound statement allows declaring local variables and exception handlers. See 'Scope Rules' above for a description of the scope of declarations.

Labeled statements (goto targets) and declarations can only occur within a compound statement.

See:

Create Procedure statement

11.7.7. goto, return statements

```
goto_statement
        : GOTO label
        ;

label   : NAME

return_statement
        : RETURN scalar_exp
        | RETURN
        ;
```

The goto statement unconditionally transfers control to the label following it. The label can be anywhere within the same procedure. It is in principle possible to jump into a block (e.g. loop body) from outside.

The return statement causes the executing procedure to return. If a return value is specified the expression is evaluated and returned as the return value of the procedure. If no return value is specified the procedure returns an undefined value.

Returning from a procedure automatically frees any resources associated with the procedure. This includes values in local variables or call by value (IN) parameters and any cursors that may be open.

See:

Create Procedure statement

11.7.8. whenever statement

```
condition
        : NOT FOUND
        | SQLSTATE STRING
        ;

handler_declaration
        : WHENEVER condition GOTO NAME
        ;
```

This declares that control should be transferred to a particular label in the procedure whenever a condition occurs within the lexical scope of the WHENEVER declaration. This is similar to the statement of the same name found in most embedded SQL implementations.

The scope of the declaration is all the lines lexically following the declaration. A previous declaration is replaced by a new declaration for the same <condition>.

```
CREATE PROCEDURE COUNT_CUSTOMERS (IN C_NAME VARCHAR)
{
    DECLARE COUNT INTEGER;

    COUNT := 0;

    DECLARE C CURSOR FOR SELECT C_ID FROM CUSTOMER WHERE C_NAME = C_NAME;

    WHENEVER SQLSTATE '4001' GOTO DEADLOCK;

    WHENEVER NOT FOUND GOTO DONE;

    OPEN C;
    WHILE (1=1)
    {
        FETCH C INTO N;
        COUNT := COUNT + 1;
    }

DONE:
    RETURN COUNT;

DEADLOCK:
    RETURN -1;
}
```

Note:
This is about the same as select count (*) from CUSTOMER where C_NAME = ?;

11.7.9. call, assignment statements

```
function_call
        : NAME '(' opt_scalar_exp_commalist ')'
        | call '(' scalar_exp ')' '('opt_scalar_exp_commalist ')'
        ;

call_statement
        : CALL NAME '(' opt_scalar_exp_commalist ')'
        | function_call
        ;

assignment_statement
        : lvalue EQUALS scalar_exp
        | lvalue '=' scalar_exp
        ;

lvalue  :  NAME
```

The call statement calls a specified procedure with the given arguments. The procedure to call is resolved at run time, i.e. the latest definition prevails, even if it has been made after the calling procedure was defined. The CALL reserved word is optional and is supported for compatibility.

If the called procedure has reference parameters (OUT or INOUT) the matching actual parameter must be a variable or parameter.

There is a computed function call form of function_call. In this, the scalar expression in parentheses following the call keyword should evaluate to a string which then identifies the function to be called.

The assignment statement sets a value to a variable. The variable must be either a local variable declared with declare or a procedure argument declared in the procedure argument list. If the variable in question is a reference parameter the assignment takes effect in the actual parameter as will, i.e. the value of the argument variable in the caller is set.

```
CREATE PROCEDURE COMPUTED_CALL (IN Q INTEGER)
{
    DECLARE FN VARCHAR;

    FN := 'F';

    --- CALL FUNCTION FF WITH ARGUMENT 11.
    R := CALL (CONCATENATE (FN, 'F')) (11);
}
```

11.7.10. open, fetch, close, select ... into statements

```
SELECT opt_all_distinct selection
    INTO target_commalist
    table_exp
    with_opt_cursor_options_list
    ;

opt_all_distinct
        : /* empty */
        | ALL
        | DISTINCT
        ;

with_opt_cursor_options_list
        : /* empty */
        | WITH opt_cursor_options_list
        ;

cursor_option
        : EXCLUSIVE
        ;

cursor_options_commalist
        : cursor_option
        | cursor_options_commalist ',' cursor_option
        ;

opt_cursor_options_list
        : /* empty */
        | '(' cursor_options_commalist ')'
        ;

cursor_def : DECLARE NAME CURSOR FOR query_exp
           | DECLARE NAME (DYNAMIC|KEYSET|STATIC) CURSOR FOR query_exp

open_statement
        : OPEN cursor opt_cursor_options_list
        ;

fetch_statement : FETCH cursor INTO target_commalist
                | FETCH cursor (FIRST|NEXT|PREVIOUS|LAST) INTO target_commalist
                | FETCH cursor BOOKMARK scalar_exp INTO target_commalist

target_commalist
        : variable
        | target_commalist ',' variable
        ;

close_statement
        : CLOSE cursor
        ;
```

The open, fetch and close statements manipulate cursors in Virtuoso/PL statements. Cursors are declared with the declare cursor statement. The select into statement is a shorthand for a cursor declaration, open, fetch and close.

A forward-only cursor declaration is a declaration only and executing one does not take time. The open statement effectively starts the search associated with the forward-only cursor.

The forward-only cursor options used with open and select into allow controlling how the cursor sets locks on selected rows and how many rows it fetches at a time. The EXCLUSIVE option should be used if intending to update or delete a row in the cursor's evaluation. This causes selected rows to be locked with exclusive (write) locks.

```
The statements:

{
    DECLARE CR CURSOR FOR SELECT C_NAME FROM CUSTOMER WHERE C_ID = ID;
    OPEN CR;
    FETCH CR INTO NAME;
    CLOSE CR;
}

and

SELECT C_NAME INTO NAME FROM CUSTOMER WHERE C_ID = ID;

have the same effect.
```

See:
the TPC C Bench Marking chapter for more examples.

11.7.11. FOR Select Statement

```
<for statement> ::=
     FOR <query exp> DO statement
```

The FOR statement provides a compact notation for iterating over the result set of a cursor. The body is executed once for each row in the query expression's result set. The result columns produced by the query expression are accessible as variables of the same name inside the body. All result columns do therefore have to be named with the AS declaration if they are not simple columns, in which case the name defaults to the column's name.

The body can be exited in mid loop with a goto. The cursor of the FOR does not have to be specifically closed or opened. FOR statements can be freely nested. If a WHENEVER NOT FOUND declaration is in effect before the FOR it will be canceled by it, so that it is not in effect after the loop's body.

Example 11.2. Examples

```
for select C_NAME, sum (O_VALUE) as value  from CUSTOMER, ORDER group by C_NAME DO
  {
    result (C_NAME, value);
  })
```

The equivalent code is

```
declare C_NAME, value any;
whenever not found goto done;
declare cr cursor for select ....;
open cr;
while (1) {
  fetch cr into C_NAME, value;
  whenever not found default;
  ...
}
done: ;
```

The cursor and end label names are generated to be unique by the FOR expansion.

11.7.12. SET statement

```
Set_statement:
        SET option '=' scalar_exp
        | SET option OFF
        | SET option ON
        ;

option:
        ISOLATION
        | LOCK_ESCALATION_PCT
        | TRIGGERS
        | PARAM_BATCH
        ;
```

The SET statement sets an option to a value. Options may control trigger invocation, transaction isolation and other settable parameters of the engine. A SET inside a procedure takes effect inside the procedure and invoked procedures, counting from time of execution. Control must pass through the SET statement for it to take effect, i.e. SET is not a declaration. The effect of a SET does typically not persist across procedure return.

A SET given at top level, i.e. directly executed and by a client as the statement of a SQLExecute sets an option at the connection level. This may only be reversed by another SET.

The option may be:

11.7.13. SET Triggers

A value of OFF or 0 causes triggers not to be invoked even if there may be applicable triggers. This is mostly useful for controlling recursion of triggers or for debugging triggers.

The value of TRIGGERS is passed into called procedures but other options are not.

See Also:
SET statement.

11.7.14. Vectored Procedures

Note: This feature only applies to Virtuoso 7.0 and later.

A stored procedure may be declared vectored. This means that when called from a statement operating on multiple values, a single call of the procedure can take the whole batch of variable bindings the statement is operating on in a single invocation. This saves invocation and interpretation overhead, and, most importantly, allows running any SQL statements inside the procedure on multiple values at once, creating possibilities for parallelization and exploitation of locality. The vectored declaration consists of the VECTORED reserved word at the start of the procedure body.

Consider the example of a lookup table:

```
CREATE TABLE person
  (
    p_id   INT PRIMARY KEY,
    p_name VARCHAR
  );

CREATE TABLE knows
  ( p1 INT REFERENCES PERSON,
    p2 INT REFERENCES PERSON,
    PRIMARY KEY ( p1, p2 )
  );

CREATE PROCEDURE p_name
  ( IN code INT )
  RETURNS VARCHAR
  {
    VECTORED;
    RETURN
      (
```

```
            SELECT p_name
              FROM person
             WHERE p_id  = id
          ) ;
      }

SELECT  p_name (p1)
  FROM  knows
 WHERE  p2 = 123;
```

This last statement is equivalent to:

```
SELECT  p_name
  FROM  knows,
        person
 WHERE  p_id = p1
   AND  p2 = 123;
```

For non-trivial transformations, hiding the logic inside a procedure makes sense, Running the procedure vectored makes it so that efficiency is not lost. For example, if person 123 knows 1000 people, there will not be 1000 random lookups in person for the names but rather a single, vectored, merge-style lookup, accessing the rows in order of ID, saving time if the IDs are nearby each other. Furthermore, if the lookup is in READ COMMITTED isolation, the multiple lookups can be scheduled on multiple threads.

The restrictions for vectored statement bodies also apply to function bodies that are declared vectored.

A vectored procedure can be called from a non-vectored procedure. In this case, the vectored procedure simply executes on a single set of values, as if it were not vectored.

A vectored procedure can call a non-vectored procedure. When this happens, the non-vectored procedure is called once for each set of eligible values, i.e., once for the first values of the arguments, once for the second values of the arguments, and so forth.

A vectored procedure, if called from vectored code, returns a return value for each set of arguments.

A vectored procedure can have IN and OUT parameters. These have the same semantics as in single-value execution. When calling a non-vectored procedure with an OUT or INOUT parameter, the argument in the vectored caller must be declared to be of a boxed data type. (See the section on vectoring and data types below.)

11.7.15. FOR VECTORED Statement

Note: This feature only applies to Virtuoso 7.0 and later.

```
FOR VECTORED
  ( {IN|OUT} <variable> <data_type> [ := <value>],
    ...
  )
  <compound_statement>
```

The FOR VECTORED statement allows executing a block of code on several sets of variable bindings at once. The benefit of this is that any database operations in such a block can be run on multiple sets of parameters at once, allowing exploitation of locality and, in some cases, running the operation on different bindings on different threads. Additionally, if vectored procedures are called from inside such a block, the call is made with multiple bindings for the parameters. The input variables of FOR VECTORED are initialized from an array of scalar values. The statements inside the body are then executed vectored, as if the operation were first made on all the first values of the vectors, then on the second values, and so forth. Operations combining values from different places in the vectors are not possible in the FOR VECTORED body, but, since vectored results can be seen as arrays after the return of FOR VECTORED, any aggregation or comparison between values in different positions of the same vector can be done after the FOR VECTORED, simply accessing different elements of the arrays produced.

The FOR VECTORED statement communicates with its environment through a list of input and output variables. The input variables are marked with the syntax:

```
IN <variable> <data_type> := <value>
```

The <value> must be an expression evaluating to an array. The data type must correspond to the element type of the array. When multiple input variables are specified, the arrays initializing each must be of equal length.

An output variable is marked with:

```
OUT <variable> := <value>
```

The variable must be declared in a context outside of the FOR VECTORED statement. The value of the variable will be an array where each value of the vectored expression <value> is represented as a separate value.

Variables declared outside of a FOR VECTORED statement are visible in the body of FOR VECTORED and they appear as a single value for all rows of the vectored section.

Consider the task of pair-wise adding the elements of two arrays:

```
CREATE PROCEDURE a_add
  (
    IN  a1  INT ARRAY ,
    IN  a2  INT ARRAY
  )
  {
  DECLARE  res  INT ARRAY ;
  res := make_array (LENGTH (a1, ' any' );
  FOR (i := 0; i < LENGTH (a1); i := i + 1)
    res[i] := a1[i] + a2[i];
  RETURN res;
}
```

This can be expressed as:

```
CREATE PROCEDURE a_add_v
  (
    IN  a1  INT ARRAY ,
    IN  a2  INT ARRAY
  )
  {
    DECLARE  res  INT ARRAY;
    FOR VECTORED
      ( IN   n1  INT  := a1 ,
        IN   i2  INT  := a2 ,
        OUT  res      := r
      )
      {
         DECLARE  r  INT ;
         r := i1 + i2 ;
      }
    RETURN res;
  }
```

The two procedures are identical in function. The second will make use of vector instructions in the host CPU, if available, and will incur less interpretation overhead, since the SQL run time will not need to run a loop. In practice, substantial benefit, up to an order of magnitude, can be had from vectored execution with database operations exhibiting significant locality. Bulk loads and bulk lookups are a typical example.

11.7.16. Limitations on Vectored Code

Note: This feature only applies to Virtuoso 7.0 and later.

The body of FOR VECTORED or a vectored procedure may contain arbitrary Virtuoso PL, except for LOOPs and backward GOTOs. Conditional expressions and statements are allowed, as well as any subqueries or DML statements. Looping over a cursor is not allowed, since this is a loop, but scalar subqueries and selecting-into-variables in *SELECT ... INTO* is allowed. Exception handlers are not allowed inside, but an exception handler outside of FOR VECTORED will catch errors signaled from inside FOR VECTORED. FOR VECTORED statements may not be nested and may not occur in the body of a vectored procedure. The handler, being itself not in vectored code, will not be able to see which specific value in a vectored section gave rise to the exception.

11.7.17. Data Types and Vectoring

Note: This feature only applies to Virtuoso 7.0 and later.

Parameters in vectored procedures or FOR VECTORED blocks can be declared to be of the corresponding scalar data type. The vectoring is thus in most cases transparent; the variable will simply have multiple scalar values instead of one. The ANY type in a vectored code section is represented as an array of serialized values. Thus types that are represented as data structures in allocated memory (e.g., arrays, hash tables, XML elements, etc.) will not work efficiently with ANY vectored variables. In some cases (for example, with streams or dictionaries), assigning to a vectored ANY will lose the information.

Therefore, if dealing with vectors of complex data types in vectored code, the variable holding these must be declared as an ANY ARRAY. With this type, the representation will be an array of pointers to allocated memory, not an array of flat serialized values. The ANY ARRAY type must be used instead of the customary ANY in all cases involving complex values in vectored code. If dealing with vectors of simple scalars like strings or numbers, the ANY type is generally more efficient.

11.8. Execute Stored Procedures via SELECT statement

Stored SQL Procedures can be executed via SELECT statement:

```
SELECT PROCEDURE_NAME (parameter , parameter...);
```

For ex.:

```
create procedure mytest ( in ss varchar)
{
  return concat('My simple test with ', ss);
}
;

SQL> select mytest('Virtuoso');
callret
VARCHAR
_______________________________________________________________________________

My simple test with Virtuoso

1 Rows. -- 0 msec.
```

11.9. Execute Stored Procedures In Background

You can start procedure in background using the [name of the procedure][params]& syntax. This feature forks another ISQL process and leaves the other on background so there will be two separate clients running separate client connections:

```
SQL>create procedure test()
{
return 'my simple test';
}
;
Done. -- 0 msec.
SQL>test()&
SQL> Connected to OpenLink Virtuoso
Driver: 05.07.3033 OpenLink Virtuoso ODBC Driver
OpenLink Interactive SQL (Virtuoso), version 0.9849b.
Type HELP; for help and EXIT; to exit.

Done. -- 10 msec.
```

See Asynchronous Execution and Multithreading in Virtuoso/PL for background jobs execution details.

11.10. CREATE ASSEMBLY Syntax - External Libraries

External CLR libraries can be hosted inside Virtuoso by creating an assembly from the library itself using the syntax as follows:

```
CREATE ASSEMBLY <assembly_name> FROM <assembly_location>
  [WITH PERMISSION_SET = <perm>] [WITH AUTOREGISTER];
```

assembly_name - is how Virtuoso will reference the library.

assembly_location - is where Virtuoso will find the library within the CLR.

Every .NET assembly deployed inside Virtuoso will be verifiable, which means it will contain code the CLR can verify to be safe in the way it writes to memory.

Virtuoso also respects the Common Language Runtime's code access security model. By default, code does not have any permissions to create a graphical user interface, create threads, access the file system, or call unmanaged code. The only permissions implemented are those granted for in-process data access.

Administrators will control the permissions granted to assemblies using a standard .NET machine and user-level security policy. At runtime, any code accessing protected resources produces a stack walk that triggers a permissions check against that code and any code that called it.

To simplify security administration, Virtuoso supports these standard permission sets for .NET assemblies:

SAFE - This is the default permission set. It allows internal computation and data access. There is no access to resources outside of Virtuoso. Calls to unmanaged code are not allowed. Code must be verifiable.

UNRESTRICTED - Code can access any resource. Only system administrators (dba group/role) can run unrestricted code. This level allows calls to unmanaged code, and can be unverifiable.

The restricted assemblies (SAFE mode) are not permitted to execute any code that infringes upon any of the following permissions:

AspNetHostingPermission
EnvironmentPermission
FileIOPermission
IsolatedStoragePermission
ReflectionPermission
RegistryPermission
SecurityPermission
SocketPermission
WebPermissionPermission
DNSPermission
PrintingPermission
OleDBPermissionPermission
SqlClientPermissionPermission
EventLogPermission
MessageQueuePermission
ServiceControllerPermission
PerformanceCountersPermission
DirectoryServicePermission

If the assembly generates a security exception the error text will be returned to the client.

Note:

Currently on the Microsoft .Net Framework implementation supports permission sets. Virtuoso does not currently support the `EXTERNAL_ACCESS` permission set.

WITH AUTOREGISTER marks the assembly as a stored procedure, trigger, user-defined function, etc., based on custom attributes you add to your .NET code.

Assemblies are stored in the database and are therefore backed-up and restored with the data. Once assemblies are registered using the `CREATE ASSEMBLY` syntax there will be no further dependency on the library file (dll or exe) itself.

You can remove assemblies using the familiar SQL DROP statement:

```
DROP ASSEMBLY <assembly_name>;
```

See Also:

import_clr()

Example 11.3. Working with assemblies

This example is based on the tutorial HO_S_10. we start by obtaining a C# library compile from the following code (included in the tutorial):

```
using System;

[Serializable]
public class Point_10
{
  public Double x;
  public Double y;

  public Point_10 ()
    {
      x = 0;
      y = 0;
    }
  public Point_10 (Double new_x, Double new_y)
    {
      x = new_x;
      y = new_y;
    }

  public Double distance (Point_10 p)
    {
      Double ret;

      ret =  Math.Sqrt ((p.x - this.x) * (p.x - this.x) + (p.y - this.y) * (p.y - this.y));

      return ret;
    }
}
```

This gives us the Point_10 class with two constructors and one method for finding the distance between two points.

Now we must create the library reference in Virtuoso using the following:

```
DROP ASSEMBLY "myPoint";

CREATE ASSEMBLY "myPoint" as concat (http_root() , '\\tutorial\\hosting\\ho_s_10\\Point_ho_s_10.dll')
  WITH PERMISSION_SET = SAFE WITH AUTOREGISTER;
```

Now for a quick test, we will find the distance between two points:

```
SQL> select new Point_10(0,0).distance(Point_10(3,4));
callret
DOUBLE PRECISION
_______________________________________________________

               5
```

Now we will create a table with a column of type Point_10 and then insert some test data:

```
drop table CLR..Supplier_ho_s_10;

create table CLR..Supplier_ho_s_10 (id integer primary key, name varchar (20), location Point_10);

insert into CLR..Supplier_ho_s_10 (id, name, location) values (1, 'S1', new Point_10 (1, 1));
insert into CLR..Supplier_ho_s_10 (id, name, location) values (2, 'S2', new Point_10 (3, 3));
insert into CLR..Supplier_ho_s_10 (id, name, location) values (3, 'S3', new Point_10 (5, 5));
```

Now we will demonstrate how this assembly's class can be used in SQL by showing some queries on the sample data:

```
SQL> select name, s.location.x from CLR..Supplier_ho_s_10 s;
name                  callret
VARCHAR               DOUBLE PRECISION
```

```
S1                                1
S2                                3
S3                                5
```

The distances from (0, 0):

```
SQL> select name, s.location.distance(Point_10(0,0)) from CLR..Supplier_ho_s_10 s ;
name                  callret
VARCHAR               DOUBLE PRECISION
_______________________________________________________________________________

S1                    1.414213562373095
S2                    4.242640687119285
S3                    7.071067811865476
```

Now, the points that are more than 3 units away from it:

```
SQL> select name from CLR..Supplier_ho_s_10 s where s.location.distance(Point_10(0,0)) > 3;
name
VARCHAR
_______________________________________________________________________________

S2
S3
```

Example 11.4. Using CREATE ASSEMBLY

This example demonstrates the creation of trivial CLR classes and referencing them from Virtuoso.

- *lib.cs*

```
namespace lib
{
  public class t1
  {
     public static int addit (int a1, int a2) { return a1 + a2; }
  }
}
```

- *exe.cs*

```
using lib;

public class exe
{
   public static int call_addit (int a1, int a2)
   {
      return t1.addit (a1, a2);
   }

   public static void Main (String [] args)
   {
      Console.WriteLine ("result=" + call_addit (12, 13));
   }
}
```

- *compilation*

```
csc /t:library lib.cs
csc /r:lib.dll exe.cs
```

- *Now Virtuoso can use*

```
create assembly sql_lib from 'c:\sample\lib.dll'
create assembly sql_exe from 'c:\sample\exe.exe'
```

Example 11.5. Creating Assemblies with Permission Sets

These examples will use an assembly called test.dll, whose source code is:

```
using System;
using System.IO;

public class Sample
{
   public static String GetEnv ()
     {
       return Environment.GetEnvironmentVariable("PATH");
     }
}
```

The assembly will be registered using:

```
CREATE ASSEMBLY "test" from 'test.dll' WITH PERMISSION_SET = SAFE WITH AUTOREGISTER;
```

and subsequently called using:

```
SQL> select Sample::GetEnv ();
```

returning the following error for attempting to exceed the SAFE permission set.

```
*** Error 42000: [Virtuoso Driver][Virtuoso Server]CLR05: Request for the permission of type System.Secur
      mscorlib, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089 failed.
in
__udt_method_call:(BIF),
<Top Level>
at line 4 of Top-Level:
select Sample::GetEnv ()
```

Now we can try the same sample using PERMISSION_SET = UNRESTRICTED.

```
drop ASSEMBLY "test";

CREATE ASSEMBLY "test" from 'test.dll' WITH PERMISSION_SET = UNRESTRICTED WITH AUTOREGISTER;

SQL> select Sample::GetEnv ();
callret
VARCHAR
_______________________________________________________________________________

D:\Virtuoso\bin...;

1 Rows. -- 32 msec.
<
```

Unrestricted assemblies do not have any restrictions on usage.

11.11. CREATE PROCEDURE Syntax - External hosted procedures

Virtuoso provides a syntax shortcut for calling static method from hosted user defined types without first defining a Virtuoso external hosted user defined type:

```
CREATE (PROCEDURE|FUNCTION) <local_name> ([<arg_def1>, ...])
        [returns <sql_datatype>] LANGUAGE [JAVA|CLR] EXTERNAL NAME '<external_static_proc_name_literal>'

<arg_def> := [IN|OUT|INOUT] param_name <data_type_spec>

<external_static_proc_name_literal> = <external_type_name_literal>.<static_proc_name>
```

This compiles into an functional equivalent of :

```
create procedure <local_name) ([<arg_def1>, ....])
```

```
{
  declare ret any;

  exec ('
    create type <local_name>
      temporary self as ref
        static method m1 ([<arg_def1>, ....])
           returns <datatype> EXTERNAL NAME ''<static_proc_name>''
   ');

  ret := <local_name>::m1 (....);

  exec ('drop type <local_name>');

  return ret;
  }
```

For more details see CREATE TYPE and Runtime hosting chapters.

Example 11.6. CREATE PROCEDURE for a Java method:

Here is an example for CREATE PROCEDURE and the hosted Java VM:

```
create procedure get_property (in x varchar) returns varchar language java external name 'java.lang.Syste
```

Here's how that procedure is called:

```
SQL> select get_property ('java.vm.name');
callret
VARCHAR
_______________________________________________________________________________

Java HotSpot(TM) Client VM
```

Example 11.7. CREATE PROCEDURE for a CLR method:

Here is an example for CREATE PROCEDURE and the hosted CLR:

```
create procedure curr_thr_id () returns integer language CLR external name 'mscorlib/System.AppDomain.Get
```

Here's how that procedure is called:

```
SQL> select curr_thr_id();
callret
INTEGER
_______________________________________________________________________________

2156
```

11.12. Asynchronous Execution and Multithreading in Virtuoso/PL

Many application tasks benefit from parallel execution. This is specially true of I/O intensive workloads where each thread spends a large amount of time waiting for the network or disks. Typical tasks include crawling the web and importing large data sets. The whole process must not stop just because there is a file cache miss or because there is round trip latency or a name resolution delay on the net.

To this effect, Virtuoso/PL provides the async_queue object. A stored procedure may create an async_queue that will be served by a pool of worker threads. The size of this pool can be set when creating the queue.

The thread which made the queue can use the queue to pass procedure/parameter list pairs to the threads. If a thread is available, the thread will execute the request, if not, the next thread of the pool to become free will take the oldest queued item and execute it. Thus the queue is served in a FIFO fashion multiplexed over n threads.

The owner of the queue can check on the results of execution either collectively or individually. Each worker thread has its own transaction and worker threads may end up waiting for each other own database locks and can deadlock. The worker thread code is responsible for committing its own transaction and handling any deadlock retries or such.

When the thread owning the queue makes a request, a request number is returned. This number can be used to later request the return value and error status of the request. A queue cannot be persisted in a database table and cannot be passed between threads. A queue can be passed between procedures and kept in data structures such as arrays. Queue are internally reference counted and when the last reference drops the queue is freed. If a queue is freed while there is still activity on behalf of the queue, the activities that are ongoing are finished, all requests that are not started are discarded and all values and error states are discarded. The queue and associated resources are thereafter freed.

There is a pool of 20 threads that are shared among all async_queues on the system. Thus, the count of threads given for the queue is a maximum and does not guarantee that this quantity of threads be used in reality. If no thread is available in the pool, i.e. other async queues have exhausted the entire pool, the thread making the request ends up executing the item synchronously. One should take this possibility into account when deciding transaction boundaries, otherwise this is transparent and the calling thread still gets a request handle and can later check for its completion.

Queues take procedures and argument lists instead of texts of SQL statements in order to save the time of compiling the text. It is desirable for best performance to supply the name of the procedure in its full form, with full qualification and matching case.

Consider the following code samples:

```
create table aqi (n int);

-- The worker procedure. Insert one row and commit.

create procedure INS1 (in n int)
{
  --dbg_obj_print ('ins1 ', n);
  insert into AQI (N) values (n);
  commit work;
  return '22';
}

create procedure taq1 (in x int, in thrs int := 1)
{
  declare aq, res, err any;
  declare n int;
  aq := async_queue (thrs);
  for (n:= 0; n < x; n:=n+1)
    {
      res := aq_request (aq, 'DB.DBA.INS1', vector (n));
    }
  return (aq_wait (aq, res, 1, err));
}

-- This procedure makes a queue with a given number of worker threads, then makes a set of requests and w
of the last one. Note that this is not necessarily the last to complete if there are multiple threads ser
```

11.12.1. Synchronization

It is possible to add requests to a queue at all times. It is also possible to check for the result of any request made so far, by either blocking to wait for it or just checking its status.

```
aq_wait (in aq any, in req_no int, in block int, out err any) returns any
```

The aq_wait function takes the queue, a request number returned by aq_request, a blocking flag and an output parameter for the error state.

If there was no error, the error state is set to 0. If the procedure was undefined, the error state is set to 2. If there was a SQL state signalled from the procedure called on the worker thread, the error state is set to an array of three elements: The integer 3, the SQL state string and the text of the message. If aq_wait is terminated by an external event, then an error indicating this is signalled and the state waited for is lost. This can only happen when all transactions are killed by shutdown or going to a single user state.

If the blocking flag was zero and the request was not complete, then the error output parameter is set to 1 and 1 is returned.

Once aq_wait has retrieved a state, the state is no longer retained in the queue.

The aq_wait_all function allows waiting for all activity to complete but discards individual return states. If some of running activities is terminated by sql error, this error will be raised in the thread executing aq_wait_all function call.

Note that it is possible to get a deadlock between the requesting thread and a worker thread and that this deadlock cannot be detected by the database engine since this does not involve a cycle in database locks themselves. Thus, aq_wait signals an error if the thread calling it holds database locks. Manually committing or rolling back before calling aq_wait is necessary if the thread can belong to a transaction that holds locks.

Thus, it is most practical to explicitly commit all work on the requesting thread before calling aq_wait or aq_wait_all.

```
create procedure taq_all (in x int, in thrs int := 1)
{
  declare aq, res, err any;
  declare n int;
  aq := async_queue (thrs);
  for (n:= 0; n < x; n:=n+1)
    {
      res := aq_request (aq, 'DB.DBA.INS1', vector (n));
    }
  aq_wait_all (aq);
}

-- This procedure is guaranteed to wait for all requests to be completed but will discard individual erro
```

11.13. Performance Tips

11.13.1. Remember the following:

- Reference parameters (inout and out) are faster then value parameters (in).
- Use cursors and positioned delete/update (where current of) for read-modify transactions instead of a select .. into and searched update.
- Make an EXCLUSIVE read in read-modify transactions.
- When using ORDER BY and wishing to use a particular index, specify ALL key parts of the index, including those that have a '=' condition. If you want to have descending order using an index, specify DESC on ALL key parts.

11.14. Procedures and Transactions

A procedure call executed by a client is just like any other SQL statement. It executes in the context of the client's active transaction. If the connection is in autocommit mode the transaction is automatically committed if the procedure returns successfully and rolled back if the procedure returns with an error. If the connection is in manual commit mode, a possible procedure error has no effect on the client's transaction, unless the error is a transaction error, e.g. timeout or deadlock.

For best performance, we recommend using procedures in autocommit mode. In this way, a single client-server exchange will suffice to carry out the whole transaction. This will also conveniently roll back the transaction if the procedure exited as a result of an unhandled SQLSTATE or a 'not found' condition.

Procedures can commit or rollback transactions using commit work and rollback work statements.

11.15. Distributed Transaction & Two Phase Commit

2PC is an acronym for 2 Phase Commit. This is a protocol by which data being committed to a database is committed in two phases. In the first phase, the transaction processor checks that all parts of the transaction can be committed. In the second phase, all parts of the transaction are committed. If any part of the transaction indicates in the first phase that it cannot be committed, the second phase does not occur. ODBC does not support two-phase commits.

Transactions in SQL databases are expected to have "ACID" features: Atomicity, Consistency, Isolation, Durability. A two-phase commit (2PC) protocol is needed for guaranteeing ACID properties of transactions which involve changing data in more than one database. This can be the case in a transaction involving tables attached from other databases or explicit access to remote databases with rexecute().

The 2PC protocol needs to have a third party Distributed Transaction Coordinator (DTC). Virtuoso supports Microsoft Transaction Server (or MS DTC).

There are two ways of using MTS-driven distributed transactions in Virtuoso. Virtuoso either initiates the transaction, or it responds to a transaction.

11.15.1. Initiating Distributed Transactions

In this case the transactions are initiated by Virtuoso itself. This causes all remote connections of linked tables to be automatically enlisted in a distributed transaction controlled by MTS. To enable this, Virtuoso's transaction must be set to a special state with the 'SET' statement as follows:

```
SET MTS_2PC=1;
```

This statement turns distributed transaction support on. All transactions started on remote databases shall automatically be enlisted as branches of a distributed transaction managed by MS DTC. The effect of SET, in this case, lasts until the commit or rollback of the transaction. The SET statement should be at the beginning of the transaction, before any distributed operations are undertaken.

Example of money transfer from one attached table to another:

```
CREATE PROCEDURE TWOPC_TRANSFER_MONEY(IN person_id INTEGER)
{
  IF (MTS_STATUS('MTS') = 'disconnected') -- check connection to MS DTC
  {
    MTS_CONNECT(0); -- connect to MS DTC
  }
  SET MTS_2PC=1; --  transaction of this procedure is now in distributed
  MTS_SET_TIMEOUT (1000); -- 1sec timeout on distributed transactions
  UPDATE linked_account1 SET amount=amount+100 WHERE id=person_id;
  UPDATE linked_account2 SET amount=amount-100 WHERE id=person_id;
  commit work;
}
```

This money transfer is under 2PC control of MTS. If one of the two participating databases crashes (or rolls back due to deadlock or timeout), Virtuoso will roll back the whole distributed transaction.

Note that if a transaction modifies the local Virtuoso database, and not more than one remote database, 2 phase commit is not needed for guaranteeing integrity.

Deadlocks are detected for local transactions using a wait graph. Deadlocks are detected for distributed transactions based on timeouts. Use mts_set_timeout() for explicitly setting a timeout. See MS DTC for a definition of timeouts.

11.15.2. Responding to Distributed Transactions

In this situation a distributed transaction is initiated by an ODBC client of Virtuoso. The application enlists one or more Virtuoso hdbcs in an OLE/DB distributed transaction, and then works with that hdbcs and commits or rolls back the distributed transaction.

c++ example:

```
/* begin of example */
  ITransaction* transaction;
  ITransactionDispenser* disp;
  HRESULT hr =
      DtcGetTransactionManager (0, 0, &IID_ITransactionDispenser, 0, 0, 0,
      &disp);
  hr = disp->BeginTransaction (0, ISOLATIONLEVEL_ISOLATED,
      0, 0, &transaction); /* initialize transaction */
  SQLSetConnectOption (hdbc1, SQL_COPT_SS_ENLIST_IN_DTC,
      (DWORD) transaction); /* enlist 1st hdbc in transaction */
  SQLSetConnectOption (hdbc2, SQL_COPT_SS_ENLIST_IN_DTC,
      (DWORD) transaction); /* enlist 2nd hdbc in transaction */

  ..... /* some work with ODBC connections */

  transaction->Commit (0, 0, 0); /* commit the transaction */
/* end of example */
```

If a Virtuoso connection is enlisted into a distributed transaction managed by MS DTC, and a Virtuoso statement executed in this transaction accesses attached tables, or otherwise uses other databases, then Virtuoso automatically enlists these remote databases into the original distributed transaction. If the remote database does not support MS DTC, then it signals the special error (see error list below).

For more information, see Microsoft's documentation for MTS and OLE DB.

If you want Virtuoso to start connected to MTS, add the following string in the [VDB] section of virtuoso.ini file:

```
UseMTS = 1
```

See also:
mts_connect , mts_status , mts_set_timeout , mts_get_timeout .

11.15.3. 2PC Log & Recovery

If one branch of a distributed transaction crashes during the second phase of a commit, the recovery cycle will be performed during the next start up of the server. Information about a distributed transaction is stored in the transaction log file.

When Virtuoso connects to MS DTC, it creates a guid.bin file in the working directory. This file contains a unique ID of the server and is require for the recovery cycle.

11.15.4. Error Codes

Table 11.1. 2PC & MS DTC error list

Code	Description	Possible courses
MX000	connection to MS DTC is failed	MS DTC service is not started (in case of NT4.0 MTS is not started).
37100	MTS support is not enabled	Current or involved in distributed transaction database has not connected to MS DTC.

11.16. Triggers

A trigger is a procedure body associated with a table and an event. A trigger can take effect before, after or instead of the event on the subject table. Several before, after or instead of triggers may exist for a given event on a given table, which can be fired in a specified order.

Triggers are useful for enforcing integrity rules, maintaining the validity of data computed from other data, accumulating history data etc.

A trigger body has no arguments in the sense a procedure does. A trigger body implicitly sees the columns of the subject table as read-only parameters. An update trigger may see both the new and old values of the row of the subject table. These are differentiated by correlation names in the REFERENCING clause.

Triggers are capable of cascading; the code of a trigger may cause another trigger to be activated. This may lead to non-terminating recursion in some cases. Triggers may be turned off either inside a compound statement or inside a connection with the SET TRIGGERS OFF statement.

An update trigger may have a set of sensitive columns whose update will cause the trigger code to be run. Update of non-sensitive columns will not invoke the trigger. If no column list is specified any update will invoke the trigger.

11.16.1. The CREATE TRIGGER statement

Triggers can be defined to act upon a table or column and fire upon:

UPDATE
INSERT
DELETE
at the following times during the operation on a table or column:

BEFORE

AFTER

INSTEAD OF

Triggers have a unique name which is qualified by the current catalog and owner. The trigger name is only really relevant for the purposes of dropping triggers. Triggers operate on a table or column which must be adequately qualified.

The trigger body has read-only access to the values of the data manipulation operation that triggered the trigger. In the case of an update statement it has access to both old and new values for each effected column. These values cannot be changed directly. If the trigger is to influence any data in a table, even from the current operation, it must be achieved by another SQL statement. The REFERENCING clause allows specifying a correlation name for new and old values of columns. By default, the new values are seen under the column names without a correlation name. If old values of updated columns are needed, the REFERENCING OLD AS <alias> will make <alias>.<column> refer to the old value.

Triggers defined to make further operations within the same or other table may fire further triggers, or even the same trigger again. Care must be taken to understand the implications of this and when triggers cane be allowed to continue firing after the current trigger. For example, an after update trigger that makes a further update to the same table will fire the same trigger again and may continue looping in this way endlessly. The `SET TRIGGER` statement can be issued to control this:

SET TRIGGERS on; -- (default state) further triggers within this transaction are allowed to fire.

SET TRIGGERS off; -- further triggers within this transaction are disabled.

A table may have more than one trigger. Their execution order can be specified using the ORDER clause. Each trigger gets an order number, triggers are called starting at the lowest order number in ascending order.

Syntax:

```
CREATE TRIGGER NAME action_time event ON q_table_name
  opt_order opt_old_ref trig_action

action_time
        : BEFORE
        | AFTER

event
        : INSERT
        | UPDATE opt_column_commalist
        | DELETE

opt_order
        | ORDER INTNUM

opt_old_ref
        | REFERENCING old_commalist

trig_action
        : compound_statement

old_commalist
        : old_alias
        | old_commalist ',' old_alias

old_alias
        : OLD AS NAME
        | NEW AS NAME
```

Example 11.8. Creating a simple trigger

This trigger is a simple example of one that would cause an endless loop if further triggering were not disabled.

```
create trigger update_mydate after
   update on mytable referencing old as O, new as N
{
  set triggers off;
  update mytable
    set
      previousdate = O.mydate,
      mydate=now()
```

```
    where id=N.id;
}
;
```

The trigger makes aliases for the values of the column that are part of the SQL manipulation transaction that will be in progress, hence the values of the columns can be accessed as "O.column" and "N.column" for old and new values respectively.

The set statement is scope to the procedure or trigger body where it occurs, plus procedures called from there , thus when the trigger finishes no other triggers are effected by it.

Example 11.9. Creating a simple trigger using INSTEAD OF

This trigger example will show how INSTEAD OF can be used to intercept the values of an insert statement and re-write it. In this case the purpose is to deliberately truncate VARCHAR inserts to prevent an error if the data type bounds are exceeded:

First we create a test table with a 30 character limitation in one of the columns:

```
SQL>create table test_trunc (
      id integer not null primary key,
      txt varchar (30)
      )
    ;

Done. -- 10 msec.
```

Then we attempt to insert 33 characters into it with the following results:

```
SQL>insert into test_trunc (id, txt)
      values (1, 'aaaaaaaaaabbbbbbbbbbccccccccccxxx');

*** Error 22026: [Virtuoso ODBC Driver][Virtuoso Server]SR319: Max column length (30) of column [txt] exc
```

Now we make a trigger to fire instead of insert statements that can perform some custom error correction, in this case we simply want to chop-off any extra characters that will cause an insert to fail.

```
SQL>create trigger test_trunc_it
      instead of insert on test_trunc
        referencing new as N
    {
      set triggers off; -- we do not want this looping...
      insert into test_trunc (id, txt) values (N.id, left(N.txt, 30));
    }
    ;

Done. -- 10 msec.
```

We perform the same test insert, now without errors:

```
SQL>insert into test_trunc (id, txt)
  values (1, 'aaaaaaaaaabbbbbbbbbbccccccccccxxx');

Done. -- 10 msec.
```

And to see what we have in the database, a quick select:

```
SQL> select * from test_trunc;
id                txt
INTEGER NOT NULL  VARCHAR
_____________________________________________________

1                 aaaaaaaaaabbbbbbbbbbccccccccccc

1 Rows. -- 20 msec.
```

11.16.2. Triggers on Views

In virtuoso you can create a trigger on a view. To accomplish this there is only one condition: The first trigger for a given type of event (INSERT/DELETE/UPDATE) must be an INSTEAD OF trigger. After such a trigger is defined then any type of triggers (AFTER/BEFORE) can be added.

Example 11.10. Creating a trigger on view

We will make two tables and an union view for them. Then we will create a trigger which inserts a new record in one of the tables according to values.

First lets create the tables and the view.

```
create table first_table(
  id integer not null primary key,
  txt varchar
);

create table second_table(
  id integer not null primary key,
  txt varchar
);

create view all_tables (id,from_table,txt)
as select id,'first',txt from first_table
union all
select id,'second',txt from second_table;
```

Now lets create a trigger instead of insert for the view and insert some data.

```
create trigger insert_all_tables
  instead of insert on all_tables referencing new as N{
    if(N.from_table = 'first' or N.from_table = 'all')
      insert into first_table (id,txt) values(N.id,N.txt);

    if(N.from_table = 'second' or N.from_table = 'all')
      insert into second_table (id,txt) values(N.id,N.txt);
  };

  insert into all_tables (id,from_table,txt) values (1,'first','into first');
  insert into all_tables (id,from_table,txt) values (2,'second','into second');
  insert into all_tables (id,from_table,txt) values (3,'all','into all');

  select * from all_tables;

  id       from_table  txt
  INTEGER  VARCHAR  VARCHAR
  _____________________________________________________________________________

  1        first    into first
  3        first    into all
  2        second   into second
  3        second   into all
```

You can see that the trigger inserted the data in the two tables according the value of from_table.

11.16.3. The DROP TRIGGER statement

```
DROP TRIGGER qualified_name
```

This drops a trigger of the given name. The name may optionally have a qualifier and owner, in which case these should be the qualifier and owner of the subject table of the trigger. Identical trigger names may exist for identically named tables in different namespaces.

11.16.4. Triggers and Virtual Database

Triggers may be defined on tables residing on remote databases. The semantic of triggers is identical but will of course only take place when the manipulation takes place through the Virtuoso defining the triggers. Trigger bodies may reference remote tables just as any other procedure bodies can. Note that triggers can be used for replication, i.e. one may define a local change to be mirrored to a remote table using a trigger.

Consider an application with a warehouse supplying orders. There is a total value of all orders kept at the warehouse level and there is the total value of all order lines kept at the order level. When an order line is added, both the order value and consequently the total order value are updates. These values are maintained for insert, update and delete of order line. On the other have, when an order is deleted, all corresponding order lines must be deleted.

These rules are maintained with the below set of triggers.

```
drop table T_WAREHOUSE;

drop table T_ORDER;

drop table T_ORDER_LINE;

create table T_WAREHOUSE (W_ID integer default 1,
                          W_ORDER_VALUE float default 0,
                          W_DATA varchar,
                          primary key (W_ID));

create table T_ORDER (O_ID integer not null primary key, O_C_ID integer,
                      O_W_ID integer default 1,
                      O_VALUE numeric default 0,
                      O_MODIFIED datetime);

create table T_ORDER_LINE (OL_O_ID integer,
                           OL_I_ID integer,
                           OL_QTY integer,
                           OL_MODIFIED timestamp,
                           OL_I_PRICE float default 1,
                           primary key (OL_O_ID, OL_I_ID));

create index OL_I_ID on T_ORDER_LINE (OL_I_ID);

create trigger AMT_INS after insert on T_ORDER_LINE
{
  update T_ORDER
    set O_VALUE = O_VALUE + OL_QTY * OL_I_PRICE
    where O_ID = OL_O_ID;
}

create trigger AMT_DEL after delete on T_ORDER_LINE
{
  update T_ORDER
    set O_VALUE = O_VALUE - OL_QTY * OL_I_PRICE
    where O_ID = OL_O_ID;
}

create trigger AMT before update on T_ORDER_LINE referencing old as O
{
  update T_ORDER
    set O_VALUE = O_VALUE - O.OL_QTY * O.OL_I_PRICE + OL_QTY * OL_I_PRICE
    where O_ID = OL_O_ID;
}

create trigger W_VALUE before update (O_VALUE) on T_ORDER
     referencing old as O, new as N
{
  update T_WAREHOUSE
    set W_ORDER_VALUE = W_ORDER_VALUE - O.O_VALUE + N.O_VALUE
    where W_ID = O.O_W_ID;
}

create trigger O_DEL_OL after delete on T_ORDER order 2
{
```

```
  set triggers off;
  delete from T_ORDER_LINE where OL_O_ID = O_ID;
}

create trigger O_DEL_W  after delete on T_ORDER order 1
{
  update T_WAREHOUSE
    set W_ORDER_VALUE = W_ORDER_VALUE - O_VALUE
    where W_ID = O_W_ID;
}

create procedure ol_reprice_1 (in i_id integer, in i_price float)
{
  declare id integer;
  declare cr cursor for
    select OL_I_ID from T_ORDER_LINE;
  whenever not found goto done;
  open cr;
  while (1) {
    fetch cr into id;
    if (id = i_id)
      update T_ORDER_LINE set OL_I_PRICE = i_price where current of cr;
  }
 done:
  return;
}

create procedure ol_reprice_2 (in i_id integer, in i_price float)
{
  declare id integer;
  declare cr cursor for
    select OL_I_ID from T_ORDER_LINE order by OL_I_ID;
  whenever not found goto done;
  open cr;
  while (1) {
    fetch cr into id;
    if (id = i_id)
      update T_ORDER_LINE set OL_I_PRICE = i_price where current of cr;
  }
 done:
  return;
}

create procedure ol_del_i_id_2 (in i_id integer)
{
  declare id integer;
  declare cr cursor for
    select OL_I_ID from T_ORDER_LINE order by OL_I_ID;
  whenever not found goto done;
  open cr;
  while (1) {
    fetch cr into id;
    if (id = i_id)
      delete from T_ORDER_LINE where current of cr;
  }
 done:
  return;
}
```

Compatibility:

Virtuoso triggers are modeled after SQL 3. Omitted are the FOR EACH STATEMENT and related OLD TABLE AS phrases as well as the WHEN in the trigger body. The implementation is otherwise complete.

11.17. Character Escaping

The C style escape character can be used to include special characters inside literals. The backslash character, '\', followed by an octal character code or a special character provides a notation for characters that are normally not typable in a string literal such as tab or crlf. Backslash support can be turned on or off at the statement level, the connection level or server default level.

11.17.1. Statement Level

If you want to activate or deactivate the backslash support in a stored procedure you can use the following two special comments (on a separate line) :

```
--no_c_escapes+
```

turns the backslash escaping support off (insert into x values ('c:\test') will result in 'c:\test' in the column

```
--no_c_escapes-
```

turns the backslash escaping support on. (same as above will insert 'c:test' in the column.)

11.17.2. Connection Level

The switch SET SQL_NO_CHAR_C_ESCAPE can be set to 'on' or 'off' to respectively turn backslash support on or off for the current connection

There is an ODBC connection attribute that can be set for the same effect in an ODBC connection. SQLGetConnectAttr/SQLSetConnectAttr with option ID of 5002 takes values 0 or 1 to facilitate this

11.17.3. Server Default

SQL_NO_CHAR_C_ESCAPE=0/1 can be set in the "Client" section of the virtuoso.ini file to set the connection default backslash handling behavior. The default value is 0.

Note

When a 'create procedure' is executed and that mode is "ON" the procedure is stored in such a way that it will preserve the setting for it's text no matter what the current default is.

11.18. Virtuoso/PL Scrollable Cursors

Virtuoso/PL supports scrollable cursors, providing functionality similar to the ODBC scrollable cursor support. Scrollable Cursor support extends the basic (forward-only) syntax of DECLARE CURSOR and FETCH to support the various fetch directions & cursor modes. The Virtuoso/PL scrollable cursors always operate with a rowset size equal to 1. The keyset size (where applicable) is as per the default.

Note:

If a Virtuoso/PL cursor is declared forward only it supports only FETCH .. NEXT scroll direction. The FETCH defaults its direction to NEXT (if omitted) so this is how the syntax extensions to DECLARE CURSOR & FETCH interoperate with the forward-only cursors syntax.

11.18.1. Declaring a Scrollable Cursor

Virtuoso/PL cursor types are specified at declaration time. Unlike the forward-only cursor declaration the scrollable cursor DECLARE CURSOR causes some actions (cursor statement preparation & cursor variable assignment). The cursor variable's value can not be copied, it should be passed only by reference in procedure calls. Scrollable cursors have an appropriate destructor, which will close the cursor when the cursor variable goes out of scope. Variables in the surrounding context are referenced similarly to the forward-only cursor.

Note:

Some types of statements do not allow other cursor types than static. For example SELECT DISTINCT will always result in a static cursor, ignoring the cursor declared type.

11.18.2. Opening a Scrollable Cursor

The OPEN on a scrollable cursor opens the cursor and sets it's position right before the first resultset row. So before taking the bookmark value at least one FETCH should be issued.

11.18.3. Fetching Data From a Scrollable Cursor

The FETCH on a scrollable cursor allows specification of a direction. If there is no more data in the specified fetch direction this causes the NOT FOUND exception to be raised, as with the forward-only cursors. In addition to that if the row on which the cursor is about to position has been deleted and the isolation level & cursor type allows detecting that, then the exception SQLSTATE 'HY109' (Row deleted) is raised.

Positioning on a bookmark is done the following way:

A bookmark value should be retrieved using the bookmark() function . The value returned by that function can be stored, copied and retrieved. This value can also survive a cursor close and reopen, even between transactions. How the cursor will behave if a bookmark from a cursor with different select statement or scroll type is used for positioning is undefined and should be avoided. On some occasions it may signal an error, on others it will position on a wrong or non-existing row. As a general rule bookmark values should be used only on the cursor from which they are generated.

The cursor should be in opened state. Now a FETCH .. BOOKMARK bm_value INTO ... can be issued with the bookmark variable.

Bookmarks can serve for persisting the cursor position in an VSP context. One can imagine a VSP page which on it's first go will execute a cursor and will show the first so-many rows. Then it can retrieve the bookmark value of the last displayed row, persist it somehow (for example as an HTTP session variable), then close the cursor and exit. On each subsequent hit it will open again the same cursor, position on the bookmark persisted and return the next, previous, first or last so-many rows.

11.18.4. Virtuoso/PL Scrollable Cursor Examples

Example 11.11. Procedure using scrollable cursor to read the keys in batches of 20

```
create procedure READ_KEYS_NEXT_20 (in mask varchar, inout bm any) returns integer
{
  --- This procedure reads the next 20 table names based on a bookmark value.
  declare cr static cursor for
      select distinct KEY_TABLE
        from DB.DBA.SYS_KEYS
        where
          KEY_IS_MAIN = 1 and
          KEY_MIGRATE_TO is NULL and
          KEY_TABLE like mask;
  declare table_name varchar;
  declare inx integer;

  inx := 1;

  -- no 'Row Deleted' (HY109) handling as the static cursors doesn't show the deleted rows.
  whenever not found goto done;
  open cr;

  -- positions on the bookmark or on the first if it is null
  -- and fetches the value into table_name
  if (bm is not null)
    {
      fetch cr bookmark bm into table_name;
      -- note that the value from fetch bookmark is omitted
      fetch cr next into table_name;
    }
  else
    fetch cr first into table_name;

  -- fetches the next 20 rows (or less)
  while (inx < 20)
    {
      result (table_name);
      inx := inx + 1;
      fetch cr next into table_name;
    }

  -- 20 rows were fetched - get the bookmark of the last row fetched
  bm := bookmark (cr);
  close cr;
```

```
  return;

done:

  -- no more rows - set the bookmark to NULL
  close cr;
  bm := NULL;
};

create procedure READ_KEYS (in mask varchar)
{
  -- the main function (mask is a mask to be applied over the select

  declare table_name varchar;
  declare bm any;

  -- it'll return a resultset with a single column
  result_names (table_name);

  -- sets the bookmark to empty
  bm := NULL;

  while (1)
    {
       READ_KEYS_NEXT_20 (mask, bm);

       -- that's the flag for no more rows
       if (bm is NULL)
         return;
    }
};

READ_KEYS ('%');
```

11.18.5. FORWARD-ONLY (traditional cursor statement) Example

Example 11.12. Procedure using forward only cursor

```
create procedure TEST_FW()
{
  declare cr cursor for select KEY_ID from DB.DBA.SYS_KEYS;
  declare inx, data integer;
  inx := 0;

  whenever not found goto done;
  open cr;
  while (1)
    {
      fetch cr into data;
      inx := inx + 1;
    }
done:
  close cr;
  result_names (data);
  result (inx);
};
TEST_FW();
```

11.18.6. DYNAMIC (traditional cursor statement) Example

Example 11.13. Procedure using dynamic cursor

```
create procedure TEST_DYNAMIC ()
{
  declare cr dynamic cursor for select KEY_ID from DB.DBA.SYS_KEYS;
  declare inx, data integer;
  inx := 0;

  whenever not found goto done;
```

```
  open cr;
  while (1)
    {
      fetch cr into data;
      inx := inx + 1;
    }
done:
  close cr;
  result_names (data);
  result (inx);
};
TEST_DYNAMIC ();
```

11.18.7. KEYSET (traditional cursor statement) Example

Example 11.14. Procedure using keyset cursor

```
create procedure TEST_KEYSET ()
{
  declare cr keyset cursor for select KEY_ID from DB.DBA.SYS_KEYS;
  declare inx, data integer;
  inx := 0;

  whenever not found goto done;
  open cr;
  while (1)
    {
      fetch cr into data;
      inx := inx + 1;
    }
done:
  close cr;
  result_names (data);
  result (inx);
};
TEST_KEYSET ();
```

11.19. Virtuoso PL Modules

Modules are packages of procedures which compile together. Procedure names in module definitions are not fully qualified names, but consist only of a single identifier that it is appended to the name of the module (which is a 3-part name) to make the 4-part module procedure name.

Module procedures do not appear in SQLProcedures output. Module names are in the same domain as the procedure names, so it is not possible to have a procedure with the same name as an existing module.

11.19.1. Syntax

```
CREATE MODULE
  m_name
{
  [PROCEDURE|FUNCTION] p_name1 (...) { ...};
  [PROCEDURE|FUNCTION] p_name2 (...) { ...};
  ...
  [PROCEDURE|FUNCTION] p_nameN (...) { ...};
}

DROP MODULE m_name;
```

Example 11.15. Procedure Modules

```
create module
  DB.DBA.MOD
{
  function MOD1 () returns varchar {
    return ('MOD1');
```

```
  };

  procedure MOD2 () {
    return concat (MOD1(), 'MOD2');
  };
};
```

This example creates a module, MOD, with 2 procedures: MOD1 & MOD2. Their fully-qualified names are DB.DBA.MOD.MOD1 and DB.DBA.MOD.MOD2.

Note the call to MOD1 in MOD2 - it is not fully qualified, but it resolves to the module procedure MOD1, instead of any procedure external to the module.

A single part procedure name in a call inside a module is first matched against procedures defined in the module. If the above example were executed by DBA (in the DB qualifier), then the below statements are equivalent:

```
select DB.DBA.MOD.MOD1()
select DB..MOD.MOD1()
```

The statement:

```
select MOD.MOD1()
```

will result in calling the DB.DBA.MOD.MOD1() only if a function DB.MOD.MOD1 does not exist. If it exists, it will be preferred over DB.DBA.MOD.MOD1 when using this notation.

11.19.2. Security

Module procedures can be granted to users. Modules can also be granted to users. Granting execute to a module is equivalent to granting execute for all of the module's procedures.

11.20. Handling Conditions In Virtuoso/PL Procedures

Condition handlers determine the behavior of a Virtuoso/PL procedure when a condition occurs. You can declare one or more condition handlers in your Virtuoso/PL procedure for general SQL conditions or specific SQLSTATE values.

If a statement in your procedure raises an SQLEXCEPTION condition and you declared a handler for the specific SQLSTATE or SQLEXCEPTION condition the server passes control to that handler.

If a statement in your Virtuoso/PL procedure raises an SQLEXCEPTION condition, and you have not declared a handler for the specific SQLSTATE or the SQLEXCEPTION condition, the server passes the exception to the calling procedure (if any). If the procedure call is at the top-level, then the exception is signalled to the calling client.

Handlers are active only for the duration of the enclosing compound statement. When an exception is thrown outside the handler's scope then this handler is never called.

11.20.1. Declaring Condition Handlers

The general form of handler declaration is:

```
DECLARE <handler_type> HANDLER FOR
        <condition> [, <condition [...]]
        <sql_procedure_statement>;
```

For compatibility handlers can be declared also as :

```
WHENEVER <condition> [GOTO <label>|DEFAULT];
```

When Virtuoso raises a condition that matches the <condition>, the <sql_procedure_statement> gets executed and when (and if) it finishes the execution continues according to the <handler_type>.

```
<handler_type>
```

CONTINUE - Specifies that after <sql_procedure_statement> completes, execution continues with the statement after the statement which caused the error.

EXIT - Specifies that after <sql_procedure_statement> completes, execution continues after the end of the compound statement that contains the declaration of the handler.

```
<condition>
```

NOT FOUND - Identifies any condition that results in SQL STATE = SQL_NO_DATA_FOUND (+100)

SQLEXCEPTION - Identifies all character SQL STATEs excluding ones starting with '01', '02' and '00'

SQLWARNING - Identifies character SQL STATEs starting with '01'. This is a shortcut for SQLSTATE '01*'

SQLSTATE [VALUE] '<sql_state_mask>' - Identifies character SQL STATEs. The <sql_state_mask> can be a full 5 character value or 0-4 characters followed by an '*'. When a '*' is present then any SQL STATE signal led which starts with the same characters as the <sql_state_mask> before the '*' will cause that handler to execute. For example the exception with SQL state '42S22' will match all the following

```
SQLSTATE <conditions> :
SQLSTATE '*' SQLSTATE '42*'
SQLSTATE '42S22'
```

Note:

Handler's call priority is determined by the number of matching characters in this mask.

```
<sql_procedure_statement>
```

This can be any allowed Virtuoso/PL statement as well as an compound statement. This statement is executed in the same procedure context as the procedure body itself, so any labels and variables in the procedure body can be used and RETURN causes the procedure to end. No handler is active while the <sql_procedure_statement> is executed. So any exception raised is passed directly to the procedure caller. The <sql_procedure_statement> can be empty resulting in the <handler_type> action being taken right after setting the __SQL_STATE & __SQL_MESSAGE variables.

When multiple active handlers <condition>s match the exception being raised Virtuoso chooses the closest to the statement raised the exception that has a largest call priority. This means that if an exception is there are two handlers with condition SQLSTATE '4*' and SQLSTATE '42*' and an exception with SQL STATE '42S22' is raised the handler with <condition> '42*' will be called.

```
WHENEVER <condition> GOTO <label>
```

is an equivalent of:

```
DECLARE EXIT HANDLER FOR <condition> GOTO label.

WHENEVER <condition> DEFAULT
```

is equivalent of:

```
DECLARE EXIT HANDLER FOR <condition> RESIGNAL;
```

The following examples demonstrate simple common handlers:

CONTINUE handler: The handler assigns a value of 1 to a local variable at_end when a NOT FOUND condition is raised. The execution then continues with the statement after the signal.

```
create procedure test1 ()
{
  declare at_end integer;

  at_end := 0;
  declare continue handler for NOT FOUND at_end := 1;

  result_names (at_end);
```

```
  result (at_end);
  signal (100);
  result (at_end);
}
```

When this procedure gets executed it returns the following result set :

```
0
1
```

EXIT handler: The handler assigns a value of 2 to a local variable at_end when a NOT_FOUND condition is raised. The execution then continues with the statement after the compound statement containing the signal.

```
create procedure test2 ()
{
  declare at_end integer;

  result_names (at_end);

  at_end := 0;
  declare exit handler for NOT FOUND at_end := 1;

    {
      result (at_end);
      signal (100);
      result (3);
    }
  result (at_end);
}
```

When this procedure gets executed it returns the following result set :

```
0
1
```

```
__SQL_STATE and __SQL_MESSAGE variables.
```

All Virtuoso/PL procedure have two variables implicitly declared :

```
declare __SQL_STATE any;
declare __SQL_MESSAGE varchar;
```

Initially they are set to 0.

When an exception is raised these variables are set as follows :

```
__SQL_STATE gets the SQL STATE (character string or integer 100 for NOT FOUND)
__SQL_MESSAGE gets the SQL MESSAGE (character) or NULL if no message.
```

Their values are preserved until the next exception overwrites them.

```
RESIGNAL statement

Syntax :
RESIGNAL [ '<new_sql_state>' ]

RESIGNAL  is a shortcut for    signal (__SQL_STATE, __SQL_MESSAGE)
RESIGNAL '<new_sql_state>' is a shortcut for     signal ('<new_sql_state>', __SQL_MESSAGE);
```

This statement resignals the current exception to the caller of the procedure.

11.20.2. Stack Trace Reporting On Sql Error Generation

When an exception occurs the Virtuoso server has the ability to provide information about the procedure call stack. It appends the call stack information to the error message text. There are also line numbers besides each level of the call stack which are a Virtuoso/PL procedure. The line numbers mark the beginning of the innermost compound statement.

The call stack reporting mode is controlled by the "CallstackOnException" option in the *Parameters* section of the Virtuoso INI file .

This parameter takes the following values:

0 (default) - Call stack reporting disabled.

1 - Call stack is reported but does not include values of arguments.

2 - Call stack is reported and contains all available information.

Call stack reporting can be a security hole because it can demonstrate internal logic of the system to the end user; this is especially important for dynamic web pages. Mode 2 is especially insecure because it may print values of function arguments that may contain confidential information.

Some client applications are unable to handle long error messages properly. Client-side APIs for ODBC and similar protocols assume that client should allocate a buffer for error message string and then ask the API to save the message string to the specified buffer of the specified size. Not all client applications work properly if a message does not fit in the buffer. If an client application you use reports an empty string instead of error message or displays a message like 'Error message is too long' then you may wish to decrease the value of the "CallstackOnException" option to keep messages shorter.

11.21. Procedure Language Debugger

Virtuoso has step by step PL debugging capabilities and a call stack trace option. When Virtuoso is running with a console or with stdout going somewhere like a file you can print debug messages directly to that using `dbg_printf()` and `dbg_obj_print()` functions.

Procedure source code can be written to an ASCII file using any text/programming editor. This file can be loaded into ISQL using the load command or directly from the command line using the ISQL batch mode:

```
isql data-source user password file
```

ISQL can then be used to test the procedures functionality. Procedures can return result sets to ISQL for data debugging purposes too. When dealing with results sets in ISQL remember to call procedures using the `CALL` keyword to ensure that ISQL outputs all result sets:

```
CALL procedure(arguments);
```

The PL procedure debugger allows you to execute procedures step-by step, or stop execution based upon a predefined condition within Virtuoso/PL procedure(s). This feature is not intended for SQL statements debugging. SQL statements are treated as atomic units of execution, it is not possible to step into these. It is possible to step over a procedure which loops over a cursor, though. Procedures are compiled with extra information for debugging, thus the speed of execution may be somewhat reduced. It is possible to step in or step over statements, however this does not apply to SQL statements. In order to start debugging on some event, breakpoints are required. The breakpoints can be global or temporary. The global breakpoint means the process will stop with or without a debug session. The temporary breakpoints are to stop execution only in the current debug session.

Procedure debugging can be activated in two ways. You can instruct Virtuoso to enable debugging information for all procedures using the PLDebug parameter in the Virtuoso INI file, or for selected procedures by including a special comment "`--pl_debug+` " at the top of the procedure. Debug information is visible only from a debugging tool such as ISQL

ISQL can be used as a procedure debugging tool. ISQL is started in debug mode using the -D switch:

```
isql <host:port> dba <dba-password> -D
```

Debugging of the server is restricted to dba privileged accounts for security reasons. The summary of the debug commands available can be viewed from typing help in the debugger. The following list will be shown:

```
DEBUG> help
OpenLink Interactive PL Debugger (Virtuoso).

Available commands:
    BREAK procedure_name [line number] - Set breakpoint at specified line or PL function
    NEXT  - Step program, proceeding through PL subroutine calls.
    INFO (THREAD|CLIENT|BREAK) - Generic command for showing things about the program/process being debug
            THREAD - Running threads
```

```
            CLIENT - Connected SQL/ODBC clients
            BREAK - Active breakpoints
     ATTACH thread_id|client_id - Attach to a running process.
     STEP  - Step PL program until it reaches a different source line.
     LIST [procedure name] [line number] - List specified procedure or line.
     WHERE  - Print backtrace of all stack frames.
     CONTINUE  - Continue PL program being debugged after breakpoint.
     PRINT variable_name - Print value of variables or arguments.
     SET variable_name new_value - Assign a specified value to a variable.
     DELETE ([breakpoint_number]|[procedure_name] [line_number]) - Delete some breakpoints.
     FRAME frame_number - Select and print a stack frame.
     FINISH  - Execute until returns.
     UNTIL line_number - Execute until the program reaches a source line greater than the current.
```

There are two debug modes: local, for issuing local breakpoints and step commands; global, for making global breakpoints.

There are two console or screen modes to debug/attach-to. These are: the global console, which allows you to issue the INFO, ATTACH, BREAK, DELETE commands; the local console, which once attached (using the ATTACH command) via the global console becomes local and all possible commands are available.

There are two syntaxes for specifying breakpoint locations, `BREAK FFFF NNN` and `BREAK FFFF` where FFFF is a name of a procedure.

Local breakpoints will interrupt execution only if there is a connected debug session. Global break points will cause the process to stop regardless.

The source identification and line number output can be controlled with a special comment in the PL code. This can be useful when you want to alter the compilers notion of the current file and line number. `--src FILENAME:LINENUMBER` can be used to reset the counters to the specified filename and line number. This is also useful for handling test coverage output - see the next section.

Example 11.16. Debugging simple procedures

Consider the following PL procedures:

```
create procedure
db.dba.test_proc1 (in b varchar)
{
  --pl_debug+
   declare c integer;
   c := 0;
   for select u_id, u_name from sys_users do
     {
       c := u_id;
     }

   c := vector (1);

   c := test_proc2 ();

   return c;
};

create procedure
db.dba.test_proc2 ()
{
  --pl_debug+
  declare x, y any;
  x := 1;
  if (x = 1)
     {
       y := x + 1;
     }
   else
     {
       y := 2;
     }
   return y;
};
```

Here is a simple debugging session on the above procedures:

```
$ isql localhost:1111 dba dba -D
Debug session established to localhost:1111
```

Display the test procedures to familiarize yourself with the lines and their line numbers.

```
DEBUG> list db.dba.test_proc1;
1 create procedure
2 test_proc1 (in b varchar)
3 {
4   --pl_debug+
5   declare c integer;
6   c := 0;
7   for select u_id, u_name from sys_users do
8     {
9       c := u_id;
10     }

DEBUG> list db.dba.test_proc1 11;
11
12   c := vector (1);
13
14   c := test_proc2 ();
15
16   return c;
17 }
18
19
20
```

put a break after for select statement.

```
DEBUG> break db.dba.test_proc1 11
Breakpoint at: procedure DB.DBA.test_proc1, line 11
```

Now, start another ISQL session to Virtuoso and run the procedure that we are debugging, DB.DBA.test_proc1('), and wait for it to hit the breakpoint. Back to the debug ISQL, get the list of breakpoints:

```
DEBUG> info thre
@1111:4 in DB.DBA.test_proc1 () at 11
```

attach to thread of interest:

```
DEBUG> attach @1111:4
1111:4
```

Find out where it stopped:

```
DEBUG> where
#0 DB.DBA.test_proc1 () at 11
#1 ?? () at 0
```

print the value of 'c' now, and continue step by step

```
DEBUG> print c
$25 "c" INTEGER (189) 4
DEBUG> next
11
DEBUG> print c
$25 "c" INTEGER (189) 4
DEBUG> next
12   c := vector (1);
DEBUG> print c
$25 "c" ARRAY_OF_POINTER (193) (1 )
DEBUG> next
13
```

step into the test_proc2 procedure:

```
DEBUG> step
14   c := test_proc2 ();
```

examine the stack

```
DEBUG> where
#0 DB.DBA.test_proc2 () at 4
#1 DB.DBA.test_proc1 () at 14
#2 ?? () at 0
```

step through the test_proc2 procedure

```
DEBUG> next
7   x := 1;
DEBUG> next
8   if (x = 1)
DEBUG> next
10       y := x + 1;
DEBUG> next
11     }
DEBUG> print y
$25 "y" INTEGER (189) 2
```

set the 'y' to be 100

```
DEBUG> set y 100
The 'y' is set to '100'.
DEBUG> next
16   return y;
DEBUG> print c
$25 "c" INTEGER (189) 100
```

continue the execution

```
DEBUG> cont
Execution resumed
DEBUG>
```

11.21.1. Branch Coverage

The Virtuoso ini file contains a parameter in [Parameters] section called TestCoverage whose value is the name of an XML file that will accumulate test coverage data.

The PLDebug switch controls the type of debugging enabled:

PLDebug = 0 - default debugging information and test coverage disabled.

PLDebug = 1 - debugging information enabled.

PLDebug = 2 - debugging information enabled, test coverage data will be written to file specified in TestCoverage Virtuoso ini file parameter.

If the test coverage file is specified the initial state of the accumulation is read from the coverage file if one exists. If the file does not exist then coverage starts from scratch.

The format of the file for store/restore of coverage data is:

```
<pl_stats>
  <proc name="xxx" calls="nn" file="ff" >
    <line no="xx" ctr="nn" /> ..
    <caller name="nnn" ct="nn" /> ..
  </proc>
  ...
</pl_stats>
```

An example of file:

```
<pl_stats>
  <proc name="DB.DBA.pop_get_command" calls="00000133" file="pop3_svr.sql" >
    <line no="000152" ctr="000133" />
    <line no="000153" ctr="000133" />
    <line no="000156" ctr="000133" />
    <caller name="DB.DBA.POP_IF_COMMAND" ct="000070" />
    <caller name="WS.WS.NN_SRV" ct="000063" />
  </proc>
</pl_stats>
```

For the purpose of making a Coverage report the format is extended. The lct attribute of the <proc> element contains a total number of lines for the procedure. The CDATA sections of the <line> element contain the excerpt of the line in question up to a maximum of 50 characters.

An example of the extended file format (made with cov_store ('ccc.xml', 1)) is:

```
<proc name="DB.DBA.pop_is_ok_" calls="00000034" file="pop3_svr.sql" lct="22" >
 <line no="000239" ctr="000034"><![CDATA[6    _idx := pop_atoi (num); ]]></line>
 <line no="000240" ctr="000034"><![CDATA[7    _all := length (_stat); ]]></line>
 <line no="000242" ctr="000034"><![CDATA[9    if ((_idx < 1) or (_idx > _all - 1)) ]]></line>
 <line no="000247" ctr="000034"><![CDATA[14    if (pop_is_deleted (_idx, _stat)) ]]></line>
 <line no="000253" ctr="000034"><![CDATA[20    return 1; ]]></line>
 <caller name="DB.DBA.POP_DELE" ct="000011" />
 <caller name="DB.DBA.POP_RETR" ct="000023" />
</proc>
```

The comment `--src xxxx:yyy` indicates the file xxx and line yyy. This comment will be added at the end line of each procedure when loading a file via isql or when generating SQL procedure constants for C code from sql files. These files and lines will be used for annotating the coverage report.

The function `cov_load()` is used to cumulatively load a test coverage file into a running system. Repeatedly calling this allows making a union of multiple separately obtained coverage files which will all show in the ultimate report.

The function `cov_store (in out_file varchar, in add_comments integer := 0)` is used to make an XML file for further XSL-T processing with `cov_report()` (see below) or to make a snapshot of current coverage statistics. The add_comments parameter is used to add a line info and count, usually it takes 1 for making the report, so internally it is used with add_comments equal to 0 to store the cumulative report in the TestCoverage file.

The function `cov_report (in file varchar, in outdir varchar)` makes the full coverage report based on the current state and writes it out.

The report consists of a summary file called profile.prof, containing execution counts and times for individual functions, sorted by self time as well as by time including called functions. For each distinct file mentioned in loaded procedures there is a file called <outdir>/<orgname>.sql.cov under the server's working directory. This file contains listings of all procedures from the file, annotated with line numbers at the start of each line. Each line with executable code will have an execution count. The line number field has six leading characters, the exec count is the next six characters, the line follows. The lines that are not executed are denoted with ## on place of exec count number.

Procedures that do not have source info associated are listed in name order in the file unnamed.sql.cov in the same directory.

An example of files generated by cov_report ('ccc.xml', 'cov/') are:

```
---->

>> begin <<<----
Lines: 806
TotalLines: 2741
Coverage: 29 %

[times called] [proc name]
    6250 : WS.WS.FINDPARAM
    --- code info
    [line in file] : [how many times executed] : [offset from beginning] [line excerpt]
       664 :   6250 : 5    if (pkey is null)
       665 :     ## : 6      pkey := '';
       666 :  12500 : 7    i := 0; l := length (params);
       667 :   6250 : 8    pkey := rtrim (pkey,': ');
```

```
   668 :   6250 : 9    while (i < l)
   670 :  35966 : 11        ret := http_request_header (vector (para
   671 :  35966 : 12        if (ret is not null)
   672 :   3014 : 13          return ret;
   673 :  32952 : 14        i := i + 1;
   675 :   3236 : 16    return '';

-- caller information
[time called] [from proc]
     6 : WS.WS.MOVE
     9 : WS.WS.COPY
   267 : WS.WS.UNLOCK
   269 : WS.WS.GET
   982 : WS.WS.PUT
  1028 : WS.WS.LOCK
  3689 : WS.WS.ISLOCKED
..... more sections follow .....
```

11.21.2. Coverage Functions

`cov_load()`

`cov_store()`

`cov_report()`

`pldbg_stats()`

`pldbg_stats_load()`

11.22. Row Level Security

Organizations often need to compartmentalize access to data. This may be implemented at the level of physically segregating systems, building specific application logic or within the database.

Physically disconnected systems can be troublesome in an increasingly integrated corporate IS environment. Building access rules into application logic, typically in a middle tier is possible and flexible but the protection runs the risk of being circumvented by direct access to the database, through business intelligence tools for example.

For these reasons some database level security enforcement is needed in most applications. SQL provides table and column level privileges which can be granted to users and roles. These do not address issues where one department's data should be accessible when another department's data should not. Such segregation within a table is usually done with views which hard-code selection criteria. The table itself will not be granted but views to specific ranges of rows will be granted to users.

This has the inconvenience of requiring administration of views and requiring applications to use different views for different end users.

Row level security or policy based security allows the SQL compiler to make choices according to which user is accessing any given table. Extra conditions will be introduced into a SQL statement at compile time in order to limit the user to a specific range of rows. This will apply equally to reading and modifying the table.

A policy is a SQL function that will be called by the SQL compiler each time a table having the policy is accessed in a user's query or stored procedure. The policy can return extra conditions which will be and'ed to the conditions in the query. After this is done, the new query is optimized and compiled. This mechanism makes it possible to transparently customize which information a user account sees without having to maintain a multiplicity of static views. This mechanism cannot be subverted by a user and will work no matter what application is used against the database.

Let us consider the example of a table of classified documents. To access a document, the user needs to have a record in a need-to-know table which forms a many-to-many between classifications and users. If the user is a member of the security_auditor role, all documents will be accessible. Users themselves may neither read nor update the need-to-know table.

```
create role staff;
create role security_auditor;
grant staff to security_auditor;
```

```
create table document (d_id varchar, d_changed datetime, d_author varchar, d_classification int, d_text l

create table document_access (da_classification int, da_user  varchar,
primary key (da_d_id, da_user));

grant all privileges on document to staff;
grant all privileges on document_access to security_auditor;

create procedure d_policy (in tb varchar, in op varchar)
{
  if (user_has_role (user, 'security_auditor')) return '';
   if (user_has_role (user, 'staff'))
    return 'exists (select 1 from document_access where da_user = user and da_classification = d_classifi
  return '1=2';
}

table_set_policy ('document', 'd_policy', 'IDUS');
```

These operations first define two roles, staff and security_auditor. The staff is not granted access to document_access, which is configured by security_auditor.

The policy function will, for each query accessing document, check if the user is a member of security_auditor. If so, there will be no extra conditions and all rows will appear. If the user is a member of staff, an extra condition checking the existence of a need to know is added. This will check that there is an entry in the document_access where the accessing user is granted access to the document's classification. If the user is neither member of staff nor of security_auditor, no rows will be returned since the always false condition of 1=2 is added.

Besides selects, the policy rule will apply to inserts, updates and deletes. This is caused by the 'IDUS' parameter to table_set_policy. Different policies may be defined for select, insert, update and delete. Most often these will be the same. Only one policy function is allowed per table and operation. In order to set a policy, one must either be the table's owner or dba. Policies do not apply to dba users.

We note that the need to know check accesses a table which is not granted to staff, even for reading. This is OK, since the predicate coming from the policy function has access rights of the owner of the policy function. The predicate is treated like a view body that were granted to the user making the query without requiring that component tables of the view were granted.

Since policies are processed at the level of SQL compilation, these will equally apply to local and remote tables.

Queries inside stored procedures are subject to policy. The user whose policies are consulted is the owner of the stored procedure. The policy conditions are fixed at the time of compiling the procedure. Reassigning a policy counts as a change to the table, causing recompilation of all concerned procedures, triggers and client statements.

Conclusions

Policies are an effective way of implementing a 'virtual private database,' providing for isolation between classes of users of the same application. The maintenance overhead is small compared to the trouble of maintaining views for each class of users. The table concerned remains one schema object with one name. Policies can be set and maintained without modification to applications and existing applications can be made to use policy based access control at no additional cost.

11.22.1. Row Level Security Functions

`table_set_policy()`

`table_drop_policy()`

11.23. Vectored Execution and Query Parallelization

Note: This feature only applies to Virtuoso 7.0 and later.

Vectored execution means executing queries or stored procedures simultaneously on multiple sets of parameters. Further, when a query contains a *JOIN* , a single invocation of the query will, with vectored execution, execute every consecutive *JOIN* step with multiple inputs. When every stage of a query's evaluation is performed on a large number of intermediate result rows at a time, two benefits are obtained:

1. The interpretation overhead disappears and
2. Locality of reference in

   JOINs

   can be better exploited.

For example, with a *JOIN* like:

```
SELECT  COUNT(*)
  FROM  part,
        lineitem
 WHERE  l_partkey = p_partkey
   AND     p_size < 23
OPTION  ( LOOP, ORDER );
```

the outermost *LOOP* of the query will look for parts with *p_size < 23* . The part keys of these are used as lookup keys for an index on *l_partkey* in *lineitem* . This index translates these values into values of the primary key, *l_orderkey* , *l_linenumber* , which is then used to get the data row with the price and discount. When each of these steps is done with tens of thousands of values at the same time, the SQL interpretation overhead is almost completely eliminated and locality can be exploited when accessing nearby rows. The chance of hitting nearby rows also increases when the size of the intermediate result batch increases.

11.23.1. Automatic Query Parallelization

If a query does not modify data, executes in *READ COMMITTED* isolation, and contains some form of aggregation or *ORDER BY* , it can be automatically parallelized. Parallelization typically splits the query's outermost *LOOP* into approximately equal size chunks, which are independently evaluated each on its own thread. The results are merged together when all are ready, and are combined in an aggregation or *ORDER BY* . This is entirely transparent to the user.

11.23.2. Configuration Parameters for Vectoring and Parallelization

The following virtuoso.ini [Parameters] section entries concern query parallelization and vectoring:

Note: These settings only apply to Virtuoso 7.0 and later.

- AsyncQueueMaxThreads :

  Sets the number of threads in a pool that is used for getting extra threads for running queries and for aq_request. Each running statement has at least one thread that is not allocated from this pool plus zero or more threads from this pool.

  Setting the pool size to the number of cores plus a few is a reasonable default. On platforms with core multithreading, one can count a core thread as a core for purposes of this parameter.

  If one expects to run many slow `aq_requests()` (see `async_queue()` , `aq_request()` , etc.), then the number of threads should be increased by the number of slow threads one expects.

  Slow threads are typically I/O bound threads used for web crawling or similar long-latency, low-CPU activity.

  Note: Only effective with Virtuoso 7.0 and later.
- ThreadsPerQuery :

  This is maximum number of threads that can be claimed from the thread pool by a single query. A value of one means that no query parallelization will take place, and all queries will run single threaded.

  The number of cores on the machine is a reasonable default if running large queries.

  Note that since every query is served by at least one thread, a single query taking all the extra threads will not prevent other queries from progressing.

  Note: Only effective with Virtuoso 7.0 and later.
- VectorSize This the number of simultaneous sets of query variable bindings processed at one time. The default is 10,000, which is good for most cases.

If we are evaluating the query:

```
SELECT COUNT (*)
FROM t1 a
NATURAL JOIN t2 b
WHERE a.row_no + 1 = b.row_no
OPTION (LOOP, ORDER)
```

with vector size of 10,000, then 10,000 rows of t1 a will be fetched first; 1 will be added to the 10,000 row_no values; and then the corresponding row of t1 b will be fetched for the 10,000 row_no of t1 a. This process will repeat until enough batches of t1 a have been fetched to come to its end.

Note: Only effective with Virtuoso 7.0 and later.

- AdjustVectorSize :

  Using a larger vector size when evaluating large queries with indexed random-access can yield up to a 3x speed-up relative to using the default vector size. However, always using a large vector size will prohibitively increase the overhead of running small queries. For this reason, there is the option to adaptively select the vector size. Set AdjustVectorSize = 1 to enable this feature. The SQL execution engine will increase the vector size when it sees an index lookup that does not get good locality, (e.g., after sorting the keys to look for, too few consecutive lookups fall on the same page). Having more keys to look up increases the chance that consecutive keys should be found on the same page, thus eliminating much of the index lookup cost.

  Note: Only effective with Virtuoso 7.0 and later.

- MaxVectorSize :

  When AdjustVectorSize is on, this setting gives the maximum vector size. The default is 1,000,000 and the largest allowed value is about 3,500,000.

  Note: Only effective with Virtuoso 7.0 and later.

Chapter 12. Database Event Hooks

Abstract

Virtuoso provides a number of hooks that enable you to traps events within the database such as startup, shutdown, connection, disconnection and SQL compilation events.

Virtuoso allows the dba to set hooks for various events, such as:

Startup
Shutdown
Client Connect
Client Disconnect
Compilation of a Dynamic SQL Statement

These events are intercepted by calling a SQL procedure if one is defined.

In the following we will examine each hook in the context of a simplified database security system. The system will record all logins and logouts and will enforce custom security rules on reading a specific application table.

12.1. Database Startup

`DB.DBA.DBEV_STARTUP()`

If defined, this is called after the server initialization, including roll forward of transaction logs, compilation of stored procedures, etc..., but before starting to listen for clients. Hence this is guaranteed to run before any clients connect or any HTTP messages are serviced.

The procedure will run in its own transaction, which will be automatically committed upon return, regardless of run time errors. Result sets are not allowed and return values are discarded. All custom Virtuoso Server Extensions (VSEs) will be defined when calling this procedure.

Example 12.1. Sample Startup Procedure Hook

```
create procedure DB.DBA.DBEV_STARTUP ()
{
  dbg_obj_print (' server started ');
}
```

12.2. Database Connections

`DB.DBA.DBEV_CONNECT()`

If defined, this hook is called for each successful ODBC or JDBC connection. The hook is only called after password and license checks have passed. The hook is called before the server acknowledges a successful connect and is therefore guaranteed to run before any other action by the connected client.

The function runs in its own transaction, which is automatically committed following successful return. Result sets may not be generated and return values are discarded. The connection_id () and user are defined in the function. SQL states signalled inside this hook will be sent to the client and will cause the connection to be closed server side.

Example 12.2. Simple Connection Logging

```
create table security_log (
  sl_user varchar,
  sl_logged_in datetime,
  sl_logged_out datetime,
  primary key (sl_user, sl_logged_in)
);

create procedure DB.DBA.DBEV_CONNECT ()
{
  dbg_obj_print (user, ' connected');
  if (user = 'NOGO')
    signal ('EAUTH', '  External Authorization Failed');
  insert into security_log (sl_user, sl_logged_in) values (user, curdatetime ());
  connection_set ('login_time', curdatetime ());
}
```

This function will print a message to the server's standard output, check if the username of the logged in user is NOGO and disconnect the user if so, This will thereafter insert a row into the security_log table and set the connection variable 'login_time' to be the current time. This information will be used at disconnect time to identify which entry to mark closed.

`DB.DBA.DBEV_DSN_LOGIN (inout dsn varchar, inout user_id varchar, inout pwd varchar, )`

If defined this function is executed just before Virtuoso makes connections to remote data sources. It can change all of it's inout parameters depending on it's logic.

Example 12.3. Remote Connection Hook

This examples contains a sample DBEV_DSN_LOGIN hook that will be called just before the Virtual Database connection to a datasource is made. The following parameters can be used in this function as demonstrated in this example:

dsn : the dsn to connect to

user_id : the user id used in connecting to the DSN

pwd : the password used in connecting to the DSN

```
create procedure "DB"."DBA"."DBEV_DSN_LOGIN"
(in dsn varchar,
 inout user_id varchar,
 inout pwd varchar)
{
  if (user_id = 'U1' and pwd = 'U1')
    {
      -- map U1 to U2
      dbg_obj_print ('mapping U1 to U2');
      user_id := 'U2';
      pwd := 'U2';
    }
   dbg_obj_print (dsn, user_id, pwd);
};
```

12.3. Database Logins

`DB.DBA.DBEV_LOGIN (inout user_name varchar, in digest varchar, in session_random varchar, )`

This function, if defined, will always be called by Virtuoso just before a client is authenticated against the Virtuoso Server. Three parameters are available for audit purposes or any other pre-processing purpose totally user definable.

1. *user_name:*

   IN : the user name from the login data

   OUT : the user name used by Virtuoso for this connection
2. *digest* : the password digest calculated by the client (or the password itself for older clients)
3. *session_random* : the random key used to calculate the digest

This hook can be used to control how Virtuoso proceeds with the client login by responding to 3 possible return values:

-1 - continue with normal verification

0 - reject the login

1 - allow the login (the user returned should be a valid Virtuoso local user name

Example 12.4. Sample Database Login Hook

```
create procedure "DB"."DBA"."DBEV_LOGIN" (
    inout user_name varchar,
    in digest varchar,
    in session_random varchar)
{
  -- the hook runs as DBA
  dbg_obj_print ('user=', user);

  -- and is compiled as DBA
  declare uid_cnt integer;
  uid_cnt := 0;
  select count (*) into uid_cnt from SYS_USERS where U_NAME = user_name;
  dbg_printf ('%ld local users found with this name', uid_cnt);

  if (user_name = 'masterdba')
    {
      -- 'masterdba' is a valid remotely defined user
      -- it's password is 'masterdbapwd'
      -- we're going to check it's password and then map it to dba

      declare md5_ctx any;
      declare my_digest varchar;
      declare pwd varchar;

      -- this is our assumption for the user's password
      -- this can come from an external source as well
      pwd := 'masterdbapwd';

      -- calculate the MD5 digest for checking the password that the client supplied.
      -- note that it uses the 'masterdba'/'masterdbapwd' to calculate it since we
      -- assume that these are the data the client supplied as user id and password
      --
      -- The way to calculate this is FIXED.
      -- The ONLY variables are the user id and the password

      -- START of the FIXED digest calculation sequence
      md5_ctx := md5_init();
      md5_ctx := md5_update(md5_ctx, session_random);
      md5_ctx := md5_update(md5_ctx, user_name);
      md5_ctx := md5_update(md5_ctx, pwd);
      -- the 0 parameter to the md5_final causes it to return the bytes
      -- instead of representing it as hexadecimal characters
      my_digest := md5_final (md5_ctx, 0);
      -- END of the FIXED digest calculation sequence

      -- now compare the calculated digest with the one supplied by the client
      -- note the OR here - some older clients MAY supply the password in plain text
      if (my_digest = digest or pwd = digest)
        {
          -- the match says that the client indeed supplied 'masterdbapwd' as a password
          dbg_obj_print ('masterdba validated');

          -- so map it to the local 'dba' user
          user_name := 'dba';
```

```
          -- and skip further verification
          return 1;
        }
      else
        {
          -- the password is not the one that we verify against
          dbg_obj_print ('masterdba pwd wrong');

          -- not allow the login at all
          return 0;
        }
    }
  else if (user_name = 'nobody')
    {
      -- the hook can signal :
      -- this is equal to returning -1, but has the additional benefit of an error message printed into t
      signal ('28000', 'we don''t map nobody');
    }
  else if (user_name = 'attacker')
    {
      -- that's a way to print a message to the log and reject the login
      log_message ('trying invalid user');
      return 0;
    }
  else
    {
      -- all local user_name/pwd are subject to normal verification
      dbg_obj_print (user_name);
      return -1;
    }
  -- note that not returning value causes the Virtuoso to print an appropriate message to the log
  -- and then continue with the normal verification
  -- same happens if the value returned is not 0, -1 or 1
};
```

`DB.DBA.USER_FIND (in name varchar, )`

This is a user-defined PL function hook which, if it exists, will be executed before doing the SQL/ODBC login. In this hook the user can find a user account from some other server and register it in the local database. Or, this can be used to perform some pre-login actions. It is similar to the DBEV_LOGIN, but it does not change any account validation rule, it is purely for pre-processing.

12.4. Database Disconnections

`DB.DBA.DBEV_DISCONNECT()`

If defined, this procedure is called after a connection has been found to be disconnected, either as a result of the client process disconnecting, the client process terminating or the server deciding to disconnect, see disconnect_user() . This will also be called if the DBEV_CONNECT hook signals an error, leading to the connection being closed during the connect sequence.

All activity on behalf of the disconnecting client is terminated by the time this function is called. This function runs in its own transaction. Result sets are prohibited, return values discarded and errors are logged in the error log file but not otherwise processed. The transaction is committed regardless of errors. The user and connection_id and any connection variables are defined during this hook.

Example 12.5. Disconnect Interception

```
create procedure DB.DBA.DBEV_DISCONNECT ()
{
  declare ctime datetime;
  dbg_obj_print (user, ' disconnected');
  ctime := connection_get ('login_time');
  update security_log set sl_logged_out = now () where
    sl_user = user and sl_logged_in = ctime;
  if (row_count () = 0)
    signal ('ELOGO', 'Logout by user with no login record.  This occurs when DBEV_CONNECT denied permissi
```

```
}
```

This function updates the row created by the connect hook to set the logout time.

12.5. Database Shutdown

`DB.DBA.DBEV_SHUTDOWN()`

If defined, this function is called when shutting down the server, following disconnection of all clients and making a checkpoint. When a disconnect occurs as a result of server shutdown the DBEV_DISCONNECT hook is not called and this function is expected to perform any logout processing. The rationale is that all disconnect hooks would be called at the same time, creating likely deadlocks, resource contention and there could be no guarantee of time consumed by them or of even whether they would terminate at all.

The shutdown will do a checkpoint before calling this hook. This checkpoint will terminate all transactions with a deadlock state. If transactions contain automatic retries, etc..., we cannot guarantee that all activity would have terminated when this hook starts. However, the hook function can try

```
txn_killall (6);
rollback work;
```

to signal an error on all other transactions to prompt them to terminate. Again, this is not a sure-fire termination since this could be handled by procedures.

When this hook returns the server commits the transaction in which this was running and exits regardless of any lingering activity. There is no hard time limit for this function. Killing the process during this function has no specific ill effect, besides losing uncommitted work by said function.

Result sets are prohibited, return values are discarded, errors are logged but not otherwise processed.

Example 12.6. The Shutdown Hook

```
create procedure DB.DBA.DBEV_SHUTDOWN ()
{
  dbg_obj_print (' server shut down.');
  update security_log set sl_logged_out = now () where sl_logged_out is null;
}
```

This just marks all open connections to be disconnected at the current time.

12.6. SQL Statement Preparation

`DB.DBA.DBEV_PREPARE(`*`inout tree any`*` )`

```
DB.DBA.DBEV_PREPARE (inout tree any)
```

If defined, this function is called after parsing any dynamic SQL statements by any users. The parse tree will be a syntactically correct SQL parse tree. The user and connection variables are defined. The hook should not produce a result set and any return values are discarded. The function runs in the transaction which is current on the connection and the transaction is not automatically committed, so that the hook does not modify application transaction boundaries.

The tree may be modified by replacing it with any other correct parse tree or destructively splicing it. The tree is a regular SQL heterogeneous array. If the tree is modified incorrectly, it is probable that the server will crash.

The parse tree manipulation is best written in C as a Virtuoso Server Extension using the supplied SQL parse tree typedef and constants.

If an error occurs inside this hook the error is simply ignored and the unmodified parse tree is used. To signal an error to a user it is possible to change the parse tree into a call to the signal SQL function.

Example 12.7. SQL Prepare Hook

```
CREATE TABLE REPORT (
  R_AUTHOR VARCHAR,
  R_ID INTEGER IDENTITY,
  R_CLASS INTEGER,
  R_TEXT LONG VARCHAR,
  PRIMARY KEY (R_ID)
);

CREATE TABLE NEED_TO_KNOW (
  NK_CLASS INTEGER,
  NK_USER INTEGER,
  PRIMARY KEY (NK_CLASS, NK_USER)
);

grant select on REPORT to public;

create procedure DB.DBA.DBEV_PREPARE (inout tree any)
{
  declare uid integer;
  uid := (select U_ID from SYS_USERS where U_NAME = user);
  need_to_know (uid, tree);
  dbg_obj_print ('compiled by ', user, ': ', tree);
}
```

This example has a table of variously secret reports, each having a class or compartment and different users having a need to know about a certain collection of compartments. The need_to_know table references U_ID in SYS_USERS and R_CLASS in REPORT. Each select referencing REPORT is modified by the `need_to_know` VSE in order to add a check for the need to know.

For example,

```
select * from REPORT
```

becomes

```
select * from REPORT
  where exists (select 1 from NEED_TO_KNOW
    where NK_CLASS = R_CLASS and NK_USER = <user>)
```

where <user> is the id of the user preparing the query.

As a result, all queries referencing the REPORT table, no matter how they are phrased, will not access rows for which the user does not have a need to know. Note that the REPORT table can be granted to public, unauthorized users will just get an empty result. Further, note that the NEED_TO_KNOW table is not granted to anyone, hence the user does not even need to know the extent of his need to know let alone that of any other user. The expansion of the need to know test inserts the table reference as in a view expansion, where it's privileges are not those of the user but of the view owner, or in this case the procedure owner, which is always dba.

12.7. SQL Parse Tree

The SQL parse tree is composed of DV_ARRAY_OF_POINTER boxes. Other types of boxes may occur as leaves, where they are interpreted as literals. All nodes' first element (index 0) is the type of the node, one of the constants in sqlparext.h.

The nodes' various fields can be accessed through the data members of the sql_tree_t union. Most data members are pointers to the same type. Sometimes they are double pointers, denoting a variable length array of pointers to the struct. The caddr_t type is used to denote a terminal, like a string or other constant. The types are only for declarative value, the entire structure is a self describing, run time typed tree of boxes.

The correspondence of the tree to the SQL syntax is documented by the yacc grammar supplied as appendix.

We will next examine the need to know example:

```
ST *
nk_tree_and (ST* left, ST * right)
{
  if (left && right)
    return ((ST*) list (4, BOP_AND, left, right, NULL));
  if (left)
    return left;
  return right;
}
```

This function adds anAND operation between 2 sub trees. If either is NULL, the non-null one is returned. The list function is used as a universal constructor of the parse tree, where the first argument is the count of arguments to follow.

The above could have been written as follows without list:

```
ST * r = (ST*) dk_alloc_box (4 * sizeof (caddr_t), DV_ARRAY_OF_POINTER);
r->type = BOP_AND;
r->_.bin_exp.left = left;
r->_.bin_exp.right = right;
r->_.bin_exp.more = NULL;
```

We see that the list notation is more concise.

```
void
nk_test_add (ST * outer_texp, char * corr_name, int uid)
{
  /* add a exists (select 1 from need_to_know where nk_class = <corr_name>.r_class) */
  ST * sel, * exists, * texp, **from;
  ST * where = (ST*) list (4, BOP_EQ, list (3, COL_DOTTED, NULL, box_string ("NK_CLASS")),
                           list (3, COL_DOTTED, box_string (corr_name), box_string ("R_CLASS")), NULL);
  where = nk_tree_and (where, listst (4, BOP_EQ, list (3, COL_DOTTED, NULL, box_string ("NK_USER")), box_
  from = (ST**)
    list (1, list (3, TABLE_REF,
                   list (5, TABLE_DOTTED,
                         box_string ("DB.DBA.NEED_TO_KNOW"),
                         NULL, box_num (0), box_num (0)),
                   NULL));
  texp = listst (7, TABLE_EXP, from,
                 where, NULL, NULL, NULL, -1);
  sel = listst (5, SELECT_STMT, NULL, list (1, box_num (1)), NULL, texp);
  exists = (ST*) list (5, EXISTS_PRED, NULL, sel, NULL, NULL);
  outer_texp->_.table_exp.where = nk_tree_and (outer_texp->_.table_exp.where, exists);
}
```

This function takes the table expression referencing the REPORT table and the correlation name used for the table and the user id of the querying user. It adds the existence test for the need to know. The logic is self evident when examined in the context of the yacc grammar. Note that the table in the FROM of the sub query is a TABLE_REF node with a TABLE_DOTTED node, which finally contains the table and correlation names. Note that all strings must be allocated as string boxes. This is because the tree may only reference other boxes. Note that all numbers except for the types of the nodes are boxed with box_num. This will make sure that numbers and pointers are always distinguishable. The node types are distinguishable by definition due to their small absolute value.

```
caddr_t
bif_need_to_know (caddr_t * qst, caddr_t * err_ret, state_slot_t ** args)
{
  unsigned inx;
  caddr_t uid = (caddr_t) bif_long_arg (qst, args, 0, "need_to_know");
  ST * tree = (ST*) bif_array_arg (qst, args, 1, "need_to_know");
  if (ST_P (tree, SELECT_STMT))
    {
      ST * texp = tree->_.select_stmt.table_exp;
      if (!texp)
        return 0; /* select w/o a from */
      for (inx = 0; inx < BOX_ELEMENTS (texp->_.table_exp.from); inx++)
        {
          ST * tref = texp->_.table_exp.from[inx];
          if (ST_P (tref, TABLE_REF))
            tref = tref->_.table_ref.table;
          if (ST_P (tref, TABLE_DOTTED))
            {
```

```
                char * corr_name;
                if (tref->_.table.prefix)
                  corr_name = tref->_.table.prefix;
                else
                  corr_name = tref->_.table.name;
                if (strstr (tref->_.table.name, "REPORT"))
                  nk_test_add (texp, corr_name, (int)uid);
              }
          }
      }
    return 0;
  }
```

This is the top level need to know function. It first checks to see that the statement is a select, that it has a FROM clause (table expression). For each table in the FROM which has REPORT as part of its name the function adds an existence test to check the user's need to know. Note that the correlation name is taken from the table and that the table name is used in the absence of a correlation name to disambiguate the reference. Note that the table_exp.from is of type ST **, meaning a variable length array of boxes. Note that BOX_ELEMENTS returns the length in pointer-size units.

Note that in this example the tree is spliced in place, only adding nodes. There is no need to free data or to modify the top node.

12.7.1. Notes on Special Features of the Parse Tree

The parse tree structure has a 1 to 1 correspondence with the yacc grammar. The members of the union in sql_tree_t are mostly named after the syntactic element and the constant that identifies them. There are however some points worth noting:

Literals - All boxes whose tag is not DV_ARRAY_OF_POINTER are considered literals and may appear in all places where a literal is allowed.

Identifier Case - The case conversion of identifiers is controlled by the CaseMode ini file setting. When introducing a constant name referencing a VSE, one should use sqlp_box_id_upcase to get the right case for all modes.

The identifiers, once in the tree, are compared case sensitively in all modes case modes except 2.

Hook functions should use the appropriate comparison functions for the desired case sensitivity.

When generating references to names the case should be correct as defined by the case mode.

12.7.2. SQL Security and Parse Trees

When compiling table access the access rights granted on the table are checked against the user and group marked on the table reference (TABLE_DOTTED) parse tree node. When a user writes a table reference the user id and group of the user are marked in the parse tree and these are compared against the grants in effect.

When a table reference comes from a view the reference is annotated with the view owner's user id and group. Hence a view may introduce references that would not otherwise be granted to a user.

This is used in the need to know example, marking the group and user of the reference to NEED_TO_KNOW to be 0, meaning dba.

There are no other security related features in the parse tree. Procedure security is resolved exclusively at run time.

12.7.3. Debugging with Parse Trees

It is extremely easy to make errors when manipulating parse trees in code. A useful technique will be to print out the tree before and after the operation. This can be conveniently done in the hook function with dbg_obj_print, as shown in the example.

Once the modified tree is returned to the SQL compilation, the system checks for it for absence of cycles, so that no two branches are the same. There is an assertion failure if this happens. Otherwise the tree is not checked. Hence a malformed tree will in all likelihood crash the server during the compilation.

12.8. WebDAV Logins

```
DB.DBA.DBEV_DAV_LOGIN (inout user_name varchar, in password varchar, in http_auth any,
)
```

This function, if defined, will always be called by Virtuoso just before a HTTP client is authenticated against the WebDAV Server. Three parameters are available for audit purposes or any other pre-processing purpose totally user definable.

1. *user_name:*

   IN : the user name from the login data

   OUT : the user name used by WebDAV server for this request
2. *password* : the encrypted password that corresponding to the `user_name`
3. *http_auth* : HTTP authentication information (see below)

The data structure of the `http_auth` is an array containing name/value pairs as described below.

For HTTP Basic authentication:

method - HTTP method : GET/POST/PUT etc.
authtype - string 'Basic'
username - the same as `user_name` parameter
pass - the password (in case of basic authentication only)

For HTTP Digest authentication:

method - HTTP method : GET/POST/PUT etc.
authtype - string 'Digest'
username - the same as `user_name` parameter
realm - authentication realm
qop - Indicates what "quality of protection" the client has applied to the message
algorithm - hashing algorithm used to create the checksum or digest
uri - the URI from Request-URI of the Request-Line
nonce - data string which may be uniquely generated
nc - The nc-value is the hexadecimal count of the number of requests (including the current request)
cnonce - The cnonce-value is an opaque quoted string value provided by the client
opaque - string value provided by the server and returned by the client unchanged
response - string containing password hash as described in RFC 2617

An example of the `http_auth` value:

```
vector ('method', 'GET', 'authtype', 'basic', 'username', 'MyUser', 'pass', 'My!Secret')
```

This hook can be used to control how Virtuoso proceeds with the WebDAV client login by responding to 3 possible return values:

-1 - continue with normal verification
0 - reject the login
1 - allow the login (the user returned should be a valid Virtuoso local user name)

Example 12.8. Sample WebDAV Login Hook

```
create procedure
DB.DBA.DBEV_DAV_LOGIN (inout user_name varchar, in pwd any, in auth any)
{
  declare result any;

  WHENEVER SQLSTATE '28000' GOTO validation_failure;

  -- All accounts that are not WebDAV admin are going here
  if (lcase(user_name) <> 'dav')
    {
      declare pass any;

      -- use password from request if basic HTTP authentication is used
      if (get_keyword ('authtype', auth) = 'basic')
        pass := get_keyword ('pass', auth);
```

```
      else -- or use the password from database if digest
        pass := pwd_magic_calc (user_name, pwd, 1);

      -- set appropriate LDAP protocol version
      connection_set ('LDAP_VERSION', 2);
      commit work;
      result := LDAP_SEARCH('ldap://mail2.openlinksw.com:389',
                0, 'ou=Accounts, o=OpenLink Software, c=US', sprintf ('(uid=%s)', user_name),
                sprintf('uid=%s, ou=Accounts, o=OpenLink Software, c=US', user_name),
                pass);
      return 1;
    }
  -- normal authentication for WebDAV admin
  return -1;

  -- all accounts that are not authenticated by LDAP are rejected
validation_failure:
  return 0;
};
```

12.9. Associating Auxiliary Data With A Connection

The following functions allow you to set and retrieve connection variables:

- connection_id()
- connection_set()
- connection_get()

Chapter 13. Data Replication, Synchronization and Transformation Services

Abstract

This chapter describes how to replicate data between Virtuoso and non-Virtuoso servers.

The material in this chapter covers the programmatic means of performing these operations. You can also use the graphical interface to do replication and synchronization. This is covered in the Replication & Synchronization section of the Visual Server Administration Interface chapter.

13.1. Introduction

Virtuoso provides several replication methods:

13.1.1. Snapshot replication

Virtuoso provides the following flavors of snapshot replication:

Non-incremental snapshot replicationNon-incremental snapshot replication is useful when the data is changed infrequently or when data is modified in large portions at a time.

Incremental snapshot replication Incremental snapshot replication is useful when the data is changed frequently and implements incremental updates using a snapshot log.

Bi-directional snapshot replication Bi-directional snapshot replication allows data to be modified on both publisher and subscribers. This is useful read-write access to the replicas is needed.

Note:

Snapshot replication can be used in heterogeneous environments to set up replication between non-Virtuoso databases.

13.1.2. Transactional replication

Transactional replication allows subscribers to receive data in near-real time. ACID properties of transactions are maintained in transactional replication as well. Virtuoso implements the following flavors of transactional replication:

Ordinary transactional replicationOrdinary transactional replication flavor implements one-way data replication and is useful when data can be modified on only one server with other servers participating in replication providing read-only data access.

Bi-directional transactional replication Bi-directional transactional replication is useful when there is a requirement to allow data updates on multiple servers.

13.2. Snapshot Replication

13.2.1. Non incremental snapshot replication

The Virtuoso Server periodically evaluates a query and inserts the result into a table, replacing the content. The source and target can be anywhere but typically will be on different databases of which at least one is remote.

This makes a two party distributed transaction. Only one of the transaction branches is writing, hence this goes with a one phase commit.

Note:

This can be used to replicate between two non-Virtuoso databases.

The prerequisite of this mode of replication is that all tables exist. The schema is never replicated.

13.2.2. Incremental snapshot replication

A table can be replicated from a generic, possibly non-Virtuoso source in an incremental fashion if the data source meets certain criteria:

The data should come from a single table.

The source table should have an explicit primary key.

There source query should have the form

```
SELECT fields FROM source_table WHERE scalar_conditions
```

A snapshot log on the source table should exist.

There should be insert, delete and update triggers on the source table to update the snapshot log.

The incremental update is done in the following way:

All the changed records with a snaptime greater or equal from REPL_START_TIME(SN_LAST_TS) are processed. If the record to insert already exists, then it is updated. If the record to update does not exist in the destination table, then it is inserted. This contributes for conflict resolving.

13.2.3. Command reference

CREATE SNAPSHOT LOG FOR source_table

```
CREATE SNAPSHOT LOG FOR source_table
```

```
repl_create_snapshot_log(in source_table varchar)
```

Creates a snapshot log of a native (non VDB) table and the appropriate log update triggers.

Note that when replicating data from a remote Virtuoso a snapshot log table can be created on the remote virtuoso and then the source and log tables can be attached into the local Virtuoso. However the snapshot log table and the update triggers should be created manually on a non-Virtuoso data source.

Examples (based on Virtuoso demo database):

```
create snapshot log for Orders;
```

DROP SNAPSHOT LOG FOR source_table

```
DROP SNAPSHOT LOG FOR source_table

repl_drop_snapshot_log(in source_table varchar)
```

Drops a snapshot log of a table and the log update triggers.

This results in 4 SQL commands :

```
drop trigger xxx_I_log;
drop trigger xxx_U_log;
drop trigger xxx_D_log;
drop table RLOG_xxx;
```

where xxx is the source table name (fully qualified - DB.DBA.Orders becomes DB_DBA_Orders).

Examples (based on Virtuoso demo database):

```
drop snapshot log for Orders;
```

PURGE SNAPSHOT LOG FOR source_table

```
PURGE SNAPSHOT LOG FOR source_table

repl_purge_snapshot_log(in source_table varchar)
```

There can be old snapshot log rows which have been replayed in all the snapshots of a given source table. These rows are no longer needed and can be purged from the snapshot log for faster operation.

This command checks if there are any rows in SYS_REPLICATION referring to that source table and if there are, then it deletes all the rows in the log table with SNAPTIME earlier then REPL_START_TIME(MIN(SN_LAST_TS)).

If there are no rows in SYS_SNAPSHOT it assumes that the log is used for "pull" replication and does not do anything.

Examples (based on Virtuoso demo database):

```
purge snapshot log for Orders;
```

CREATE NONINCREMENTAL SNAPSHOT

```
CREATE NONINCREMENTAL SNAPSHOT dest_table as 'query_def'

repl_create_snapshot(in query_def varchar, in dest_table varchar)
```

Creates a non-incremental snapshot log of the data returned by the query query_def.

The server first checks for the existence of a table with the same name. If it does not exist, then a table with a layout to accommodate the resultset produced by query_def is created.

Then a entry containing query_def and tablename is added to SYS_SNAPSHOT and the destination table is filled in with an insert into tablename query_def.

Examples (based on Virtuoso demo database):

```
create nonincremental snapshot sub_orders as
                'select * from Orders where OrderID < 5'
```

CREATE INCREMENTAL SNAPSHOT

```
CREATE SNAPSHOT dest_table FROM source_table [ 'source_column_list' ]
                [ WHERE 'condition' ]

repl_create_inc_snapshot(
    in source_column_list varchar,
    in source_table varchar,
    in condition varchar,
    in dest_table varchar)
```

Creates a incremental snapshot log using source query like that :

```
SELECT source_table_pk_cols, source_column_list FROM source_table WHERE condition
```

If the dest table already exists, then it's column count is checked to be greater than the primary key parts count of the source table.

If the source_column_list is omitted, then the destination table is created to have the same columns as the source table.

Examples (based on Virtuoso demo database):

```
CREATE SNAPSHOT sub_orders FROM Orders WHERE 'OrderID < 5'
```

UPDATE SNAPSHOT

```
UPDATE SNAPSHOT snapshot_table_name [ AS NONINCREMENTAL ]

repl_refresh_inc_snapshot(
    in snapshot_table_name varchar)

repl_refresh_noninc_snapshot(
    in snapshot_table_name varchar)
```

Without the optional argument updates the snapshot according to it's type. If an incremental snapshot needs to be updated as a nonincremental, then the full form should be used.

Examples (based on Virtuoso demo database):

```
UPDATE SNAPSHOT sub_orders
```

DROP SNAPSHOT

```
DROP SNAPSHOT snapshot_table_name [ WITH DELETE ]

repl_drop_snapshot(
    in snapshot_table_name varchar,
    in delete integer)
```

Removes a snapshot definition from SYS_SNAPSHOT and optionally drops the snapshot destination table.

Examples (based on Virtuoso demo database):

```
DROP SNAPSHOT sub_orders WITH DELETE
```

13.2.4. Bi-Directional Snapshot Replication

Bi-directional snapshot replication allows you to set up snapshot replication between multiple servers where updates can be performed on all servers. Bidirectional snapshot replication uses the publisher-subscriber model where each table or DAV collection has only one publisher and when an update is performed on subscriber it goes to publisher first, and then to all other

subscribers. Conflict resolution may need to take place on the publisher when data coming from a subscriber is processed. Bi-directional snapshot replication uses snapshot logs on publisher and subscribers to track changes in published table and its replicas.

It is assumed that all tables published using bi-directional snapshot replication have primary keys columns that are never modified.

To perform conflict resolution a ROWGUID column is added to every published table. This column should never be modified manually. All INSERT statements should specify exact column-select-lists that do not include the ROWGUID column. Likewise the ROWGUID column should never feature in any UPDATE statements.

Each server participating in bi-directional snapshot replication must have unique name (replication name). For Virtuoso servers replication name is assigned to the server in its virtuoso.ini file in the DBName setting. For other RDBMS servers replication name is the name of instance or the name of database. Replication name of remote server can be obtained using `REPL_SERVER_NAME()` (after this server is defined, see below).

`REPL_CREATE_SNAPSHOT_PUB()` function should be used to create bi-directional snapshot publication.

To create subscription for the publication, replication subscriber server should be defined first using `REPL_SNP_SERVER()` function. The name of the server can be obtained later using `REPL_SERVER_NAME()` function. After this `REPL_CREATE_SNAPSHOT_SUB()` function should be used to create a subscription (and create replicated table on subscriber if table replication takes place).

To load initial data on subscriber `REPL_INIT_SNAPSHOT()` should be used after subscription is created. Loading initial data can take some time so `REPL_INIT_SNAPSHOT()` performs commits after every 100 rows are copied from source table to the table on subscriber to prevent running out of transaction log or deadlocks. It is possible to specify an alternative value for number of rows per transaction (last parameter of `REPL_INIT_SNAPSHOT()` function). In DAV case commits are performed per copied resource.

`REPL_UPDATE_SNAPSHOT()` should be called periodically after initial data is loaded on subscriber to sync published items (tables or DAV collections) on publisher and subscribers. This function reads snapshot logs attached from subscribers and replays them with possible conflict resolution. After all snapshot logs from subscribers are processed an updating procedure reads snapshot log on publisher and replays it on all subscribers.

Note:
Please note that all operations in bi-directional snapshot replication (publication, subscription, doing initial copy, syncing) should be performed on publisher.

Example 13.1. Creating bi-directional snapshot publication

This example demonstrates creating bi-directional snapshot publication of table 'Demo.demo.Shippers'.

```
SQL> REPL_CREATE_SNAPSHOT_PUB ('Demo.demo.Shippers', 2);
```

Example 13.2. Creating bi-directional snapshot subscription

This example demonstrates creating bi-directional snapshot subscription for table 'Demo.demo.Shippers' and loading initial data on subscriber with DSN 'localhost:1121'.

```
SQL> REPL_SNP_SERVER ('localhost:1121', 'dba', 'dba');
SQL> REPL_CREATE_SNAPSHOT_SUB (REPL_SERVER_NAME ('localhost:1121'), 'Demo.demo.Shippers', 2);
SQL> REPL_INIT_SNAPSHOT (REPL_SERVER_NAME ('localhost:1121'), 'Demo.demo.Shippers', 2);
```

Example 13.3. Syncing bi-directional snapshot publication

This example demonstrates syncing bi-directional snapshot publication of table 'Demo.demo.Shippers'.

```
SQL> REPL_UPDATE_SNAPSHOT ('Demo.demo.Shippers', 2);
```

See Also:

The following functions are used for creating, dropping and updating publications and subscriptions to them:

`REPL_CREATE_SNAPSHOT_SUB()`

`REPL_CREATE_SNAPSHOT_PUB()`

`REPL_DROP_SNAPSHOT_SUB()`

`REPL_DROP_SNAPSHOT_PUB()`

`REPL_INIT_SNAPSHOT()`

`REPL_UPDATE_SNAPSHOT()`

`REPL_SNP_SERVER()`

`REPL_SERVER_NAME()`

Conflict Resolution

Since every table can have only one publisher, conflicts can only occur on the publisher when modifications from a subscriber are attempted. When DML operations originating on a subscriber are being replayed on the publisher, three types of conflicts can arise:

1. **uniqueness conflict (insert conflict).** occurs when the row with some primary key <PK> already exists in publisher's table.
2. **update conflict.** occurs when UPDATE modifies a row which has already been modified on publisher (by the publisher or another subscriber)
3. **delete conflict.** occurs when DELETE deletes a row that does not exist on publisher anymore.

Delete conflicts when UPDATE modifies a row that does not exist on publisher can't be detected in snapshot replication case.

Every table has a number of conflict resolvers which are used for conflict resolution which are enlisted in the `DB.DBA.SYS_SNAPSHOT_CR` system table. Each conflict resolver has a type, one of ('I', 'U', or 'D'), and an order. Conflict resolvers are applied in ascending order.

The conflict resolver is a Virtuoso/PL procedure that receives a conflicting row from a subscriber and some other arguments. The conflict resolver can modify the row, which is passed as an 'inout' argument. The conflict resolver should return an integer value, which will be used for conflict resolution.

Conflict resolvers of different types have different signatures:

'I' - Insert conflict resolvers (<ALLCOLS>, inout _origin varchar)

'U' - Update conflict resolvers (<ALLCOLS>, inout _origin varchar)

'D' - Deletion conflict resolvers (<PK>, inout _origin varchar)

where

<ALLCOLS> are the new values of all columns (including the ROWGUID column), <PK> are the values of primary key columns, and _origin is transaction originator.

Conflict resolvers can return the following integer values; The conflict resolver types concerned for each are listed in parentheses:

- **0 - un-decide (I, U, D).** next conflict resolver will be fired.
- **1 - subscriber wins (I, U, D).** DML operation will be applied with <ALLCOLS> All the subscribers except originator will receive modifications (originator already has them).
- **2 - subscriber wins, change origin (I, U).** DML operation will be applied with <ALLCOLS> and origin of transaction will be changed to publisher's server name. All the subscribers (including originator) will receive modifications. This return value is useful when conflict resolver changed some of the columns of the row that were passed in. Although all

parameters of conflict resolver are inout only changing of <ALLCOLS> (non-PK columns) parameters makes sense.

- **3 - publisher wins (U).** DML operation will be applied with <ALLCOLS> taken from publisher's table. All the subscribers will receive modifications.
- **4 - reserved.**
- **5 - ignore (D).** DML operation is ignored.

Conflict resolution stops when some conflict resolver returns a non-zero value meaning that it has made a decision.

Example 13.4. Conflict Resolution

Suppose we have the following table:

```
create table items(
  item_id integer primary key,

  name varchar,
  price decimal
);
```

"Publisher wins" 'I' conflict resolver will look like:

```
create procedure items_cr(
    inout _item_id integer,
    inout _name varchar,
    inout _price decimal,
    inout _origin varchar)
  returns integer
{
  return 3;
}
```

The conflict resolver that will make a decision based on the minimal price column will look like:

```
create procedure items_cr(
    inout _item_id integer,
    inout _name varchar,
    inout _price decimal,
    inout _rowguid varchar,
    inout _origin varchar)
  returns integer
{
  declare p decimal;
  -- get current price value
  select price into p from items where item_id = _item_id;
  if (p < _price)
    return 3;                   -- publisher wins
  else if (p > _price)
    return 1;                   -- subscriber wins
  return 0;                     -- can't decide
}
```

The conflict resolver that will change the price to the minimal value will look like:

```
create procedure items_cr(
    inout _item_id integer,
    inout _name varchar,
    inout _price decimal,
    inout _rowguid varchar,
    inout _origin varchar)
  returns integer
{
  declare p decimal;
  -- get current price value
  select price into p from items where item_id = _item_id;
  if (p < _price)
    {
      _price := p;
      return 2;                 -- publisher wins, change origin
    }
```

```
  return 1;                     -- subscriber wins
}
```

Conflict resolution occurs differently for each kind of DML operation:

- **INSERT.** When INSERT of some row with primary key <PK> is replayed, the row in the publisher's table with such <PK> is looked-up. If the row does not exist then there is no conflict, conflict resolution stops and the INSERT is replayed. If the row exists then we have a "uniqueness conflict". In this case 'I' conflict resolvers are fired-up. If none of the 'I' conflict resolvers were able to make a decision (return non-zero value) the default action is 'publisher wins'.
- **UPDATE.** When there is an UPDATE of some row with primary key <PK> is replayed, the row (and its ROWGUID) in publisher's table with such <PK> is looked-up. If the row does not exist then we have a "delete conflict", 'D' conflict resolvers are fired up. If none of the 'D' conflict resolvers were able to make a decision the default action will be to 'ignore'. If the row exists in the publisher's table and its ROWGUID is the same as that from the subscriber then there is no conflict. Conflict resolution stops and the UPDATE is replayed. If the row exists and its ROWGUID differs from the one that came from subscriber then we have an "update conflict". In this case the 'U' conflict resolvers are fired-up. If none of the 'U' conflict resolvers were able to make a decision (return non-zero value) the default action will be 'publisher wins'.
- **DELETE.** When DELETE operation of some row with primary key <PK> is replayed, the row in the publisher's table with such <PK> is looked-up. If the row does not exist or if the row exists but its ROWGUID differs from the one that came from subscriber then we have "delete conflict". The 'D' conflict resolvers are fired-up. If none of the 'D' conflict resolvers were able to make a decision then the default action will be taken to 'ignore'. Otherwise it is assumed that there is no conflict and DELETE statement is replayed.

Conflict Resolution in WebDAV

Conflict resolvers in DAV are found based on the collection. The closest collection that specifies resolvers will be the one providing the resolver set. Resolvers from any enclosing collection will not be invoked.

There is special subcollection in each replicated collection named '_SYS_REPL_BACKUP'. This subcollection is never replicated and is used to store backup copies of resources that lose conflict resolution.

If a resource is locked during an update it will be stored in temporary location. Conflict resolution will take place after the lock is released. If the update was performed on the publisher then no conflict resolution will be performed on the subscribers and the resource from temporary location will simply replace an existing resource after the lock is released.

Default conflict resolution is 'publisher wins'.

Delete conflicts are not handled when replicating DAV because updates are the same as inserts -- if the resource that was updated does not exist it will be created. INSERT or uniqueness conflict for the same reasons.

Subcollections are not considered in conflict resolution. Subcollections are always created as needed during an update. This means that if a resource is updated locally but collection that holds this resource does not exist on remote peer it will be created.

Automatically Generated Conflict Resolvers

Simple table conflict resolvers can be generated automatically by calling the `REPL_ADD_SNAPSHOT_CR()` function. DAV conflict resolvers be generated by calling `REPL_ADD_DAV_CR()` function.

See Also:

`REPL_ADD_DAV_CR()`

`REPL_ADD_SNAPSHOT_CR()`

The generated procedures can be modified afterwards. In particular it is possible to change the notification e-mail address by setting the _notify_email parameter, and the notification text by setting the _notify_text parameter.

The default behaviour for generated procedures is 'pub_wins', making a backup and notifying the owner by e-mail.

13.2.5. Registry variables

1. *snp_repl_tolerance_offset* (default 15) In incremental and bi-directional snapshot time stamp of last replayed snapshot log entry (LAST_TS) is kept in system tables. LAST_TS is used to determine the starting point of the next update. This value can't be used as is for the following reasons:

   Local time may be adjusted occasionally on the servers which participate in replication.

   Virtuoso snapshot replication triggers use `now()` function to get snapshot log entry time stamp. If transaction is started before updating snapshot log and ended after updating snapshot log is finished, data modifications made in this transaction will be missed.

   In order to resolve the issues mentioned above Virtuoso reads snapshot logs starting from LAST_TS minus some tolerance offset. The value of tolerance offset (in minutes) is kept in "snp_repl_tolerance_offset" registry variable. For the reasons mentioned above tolerance offset should be longer than length of any transaction which modifies published tables.
2. *snp_repl_purge_offset* (default 30) In order to prevent snapshot log entries to be replayed more than once Virtuoso keeps log of replayed snapshot log entries (rplog). Rplog entries need to be purged periodically. Virtuoso automatically schedules rplog purger when snapshot log for some table is created. All the records in rplog with time stamp less than MIN(LAST_TS) minus some offset are purged. The value of purge offset (in minutes) is kept in "snp_repl_purge_offset" registry variable. Normally, purge offset should be greater than tolerance offset.

13.2.6. Heterogeneous snapshot replication

Virtuoso allows incremental and bi-directional snapshot replication flavors to be used with non-Virtuoso databases.

The following databases are supported in incremental snapshot replication:

SQL Server 2000 and later
Oracle 8i and later
IBM DB2 8.1 and later
Informix IDS 9.40 and later

The following databases are supported in bi-directional snapshot replication:

SQL Server 2000 and later
Oracle 8i and later
IBM DB2 8.1 and later

DBMS-specific notes:

1. *SQL Server 2000 and later* Bi-directional snapshot triggers for SQL Server require "nested triggers" server option to be "On" (which is "On" by default).
2. *Oracle 8i*

   a. Oracle 8i does not have support for "localtimestamp" data type out of box. Required functionality can be enabled by editing $ORACLE_HOME/rdbms/admin/standard.sql script. The following statement

   ```
   pragma new_names('8.1.6',
                    dbtimezone, sessiontimezone, localtimestamp,
                    localtime, to_local_tz, to_db_tz,
                    cube, rollup, grouping, "TIMESTAMP WITH LOCAL TIME ZONE");
   ```

   should be changed to

   ```
   --pragma new_names('8.1.6',
   --                  dbtimezone, sessiontimezone, localtimestamp,
   --                  localtime, to_local_tz, to_db_tz,
   --                  cube, rollup, grouping, "TIMESTAMP WITH LOCAL TIME ZONE");
   ```

   After $ORACLE_HOME/rdbms/admin/standard.sql is edited it should be executed with SYS user privileges:

```
$sqlplus SYS/CHANGE_ON_INSTALL @$ORACLE_HOME/rdbms/admin/standard.sql
```

where "CHANGE_ON_INSTALL" is password for user SYS.Oracle 9i and later has this type in the default installation and this step is not necessary.

b. ODBC driver from Oracle 8.1.6 (and earlier versions) is known to be buggy and may cause Virtuoso server to crash when snapshot replication with Oracle database is set up. ODBC driver from Oracle 8.1.7 and later is recommended.

c. Oracle ODBC driver uses Oracle NLS settings by default for determining decimal separator. If decimal separator in the locale is not '.' (period) there will be interoperability errors with Virtuoso. Oracle ODBC driver can be forced to use US Settings for numerics: open Oracle ODBC driver configuration dialog, select "Application" tab and set "Numeric Settings" to "Use US Settings".

3. *DB2 8.1 ESE*

a. FixPak 3 is recommended because ODBC driver from earlier DB2 8 versions may cause Virtuoso server to crash when BLOB columns exist in replicated tables.

b. DB2 CLI/ODBC driver uses default locale's decimal separator. If decimal separator in the locale is not '.' (period) Virtuoso fails to parse numeric values. The default behavior of the DB2 CLI/ODBC driver can be modified by specifying PATCH2 CLI/ODBC configuration keyword in the db2cli.ini initialization file. Add the following line to your DSN config in db2cli.ini:

```
PATCH2=15
```

This will force DB2 CLI/ODBC driver to always use period ('.') as decimal separator.

4. *Informix IDS 9.40* Informix IDS can't report current time in UTC. This may cause snapshot replication to fail when timezone change occurs. In order to set up snapshot replication with Informix server properly any of the following steps must be taken:

Informix server must be run in UTC time zone.

"snp_repl_time_offset" registry variable should be set to value greater than 60.

The second method may cause additional system load on updates and will cause transactions to be replayed out of order when timezone change occurs.

13.2.7. Data type mappings

Heterogeneous replication requires data type mapping to be performed when Virtuoso table is created on replica.

Table 13.1. Data type mappings

Virtuoso	*SQL Server*	*Oracle*	*DB2*	*Informix*
varchar	varchar(8000)	VARCHAR2(4000)	VARCHAR(32000)	VARCHAR(254)
varchar(n)	varchar(n)	VARCHAR2(n)	VARCHAR(n)	VARCHAR(n)
integer	int	INTEGER	INTEGER	INTEGER
smallint	smallint	SMALLINT	SMALLINT	SMALLINT
real	real	FLOAT	REAL	SMALLFLOAT
double precision	float	DOUBLE PRECISION	DOUBLE	FLOAT
numeric	numeric(38, 15)	NUMERIC(38, 15)	NUMERIC(32, 15)	DECIMAL(31, 15)
numeric(p, s)	numeric(p, s)	NUMERIC(p, s)	NUMERIC(p, s)	DECIMAL(p, s)
date	datetime	DATE	DATE	DATE
time	datetime	DATE	TIME	DATETIME HOUR TO SECOND
datetime, timestamp	datetime	DATE	TIMESTAMP	DATETIME YEAR TO FRACTION(5)
varbinary	varbinary(8000)	RAW(2000)	VARCHAR(32000) FOR BIT DATA	BYTE
varbinary(n)	varbinary(n)	RAW(n)	VARCHAR(n) FOR BIT DATA	BYTE
long varbinary	image	BLOB	BLOB	BYTE

long varchar	text	CLOB	CLOB	TEXT
nvarchar	nvarchar(4000)	NVARCHAR2(4000)	VARCHAR(32000) FOR MIXED DATA	NVARCHAR(254)
nvarchar(n)	nvarchar(n)	NVARCHAR2(n)	VARCHAR(n) FOR MIXED DATA	NVARCHAR(n)
long nvarchar	ntext	NCLOB	DBCLOB	TEXT

13.2.8. Objects created by incremental snapshot replication

Table "DB.DBA.RPLOG_<name>" (replay log) is created in Virtuoso database for replayed snapshot log entries.

Table "RLOG_<name>" (snapshot log) is created in the source DSN. This table is attached as "<qual>"."<dsn>"."RLOG_<name>" if source table is an attached table. Other objects created by incremental snapshot replication in the source DSN are:

Virtuoso Triggers "<qual>_<owner>_<name>_I_log", "<qual>_<owner>_<name>_U_log" and "<qual>_<owner>_<name>_D_log" on replicated table.

SQL Server Triggers "<name>_I_log", "<name>_U_log" and "<name>_D_log" on replicated table.

Oracle Triggers "<name>_I_log", "<name>_U_log" and "<name>_D_log" on replicated table. Function OPL_GETUTCDATE().

DB2 Triggers "<name>_I", "<name>_U" and "<name>_D" on replicated table. Sequence opl_seq_rowguid.

Informix Triggers "<name>_I_log", "<name>_U_log" and "<name>_D_log" on replicated table. Sequence opl_seq_rowguid. Stored procedures "<name>_R_proc" and "<name>_U_proc".

13.2.9. Objects created by bi-directional snapshot replication

Table "DB"."DBA"."RPLOG_<name>" (replay log) is created in Virtuoso database for replayed snapshot log entries.

Table "RLOG_<name>" (snapshot log) is created on the publisher and on subscribers. "RLOG_<name>" table and replicated table itself are attached from subscribers as "<qual>"."<dsn>"."RLOG_<name>" and "<qual>"."<dsn>"."<name>" respectively. Other objects created by bi-directional snapshot replication on the publisher and on subscribers are:

Virtuoso Triggers "<qual>_<owner>_<name>_I_log", "<qual>_<owner>_<name>_U_log" and "<qual>_<owner>_<name>_D_log" on replicated table.

SQL Server Triggers "<name>_I_log", "<name>_U_log" and "<name>_D_log" on replicated table.

Oracle Triggers "<name>_I_log", "<name>_IR_log", "<name>_U_log", "<name>_UR_log", "<name>_UD_log", "<name>_UDR_log", and "<name>_D_log" on replicated table. Function OPL_GETUTCDATE(). Global temporary table "DLOG_<name>".

DB2 Triggers "<name>_I", "<name>_U", "<name>_UD" and "<name>_D" on replicated table. Sequence opl_seq_rowguid.

13.2.10. Replication system tables

SYS_SNAPSHOT

```
CREATE TABLE SYS_SNAPSHOT (
    SN_NAME     VARCHAR(255),
    SN_QUERY    VARCHAR(1024),
    SN_LAST_TS  VARBINARY(30),
    SN_IS_INCREMENTAL   INTEGER,
    SN_SOURCE_TABLE     VARCHAR(255),
    PRIMARY KEY (SN_NAME))
```

This table describes all defined snapshot replication relations. SN_NAME is the snapshot's name (destination table name). SN_QUERY is the query to get the data from. SN_LAST_TS is the time of the last update reflected for incremental snapshots (from the snapshot log's SNAPTIME). SN_IS_INCREMENTAL is null for nonincremental snapshots and non-null for the incremental ones. SN_SOURCE_TABLE is the source table for the incremental snapshots, NULL otherwise.

This implies that an incremental snapshot can be updated in two ways - incrementally and nonincrementally as the nonincremental definition is a subset of the incremental definition.

SYS_SNAPSHOT_LOG

```
CREATE TABLE SYS_SNAPSHOT_LOG (
    SNL_SOURCE  varchar (320) NOT NULL,
    SNL_RLOG    varchar (320) NOT NULL,
    SNL_RPLOG   varchar (320) NOT NULL,
    primary key (SNL_SOURCE)
)
```

This table describes all defined snapshot replication relation logs. SNL_SOURCE is the base table for which a snapshot log is defined. SNL_RLOG is the name of the RLOG snapshot log auxiliary table. SNL_RPLOG is the name of the RPLOG snapshot log auxiliary table.

13.2.11. Table snapshot logs

In order to be able to create incremental snapshots on a table there should be a snapshot log table defined in the following fashion:

```
create table RLOG_sourcetable (
RLOG_sourcetable_primarykey_parts,
...
SNAPTIME DATETIME,
DMLTYPE  CHAR(1),
RLOG_ROWGUID VARCHAR(255),
PRIMARY KEY (RLOG_*));
```

The name of the snapshot log table is constructed from the name of the source table by prefixing it with RLOG_ .

Snapshot log table contains all the primary key parts of the source table prefixed with RLOG_ .

Snapshot log table contains at most one row per updated source table row with the update time and update type in DMLTYPE (I - for insert, U - for update, D - for delete).

Snapshot log table contains RLOG_ROWGUID column which uniquely identifies this snapshow log row.

An updating procedure for incremental snapshot or bi-directional snapshot (table case) reads records from snapshot log table, ordered by SNAPTIME. Commits are performed each time an updating procedure notices that SNAPTIME has changed. In Virtuoso case (when native table is snapshot-replicated) all snapshot log records with equal SNAPTIME belong to the same transaction and all such records constitute almost a transaction (it is not exactly a transaction because only the last DML operation for each primary key in the source table is recorded in snapshot log table). So in Virtuoso case an updating procedure commits almost per transaction on source table. Snapshot log records are not transaction-bound in heterogeneous case however. Nevertheless, the technique used to determine when to commit described above prevents running out of transaction log or deadlocks even in heterogeneous case.

Replay log is created on the server which performs sync (server on which snapshot log was create in ordinary case, publisher in bi-directional snapshot case) to handle correct snapshot log replaying. Replay log is purged periodically using `REPL_PURGE_RPLOGS()` function in ordinary case and `REPL_PURGE_URPLOGS()` in bi-directional case. The calls to this functions are automatically scheduled when snapshot log or bi-directional snapshot publication is created and there is no need to call them manually.

There should be triggers on the source table to update the log like that:

```
create trigger xxx_I_log after insert on xxx
{
  insert replacing RLOG_xxx values (xxx_pk, now(), 'I', uuid());
};

create trigger xxx_D_log after delete on xxx
{
  insert replacing RLOG_xxx values (xxx_pk, now(), 'D', uuid());
};

create trigger xxx_U_log after update on xxx
{
  if (OLD.xxx_pk differs from NEW.xxx_pk)
    {
      insert replacing RLOG_xxx values (OLD.xxx_pk, now(), 'D', uuid());
```

```
      insert replacing RLOG_xxx values (NEW.xxx_pk, now(), 'I', uuid());
    }
  else
    insert replacing RLOG_xxx values (xxx_pk, now(), 'U', uuid());
};
```

where xxx is the source table name, RLOG_xxx is the snapshot log table name, xxx_pk is a comma separated list of the source table's primary key parts. The names of the triggers are constructed from the fully qualified name (DB.DBA.Orders makes DB_DBA_Orders_I_log).

Table snapshot logs and triggers are created automatically for Virtuoso and databases listed in Heterogeneous Snapshot Replication section.

13.3. Transactional Replication

The unit of replication is a publication. A publication is an ordered sequence of transaction entries. One database transaction can add data to zero or more publications. The data contributed to a publication by a transaction is appended to the publication at the time of commit. Because commits are serialized database wide, items in a publication have a well defined order.

Each transaction entry in a publication has a unique sequence number within the publication. Each subscriber of a publication has a level of synchronization, which is the serial number of the last transaction from the publication which this subscriber has processed.

Each publication has exactly one publisher and zero or more subscribers. Any multi-master merge replication schemes will be based on this notion, with data to be merged back into the original source regarded as a separate publication and the merge regarded as a process between publications.

In order to publish data for replication by others a server must have a unique name within the group of servers participating in the replication. This server name is assigned to the server in its virtuoso.ini file in the DBName setting.

To publish data the publishing server initializes a publication with the repl_publish function, where it names the publication and assigns a log file name for it. The server can then start adding transactions to the publication, which can happen either under application control or implicitly.

Tip

See the repl_text function.

To subscribe to publications a server must also have a distinct DBName. It identifies the publishing server by associating a host name and port number to its logical name with the repl_server function. It can then call `repl_subscribe()` . Replication feeds from publisher are replayed by 'dba' user by default. The default can be changed (see repl_subscribe() function). for each of the publications it subscribes to. A publication is uniquely identified on the subscriber with the publishing server name and the publication name. Note that several servers in a network may publish like named publications and these will be logically distinct, having each their own distinct publisher.

A subscriber may or may not be connected to the publisher at any point in time. If a subscriber is connected to the publisher it may either be 'in sync' or syncing'. In the syncing state it is receiving transaction entries with numbers consecutive from its sync level up until the last committed serial number committed on the server.

At the start of the sync communication the subscriber indicates the level of the last successfully processed transaction in the publication. The sync exchange terminates when the subscriber reaches the last committed item on the publication. At this point the subscriber is said to be 'in sync'. The connection to the publisher is then maintained by default and is used to send sync information as it becomes available. This means that once an entry is appended to the publication by a committing a transaction it is sent to the 'in sync' subscribers without separate request.

The publisher can terminate the replication feed by unilateral decision. It will do it if the sending of the message times out for too long or if the queue of 'to be sent' replication records exceeds a settable threshold. This essentially happens with communication failures or if the subscriber continuously processes the feed at a speed lower than the feed production speed of the publisher. A disconnected subscriber can reconnect at will, in which case it enters the 'syncing' state and will receive transactions from the point where the feed was cut.

A subscriber can disconnect from the publisher at any time without ill effect.

A table

```
SYS_REPL_ACCOUNTS (
    SERVER varchar,
    ACCOUNT varchar,

    NTH integer,
    LEVEL integer,
    IS_MANDATORY integer,
    IS_UPDATEABLE integer,
    SYNC_USER varchar,

    P_MONTH integer,
    P_DAY integer,
    P_WDAY integer,
    P_TIME time,

    primary key (SERVER, ACCOUNT))
```

is used to store information about published accounts and accounts this server is subscribed to.

A table

```
SYS_REPL_SUBSCRIBERS (
    RS_SERVER varchar,
    RS_ACCOUNT varchar,
    RS_SUBSCRIBER varchar not null,

    RS_LEVEL integer NOT NULL,
    RS_VALID integer NOT NULL,

    primary key (RS_SERVER, RS_ACCOUNT, RS_SUBSCRIBER))
```

is used to store subscribers' status (pushback accounts for updateable subscriptions are there too). Subscribers for an account are added to this table automatically on each request to sync an account from subscriber or manually from Admin UI.

SYS_REPL_SUBSCRIBERS.RS_VALID column is be used to designate subscribers whose replication account level is valid (lags not more than REPL_MAX_DELTA behind the publisher's level).

RS_VALID state of subscriber is checked and updated on every sync request from subscriber. If subscriber is found to be invalid all further sync requests from it are ignored. Such subscriber need to be reinitialized manually and marked as valid using Admin UI.

13.3.1. Publishable Items

Tables, stored procedures and DAV collections may be added to a transactional publication. When a table is added, triggers are created for capturing changes to the table. When a procedure is added, all calls to this procedure will be recorded in the publication's log and the same procedure will be called on the subscriber. When a procedure is published in this manner, actions performed inside the procedure are not themselves recorded even if they touch on items that are part of a publication. It is assumed that the procedure on the subscriber will produce the equivalent effect. When a DAV collection is published, operations on direct and indirect members of the collection are logged into the publication's log. When this is replayed on the subscriber, the operations are repeated on like-named DAV resources, creating collections and resources as needed.

When a table or procedure is added to a publication, the creating statement is added also, so that the subscribers come to create the table or procedure before receiving any replication operations on said table or procedure. Also when the table or procedure is altered, the altering statements are added to the publication so s to be reflected on all subscribers. This can be overridden for procedures, since in some cases it is desirable to have a different definition on the subscriber. Table constraints are also replicated, except for foreign key constraints with tables outside the publication, since these would not be meaningful on the subscriber, as there is no knowledge on what tables may exist there outside of the ones in the publication. Changes to schema on subscribers are never replicated to the publisher, even if we had bidirectional replication. Identity and timestamp columns are replicated so that the values on the subscriber are assigned by the publisher.

When a procedure is published, it is possible to specify whether the definitions, calls or both are replicated. Usually specifying both calls and definition is reasonable, sometimes only calls are to be replicated if the procedure intentionally has a different definition in the subscriber. This is useful for example when the subscriber gathers statistics or maintains a data warehouse where

the storage schema is not identical with that of the publisher. Only procedures with input parameters can be replicated. The rationale is that replication is a one way stream and no return values, result sets or output parameters can be captured by either publisher or subscriber. Procedure calls cannot be replicated bidirectionally, these go from publisher to subscriber exclusively.

Even though replication can carry schema changes, it is in no way a means of keeping software installations in sync. Many schema elements such as triggers are not covered and a software upgrade is more complex than can be represented by replication alone.

13.3.2. Errors in Replication

A statement received for replication from a publisher may encounter an error. All Such errors are logged into the subscriber's error log file. If the error is retryable, like a deadlock, retries are made until the operation succeeds. If Other errors are simply skipped and replication proceeds. A replication subscription can be permanently broken and out of sync if for example columns are dropped from the subscriber copy of the tables or if the subscriber runs out of disk and can replay some transactions but not all. Note that in such situations gaps may be formed into the received transaction sequence. In such cases, it is best to drop the subscription, drop the tables and remake the subscription.

13.3.3. Publisher Transactional Replication Functions

The most generic form of the replication element is a stored procedure call. This associates a procedure name and set of parameters to a publication inside a transaction. When the transaction commits, all the publication entries are appended to the publication, forming a replication entry with its unique number.

These functions are available to the publishing Virtuoso server:

- `repl_publish()`
- `repl_unpublish()`
- `repl_pub_add()`
- `repl_pub_remove()`
- `repl_pub_init_image()`
- `repl_new_log()`
- `repl_text()`
- `repl_grant()`
- `repl_revoke()`

13.3.4. Subscriber Functions

These are the functions that are available to the subscribing Virtuoso server:

- `repl_server()`
- `repl_subscribe()`
- `repl_unsubscribe()`
- `repl_init_copy()`
- `repl_sync()`
- `repl_sync_all()`
- `repl_sched_init()`
- `sub_schedule()`
- `repl_disconnect()`
- `repl_this_server()`
- `repl_purge()`

13.3.5. Common Status Functions

- `repl_stat()`
- `repl_status()`

The status () function shows a replication status summary. The same data can be obtained with the repl_stat and repl_status procedures.

13.3.6. Bi-Directional Transactional Replication

Virtuoso supports bi-directional transactional replication via a mechanism of updateable subscriptions. The following rules and conditions must be observed:

Every table has only one publisher.

Only direct subscribers are considered.

Only replication of tables is allowed.

It is assumed that all the tables within a publication have primary keys and that the primary key columns are never modified.

Every transaction has an origin, i.e. the originating server on which the transaction was performed.

Modifications to a subscriber come from publisher only using the ordinary transactional replication technique: the subscriber initiates an update and pulls (requests) replication logs from publisher. The publisher sends the replication log from the replication log files and then places the subscriber into "synced" (or "online") mode. In this mode the replication logs are sent to subscriber immediately after each COMMIT.

Data flow from a subscriber to the publisher is very similar: the subscriber initiates the update and pushes replication logs to the publisher. After all replication log data has been sent to the publisher it is put into "synced" mode and will receive modifications immediately after each COMMIT on subscriber.

Creating Publications for Updateable Subscriptions

In order to create a publication that allows transaction-based replication and updateable subscriptions use the `repl_publish()` with a non-zero third argument. Replication feeds from subscribers can be replayed by user different from 'dba' user.

```
repl_publish('foo', 'foo.log', 1, 'demo');
```

This will create updateable publication 'foo'. Replication feeds from subscribers will be replayed as user 'demo'.

Adding Tables to a Publication

When a table is added to an updateable publication a new 'ROWGUID varchar' column is automatically added to the table. This column will be used for conflict resolution (described later). If the table already has column with such a name, an existing column will be used (with checking for appropriate data type and width). ROWGUID columns hold globally unique identifiers of a row and are modified after each UPDATE of a row. ROWGUID column values are OSF DCE 1.1 compliant Universally Unique Identifiers (UUID).

ROWGUID columns are used for conflict resolution for INSERT/UPDATE/DELETE DML operations. Basically, if a ROWGUID column that came from a subscriber does not differ from the ROWGUID column of the publisher's table then it is assumed that there is no conflict, otherwise conflict resolution must take place.

Conflict Resolution

Since every table may have only one publisher, conflicts resolution will always take place on the publisher.

Assume some DML operation that occurred on a subscriber is being replayed on publisher. There may be three types of conflicts:

1. **uniqueness conflict (insert conflict) .** occurs when the row with some primary key <PK> already exists in publisher's table.
2. **update conflict .** occurs when an UPDATE modifies a row which has already been modified on publisher (by the publisher or another subscriber).
3. **delete conflict .** occurs when an UPDATE modifies a row or a DELETE removes a row that does not exist on publisher anymore.

Every table has a number of conflict resolvers that are used for conflict resolution. These are stored in DB.DBA.SYS_REPL_CR system table. Each conflict resolver has a type ('I', 'U', or 'D') and an order. Conflict resolvers are applied in ascending order.

The conflict resolver is a Virtuoso/PL procedure that receives a conflicting row from a subscriber and some other arguments. The conflict resolver can modify the row, which is passed as an 'inout' argument. The conflict resolver should return an integer value,

which will be used for conflict resolution.

Conflict resolvers of different types have different signatures:

'I' - Insert conflict resolvers (<ALLCOLS>, inout _origin varchar)
'U' - Update conflict resolvers (<ALLCOLS>, , <ALLOLDCOLS>, inout _origin varchar)
'D' - Deletion conflict resolvers (<ALLOLDCOLS>, inout _origin varchar)
where

<ALLCOLS> are new values of all columns including the ROWGUID column, <ALLOLDCOLS> are old values of all columns, and _origin is transaction originator.

Conflict resolvers can return the following integer values; The conflict resolver types concerned for each are listed in parentheses:

- **0 - can't decide (I, U, D) .** next conflict resolver will be fired.
- **1 - subscriber wins (I, U, D) .** DML operation will be applied with <ALLCOLS> All the subscribers except originator will receive modifications (originator already has them).
- **2 - subscriber wins, change origin (I, U) .** DML operation will be applied with <ALLCOLS> and origin of transaction will be changed to publisher's server name. All the subscribers (including originator) will receive modifications. This return value is useful when conflict resolver changed some of the columns of the row that were passed in. Although all parameters of conflict resolver are inout only changing of <ALLCOLS> (non-PK columns) parameters makes sense.
- **3 - publisher wins (U) .** DML operation will be applied with <ALLCOLS> taken from publisher's table. All the subscribers will receive modifications.
- **4 - reserved .**
- **5 - ignore (D) .** DML operation is ignored.

Conflict resolution stops when conflict resolvers return a non-zero value meaning that it has made a decision.

Example 13.5. Conflict Resolution

Suppose we have the following table:

```
create table items(
  item_id integer primary key,

  name varchar,
  price decimal
);
```

"Publisher wins" 'I' conflict resolver will look like:

```
create procedure items_cr(
    inout _item_id integer,
    inout _name varchar,
    inout _price decimal,
    inout _origin varchar)
  returns integer
{
  return 3;
}
```

The conflict resolver that will make a decision based on the minimal price column will look like:

```
create procedure items_cr(
    inout _item_id integer,
    inout _name varchar,
    inout _price decimal,
    inout _rowguid varchar,
    inout _old_item_id integer,
    inout _old_name varchar,
    inout _old_price decimal,
    inout _old_rowguid varchar,
    inout _origin varchar)
  returns integer
{
```

```
  declare p decimal;
  -- get current price value
  select price into p from items where item_id = _item_id;
  if (p < _price)
    return 3;                   -- publisher wins
  else if (p > _price)
    return 1;                   -- subscriber wins
  return 0;                     -- can't decide
}
```

Conflict resolver that will change the price to the minimal value will look like:

```
create procedure items_cr(
    inout _item_id integer,
    inout _name varchar,
    inout _price decimal,
    inout _rowguid varchar,
    inout _old_item_id integer,
    inout _old_name varchar,
    inout _old_price decimal,
    inout _old_rowguid varchar,
    inout _origin varchar)
  returns integer
{
  declare p decimal;
  -- get current price value
  select price into p from items where item_id = _item_id;
  if (p < _price)
    {
      _price := p;
      return 2;                 -- publisher wins, change origin
    }
  return 1;                     -- subscriber wins
}
```

Conflict resolution occurs differently for each kind of DML operation:

- **INSERT .** When INSERT of some row with primary key <PK> is replayed, the row in the publisher's table with such <PK> is looked-up. If the row does not exist then there is no conflict, conflict resolution stops and the INSERT is replayed. If the row exists then we have a "uniqueness conflict". In this case 'I' conflict resolvers are fired-up. If none of the 'I' conflict resolvers were able to make a decision (return non-zero value) the default action is 'publisher wins'.
- **UPDATE .** When there is an UPDATE of some row with primary key <PK> is replayed, the row (and its ROWGUID) in publisher's table with such <PK> is looked-up. If the row does not exist then we have a "delete conflict", 'D' conflict resolvers are fired up. If none of the 'D' conflict resolvers were able to make a decision the default action will be to 'ignore'. If the row exists in the publisher's table and its ROWGUID is the same as that from the subscriber then there is no conflict. Conflict resolution stops and the UPDATE is replayed. If the row exists and its ROWGUID differs from the one that came from subscriber then we have an "update conflict". In this case the 'U' conflict resolvers are fired-up. If none of the 'U' conflict resolvers were able to make a decision (return non-zero value) the default action will be 'publisher wins'.
- **DELETE .** When DELETE of some row with primary key <PK> is replayed, the row in the publisher's table with such <PK> is looked-up. If the row does not exist or if the row exists but its ROWGUID differs from the one that came from subscriber then we have "delete conflict". The 'D' conflict resolvers are fired-up. If none of the 'D' conflict resolvers were able to make a decision then the default action will be taken to 'ignore'. Otherwise it is assumed that there is no conflict and DELETE statement is replayed.

Automatically Generated Conflict Resolvers

Simple conflict resolvers can be generated automatically. This can be done by calling REPL_ADD_CR function.

See Also:

`REPL_ADD_CR()`

Replication Log Data

Replication log data is different for each kind of DML operation:

- **INSERT.** (stmt, <ALLCOLS>)
- **UPDATE.** (stmt, <ALLCOLS>, <OLDPK>, <ALLOLDCOLS>, ncols)
- **DELETE.** (stmt, <OLDPK>, <ALLOLDCOLS>, ncols)

where

stmt is DML statement (varchar), <ALLCOLS> is new values of all columns, <OLDPK> is primary key, specifying a row for which (UPDATE or DELETE) DML statement is executed, <ALLOLDCOLS> is old values of all columns, ncols is number of columns in table (integer).

The format of the log replication data is the same as in simple transactional replication with addition of <ALLOLDCOLS> and ncols for logging UPDATE and DELETE statements.

13.3.7. Purging replication logs

Every replication account has an associated sequence which holds replication account level (basically, transaction number). Each subscriber also maintains a sequence where it stores its replication account level.

When subscriber issues a sync request for an account it submits its replication account level so publisher can find a point in time from where it should start to submit replication logs to publisher.

Replication account level can roll over REPL_WRAPAROUND (0x7fffffff) to 1. Old replication logs need to be purged to allow this to work correctly. Procedure repl_purge() purges replication logs for a specified account.

Columns needed to store purger configuration in SYS_REPL_ACCOUNTS table are:

P_MONTH integer (month, nullable)

P_DAY integer (day of month, nullable)

P_TIME time (time, nullable)

If P_TIME is NULL log purger will not be scheduled to run at all. If P_TIME is not NULL there can be the following combinations of P_MONTH, P_DAY and P_WDAY ('*' means NULL value, 'x' means any value):

Table 13.2. Purger config settings

P_MONTH	*P_DAY*	*P_WDAY*	*meaning*
month	day	x	purger is run yearly on specified month and day
month	*	x	purger is run yearly on 1st of month
*	day	x	purger is run monthly on specified day of month
*	*	wday	purger is run weekly on specified day of week
*	*	*	purger is run daily

An entry to call purger is inserted (or updated) into SYS_SCHEDULED_EVENT after each modification of purger settings for an account (from Admin UI) or after each successful run of repl_purge() for this account.

13.3.8. Objects created by transactional replication

Virtuoso creates triggers "<name>_I", "<name>_U" and "<name>_D" for every published table. On subscriber "DB"."DBA"."SYS_REPL_ACCOUNTS" table and "DB"."DBA"."TP_ITEM" view are attached from publisher as "DB"."<dsn>"."SYS_REPL_ACCCOUNTS" and "DB"."<dsn>"."TP_ITEM" respectively.

If publication is updateable Virtuoso additionally creates the following objects:

"<qual>"."<owner>"."replcr_<name>_I", "<qual>"."<owner>"."replcr_<name>_U" and "<qual>"."<owner>"."replcr_<name>_D" procedures for every published table. These procedures are used for conflict resolution.

Triggers "<name>_I", "<name>_U" and "<name>_D" for every subscribed table.

13.4. Virtuoso scheduler

Virtuoso scheduler allows an arbitrary SQL command to be run at certain intervals.

The scheduler process wakes up every n minutes, scans the SYS_SCHEDULED_EVENT table and executes each command which applies to the current time. Note that overdue commands are executed only once.

There is a virtuoso.ini parameter "SchedulerInterval" under Parameters section which defines the scheduler wake-up interval. Set this to 0 (the default) to disable the scheduler.

13.4.1. SYS_SCHEDULED_EVENT

```
CREATE TABLE SYS_SCHEDULED_EVENT(
    SE_NAME             varchar,
    SE_START            datetime,
    SE_SQL              varchar,
    SE_LAST_COMPLETED   datetime,
    SE_INTERVAL         integer,
    PRIMARY KEY(SE_NAME));
```

This table describes each scheduled SQL command. SE_NAME is the name of the scheduled event. SE_START is the first schedule execution time. SE_SQL is the text of the SQL command to be executed. SE_LAST_COMPLETED is the last time when the SQL command was executed successfully. SE_INTERVAL is the interval between the runs of the SQL command in minutes.

Defining a new scheduled event means adding a row to the SYS_SCHEDULED_EVENT with an insert statement like this:

```
INSERT INTO SYS_SCHEDULED_EVENT (SE_NAME, SE_SQL, SE_START, SE_INTERVAL)
                VALUES (.....)
```

13.5. Transactional Replication Example

```
-- ================================
-- ON PUBLISHER SIDE (named 'rep1')
-- ================================
set DSN=1111;
reconnect;
-- Create test tables & fill with data
create table DB.DBA.TEST (id integer, name varchar, tm datetime,
        content long varchar, primary key (id, name));
create table "ab ""cd" ("id key" integer, "ef ""gh" varchar, primary key ("id key"));
insert into DB.DBA.TEST values (1, 'a', now(), 'xxx');
insert into DB.DBA.TEST values (1, 'b', now(), 'xxx');
insert into DB.DBA.TEST values (1, 'c', now(), 'xxx');
insert into DB.DBA.TEST values (1, 'd', now(), 'xxx');
insert into "ab ""cd" values (1,'1');
-- Public one account named 'dav'
REPL_PUBLISH ('dav', 'dav.log');
-- Add an existing collection '/DAV/repl' into the 'dav' publication
REPL_PUB_ADD ('dav', '/DAV/repl/', 1, 0, null);
-- Public second account named 'tbl' for tables'
REPL_PUBLISH ('tbl', 'tbl.log');
-- Add 'TEST' table into the 'tbl' publication account
REPL_PUB_ADD ('tbl', 'DB.DBA.TEST', 2, 0, null);
-- Add 'ab "cd' table into the 'tbl' publication account
REPL_PUB_ADD ('tbl', 'DB.DBA.ab "cd', 2, 0, null);

-- ================================
-- ON SUBSCRIBER SIDE (named 'rep2')
-- ================================
set DSN=1112;
reconnect;
```

```
-- Add publication server named 'rep1' with DSN '1111' placed
-- on 'localhost' machine and using port '1111'
REPL_SERVER ('rep1', '1111', 'localhost:1111');
-- Add subscription for 'dav' publication account
REPL_SUBSCRIBE ('rep1', 'dav', null, null, 'dba', 'dba');
-- Perform initial copy of publication 'dav' data
DB..REPL_INIT_COPY ('rep1', 'dav');
-- Add subscription for 'tbl' publication
REPL_SUBSCRIBE ('rep1', 'tbl', null, null, 'dba', 'dba');
-- Perform initial copy of publication 'tbl' data
DB..REPL_INIT_COPY ('rep1', 'tbl');

-- Now we look at copied data (should return 4)
select count(*) from TEST;
-- And second table (should return 1)
select count(*) from "ab ""cd";

-- Turn subscription in 'SYNC'
SYNC_REPL();

-- ================================
-- ON PUBLISHER SIDE (named 'rep1')
-- ================================
set DSN=1111;
reconnect;
-- Insert an additional data
insert into DB.DBA.TEST values (7,'a',now(), repeat('x',1000000));
insert into "ab ""cd" values (2,'2');
insert into "ab ""cd" values (3,'3');
insert into "ab ""cd" values (4,'4');
delete from "ab ""cd" where "id key" = 4;
update "ab ""cd" set "ef ""gh" = '4' where "id key" = 3;

-- ================================
-- ON SUBSCRIBER SIDE (named 'rep2')
-- ================================
set DSN=1112;
reconnect;
-- make a procedure to check status of subscription
create procedure WAIT_FOR_SYNC (in srv varchar, in acct varchar)
{
  declare level, stat integer;
  stat := 0;
  while (level < 6)
    {
      repl_status (srv, acct, level, stat);
      if (stat = 3)
        SYNC_REPL ();
    }
};
-- run it till subscription got the right level of synchronization
WAIT_FOR_SYNC ('rep1', 'tbl');

-- Check the data (should return 5)
select count (*) from DB.DBA.TEST;
-- Check the data (should return 3)
select count(*) from "ab ""cd";

-- Check the new data entered on publisher (will return 4)
select "ef ""gh" from "ab ""cd" where "id key" = 3;

-- Check the old data deleted from publisher (will return 0)
select count(*) from "ab ""cd" where "id key" = 4;

-- ================================
-- ON PUBLISHER SIDE (named 'rep1')
-- ================================
set DSN=1111;
reconnect;
-- Create an table to use for procedure calls
create table p_test (id integer, dt varchar, primary key (id));
insert into  p_test values (1, '1');
insert into  p_test values (2, '2');
```

```
insert into  p_test values (3, '3');
insert into  p_test values (4, '4');
insert into  p_test values (5, '5');
-- And an procedure which inserts records in p_test table
create procedure t_proc (in i integer)
{
  declare d varchar;
  select dt into d from p_test where id = i;
  d := concat (d, d);
  update p_test set dt = d where id = i;
};

-- Make an account named 'proc' for procedure replication
REPL_PUBLISH ('proc', 'proc.log');
-- Add 't_proc' procedure into 'proc' publication
REPL_PUB_ADD ('proc', 'DB.DBA.t_proc', 3, 0, 3);

-- ================================
-- ON SUBSCRIBER SIDE (named 'rep2')
-- ================================
set DSN=1112;
reconnect;
-- crate the same table on subscriber
create table p_test (id integer, dt varchar, primary key (id));
-- insert an data into it
insert into  p_test values (1, '1');
insert into  p_test values (2, '2');
insert into  p_test values (3, '3');
insert into  p_test values (4, '4');
insert into  p_test values (5, '5');
-- Add subscription for 't_proc' publication
REPL_SUBSCRIBE ('rep1', 'proc', null, null, 'dba', 'dba');
-- Perform initial copy of procedure definition
DB..REPL_INIT_COPY ('rep1', 'proc');
-- Turn all subscriptions in 'SYNC' state
SYNC_REPL();

-- ================================
-- ON PUBLISHER SIDE (named 'rep1')
-- ================================
set DSN=1111;
reconnect;
-- call 5 times 't_proc'
t_proc(1);
t_proc(2);
t_proc(3);
t_proc(4);
t_proc(5);

-- ================================
-- ON SUBSCRIBER SIDE (named 'rep2')
-- ================================
set DSN=1112;
reconnect;
-- create an procedure to check synchronization level
create procedure WAIT_FOR_SYNC (in srv varchar, in acct varchar, in n integer)
{
  declare level, stat integer;
  stat := 0;
  while (level < n)
    {
      repl_status (srv, acct, level, stat);
    }
};
-- and run it till level set to 5
WAIT_FOR_SYNC ('rep1', 'proc', 5);

-- check local data (should return 5)
select count(*) from p_test where length (dt) = 2;

-- ================================
-- ON PUBLISHER SIDE (named 'rep1')
-- ================================
```

```
set DSN=1111;
reconnect;
-- modify procedure to insert 123 new records
create procedure t_proc (in i integer)
{
  declare d varchar;
  declare n integer;
  n := 128;
  while (n > 5)
    {
      insert into p_test (id, dt) values (n, cast (n as varchar));
      n := n - 1;
    }
};
-- and call it once
t_proc (1);

-- ================================
-- ON SUBSCRIBER SIDE (named 'rep2')
-- ================================
set DSN=1112;
reconnect;
-- and run check routine till level set to 7
WAIT_FOR_SYNC ('rep1', 'proc', 7);
-- check  local data (should return 128, old 5 + 123 new records)
select count(*) from p_test;
```

13.5.1. Transactional Replication Objects Example

Preconditions

publisher named 'rep' with replication address 'localhost:1111'

subscriber named 'sub'

on subscriber should be defined DSN for rep named '1111'

On publisher side. Creating publication

```
SQL> REPL_PUBLISH ('pub', 'pub.log');

-- Add row in SYS_REPL_ACCOUNTS table
SQL> select SERVER, ACCOUNT from SYS_REPL_ACCOUNTS
        where ACCOUNT = 'pub';

SERVER ACCOUNT
--------------
rep    pub
```

started new replication log file in server working directory named 'pub.log'

add entry in repl.cfg in server working directory

```
SQL> REPL_PUB_ADD ('pub', 'DB.DBA.TEST', 2, 0, null);

SQL> select * from SYS_TP_ITEM;

TI_SERVER TI_ACCT TI_TYPE TI_ITEM     TI_OPTIONS TI_IS_COPY
_____________________________________________________________
rep       pub     2       DB.DBA.TEST NULL       0
```

Adding items to the publication

Add row in SYS_TP_ITEM

On Subscriber Side. Adding a new publisher

```
SQL> REPL_SERVER ('rep', '1111', 'localhost:1111');

-- Add row in SYS_SERVERS
```

```
SQL> select * from SYS_SERVERS;

SERVER                    DB_ADDRESS                REPL_ADDRESS
------------------------------------------------------------------
rep                       1111                      localhost:1111

   SQL> REPL_SUBSCRIBE ('rep', 'pub', 'dav', 'dav_group', 'uid_for_rep', 'uid_pwd_for_rep');
```

'publication and subscription servers have identical names.' if subscriber and publisher have the same names.
'Publishing server must be declared with REPL_SERVER before subscribing' if publisher 'rep' is not defined from previous step.
'The subscription 'pub' already exist' if on subscriber 'sub' already exists subscription 'pub'
'User name and password should be supplied when subscribe to new publisher' if 'uid_for_rep' or 'uid_pwd_for_rep' not supplied
'The table 'DB.DBA.TEST' already exists' if on subscriber 'sub' already exist table in subscription.
'The WebDAV collection '/DAV/rep/' already exists' if on 'sub' already exist WebDAV collection in subscription.
a VDB error message if subscriber 'sub' cannot perform attaching TI_ITEM view from publisher 'pub'

```
SQL> select * from SYS_TP_ITEM where TI_ACCT = 'pub'

TI_SERVER TI_ACCT TI_TYPE TI_ITEM    TI_OPTIONS TI_IS_COPY TI_DAV_USER TI_DAV_GROUP
_____________________________________________________________________________
rep       pub     2       DB.DBA.TEST NULL       0           dav       dav_group

SQL> select SERVER, ACCOUNT from SYS_REPL_ACCOUNTS where ACCOUNT = 'pub';

SERVER ACCOUNT
--------------
rep    pub
```

Making a subscription

This may signal an SQL error if a precondition is not met.

Add row in SYS_TP_ITEM and row in SYS_REPL_ACCOUNTS

13.6. Replication Logger Sample

The logger directory in the samples in the distribution contains a simple load balancing sample. It implements a simplified web site hit log where there is a count of hits maintained per user name and origin IP of each hit.

Thus the transaction being replicated between the servers consists of incrementing an IP's hit count and then incrementing a user's hit count. If either IP or user do not have a count, a row is added with a count of 1. The transaction is then logged for replication, so that all servers get all hits, no matter which of the replicating servers processes the hit.

```
create table wl_ip_cnt (ic_ip varchar, ic_cnt integer,
       primary key (ic_ip));

create table wl_user (wu_user varchar, wu_cnt integer,
       primary key (wu_user));

create procedure wl_hit_repl (in ip varchar, in usr varchar)
{
  set isolation = 'serializable';
  update wl_ip_cnt set ic_cnt = ic_cnt + 1 where ic_ip = ip;
  if (0 = row_count ())
    insert into wl_ip_cnt (ic_ip, ic_cnt) values (ip, 1);
  update wl_user set wu_cnt = wu_cnt + 1 where wu_user = usr;
  if (0 = row_count ())
    insert into wl_user (wu_user, wu_cnt) values (usr, 1);
}

create procedure wl_hit (in ip varchar, in usr varchar)
{
  wl_hit_repl (ip, usr);
  repl_text ('hits', 'wl_hit_repl (?, ?)', ip, usr);
}
```

The application client calls wl_hit on one of the mutually replicating servers to log an event. The event's trace will then be propagated to all other servers. The wl_hit_repl function does the actual work. The top level function calls this plus logs the call with its arguments on the local server's hits publication for distribution to other servers.

13.6.1. Configuration of the Sample

The following sequence of calls can be used to define a network of four servers, each replicating every other server. For the sake of example, they are all on localhost and listen at ports 2001 through 2004.

```
repl_server ('log1', 'localhost:2001');
repl_server ('log2', 'localhost:2002');
repl_server ('log3', 'localhost:2003');
repl_server ('log4', 'localhost:2004');

repl_publish ('hits', 'hits.log');

repl_subscribe ('log1', 'hits');
repl_subscribe ('log2', 'hits');
repl_subscribe ('log3', 'hits');
repl_subscribe ('log4', 'hits');
```

First all the servers are identified. Next the local server declares that it has a publication 'hits'. Next it subscribes to the hits publications of all other servers. In the process it also subscribes to itself, which signals an error and has no other effect.

In this way all servers share one configuration. Each server knows which of the servers it is based on the DBName setting in its virtuoso.ini file.

13.6.2. Synchronization

```
create procedure log_sync ()
{
  for select SERVER, ACCOUNT from SYS_REPL_ACCOUNTS do
    {
      if (SERVER <> repl_this_server ())
        {
          declare err, msg varchar;
          err := '00000';
          exec ('repl_sync (?, ?, ?, ?)', err, msg, vector (SERVER, ACCOUNT, 'dba', 'dba'), 0);
        }
    }
}
```

This procedure will go through all subscriptions and request sync for each. Note that the repl_sync function is called inside exec to catch any possible exceptions, as servers may not be available etc. For the sake of simplicity this supplies the literal default dba login 'dba', 'dba' as authentication.

The replication sample schedules a call to this function to be made every minute as a background job. if all replication servers are on line and in sync or syncing the function will return without delay or effect. Otherwise it will keep trying until it gets a connection.

13.6.3. Running the Sample

The logger directory contains various scripts for starting and stopping servers etc.

log_init.sh - Creates the databases with tables and procedures loaded in the l1, l2, l3 and l4 subdirectories.

log_start.sh - starts the 4 servers and leaves them running in the background.

log_shut.sh - Shuts down the 4 test servers.

hits.sh <hist-per-hour> <no-of-hits>

Starts the hits program on each of the 4 servers. The first command line argument gives the test transaction rate for each client and the next gives the duration as a transaction count.

```
hits <dsn> <uid> <pwd> <hits-per-hour> <no-of-hits>
```

The hits executable repeatedly calls wl_hit with random arguments and collects statistics on call times. If calls complete at a rate faster than the requested rate this periodically sleeps to keep the rate close to the requested rate. It prints statistics every 1000 hits.

13.6.4. Notes on the Sample's Dynamics

When the network initially starts all the publications are at level 0 and in sync. When transactions are fed into the network at a sufficiently slow rate all the servers get to process all transactions in real time. Note that the structure is such that every server does everybody else's work in addition to its own. Thus the insertion rate of the network can't be expected to be higher than that of an individual server. However read load can be spread across servers, so that this type of configuration is effective for balancing query load but not for balancing update load.

As we increase the transaction rate at each server we reach a point at which the queue of locally committed but un-replicated transactions grows faster than the other servers will absorb the feed. The servers will each eventually disconnect all synced replication to stop the queue from growing. Once the queue that no longer grows goes empty the subscribers get disconnected. At this point all servers only process their own load without any other distraction.

Next each server will notice that it is disconnected from the network and will attempt a resync as a result of the periodic scheduled call to log_sync. Each server will then re-establish a connection to every other server and start resyncing. This will lead to the network being again in sync if the per server transaction rate slows down sufficiently to allow replicators to catch up. If this does not happen the syncing can stay in progress indefinitely, until it either reaches sync or is terminated.

Typically a server's capacity for processing local transactions is greater than its capacity for replaying replication feed. This is because one thread is responsible for all replay activity while many threads can process local transactions.

The net result of this scheduling policy is that even a heavily replicated network will scale to high peak loads and will automatically return to sync state as soon as the peak is over. If guaranteed transaction level synchronicity must be maintained between servers then the application should not be written using transactional replication but rather with distributed transactions, where each commit makes sure the transaction is fully processed on each participant before returning to the client. This is however up to several times slower and will stop the entire network if a single node fails.

Chapter 14. Web Application Development

Abstract

This chapter deals with the Virtuoso HTTP Server, its Dynamic Page generation and components used to configure it. The HTTP Servers features include:

Web server configuration and access control.

Multihosting, Virtual Hosting and Virtual Directories - A single Virtuoso server can map requests into multiple spaces of web pages based on the interface to which the requests come (multihosting) or on the Host HTTP/1.1 header (virtual hosting).

HTTP/1.0 and HTTP/1.1 support, keep alive connections and pipelining.

Proxying - Virtuoso can automatically redirect requests to certain resources or directories to another web server.

Authentication and security - Specific directories can be made to require authentication (basic or digest) or may be restricted to SSL only. Authentication can also be totally user defined.

SOAP - given directories can be defined as containing SOAP services, which can be accessed by SOAP clients, invoking Virtuoso stored procedures.

Dynamic content - Given paths can be executable, allowing defining VSP, and VSPX pages that generate HTML or XML using embedded Virtuoso/PL.

WebDAV - Paths can be mapped into DAV collections maintained inside the database. Virtuoso WebDav is a Unix file system-like mechanism allows specifying access rights and ownership of DAV resources.

WebDAV resources may optionally be indexed in a free text index.

A web robot allows copying external web resources into Virtuoso WebDAV.

Virtuoso provides a full function web server with dynamic web page generation capability using SQL procedures embedded in HTML or XML via VSP or VSPX. Runtime hosting allows Virtuoso to generate dynamic web pages from other sources also, such as ASP.Net, PHP or JSP.

Out-of-the-box Virtuoso listens for HTTP requests on the port defined in the HTTP Server section of the Virtuoso INI file. The Visual Server Administration Interface is available at this port and can be used to further configure the web server.

Web resources can be based on the file system, reside in the database or in WebDAV or any combination of all of them. WebDAV space can be specified at the path level, by default, paths beginning with /DAV are mapped to the WebDAV root collection (directory) and all other paths are assumed to be file system based. By default / is mapped to the directory specified as

ServerRoot in the HTTP Server section of the virtuoso.ini file.

Virtual Directories provide a way to make mappings from paths to other resources such as specific file system or WebDAV locations, other HTTP Servers acting as a proxy or to alter specific processing or authentication rules for a directory.

14.1. The HTTP Server

When a request comes in to one of the network interfaces where Virtuoso is listening for HTTP requests, it is matched against a set of virtual directory path mappings. A path mapping has the following attributes:

Interface - interface coming from HTTP request

Virtual Host - matched virtual host name

Logical Path - logical path

Mapped Path - physical location of the resource

Is in DAV - If the resource is placed in WebDAV domain

Default page - the name of the page to be returned if none is supplied

Browseable - allows for returning the list of files in a directory if no specific file is requested

Security - security restrictions to the resource

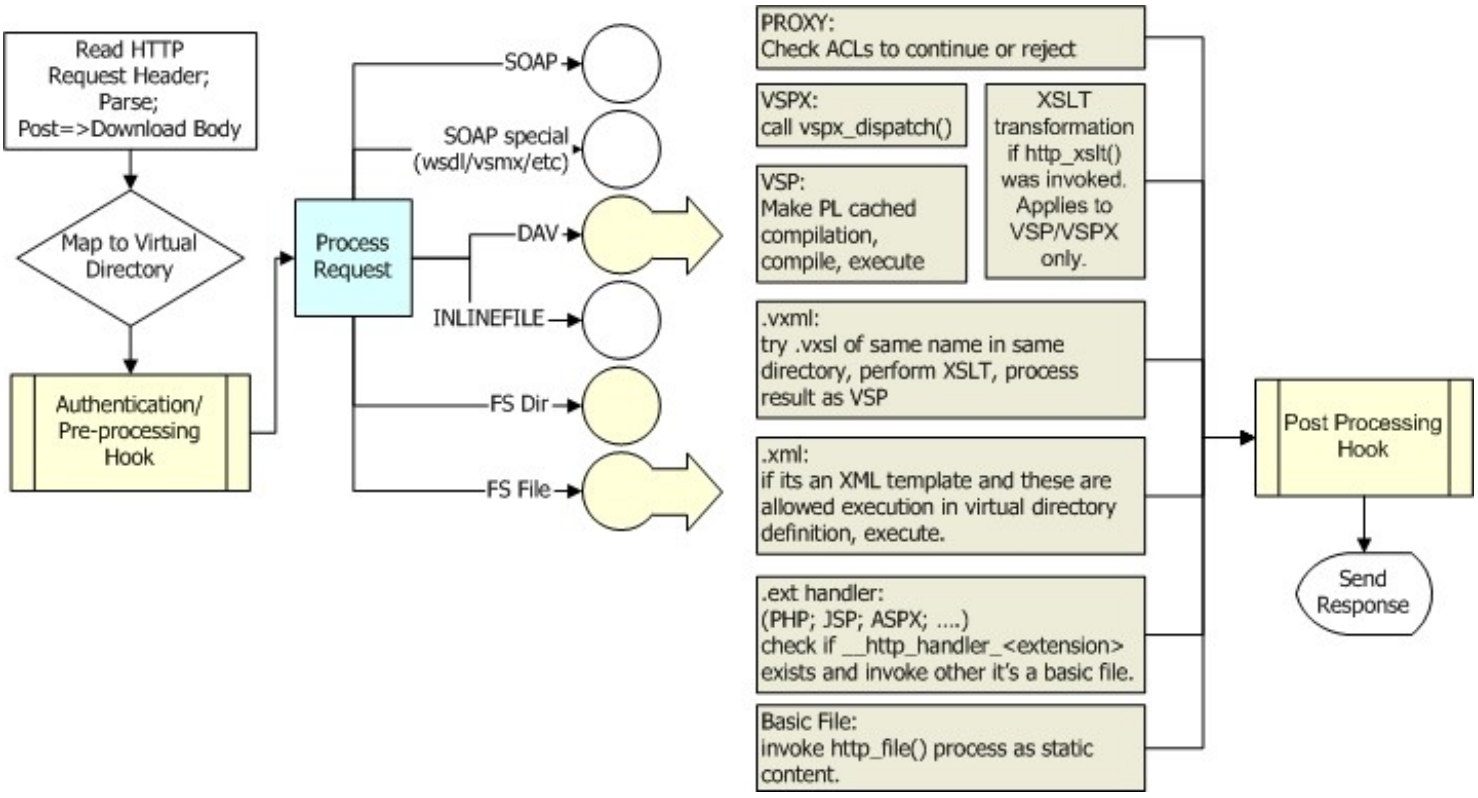
Authentication Function - Virtuoso/PL procedure name, which performs authentication

Realm - an string applied to Authentication function

Request Post-processing Function - Virtuoso/PL procedure name, which performs post-processing action, before sending the response

Database User Account - User account name, on behalf of which active content will be executed

Figure 14.1. HTTP Server Conceptual Diagram



Incoming requests are also checked against the HTTP access control lists. These lists rely on the following attributes of the connection to determine whether the request should be answered:

Client Host or IP Address - the incoming requestor is pattern matched to specify a range.

Destination Host or IP Address - Destination IP/Host match, also applicable to the proxy capabilities.

The Virtuoso HTTP server is HTTP/1.1 compliant. It can accept persistent connections from HTTP/1.1 clients. The HTTP/1.0

requests are processed as normal without the persistent connections feature.

Persistent HTTP connections offer several advantages, such as:

Reduced CPU usages by opening and closing fewer connections.

HTTP requests and responses can be pipelined on a connection. Pipelining allows a client to make multiple requests without waiting for each response, allowing a single TCP connection to be used much more efficiently with lower elapsed time.

Network traffic is reduced.

Latency on subsequent requests is reduced.

See Also:

RFC2616 for more details

14.1.1. HTTP Server Base Configuration

The `[HTTPServer]` section of the Virtuoso INI file contains parameters that directly affect the HTTP server upon server startup. After the server has been started further configuration can be performed using the Visual Server Administration Interface. Here is an example of the section in the INI file:

```
[HTTPServer]
  ServerPort = 8890
  ServerRoot = ../vsp
  ServerThreads = 2
  MaxKeepAlives = 10
  KeepAliveTimeout = 10
  DavRoot = DAV
```

The meanings of these parameters are briefly explained here as a quick reference for the most frequently used parameters:

- **ServerPort.** This specifies the HTTP listen port for the VSP server.
- **ServerThreads.** This specifies the number of concurrently serviced HTTP requests. If there are more concurrent requests, accepting the connections will be deferred until there is a thread ready to serve each.
- **ServerRoot = ../vsp.** This is the file system path of the root directory of files served by the Virtuoso web server. The index.html in that directory will be served for the / URI. If relative, the path is interpreted relative to the server's working directory.
- **MaxKeepAlives = 10.** Connections by HTTP 1.1 clients can remain open after the initial response has been sent. This parameters sets a cap on how many socket descriptors will at most be taken by keep alive connections. Such connections will be dropped by the server ahead of timeout if this number would be exceeded. Thus the maximum number of open sockets for the Virtuoso HTTP server is this number plus the number of threads. A keep alive connection is by definition not associated to any pending processing on any thread.
- **KeepAliveTimeout = 10.** This is a timeout in seconds before Virtuoso closes an idle HTTP 1.1 connection.
- **DavRoot = DAV.** This specifies the root path of DAV resources. If DAV specific HTTP methods are used on Virtuoso, these should only reference resources with paths starting with this. This is the top level DAV collection. This is matched against URI's after the translation from external to internal URI's through the virtual directory table. See Virtual Directories below.

See Also:

The Server Administration chapter contains a complete list of the Virtuoso INI file parameters. In particular, the [HTTPServer] section .

14.1.2. Virtual Directories

A Virtuoso virtual directory maps logical paths to physical resource locations accompanied by rules and/or parameters that govern how the mappings respond to user-agent (e.g. Web browser) requests. This mechanism allows physical locations to be obscured or simply reorganized. Some resource types require authentication challenges, such as the Visual Server Administration Interface, and/or special headers such as SOAP, which is another HTTP endpoint.

Virtual directories are useful when one server has to provide access to several Web sites. Redirects are not a universal solution to this, it is far better to define virtual directories that point to the other sites. Suppose that we have two companies, "a" and "b", that are to share a Virtuoso server but want to represented on the Web by `www.a.com` and `www.b.com` respectively. Their pages could be stored in directories "/a" and "/b" on the server, whilst virtual directories map requests appropriately:

```
http://www.a.com/  -->
```

```
/a
 http://www.b.com/  -->

/b
```

Hence, user-agent requests for `www.a.com` receive pages from /a, and likewise for "b". Requests under these domains are mapped back to their physical location such as the request for the URI `http://www.a.com/images/picture.jpg` retrieves the file `/a/images/picture.jpg` .

Virtual directory definitions are held within the system table DB.DBA.HTTP_PATH. Virtual directories can be administered in three basic ways:

Using the Visual Administration Interface via a Web browser.

Using the functions `vhost_define()` and `vhost_remove()` .

Updating the system table directory using SQL statements.

See Also:

Virtual Directory Administration UI

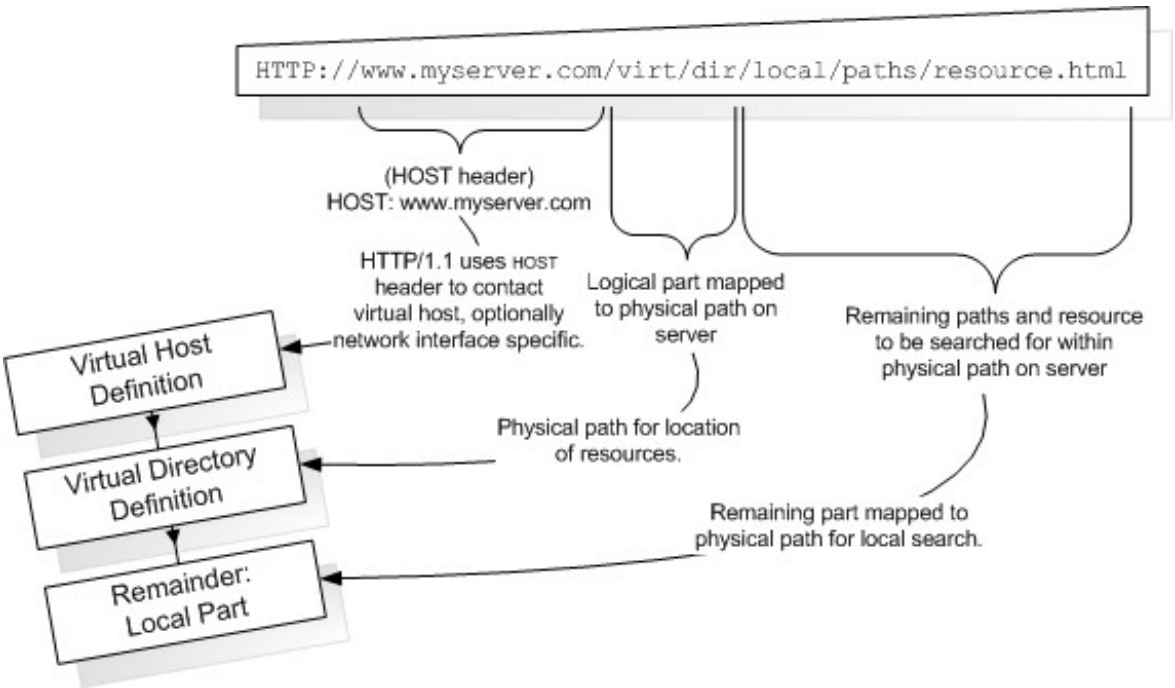
vhost_define() , vhost_remove()

HTTP_PATH system table

Virtuoso matches user-agent requests against a logical path using the longest entry that matches the path extracted from the URI. Suppose we have two entries '`/a/b` ' and '`/a` ' and a request is made of: '`http://foo.bar/a/b/c.html` ', will match the entry for '`/a/b` '.

First, Virtuoso will attempt to locate the physical path that has been mapped to a virtual host, interface and logical path. The virtual host corresponds to the 'Host' header field value from HTTP/1.1 requests. If the first step does not succeed then the server will try resolving the interface and logical path. Failing that, the default step will attempt to resolve the path directly to a physical location.

Figure 14.2. HTTP Virtual Directory Matching



Note:

HTTP 1.0 does not use the HOST header. Virtuoso will have little choice but to send HTTP 1.0 user-agents the contents of the default virtual host definition for the interface.

Thus if the following mappings are in effect:

```
/       ->  /DAV
/doc    ->  http://docs.biz.com:/
/admin  ->  /admin
```

The following translations would be made:

```
/doc/howto/intro.html      -> http://docs.biz.com:/howto/intro.html
/admin/help.vsp            -> /admin/help.vsp
/gizmo/doc.xml             -> /DAV/gizmo/doc.xml
```

Thus, the longest match is selected and the matching substring is replaced by the right hand side of the mapping. Note that this is also how automatic proxying takes place, since a physical path beginning with http:// will be passed forward to a remote server.

Default Pages And Directory Browsing

For each virtual host or logical path pair we can define a list of default pages. If the requested URL path is a directory then the server checks the default page definition for that virtual directory, if a default page exists then the path will be internally expanded to include its name, ands its contents returned.

Example 14.1. Default Page

if we have a mapping for the host:

```
www.a.com
```

with the logical path mapping of:

```
'/' mapped to '/a'
```

with default page 'index.htm', then if the URL

```
http://www.a.com/
```

is requested the server will try to send the content of '/a/index.htm'.

The same mechanism is used to determine whether a directory listing is to be returned. If a mapping is defined to have 'Browseable' set to a number greater than zero then the server, if a default page does not exist or is not defined, a directory listing will be returned to the calling client.

Virtual Hosting and Multi Hosting

The term Virtual Host refers to the practice of maintaining more than one server on one machine, differentiated by their apparent host name. It is often desirable for companies sharing a web server to have their own domains, with web servers accessible as www.company1.com and www.company2.com, without requiring the user to know any extra path information. The Virtual host can be IP-based or non-IP. The IP-based (Multi Hosting) refers to practice of having one machine listen for incoming requests on different network interfaces and respond with different pages. The non-IP-based (Virtual Hosting) refers to the practice of one machine having many DNS aliases, and requests from client to a specific alias returning a different response regarding content of 'Host' HTTP header field. Virtuoso supports IP-based, virtual IP-based, and name-based virtual hosts.

For distinct IP-based, hosts are used to determine on which interfaces Virtuoso will listen and accept HTTP requests.

Managing Host Metadata

To add metadata in /.well-known/host-meta, execute:

```
WS.WS.host_meta_add ([app-name], [xrd-xml-fragment])
```

For example:

```
WS.WS.host_meta_add
  (
```

```
    'dbpedia.page-descriptor',
    '<Link rel="http://dbpedia.org/resource-descriptor" template="http://dbpedia.org/page/{uri}"/>'
)
;
```

Virtuoso As A Proxy

The Virtuoso HTTP server can act as a proxy server on the same port as the HTTP port. You can put the host and port that the Virtuoso HTTP server is listening on, into your browser proxy settings and all requests will be processed by it. Also this can be used to retrieve a page inside VSP.

The physical path setting of a virtual directory definition can be URL to another HTTP server. In which case Virtuoso will act as a proxy to that site when the logical path for it is requested.

See Also:
Virtuoso also provides VSP functions for proxying.

`http_proxy()`.

The nature of Virtuoso's Web Proxying ability makes it easy and seamless to bind multiple websites under one roof. Existing sites do not have to move or change to be integrated under the Virtuoso Proxy. Simply map them under a logical path name. They can be mapped multiple times or from multiple ports.

If you already have pages written and working from other servers via ASP or PHP, then you will be able to run these servers concurrently with Virtuoso so they can share form data and give dynamic content from various sources, consistent with our value proposition of maximum incorporation of new technologies with minimum disruption to existing infrastructure. Whether these servers were hosted on various machines or the same machine there is no need to expose their running ports and services. This makes the end user experience cleaner, and helps maintain some server security and/or anonymity.

Note:
Virtuoso provides runtime hosting capabilities and PHP support, therefore ASP.Net and PHP and other applications can be run and hosted directly within the file system or WebDAV.

Suppose that you have two machines running existing web servers that serve various parts of your intranet. One web server may have been constructed for or by your sales department while the other server may have been a built by the support department. These servers could be resolved by http://sales.mycompany.com/ and http://support.mycompany.com/ respectively.

You can place Virtuoso on another server and start integrating your existing sites under this installation. You may use the Visual Server Administration Interface or choose to use the following commands via the isql interface:

```
DB.DBA.VHOST_DEFINE(lpath=>'/sales', ppath=>'http://sales.mycompany.com/');
DB.DBA.VHOST_DEFINE(lpath=>'/support', ppath=>'http://support.mycompany.com/');
```

This way your old servers will exist under /sales/ and /support/ of your new server. Now you can start adding virtuoso .vsp pages to your new Virtuoso server and they operate interleaved with your existing pages to add new life and functionality as required.

You may decide that you want to install Virtuoso onto a server where a web server already exists. If you plan to use Virtuoso as your default web server and the proxy to your existing server then you will need to make sure that the servers run on different ports. The default port is 80, you will have to configure Virtuoso to use this port from the virtuoso.ini file and then move your existing web server port to another number. Afterwards the procedure is similar:

```
DB.DBA.VHOST_DEFINE(lpath=>'/apache', ppath=>'http://example.com:90/');
```

Proxying Virtuoso via Apache

You may also achieve the same goal as above but in reverse, using another web server as a proxy in front of Virtuoso. If you have an existing Apache server that you want to keep as you default web server then you can set up a proxy within Apache to Virtuoso.

Firstly you will need to make sure that Apache can make use of the mod_proxy module available from most Apache distribution sites. You then have to make sure that it is referenced in your httpd.conf (or apache.conf) file. You should have something like:

```
...
LoadModule proxy_module       modules/libproxy.so
```

```
...
AddModule mod_proxy.c
...
```

Configuration steps

Below we will use the <Location> directive to simplify the configuration:

```
<Location /virtuoso/>
   ProxyPass               http://example.com:8890/
   ProxyPassReverse        /
</Location>
```

1. Set the ProxyPass directive:

   The ProxyPass directive makes Apache to change all incoming URLs and map it to the internal http endpoint.

   So when the browser makes a request for:

   ```
   http://example.com/virtuoso/conductor/login.vsp
   ```

   it is rewritten to use:

   ```
   http://example.com:8890/conductor/login.vsp
   ```

   before sending the request over to the Virtuoso server.
2. Set the ProxyPassReverse directive:

   The ProxyPassReverse directive rewrites the HTTP Headers that come back from Virtuoso to map back to the external URL. This is needed for e.g. 303 Location redirects where Virtuoso will use:

   ```
   Location: http://example.com:8890/conductor/pageXXX.vsp
   ```

   which Apache needs to rewrite to:

   ```
   Location: http://example.com/virtuoso/conductor/pageXXX.vsp
   ```

   before sending the reply back to the browser.
3. If the mapping is / ---> / instead of /virtuoso/ ---> / then the settings should be done, since ProxyPass and ProxyPassReverse only deal with rewriting urls and http headers.

   When however there is a path mapping, there is a third step to take:

   Pages can contain clickable links like:

   ```
   <a href="/conductor/mypage.vsp">Click Here</a>
   ```

   If you click on this link in your browser, it would use:

   ```
   http://example.com/conductor/mypage.vsp
   ```

   which does not map back to your /virtuoso/ vpath in apache.

   As phpBB3 has been written from outset to cater for this situation, it will always need to recalculate fully qualified host/path names everywhere in its pages, which is not always very practical.

   Thus Apache needs to be configured to do page rewriting as well as in:

   ```
   ProxyHTMLEnable         On
   ProxyHTMLURLMap         / /virtuoso/
   ProxyHTMLURLMap         http://example.com:8890/ /virtuoso/
   ```

   This will rewrite the content of every page to make sure that links inside the page are rewritten to use the external mapping of this instance.

If you have set Virtuoso to use `EnabledGzipContent=1` , then you need to tell apache it may need to gunzip the content before doing this rewrite with the following line:

```
            SetOutputFilter         INFLATE;DEFLATE
```

Although this takes a bit extra CPU power etc, it is still practical to setup a virtual path on user's own system that points to some external system.

For example, add this to your httpd.conf to get a mapping to dbpedia-live instance:

```
       <Location /dbp/>
            ProxyPass               http://dbpedia-live.openlinksw.com/
            ProxyPassReverse        /
            ProxyHTMLURLMap         / /dbp/
            ProxyHTMLURLMap         http://dbpedia-live.openlinksw.com/ /dbp/
            SetOutputFilter         INFLATE;DEFLATE
       </Location>
```

Now you should be able to use for ex.:

```
       http://example.com/dbp/page/London
```

Usage Example

```
NameVirtualHost 82.191.21.32

<VirtualHost 82.191.21.32>
ServerName www.mysite.net <http://www.mysite.net>
...

     #  Disable global proxy
     ProxyRequests       Off

     #  Pass original host to Virtuoso
     ProxyPreserveHost   On

     #  Timeout waiting for Virtuoso
     ProxyTimeout        300

     #  Set permission
     <Proxy *>
         Order deny,allow
         Allow from all
     </Proxy>

     #
     #  Map /virtuoso/ to a local Virtuoso instance.
     #
     #  Since ProxyPass and ProxyPassReverse only fix the Headers
     #  of the request, we need to use ProxyHTMLURLMap to rewrite
     #  content.
     #
     <Location /virtuoso/>
         ProxyPass               http://example.com:8890/
         ProxyPassReverse        /

         #  Enable rewrite rules
         ProxyHTMLEnable         On
         ProxyHTMLURLMap         / /virtuoso/
         ProxyHTMLURLMap         http://example.com:8890/ /virtuoso/

# Uncomment this when EnabledGzipContent=1 in virtuoso.ini
         #SetOutputFilter         INFLATE;DEFLATE
     </Location>
</VirtualHost>
```

If we map the virtual host straight through to Virtuoso, we only need header rewriting and save the time/cpu power to rewrite the content:

```
     #
```

```
#  Map / to a local Virtuoso instance
#
#  Since paths are mapped straight through, we do not have to
#  rewrite the content.
#
<Location />
    ProxyPass               http://example.com:8890/
    ProxyPassReverse        /
</Location>
```

See Also:
Apache Online Documentation

14.1.3. Authentication

If a Virtuoso/PL procedure is specified to perform authentication for this mapping then it will be executed. The procedure can take one (varchar) parameter, the 'Realm' value. The result from the procedure must be 1 if authentication successful otherwise must be zero (0). If the procedure returns 0 this causes the processing to terminate and whatever output the hook generated to be sent to the client. Typically this will be an authentication challenge.

14.1.4. Session Management

The Virtuoso HTTP session management consists of functions for session variables manipulation and an ability to define a pre- and post-processing function. The pre and post-processing functions are used to save and restore session data between HTTP posts, usually to and from a session table.

Session management must be enable by settings the flag for persistent session variables in virtual directory mapping. Virtual directory mappings use the *persistent_session_variables* flag, which when specified, session variables can be used in a post-process function to determine if the session variables content must be stored on to the session table or not.

The post-processing function hook can be any user-defined Virtuoso/PL procedure, it will be executed every time after processing of the active page.

See Also:
VSP Session Management and Session Variables

14.1.5. Writing Your Own Authentication and Session Handling

We will explain the following precompiled procedures in Virtuoso used to authenticate three branches of users.

HP_AUTH_SQL_USER - VSP authentication based on DB users
HP_AUTH_DAV_ADMIN - VSP authentication based on WebDAV users
HP_AUTH_DAV_PROTOCOL - WebDAV repository authentication

```
-- Authenticate against names and passwords in SYS_USERS, using HP_SECURITY for level

create procedure
DB.DBA.HP_AUTH_SQL_USER (in realm varchar)
{
  declare _u_name, _u_password varchar;
  declare _u_group, _u_id, allow_basic integer;

  declare auth, sec, lev varchar;
  declare _user varchar;
  declare ses_dta, lines any;

  lines := http_request_header ();         -- got a request header lines
  sec := http_map_get ('security_level');  -- got a security level from http mapping
  if (isstring (sec))
    sec := ucase (sec);
  if (sec = 'DIGEST')
    allow_basic := 0;   -- if security is only digest then allow basic is false
  else
    allow_basic := 1;

  auth :=
    DB.DBA.vsp_auth_vec (lines);        -- parse request header to got authentication credentials
```

```
  if (0 <> auth)                          -- and if client sent any authentication information try it
    {

     lev := get_keyword ('authtype', auth, ''); -- check if client tried basic but
                                                -- domain restricted to digest only
     if (allow_basic = 0 and 'basic' = lev)
        goto nf;
      _user := get_keyword ('username', auth, '');

      if ('' = _user) -- if no user name then 'bye'
        return 0;

      whenever not found goto nf;  -- if no such user in table - 'bye'

      -- got the password from users table

      select U_NAME, PWD_MAGIC_CALC (U_NAME, U_PASSWORD, 1), U_GROUP, U_ID
        into _u_name, _u_password, _u_group, _u_id from DB.DBA.SYS_USERS
        where u_name = _user;

      -- check credential against user name and password

      if (0 = _u_group and 1 = DB.DBA.vsp_auth_verify_pass (auth, _u_name,
                                              get_keyword ('realm', auth, ''),
                                              get_keyword ('uri', auth, ''),
                                              get_keyword ('nonce', auth, ''),
                                              get_keyword ('nc', auth, ''),
                                              get_keyword ('cnonce', auth, ''),
                                              get_keyword ('qop', auth, ''),
                                              _u_password))
        {
          if (http_map_get ('persist_ses_vars'))
            {                     -- if all is ok check for persistent variables storage
              declare vars any;
              declare sid varchar;
            vars := null;
            sid := http_param ('sid');
                -- got from session table stored variables
            vars := coalesce ((select deserialize (ASES_VARS)
              from DB.DBA.ADMIN_SESSION where ASES_ID = sid), null);
            if (sid is not null and vars is null or isarray (vars))
                connection_vars_set (vars);
              if (sid is not null and connection_get ('sid') is null)
                {
                  connection_set ('sid', sid);
                }
            }
          return 1; -- return authenticated
        }
    }
 nf:    -- all errors go there, we compose authentication header
        -- and send 401 unauthorized to the client
  DB.DBA.vsp_auth_get (realm, http_path (),
                md5 (datestring (now ())),
                md5 ('eEsSlLAaf'),
                'false', lines, allow_basic);
  return 0; -- return not authenticated
}
;

-- Authenticate against SYS_DAV_USER, using HP_SECURITY for level,
-- in the context of DAV administration pages accessed through regular HTTP
-- The logic is the same but in difference of password retrieval.

create procedure
DB.DBA.HP_AUTH_DAV_ADMIN (in realm varchar)
{
  declare _u_name, _u_pwd varchar;
  declare _u_group, _u_id, allow_basic integer;

  declare auth, sec, lev varchar;
  declare _user varchar;
  declare ses_dta, lines any;
```

```
  lines := http_request_header ();
  sec := http_map_get ('security_level');
  if (isstring (sec))
    sec := ucase (sec);
  if (sec = 'DIGEST')
    allow_basic := 0;
  else
    allow_basic := 1;
  auth := DB.DBA.vsp_auth_vec (lines);

  if (0 <> auth)
    {

      lev := get_keyword ('authtype', auth, '');
      if (allow_basic = 0 and 'basic' = lev)
        goto nf;
      _user := get_keyword ('username', auth, '');

      if ('' = _user)
        return 0;

      whenever not found goto nf;

      -- we tried to find out password for enabled valid WebDAV user
      select U_NAME, U_PWD, U_GROUP, U_ID
        into _u_name, _u_pwd, _u_group, _u_id from WS.WS.SYS_DAV_USER
        where u_name = _user and U_ACCOUNT_DISABLED = 0;

      if (DB.DBA.vsp_auth_verify_pass (auth, _u_name,
                                get_keyword ('realm', auth, ''),
                                get_keyword ('uri', auth, ''),
                                get_keyword ('nonce', auth, ''),
                                get_keyword ('nc', auth, ''),
                                get_keyword ('cnonce', auth, ''),
                                get_keyword ('qop', auth, ''),
                                _u_pwd))
        {
          if (http_map_get ('persist_ses_vars'))
            {
              declare vars any;
              declare sid varchar;
            vars := null;
            sid := http_param ('sid');
            vars := coalesce ((select deserialize (ASES_VARS)
              from DB.DBA.ADMIN_SESSION where ASES_ID = sid), null);
            if (vars is null or isarray (vars))
                  connection_vars_set (vars);
              if (connection_get ('sid') is null)
                {
                  connection_set ('sid', sid);
                }
            }
          return (_u_id);
        }
    }
 nf:
  DB.DBA.vsp_auth_get (realm, '/admin/admin_dav',
                md5 (datestring (now ())),
                md5 ('vVAadAnIimMDdaNnimda'),
                'false', lines, allow_basic);
  return 0;
}
;

-- The same as for DAV admin pages but in the context of DAV protocol requests.
create procedure
DB.DBA.HP_AUTH_DAV_PROTOCOL (in realm varchar)
{
  declare _u_name, _u_password, _perms varchar;
  declare _u_id, _u_group, req_user, req_group, what integer;
  declare auth varchar;
  declare _user varchar;
```

```
  declare our_auth_vec, lines, sec, path, req_perms, req_meth, cmp_perms, def_page varchar;
  declare _method, lev, allow_basic, authenticated integer;

  declare c cursor for select 1, COL_OWNER, COL_GROUP, COL_PERMS
      from WS.WS.SYS_DAV_COL where WS.WS.COL_PATH (COL_ID) = path;
  declare r cursor for select 2, RES_OWNER, RES_GROUP, RES_PERMS
      from WS.WS.SYS_DAV_RES where RES_FULL_PATH = path;

  authenticated := 0;

  lines := http_request_header ();
  path := http_physical_path ();

  if (isarray (lines))
    {
      req_meth := aref (lines, 0);
      if (strchr (req_meth, ' ') is not null)
        req_meth := lower (substring (req_meth, 1, strchr (req_meth, ' ')));
    }

  -- there we compose mask for permissions compared against
  -- resource or collection permission
  if (req_meth = 'get' or
      req_meth = 'post' or
      req_meth = 'options' or
      req_meth = 'propfind' or
      req_meth = 'head' or
      req_meth = 'trace' or
      req_meth = 'copy')
    cmp_perms := '1__'; -- in this case only read access needed
  else if (req_meth = 'mkcol' or req_meth = 'put')
    {
      if (length (path) > 1
          and strrchr (substring (path, 1, length(path) - 1), '/') is not null)
        path := substring (path, 1,
          strrchr (substring (path, 1, length(path) - 1), '/') + 1);
      cmp_perms := '11_';
    }
  else
    cmp_perms := '11_'; -- but in this we needs a read/write access

  what := 0;
  whenever not found goto fr;
  open c (prefetch 1);
  fetch c into what, req_user, req_group, req_perms;
                -- get from collections owner , owner group and permissions

  def_page := http_map_get ('default_page');
  if (isstring (def_page))
    {
      path := concat (path, def_page);
      what := 0;
    }
fr:
  close c;

  if (not what)
    {
      whenever not found goto fe;
      open r (prefetch 1);
      fetch r into what, req_user, req_group, req_perms; -- if not a collection try a resource
fe:
      close r;
    }

  sec := http_map_get ('security_level'); -- see DB user authentication
  if (isstring (sec))
    sec := ucase (sec);
  if (sec = 'DIGEST')
    allow_basic := 0;
  else
    allow_basic := 1;
```

```
  auth := DB.DBA.vsp_auth_vec (lines);

  if (0 <> auth)
    {
      lev := get_keyword ('authtype', auth, '');
      if (allow_basic = 0 and 'basic' = lev)
        goto nf;

      _user := get_keyword ('username', auth);

      if (_user = '' or isnull (_user))
        {
          goto nf;
        }

      whenever not found goto nf;

      select U_NAME, U_PWD, U_GROUP, U_ID, U_METHODS, U_DEF_PERMS
        into _u_name, _u_password, _u_group, _u_id, _method, _perms from WS.WS.SYS_DAV_USER
        where U_NAME = _user and U_ACCOUNT_DISABLED = 0;
      if (_u_password is null)
        goto nf;
      if (DB.DBA.vsp_auth_verify_pass (auth, _u_name,
                                coalesce(get_keyword ('realm', auth), ''),
                                coalesce(get_keyword ('uri', auth), ''),
                                coalesce(get_keyword ('nonce', auth), ''),
                                coalesce(get_keyword ('nc', auth),''),
                                coalesce(get_keyword ('cnonce', auth), ''),
                                coalesce(get_keyword ('qop', auth), ''),
                                _u_password))
        {
          update WS.WS.SYS_DAV_USER set U_LOGIN_TIME = now () where U_NAME = _user;
          if (http_map_get ('persist_ses_vars'))
          {
            declare vars any;
            declare sid varchar;
            vars := null;
            sid := http_param ('sid');
            vars := coalesce ((select deserialize (ASES_VARS)
            from DB.DBA.ADMIN_SESSION where ASES_ID = sid), null);
          if (vars is null or isarray (vars))
                connection_vars_set (vars);
            if (connection_get ('sid') is null)
            {
                connection_set ('sid', sid);
            }
          }
          if (connection_get ('DAVUserID') <> _u_id)
            connection_set ('DAVUserID', _u_id);
          authenticated := 1;
        }
    }

-- Check permissions
  if (authenticated and _u_id = 1) -- If user is DAV admin all rights granted
    return 1;
  else if (not authenticated and req_perms like concat ('______', cmp_perms, '%'))
    return -1;  -- if not authenticated and resource
                        -- does have public access return authenticated
  else if (authenticated and
          ((_u_id = req_user and req_perms like concat (cmp_perms, '%')) or
           (req_group = _u_group and req_perms like concat ('___', cmp_perms, '%')) or
           (req_perms like concat ('______', cmp_perms, '%'))))
    return (_u_id);     -- if user is owner or belongs to group
                               -- ownership return authenticated
  else if (authenticated)       -- if authenticated but does not access
                                       -- return false but set 403 forbidden
    {
      http_request_status ('HTTP/1.1 403 Forbidden');
      http ( concat ('<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">',
             '<HTML><HEAD>',
             '<TITLE>403 Forbidden</TITLE>',
             '</HEAD><BODY><H1>Forbidden</H1>',
```

```
                'Access to the resource is forbidden.</BODY></HTML>'));
        return 0;
      }
  -- End check permissions

  nf:     -- all error cases go there, we send authentication credentials
          -- and turn status 401 Unauthorized, and return false
    DB.DBA.vsp_auth_get (realm, '/DAV',
        md5 (datestring(now())),
        md5 ('opaakki'),
        'false', lines, allow_basic);
    return 0;
  }
  ;

  -- Post-processing function, this will executed after processing
  -- the request but before sending result to the client.
  create procedure
  DB.DBA.HP_SES_VARS_STORE ()
  {
    declare vars any;
    declare sid varchar;
    if (http_map_get ('persist_ses_vars') and connection_is_dirty ())
      -- if connection variables changed in request and persistent variables stored
      {
        vars := connection_vars (); -- get all variables
        connection_vars_set (null); -- set immediately to empty vector (prevent changing)
        sid := get_keyword ('sid', vars, null); -- got the session ID
        -- store the variables in session table
        if (sid is not null)
          update DB.DBA.ADMIN_SESSION set ASES_VARS = serialize (vars) where ASES_ID = sid;
      }
  }
  ;
```

These functions are implemented as part of Virtuoso server by default. The user can freely define their own procedures based on this logic to use for special authentication (different users information table etc.) and session management.

14.1.6. Cancellation of Web Requests

If there are many long-running tasks processing on Virtuoso it is possible to have denial of service. To prevent this we can use status and stop functions. We can retrieve the URL, client IP-address, and process status code for all currently running VSP requests, then use this information to isolated and eliminate a process.

`http_pending_req()`

See Also:
Long HTTP Transactions

14.1.7. Virtuoso WebRobot API

The Virtuoso WebRobot (WebCopy) is useful for retrieving Internet web sites and storing them on to a local WebDAV repository. Once retrieved, the local copy in the WebDAV repository can be exported to the local filesystem or another WebDAV enabled server. The common features and usages are demonstrated in the WebCopy User Interface in the Visual Server Administration Interface. This document provides the actual API's and techniques useful for some other implementations.

Creating A New Target

A new web server target is created by inserting a row in to the WS.WS.VFS_SITE table and then a row in to the WS.WS.VFS_QUEUE table.

See Also:
Web Robot System Tables for table definitions

Example 14.2. Creating a new target

This example creates a new target pointing to the site http://www.foo.com/help/, with instructions to walk across foreign links, delete a local page if it is detected that it has been removed from the remote, retrieve images, walk on entire site using HTTP GET method. The content will be stored in /DAV/sites/www_foo_com collection in the local WebDAV repository.

1. Create target for http://www.foo.com/help/

```
insert into WS.WS.VFS_SITE
    (VS_DESCR,VS_HOST,VS_URL,VS_OWN,VS_ROOT,VS_NEWER,VS_DEL,
      VS_FOLLOW,VS_NFOLLOW,VS_SRC,VS_OPTIONS,VS_METHOD,VS_OTHER)
  values
    ('My first test', 'www.foo.com', '/help/', 1, 'sites/www_foo_com', '1990-01-01',
      'checked', '/%;', '', 'checked', null, null, 'checked');
```

2. Create start queue entry

```
insert into WS.WS.VFS_QUEUE
    (VQ_HOST,VQ_TS,VQ_URL,VQ_ROOT,VQ_STAT,VQ_OTHER)
  values ('www.foo.com', now(), '/help/', 'sites/www_foo_com', 'waiting', null);
```

Creating A Custom Queue Hook

The custom queue hook can be used to extract the next entry from the robot's queue following a custom algorithm. The following example extracts the oldest entry comparing to the my_data array (this array consists of non-desirable sites) and returns if some are found.

Example 14.3. Creating A Custom Robot Queue Hook

```
create procedure
  DB.DBA.my_hook (
    in host varchar, in collection varchar, out url varchar, in my_data any
  )
{
  declare next_url varchar;
  whenever not found goto done;

  -- we trying to extract the oldest entry
  declare cr cursor for select VQ_URL from WS.WS.VFS_QUEUE
      where VQ_HOST = host and VQ_ROOT = collection and VQ_STAT = 'waiting'
      order by VQ_HOST, VQ_ROOT, VQ_TS for update;

  open cr;
  while (1)
  {
    fetch cr into next_url;
    if (get_keyword (host, my_data, null) is not null) -- process if host not in black-list
    {
      update WS.WS.VFS_QUEUE set VQ_STAT = 'pending'
          where VQ_HOST = host and VQ_ROOT = collection and VQ_URL = next_url;
      url := next_url;
      close cr;
      return 1;
    }
    else -- otherwise continue finding
        update WS.WS.VFS_QUEUE set VQ_STAT = 'retrieved'
          where VQ_HOST = host and VQ_ROOT = collection and VQ_URL = next_url;
    }
done:
  -- if we arrive at the bottom of the queue return false to stop processing
  close cr;
  return 0;
}
;
```

Note:

The default function will return the oldest entry from queue without any restriction. The follow/not-follow restrictions are applied to the path on target before inserting a new queue entry.

Starting The Site's Retrieval/Update

The site retrieval can be performed with the WS.WS.SERV_QUEUE_TOP PL function integrated in to the Virtuoso server.

```
serv_queue_top()
```

Exporting Local Content To Filesystem Or Other Webdav Enabled Server

```
lfs_exp
```

```
dav_exp
```

See Also
Web Robot System tables

14.1.8. HTTP Server Extensions

The Virtuoso shared object library enables you to create your own custom extensions to the Virtuoso HTTP server and create custom VSEs. Support for PHP page execution was implemented using this functionality. Virtuoso can automatically switch processing mode from VSP to PHP or some other custom extension based on the extension of the files being requested from the HTTP server. To enable Virtuoso to process files of a different extension you have to write a VSE handler where part of the name contains the extension: *__http_handler_[extension]* . The VSEs for HTTP handling must have the following parameters:

```
__http_handler_<extension> (  in resource varchar,
                              in parameters vector,
                              in request_header vector,
                              inout type_flag any);
```

- *resource* is either the path to a file or the content of a resource in the WebDAV store. The interpretation of this parameter by the server is dependent of the type_flag parameter.
- *parameters* for execution, the server will pass to this parameter a string session containing the entity body of the POST method request.
- *request_header* is the HTTP request header lines as an array of strings. This parameter will accept the original header as a vector. The vector will contain the complete HTTP request header.
- *type_flag* is an in/out parameter which is a flag for indicating the type of the first parameter. If the resource parameter is a file path in the file system this flag should be NULL, if the resource is located in the WebDAV store this flag should be set to URI of the WebDAV resource, something like: virt://WS.WS.SYS_DAV_RES.RES_FULL_PATH.RES_CONTENT:/DAV/dir1/subdir/myfile.php. Virtuoso will return in this parameter an array of two strings set to the http response status line and the header after execution.

See Also:
Virtuoso Server Extensions Interface

14.1.9. Chunked Transfer Encoding

Virtuoso supports HTTP 1.1 Chunking Encoding which allows Virtuoso to send the user agent chunks of output as the page is still processing. Chunking is enabled by calling `http_flush(1)` within the VSP page. Chunks are sent for every 4k worth of output generated. Chunked mode requires the following conditions:

no "Content-Length" header sent to the client using http_header()

no "Content-Encoding" header sent to the client using http_header()

use http_xslt() is not permitted

The client supports HTTP 1.1

Failing these conditions, `http_flush(1)` will be a No-Operation.

Chunked mode is not supported for static content.

See Also:

See Also:

`http_flush()`

RFC-2616

14.1.10. Using Virtuoso Server capabilities via Apache Web Server

In some situations Virtuoso services like WebDAV, JSP, PHP etc. may need to be accessed via an Apache Web Server. All this can be done through apache's mod_proxy to Virtuoso HTTP server and the Virtuoso HTTP server can be configured to take requests coming only from localhost.

The following configuration makes : The Virtuoso server to listen for HTTP requests coming ONLY from localhost. Apache proxies the outside requests to Virtuoso HTTP server.

```
line added to the /etc/httpd/conf/httpd.conf
(999.999.999.999, must be changed with actual IP address of external interface):
-----------------------------------
Listen 8080
<VirtualHost 999.999.999.999:8080 127.0.0.1:8080>
       ServerAdmin webmaster@host.example.domain
       ProxyPass  / http://example.com/
</VirtualHost>
-----------------------------------


Then changes in the virtuoso.ini

[HTTPServer] section)
...
ServerPort = 127.0.0.1:6666
...
```

14.1.11. Setting Up the Virtuoso HTTPS Listener

The Setting up of the Virtuoso HTTPS Listener can be done by using the file system or using the Virtuoso conductor.

Note that when you want to change from Virtuoso hosted Cert and Public Key to File System and vice versa, first should be stopped the listener for either setup.

Setting Up the Virtuoso HTTPS Listener to host Certificate and Public Key using File System

1. X.509 certificate Generation

   The WebID Protocol consumer needs an x509 certificate with v3 extension "Subject Alternate Name". This attribute is used for the owner's Web ID. For testing purposes we used OpenSSL demo CA to generate such certificates. If you are not using the OpenSSL demo CA, you must first setup a self-signed CA; read OpenSSL documents on how to do this.

   a. Add the following line to the `[usr_cert]` section of the `openssl.cnf` file:

      ```
      subjectAltName=$ENV::ALTNAME
      ```

   b. Set the environment variable `ALTNAME` to the owner's Web ID, e.g.,

      ```
      export ALTNAME=URI:http://example.com/dataspace/person/myname#this
      ```

   c. Make a self-signed certificate, e.g.,

      ```
      $ CA.pl -newreq (follow the dialog)
      $ CA.pl -sign
      ```

   d. When asked to commit the certificate, make sure you see several lines above, like

      ```
      X509v3 Subject Alternative Name:
          URI:http://example.com/dataspace/person/myname#this
      ```

   e. If your browser wants a `PKCS#12` bundle, you must make one

      ```
      $ openssl pkcs12 -export -in newcert.pem -inkey newkey.pem -out mycert.p12
      ```

f. Rename `newcert.pem` and `newkey.pem`, to `mycert.pem` and `mykey.pem` for example.

2. Move `newcert.pem`, `newkey.pem`, and `cacert.pem` into the server's working directory. In our test case, we put the keys in a '`keys`' sub-directory, and added the following lines to the `[HTTPServer]` section of the Virtuoso INI file, `virtuoso.ini`:

```
SSLPort                     = 4443
SSLCertificate              = ./keys/localhost.cert.pem
SSLPrivateKey               = ./keys/localhost.key.pem
X509ClientVerifyCAFile      = ./keys/localhost.ca.pem
X509ClientVerify            = 1
X509ClientVerifyDepth       = 15
```

3. Also in the Virtuoso INI file, in the `[URIQA]` section, `DefaultHost` (`example.com:8890` below) must be edited to correspond to the DNS-resolvable host name ("CNAME") of the Virtuoso host, combined with the `ServerPort` as set in the `[HTTPServer]` section of the same INI file.

```
[URIQA]
DynamicLocal = 1
DefaultHost  = example.com:8890
```

For example, if the CNAME of the host is `virtuoso.example.com`, and the `ServerPort` is `4321`, the `DefaultHost` should be set to `virtuoso.example.com:4321`

```
[URIQA]
DynamicLocal = 1
DefaultHost  = virtuoso.example.com:4321
```

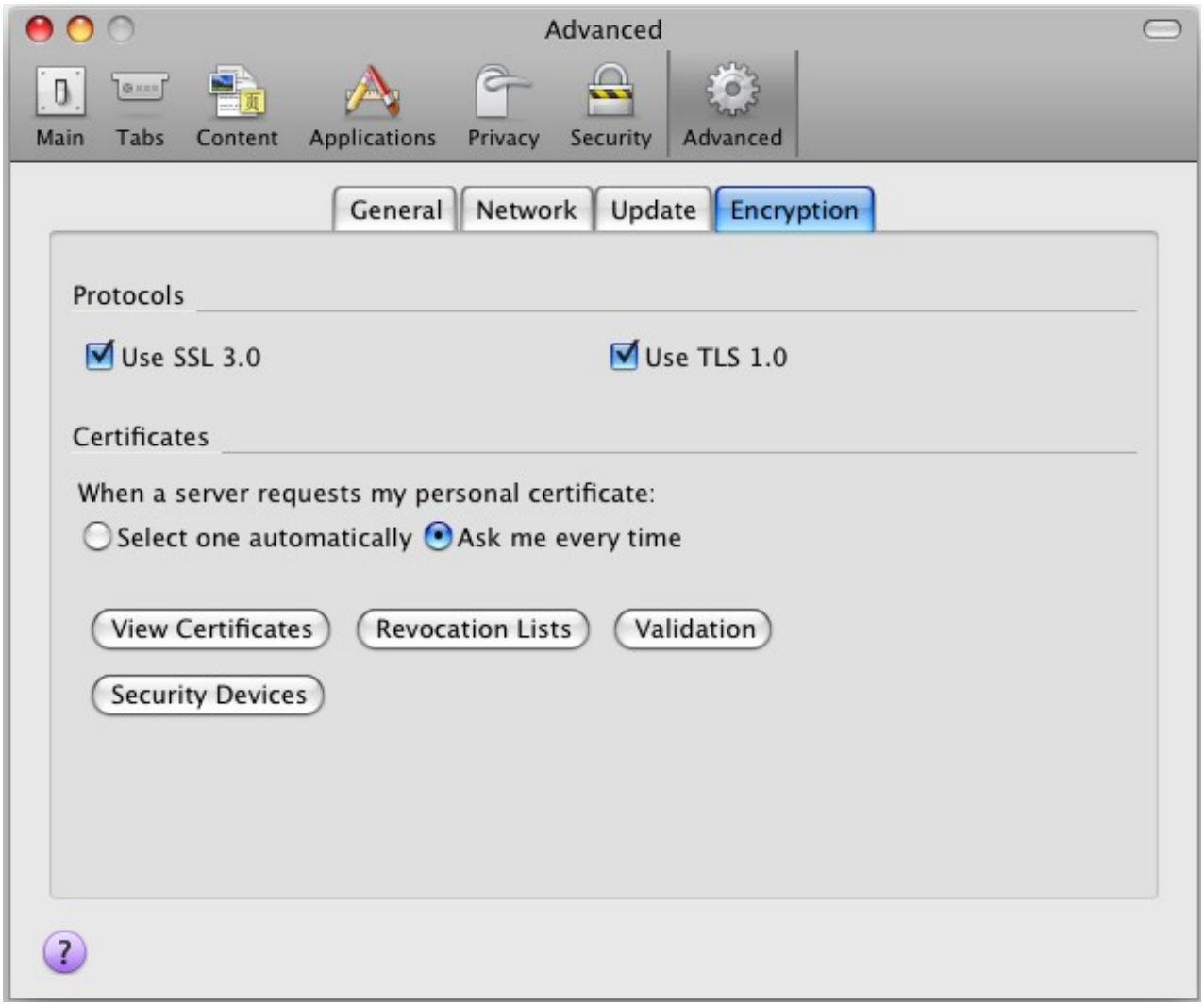
4. Start the Virtuoso server, and look at the log file. Once HTTPS is up, you should see something like:

```
HTTPS Using X509 Client CA ....
HTTPS/X509 server online at 4443
```

5. Setting Up Firefox:

a. In the Preferences dialog, open the Advanced tab, and click the "View certificates" button.

Figure 14.3. HTTPS Listener



b. Click the "Add exception" button ,and enter the address of the HTTPS server you've just configured, i.e. https://virtuoso.example.com:4443/
c. Click OK, and confirm the exception.

Figure 14.4. HTTPS Listener

d. Click to the "Your Certificates" tab, and import `mycert.p12` .

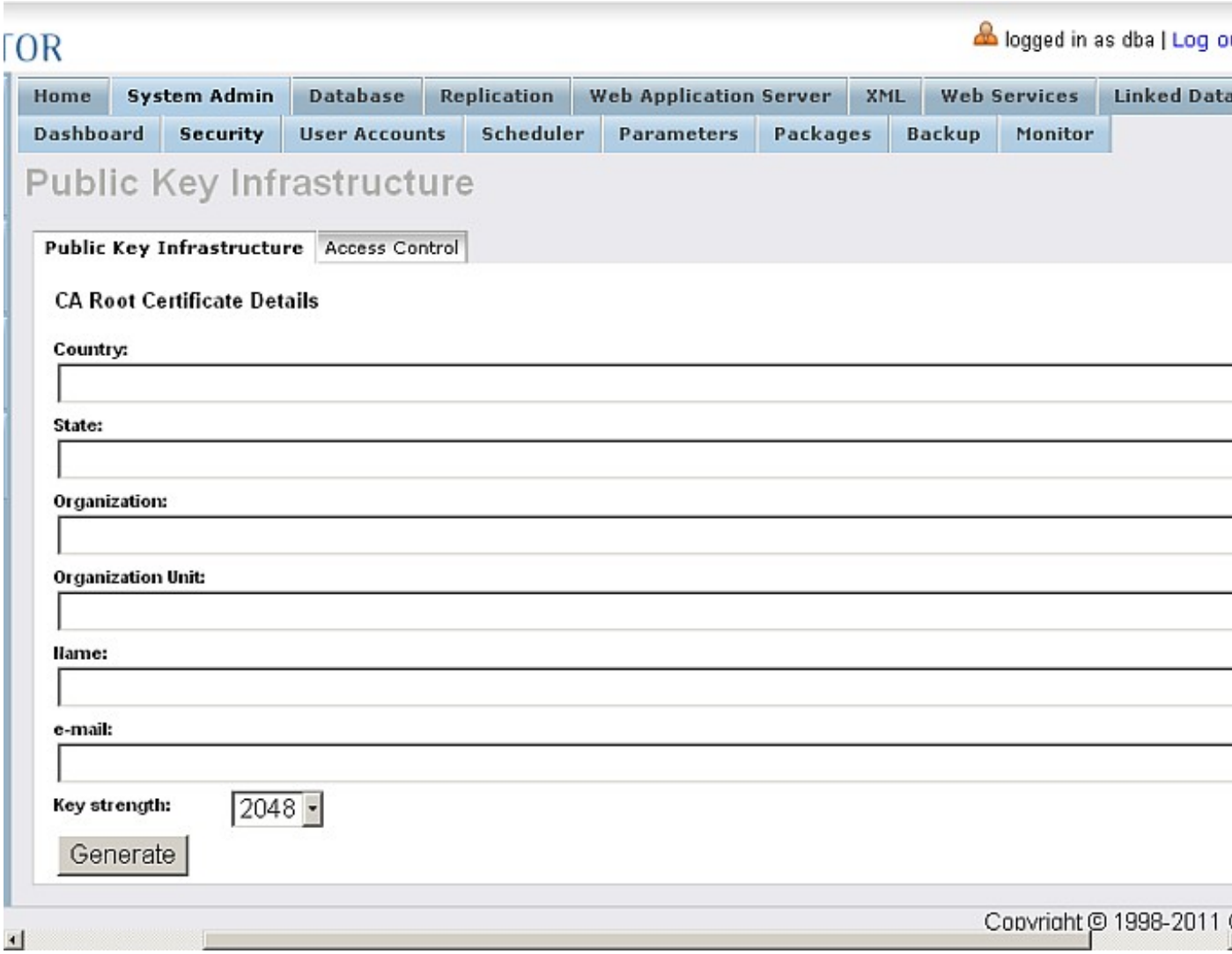
Setting Up the Virtuoso HTTPS Listener to host Certificate and Public Key using Virtuoso Conductor

1. Go to the `http://cname:port/conductor` URL, enter the DBA user credentials.

Figure 14.5. HTTPS Listener

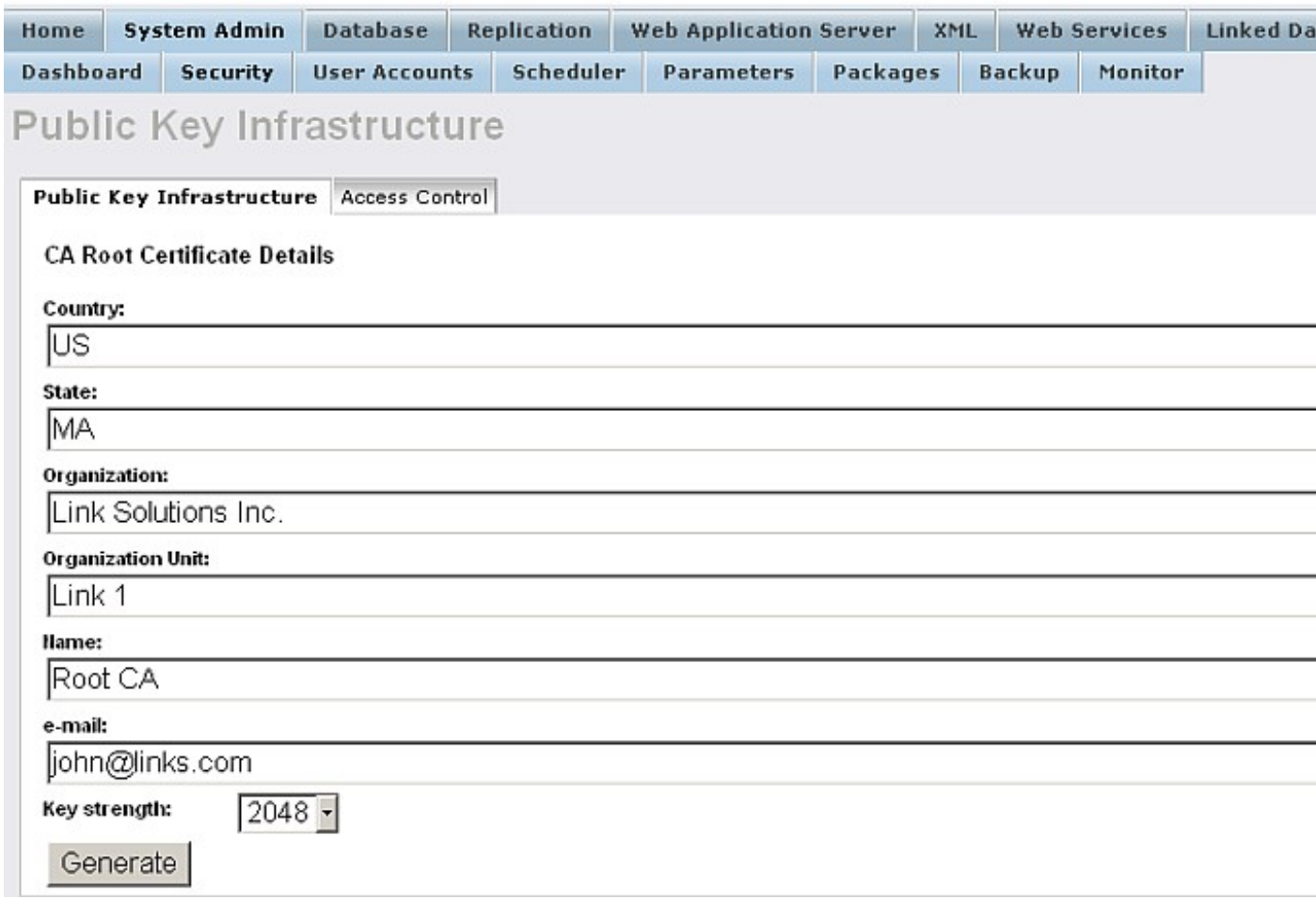
2. Go to System Admin->Security

Figure 14.6. HTTPS Listener



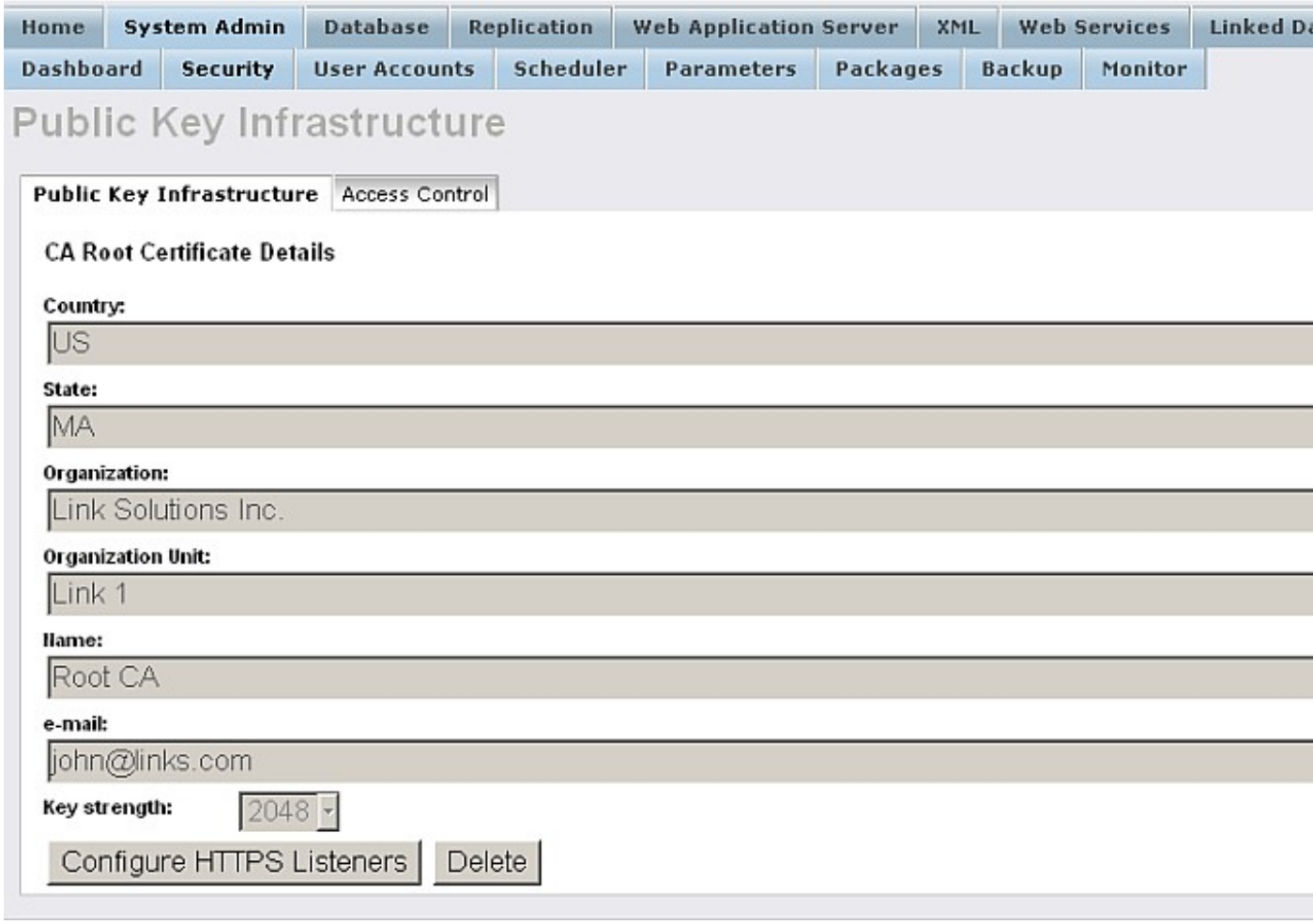
3. Enter the Issuer details:

Figure 14.7. HTTPS Listener



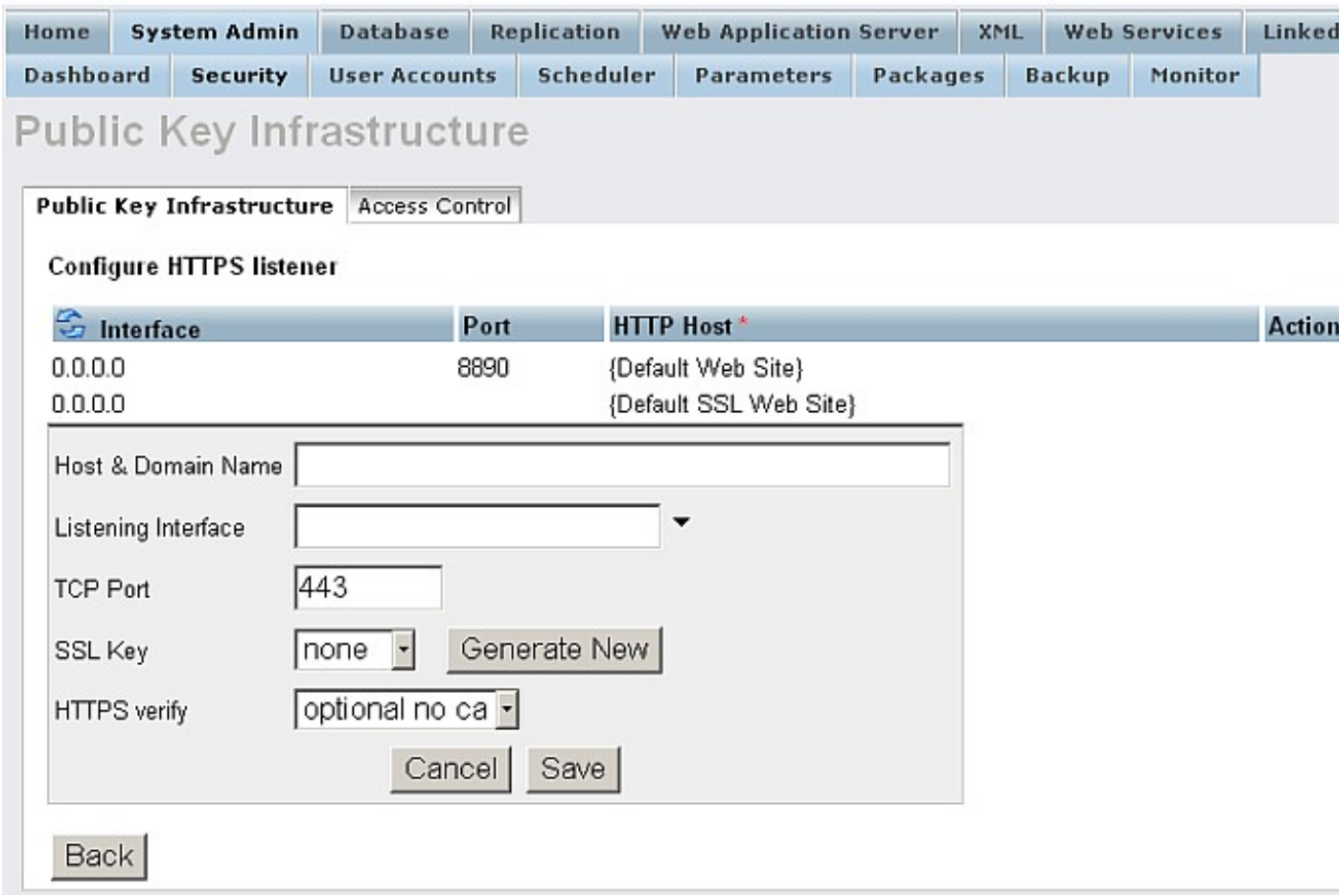
4. Click Generate

Figure 14.8. HTTPS Listener



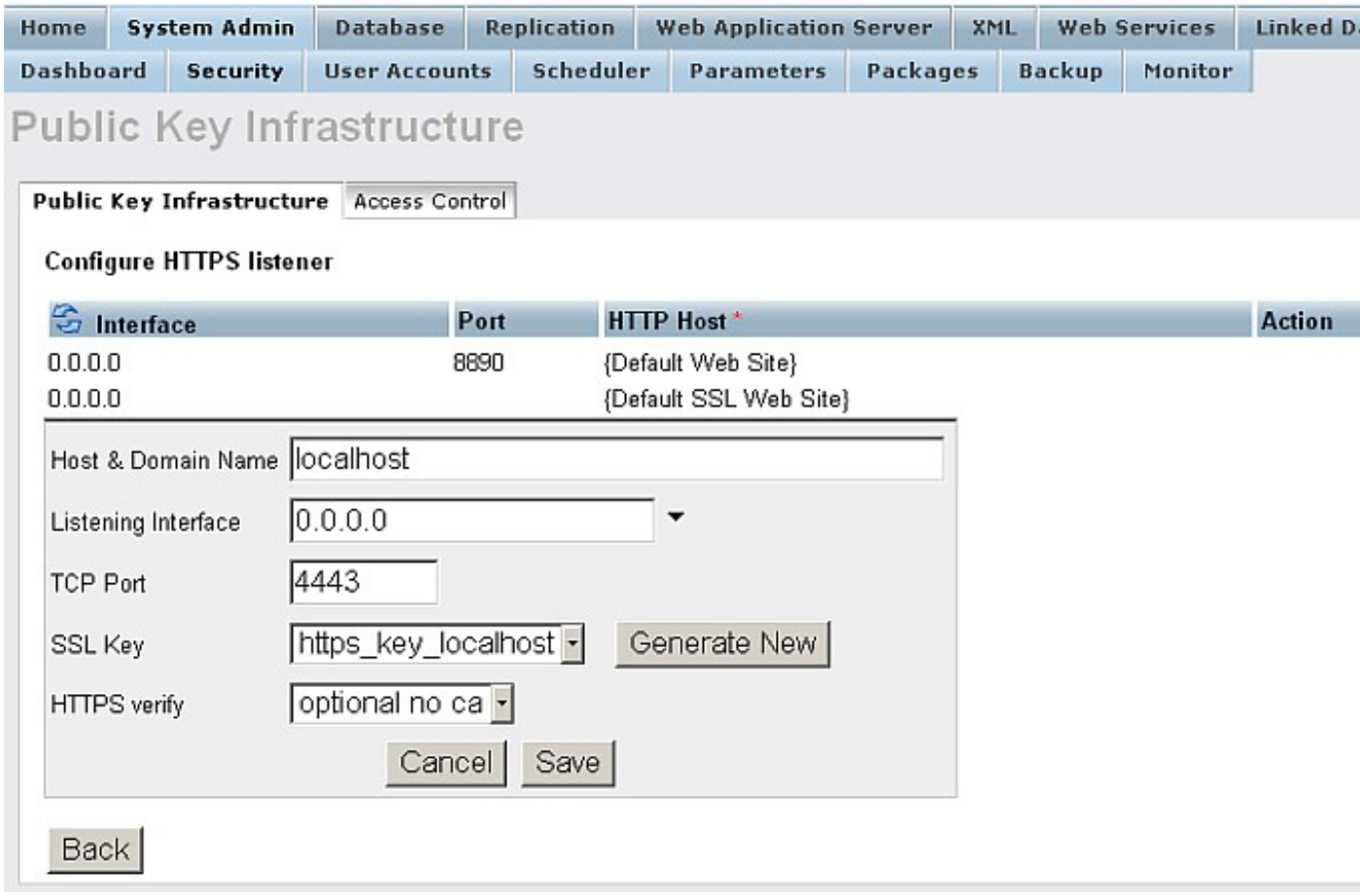
5. Click Configure HTTPS Listeners

Figure 14.9. HTTPS Listener



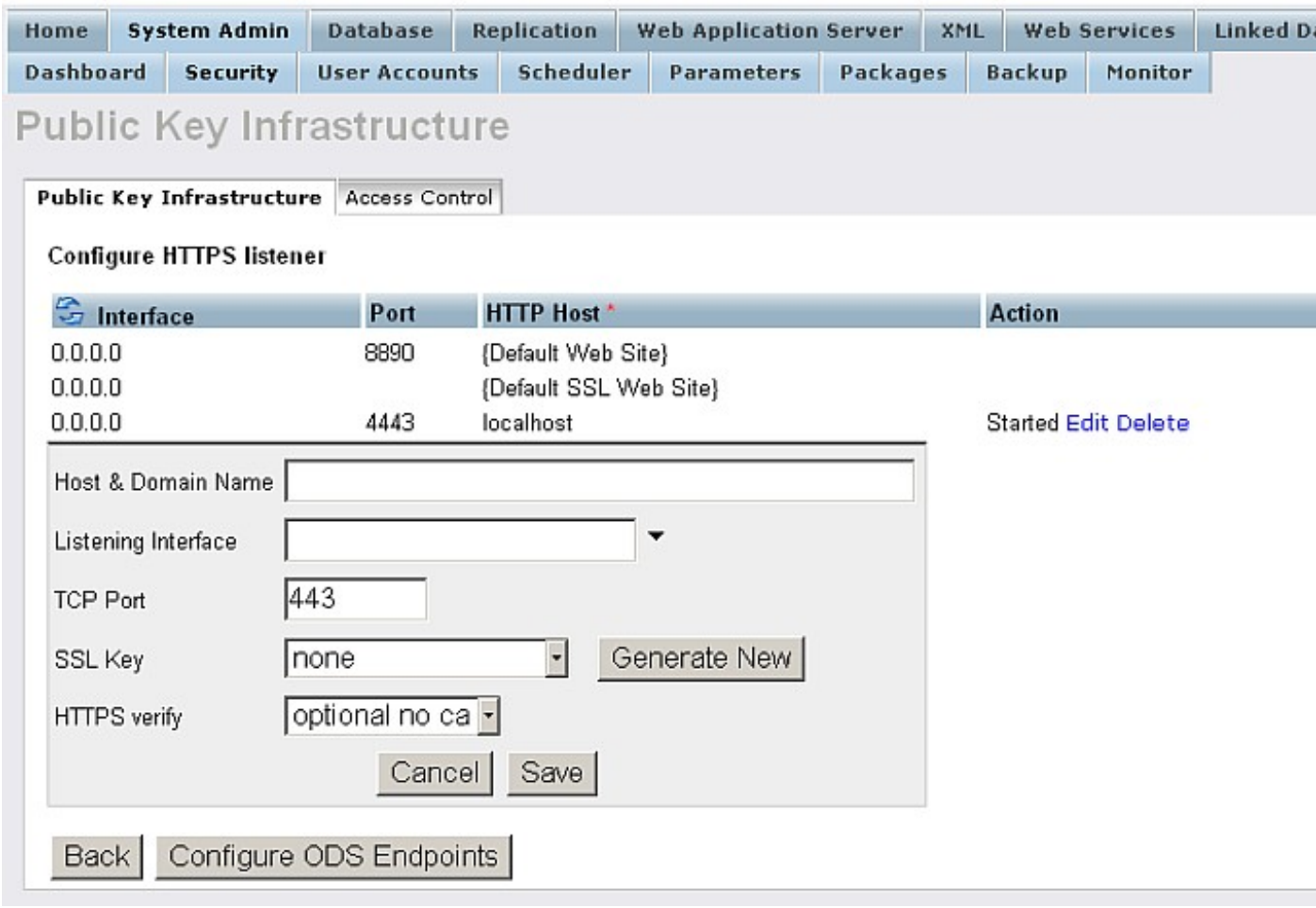
6. Edit the new listener, and click "Generate New" key.

Figure 14.10. HTTPS Listener



7. Click Save

Figure 14.11. HTTPS Listener



8. Setting Up Firefox:

   a. In the Preferences dialog, open the Advanced tab, and click the "View certificates" button.
   b. Click the "Add exception" button and enter the address of the HTTPS server you've just configured, i.e. https://virtuoso.example.com:443/
   c. Click OK, and confirm the exception.

Figure 14.12. HTTPS Listener

14.2. Web Services ACL (Access Control List)

Virtuoso provides a generic access control list for HTTP and other Internet protocol clients. This mechanism uses wildcard expressions to selectively block and allow ranges of IP addresses. An ordered set of patterns is matched against the origin of the request. The first matching pattern's allow/deny flag determines whether the client is approved or not.

The patterns for designating a range of IP addresses follow the syntax of the SQL 'LIKE' predicate; i.e. '%.foo.bar' or '*.foo.bar' for example.

See Also:
The LIKE Predicate & Search Patterns section.

The following mechanisms for services access restriction are defined by default:

Web Server ACL - controls access to the Web server (basic fire wall)
Web Proxy server ACL - controls access to the Web proxy server
News Server ACL - controls access to the Internet news groups (reading or posting)

14.2.1. General purpose ACLs

The system table `DB.DBA.HTTP_ACL` is used to persist ACL definitions, which can be managed with simple INSERT/UPDATE/DELETE statements. The table has the following layout:

Columns for DB.DBA.HTTP_ACL:

HA_LIST - ACL name
HA_ORDER - Position in the list
HA_OBJECT - Object ID (applicable to news groups also)

HA_CLIENT_IP - *PATTERN*

HA_FLAG - Allow/Deny flag, 0 - allow, 1 - deny

HA_RW - Read/Write flag, 0 - read, 1 - post

HA_DEST_IP - Destination IP/Host (applicable to the proxy also)

HA_RATE - Rate Limit

The primary key covers columns HA_LIST, HA_ORDER, HA_CLIENT_IP and HA_FLAG.

See Also

The full schema is listed in the Appendix under System Tables .

HA_LIST (required). Name of the ACL, This is a label to designate a group of rules applicable to a specific service. There are three pre-defined groups which are: HTTP, NEWS, PROXY. Please note that name of group is treated as case-insensitive string. In other words we can think about HA_LIST as domain of the ACL. Developers can use that table to add their own ACLs , and use them with API functions which are discussed in the next section.

HA_ORDER (required). Ordinal number of the rule within its list. This number should be unique within a specific group to have a well defined order of rule matching. Please note that if order is equal the one with allow flag equal to zero (HA_FLAG value see below) takes precedence. The order for applying rules for particular list is in ascending order i.e. first will be applied the rule with the smallest value of HA_ORDER.

HA_CLIENT_IP (required). A pattern to designate separate IP address or group of addresses. This is a string value as for 'LIKE' operator. Some example is: 192.168.*; will match all hosts from local private network.

HA_FLAG (required). A boolean; zero or positive number designating a rule to be applied for matching address/host. The default option is zero (false) for allowed action; positive (true) for denied action. So application may use it in reverse; but in that case it needs to take in account the order precedence (see HA_ORDER).

HA_OBJECT (optional). An integer designating an ID used in predefined ACL for Internet News groups to designate ID of the group. This can be used also in other applications where one desires to apply different rules for different objects. We can think of it as a sub list inside an ACL.

HA_RW (optional). An integer designating a action to be restricted (read/write); used in predefined Internet News groups ACL.

HA_DEST_IP (optional). A string containing pattern to match another IP number; this is used in Web Proxy server ACL. This can also be used in custom application logic.

HA_RATE (optional). An float for Rate Limit. The filter calculates hit rate average and compares with limit for http acl rule. If it is larger then will drop connection. Once per day the statistics will be reset.

- Regular expressions are not supported.
- Can be used patterns, for ex:

```
-- like '%something' or
-- something*
```

The Rate Limit UI can be configured from Conductor->System Admin->Security->Access Control where:

Rate limit is hits/per second from one IP address.

For search engine optimization statistics, for example can be set rate limit 10 (or even 100 so to start to collect statistics), and then to check with http_acl_stats () what values are returned.

14.2.2. ACL Definition/Removal

ACL's can be managed from the administration web interface using the following steps.

- From Admin UI main menu open "System Admin" node.
- Go to Security -> Access Control
- By default three default ACLs are listed:

HTTP - rules for the Web server

NEWS - rules for the Internet News

PROXY - rules for the Web Proxy Server

PSH and PSH-SSL - available only when the pubsubhub_dav.vad is installed.

Figure 14.13. Access Control

- New ACLs can be added by entering a list name and clicking "Add" button.
- Existing ACLs can be edited by clicking on the link "Edit" beside each listed ACL.
- To add new rules, enter the requested information and press button "Add".
- To change the order of the rules use the "top", "bottom", "up" and "down" links.
- ACL's can be removed using the Delete button.

Alternately the HTTP_ACL table can be directly manipulated with SQL. To add new rule:

```
INSERT INTO HTTP_ACL (HA_LIST, HA_ORDER, HA_CLIENT_IP, HA_RATE, HA_FLAG) values
('list_name', <order number>, '*pattern*', <hits_per_second_number>, [1/0]);
```

To remove existing rule:

```
DELETE from HTTP_ACL where HA_LIST = 'list_name' and HA_ORDER = <order number>
and HA_FLAG = [1/0] and HA_CLIENT_IP = '*pattern*';
```

14.2.3. Using ACL's Within Application Logic

The `http_acl_get()` function can be used to test an address against an ACL. The `http_client_ip()` function can be used to determine the IP address or DNS name of a client machine.

Example 14.4. Using ACL's with Application Logic

To restrict a 'foo.bar' (network 333.333.333.0) from accessing a SOAP service one could use the following:

```
-- deny access from '333.333.333.*'
insert into http_acl (ha_list, ha_order, ha_flag, ha_client_ip) values ('MY_SOAP', 1, 1, '333.333.333.*')
-- allow
insert into http_acl (ha_list, ha_order, ha_flag, ha_client_ip) values ('MY_SOAP', 2, 0, '*');

-- a procedure exposed as SOAP service

create procedure SumService (in a int, in b int) returns int
{
  if (0 <> http_acl_get ('MY_SOAP', http_client_ip ()))
    signal ('42000', 'Access denied');
  return (a + b);
}
```

14.2.4. Predefined ACLs

The following ACLs are predefined and have special treatment in Web Server processing:

HTTP. General Web server ACL, applying rules to it controls access to the Web server. Thus if this ACL is managed via web UI one must pay attention not to disable the current connection and render the administration UI inaccessible. The ACL rules for 'HTTP' must contain HA_ORDER, HA_CLIENT_IP (pattern) and HA_FLAG (access flag) only. The rest of column values are ignored. To add or remove rules to that list see 'ACL definition/removal' where HA_LIST must be equal to 'HTTP'. The value of client's IP address will be tested against rules.

NEWS. controls access to the separate Internet News groups, restricting POST or READ access to them.

All valid rules for 'HTTP' are also valid for 'NEWS' with following difference:

HA_OBJECT must be specified as ID of News group to control

HA_RW must be specified as 0/1 to designate which action on group to control: read or post.

The HA_LIST must be equal to 'NEWS' if control is managed with SQL statements. The HA_LIST must be equal to 'NEWS'.

PROXY. This controls access to the Web Proxy Server. Note that Proxy service is disabled by default and can be enabled with 'HTTPProxyEnabled' INI setting. This ACL is similar to the HTTP ACL. Additionally, the pattern in HA_DEST_PATTERN must match the destination server. In this way certain destinations can be restricted.

Note:
HTTP ACLs affect PROXY rules. Therefore if HTTP list rejects a request from a particular client, the proxy access from there also will be rejected.

Example 14.5. Adding/removing ACL's

To allow access from localhost only:

```
insert into http_acl (ha_list, ha_order, ha_flag, ha_client_ip) values ('HTTP', 1, 0, '127.0.0.1');
insert into http_acl (ha_list, ha_order, ha_flag, ha_client_ip) values ('HTTP', 2, 1, '*');
```

To allow only local addresses (private.net/192.168.0.0) to access proxy server.

```
insert into http_acl (ha_list, ha_order, ha_flag, ha_client_ip, ha_dest_ip)
             values ('PROXY', 1, 0, '192.168.1.*', '*');
insert into http_acl (ha_list, ha_order, ha_flag, ha_client_ip, ha_dest_ip)
             values ('PROXY', 2, 1, '*', '*');
```

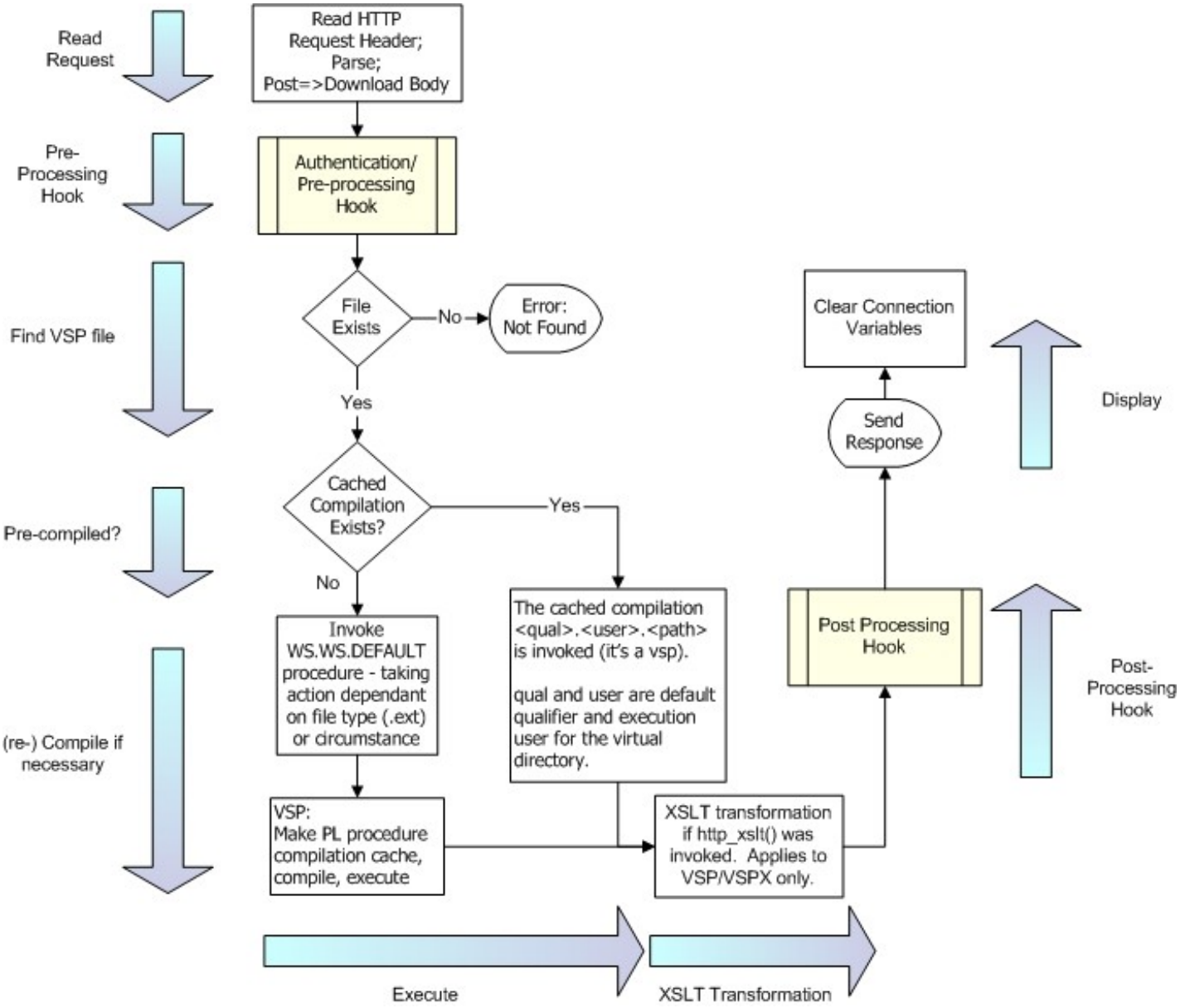
To deny web access from some domain (bad.domain/333.333.333.0)

```
insert into http_acl (ha_list, ha_order, ha_flag, ha_client_ip) values ('HTTP', 1, 0, '*');
insert into http_acl (ha_list, ha_order, ha_flag, ha_client_ip) values ('HTTP', 2, 1, '333.333.333.*');
```

14.3. Virtuoso Server Pages (VSP)

The Virtuoso Server Pages subsystem is an integral part of the Virtuoso server. A VSP page is a file system or DAV resident resource that contains Virtuoso/PL code intermixed with HTML or other static content.

Figure 14.14. VSP Conceptual Diagram



Virtuoso can serve Web pages to HTTP 1.0 and HTTP 1.1 clients. The HTTP document root is set by the ServerRoot parameter in the [HTTPServer] section of the Virtuoso INI file. By default this is set to `vsp` sub-directory of the installation. If this parameter is not set, then the default document root will be the server's working directory.

Executing VSP Pages. Directories under the ServerRoot directory, including the root itself are not automatically allowed to execute dynamic pages such as VSP. A virtual directory must first be created with an appropriate VSP user assigned to permit execution of dynamic pages. After this, Virtuoso will execute files with a "`.vsp` " extension as a VSP page.]

The VSP Stored Procedure. Each VSP page constitutes a Virtuoso stored procedure named after the URI of the page by taking the URI and pre-pending the WS.WS. qualifier and owner. Hence the page `/test.vsp` becomes the procedure `"WS"."WS"."/test.vsp"` . This is automatically performed when a page is first requested. Subsequent requests to the page will not reference the file. The VSP engine will check for changes in the `.vsp` source file before calling the procedure and re-compile as required. The ws_proc_def() SQL function can be used to explicitly update the procedure if the .vsp file is changed.

The VSP Transaction. Each URL is executed in its own transaction. All SQL statements in a page procedure, whether on local or remote data, make up one transaction, unless transaction control statements are explicitly used to divide the page into multiple transactions. If the page procedure returns through completing execution or through a return statement or as a result of a 'no data found' condition, the transaction is implicitly committed. If this commit fails, the output is discarded and the error message indicating the commit failure is sent to the HTTP client. If a VSP procedure returns as a result of an error the transaction is rolled back and the output discarded. The client gets the SQL error message as the HTML body of the reply. A VSP procedure can commit or roll back explicitly with the `commit work` or `rollback work` PL statements.

If the client closes the connection to the server while server-side processing is taking place this will be detected by the server and the transaction will be eventually interrupted and rolled back in the same way as if an ODBC client had disconnected. To control the server reacting or not on the HTTP client disconnecting the SET HTTP_IGNORE_DISCONNECT = ON/OFF should be used.

Each VSP page-procedure is called with three arguments:

path - the URI path of the VSP page itself.

params - the parameters from a previous POST to the server.

lines - the complete set of headers received from the calling client.

Each VSP procedure runs in a context that implicitly contains the stream to the HTTP client. The arguments of the VSP page procedure are represented as arrays of strings. For example, GETing the URL:

```
http://www.test.net/x/y.vsp?arg1=1&arg2=2
```

would cause the following arguments to be given to the page-procedure `WS.WS./x/y.vsp` :

```
path     ('x', 'y.vsp')
params   ('arg1', '1', 'arg2', '2')
lines    ('GET /x/y.vsp?arg1=1&arg2=2 HTTP/1.1', 'Host: www.test.net', ...)
```

Arrays are marked in parentheses with elements separated by commas. The page procedure is called as a result of either a GET or POST request in either HTTP/1.0 or HTTP/1.1. In the event of a POST request the params contains the post data.

Virtuoso can be configured to proxy certain requests to another web server. This allows using another web server for running cgi-bin's, Java servelets or other web server functions. Virtuoso, however, is capable of hosting many other dynamic engines also, such as PHP, JSP and ASP.Net.

See Also:

HTTP Server Base Configuration

Virtual Directory Administration UI

vhost_define() , vhost_remove()

Virtuoso as a Proxy

14.3.1. VSP Markup & Basic Functions

All VSP specific markup is represented as a processing instruction (<? ... ?>).

```
<?vsp
  statement ;
  statement ;
  ...
 ?>
```

This markup introduces Virtuoso PL code to a VSP page, which otherwise may normally contain HTML markup. The code enclosed must begin and end at a statement boundary but a compound statement may begin in one <?vsp tag and end in another. Code outside of these blocks is ignored by Virtuoso and placed directly on the HTTP stream to be sent to the client.

See Also:

The SQL Procedure Language Guide chapter.

Several functions exist to allow VSP code to send data to the HTTP stream. They are basically the same but offer different escaping mechanisms to suit different purposes:

```
http (in value varchar, in stream any);

http_value ( in value any, in tag varchar, in tag varchar, in stream any);

http_url ( in value any, in tag varchar, in stream any);
```

These functions output their *value* argument to the specified stream with varying escaping. The value argument may be any scalar object, i.e. string, date or number and will automatically be cast to varchar before further processing.

`http()` will print out the contents of *value* cast to string without any modification.

`http_value()` will use HTML escapes such that '< ' will be output as '`<` '.

The `http_url()` function will use URL escapes such that '+' replaces spaces and hex escapes like %25 will replace '%'. If `http_value()` gets an XML entity returned by a path expression it outputs the serialization of the entity, including children. This is not the string value since this has the entity start and end tags and other markup. The tag argument allows specifying a tag in which the value is to be enclosed. A non-string value, e.g. 0 or null will cause no tag to be put around the value.

The stream argument may be omitted, in which case it defaults to the HTTP client of the calling procedure. If present, a value of integer 0 will mean the http client. If non-0 the value must be an object returned by string_output() .

These HTTP functions are commonly combined with `sprintf()` which allows string composition based on a template. When using sprintf() to compose data to send to the user agent the %V and %U letters can be used to introduce escapes similar to http_value and http_url, respectively.

Example 14.6. HTTP Functions

```
http (' % <b>')           ' % <b>'
http_value (' % <b>')     ' % &lt;b&gt;'
http_url (' % <b>')       '+%25+<b>'
http_value (12, 'li')     '<li>12</li>'
```

Markup Short-hands

VSP markup short-hands exist for the `http_value()` and `http_url()` functions to perform the same task outside of a VSP code block. This can improve readability of VSP pages.

```
<?= expression ?>  equiv.  http_value()

<?/ expression ?>  equiv.  http_url()
```

These markups are shorthand for substituting values of expressions into the output. The `<?=` tag accepts a SQL expression and casts the value into a string, which is sent to the output. The `<?/` markup sends the value of the expression to the client with HTTP URL escapes.

Example 14.7. VSP Markup

Here is a very simple example of making a two column HTML table from the results of a "select" SQL statement. First using normal functions:

```
<html>
<h2>List of Users</h2>
<table>
<?vsp
  for (select u_name, u_password from sys_users ) do {
    http('<tr><td>');
    http (u_name);
    http('</td><td>');
    http (u_password);
    http('</td></tr>');
  }
?>
      </table>
</html>
```

This fragment outputs a table of user names and passwords. We have chosen to not end the code block until the end of the result so we have repeatedly used the `http()` function to output parts of the table also.

Now the same code but including shorthands:

```
<html>
<h2>List of Users</h2>
<table>
<?vsp for (select u_name, u_password from sys_users ) do { ?>
  <tr>
    <td><?=u_name ?></td>
    <td><?=u_password ?></td></tr>
      <?vsp } ?>
      </table>
</html>
```

14.3.2. Access Request Information

Request information, resulting from an HTTP POST, is available via the *params* vector. The params vector is always available for the purpose in a VSP context. Similarly the *lines* and *path* vectors are available for the HTTP headers and URL path respectively.

Parameters stored in the params vector are stored in keyword-value pairs, such that keywords are stored in the even elements, values in the odd numbered elements. Vectors and arrays can be accessed using the `aref()` and `aset()` functions, however, the exact position of parameters is seldom known. The `get_keyword()` function can be used to return the value of a given parameter. You can specify a default value to return in case the parameter you are looking for is not found.

```
get_keyword ('param_name' , vector [, opt_default_value' ]);
```

If a default value other than '' (empty-string) is not required then a short-hand can be used instead.

```
{?'param_name'}
```

is equivalent to `get_keyword ('param_name' , 'params' );`

Since `get_keyword()` returns only strings, you will find that `cast` ing or conversion functions such as `atoi()` are very useful here.

Example 14.8. Reading the params Vector

Consider retrieving the following page by means of the URL:

```
http://myvirtuoso/test.vsp?arg1=1&arg2=test

<html>
 <body>
  <h1>Test Params</h1>
  <?vsp
  declare _arg1 integer; --   the underscore helps to differentiate from
  declare _arg2 varchar; --   the keyword name, whereas the variable names
  declare _arg3 varchar; --   help use remember which is which.

  _arg1 := atoi(get_keyword('arg1', params));
  _arg2 := {?'arg2'};
  _arg3 := get_keyword('arg3', params, 'was empty'};

  ?>
  <p>values returned:</p>
  <p>arg1: <?=_arg1?></p>
  <p>arg2: <?=_arg2?></p>
  <p>arg3: <?=_arg3?></p>
 </body>
</html>
```

See Also:

See Also:

`get_keyword()`

`atoi()`

`aref()`

`aset()`

cast

Note:
Sometimes a POST can supply a large amount of data, such as from an <INPUT type="file">. When the length of the parameter data exceeds 5,000,000 octets the result is stored as a string session in the params array. In these cases the value should not be copied or converted to a string, it is likely to be truncated. The value should be stored as a BLOB or an external file. When processing possibly large input fields in a form either the fourth parameter to `get_keyword()`/`get_keyword_ucase()` should be set to 1 or they should be accessed through `aref_set_0()`.

14.3.3. Errors in Page Procedures

VSP pages can declare handlers for errors using the normal handler declaration or whenever ... goto construct. Any unhandled error causes the procedure to be terminated and the error message to be sent to the HTTP client instead of the output. Because terminating the page output at an arbitrary point would probably result in not well formed output all the output up to the point of the error is discarded.

If the SQL state signalled with the error is VSPRT the output generated up to the point where the error was signalled is sent as such to the user agent. This special SQL state is useful together with the http_rewrite function for things like sending a redirect based on a condition detected in the middle of page processing. The http_rewrite function will clear all output buffered so far and signaling VSPRT will make sure that whatever output is generated after http_rewrite goes unmodified to the user agent.

14.3.4. /INLINEFILE HTTP Server Pseudo-Directory

This pseudo directory provides a way for a VSP page to have full control over the data sent to the client user-agent, for example to send files to it and to handle the HTTP response attributes.

URIs starting with /INLINEFILE are handled through a VSP procedure instead of being searched for in the HTTP root directory. This special URI has the following syntax:

```
/INLINEFILE/some_file_name?VSP=some_vsp&arg1=x&arg2=y.....
```

Upon receipt of such a URI, Virtuoso will execute the "some_vsp" VSP page with the parameters following the VSP parameter. The VSP page can adjust the HTTP response header attributes using the `http_header()` function to reflect the content of the HTTP body, such as specifying the encoding through "Content-Encoding" attribute, or a MIME type through "Content-Type" attribute).

Example 14.9. Using /INLINEFILE

Here is a simple example for showing JPEG images stored in DAV. The page will list the first ten images found in the DAV resources table as hyper-links. Clicking on them will fetch the content and display using /INLINEFILE.

```
<?vsp
  if ({?'getfile'} <> '')
    {
      http_header ('Content-type: image/jpg\t\n');         -- set the header to jpg
      declare image any;
      select RES_CONTENT into image from WS..SYS_DAV_RES
        where RES_ID = atoi({?'id'});                      -- download image from WebDAV
      http(image);                                         -- table and display
      return;
    }
  ?>
<html>
 <body>
 <h1>Using /INLINEFILE to display images from the database</h1>
```

```
 <table>
  <tr>
   <td>

<?vsp for (select top 10 RES_ID, RES_NAME from WS..SYS_DAV_RES where right(RES_NAME, 3) = 'jpg') do {?>
 <a href="?id=<?=RES_ID?>"><?=RES_NAME?></a><br>
<?vsp } ?>

   </td>
   <td>
    <p><img src="/INLINEFILE/picture.jpg?VSP=<?/http_path()?>&getfile=yes&id=<?={?'id'}?>"></p>
    <p><a href="/INLINEFILE/picture.jpg?VSP=<?/http_path()?>&getfile=yes&id=<?/{?'id'}?>">download</a></p
   </td>
  </tr>
 </table>
 </body>
</html>
```

Note:

Note the use of the `http_path()` function to find the full path of the originating VSP file.

Also note the call to `http_header()` to set the appropriate content type for the returned data. When Virtuoso retrieves files normally, it will consult the system table `WS.WS.SYS_DAV_RES_TYPES` for types based on the file extension. We do not need to refer to the table in our example because we are making sure that only JPEGs are being returned.

See Also:

`http_header()`

`http_path()`

14.3.5. Beyond Basics

All output from VSP page procedures is buffered into a local string stream before being sent out. This is done so as to support the HTTP/1.1 required content length and to allow recovery from errors.

```
http_rewrite ( in stream any);
```

This clears any previous output to the stream. If the stream is omitted or 0 the stream is the HTTP client stream of the calling procedure.

```
http_file ( in path varchar);
```

This function sends a file to the HTTP client as a response to the current request. Any other output that may have been generated by the calling procedure will be discarded and the contents of the file will be the exclusive response to the current HTTP request together with appropriate headers. The file will only be sent after the procedure handling the current HTTP request has returned. The file name is thus not validated until the calling procedure has returned.

```
http_get( in uri varchar, out headers varchar);
```

This function retrieves the document specified by the URI by HTTP and returns the data of the response. The header output parameter is set to be an array with an element for each line of the response's HTTP header. Each line is a varchar object in the containing array. The header parameter is optional.

The URI is of the form `host[:port]<path>` , e.g '`www.openlinksw.com:80/index.html` '. The port, if omitted, defaults to 80. The data following the headers is not processed in any way. No content transfer encodings are decoded but an eventual content transfer encoding header can be found in the header array.

```
http_flush();
```

This function generates a response header, flushes a stream and disconnects the client, however, the server will continue with the execution of the PL code in VSP page. The final result will never be sent to the client. This is useful when a page makes a long transaction and we do not wish the client to wait until its end, which may result in a time-out. Suppose we have the retrieval of news from many targets, each a thousand messages. We start from a VSP page process, inside it we put into the internal stream

(using http, http_value etc.) redirect code and call http_flush. The client can read the response and go to a status page that can display number of messages retrieved, which may need refreshing a few times.

```
http_proxy (in host_and_port varchar, in header_lines any, in post_parameters any)
```

The function http_proxy() is used to send request in header_lines and post_parameters to the host_and_port, read the response and send it back to the client. The http_proxy() function can be used inside a VSP page to send a request to an external web server and automatically route the reply sent by this remote server to the client of the VSP page calling http_proxy(). The output which can generated (with http functions, etc.) before and after http_proxy is called will be discarded before sending the result of link retrieval to the user-agent.

Example 14.10. Virtuoso Proxy

```
...
<?vsp
  http ('this never be displayed');
  http_proxy ('www.foo.com', vector ('GET / HTTP/1.0'), NULL);
  http ('and this also');
?>
...
```

Virtuoso HTTP Server Character Set Settings

When the HTTP server returns the HTTP header to the client it appends charset=xxxx to the Content-Type: HTTP header fields. This informs the client user agent, the web browser, as to the character set of the content to be displayed correctly. It uses the Web server charset to correctly format values resulting from the `http_value()` function or the VSP equivalent <?= ...>. Wide values and XML entities, resulting from any XML processing functions like xpath_contains, get represented using the "HTML/XML transformation".

The default web server charset is governed by the *Charset* setting defined in the Virtuoso INI file . If no default charset is specified then Virtuoso will use ISO-8859-1.

The HTTP character set can be changed during an HTTP session using: *set http_charset='charset_name';* .

The XSLT output encoding can also be specified to over ride the server default setting.

Session Management and State Variables

The Virtuoso HTTP session management consists of functions for session variables manipulation and hooks for saving and restoring session variables.

Session management must be enable by setting the flag for persistent session variables in the virtual directory mapping. Virtual directory mappings use the *persistent_session_variables* flag, which when specified, session variables can be used in a post-process function to determine if the session variables content must be stored on to the session table or not.

The post-processing function hook can be any user-defined Virtuoso/PL procedure, it will be executed every time after processing of the active page.

See Also:

`connection_get()`

`connection_set()`

`connection_vars()`

`connection_vars_set()`

`connection_is_dirty()`

14.3.6. Long HTTP Transactions

Long running tasks may be invoked by web clients. In such a case, the server should return a page for the user agent much before the tasks's completion in order to provide feedback and avoid timeouts. Also the long running task should not be interrupted by the user agent disconnecting. The `http_flush()` function will send the reply accumulated thus far to the user agent and then disconnect it. In effect, this is a generic mechanism for starting an asynchronous thread.

Starting long running tasks may lead to denial of service. To prevent this we can use status and stop functions to check processes and kill them if necessary.

We can retrieve the URL, client IP-address, and process status code for all currently running VSP requests, then use this information to isolate and eliminate a process.

`http_pending_req()` Lists the processes.

http_kill() can be used to kill them.

14.3.7. Using chunked encoding in HTTP 1.1

It is sometimes desirable to use the HTTP 1.1 chunked encoding to send data to HTTP clients. Examples of such include status pages or streaming applications. Note that the possibility of using chunked encoding depends on external factors such as whether the client browser supports chunked encoding. So a server page can request turning on the chunked encoding, but should handle the case should it not be available.

Note that in order to successfully turn on the chunked encoding the page should not be using the `http_xslt()` . Also all reply headers set by the `http_header()` are silently ignored after the mode is set.

The `http_flush()` (with 1 as a value for it"s optional argument) is used to request turning on the chunked encoding. If the client"s user agent supports the encoding, then the data accumulated so far in the server"s output buffer are sent to the client as the first chunk, the request is put in special "chunked mode" and the `http_flush()` returns a non-zero integer. When in that mode a new chunk is sent to the client either when the internal 4k buffer is filled up or when the client calls `http_flush()` again to flush the buffer and send it as a chunk. The client disconnection is handled as usual .

14.3.8. Making Simple Dynamic Web Pages

The directory where the pages reside must be marked as executable. Use the vhost_define function or the Administration Interface to do this:

```
vhost_define (lpath=>'/example_location', ppath=>'/example_location/', vsp_user=>'demo');
```

The usual way to make an Dynamic page is to make a HTML skeleton and insert Virtuoso/PL code in appropriate places to fill in the rest dynamically. Consider the following example as a demonstration of such technique. Note that the example is made in four steps but in practice this can be one.

Suppose we have a table Demo.demo.Shippers (from the Demo database of the standard distribution). We can make a simple page for editing it.

1. **Building The HTML Skeleton.** We define one form for editing and adding entries and a table for listing the existing shippers.

```
<HTML>
  <BODY>

    <!-- edit form -->

    <FORM name="ShippersForm" method="POST" action="shippers.vsp">
       <INPUT type="hidden" name="ShipperID" value="">
       <TABLE>
         <TR>
           <TD>Company Name</TD>
           <TD><input type="text" name="CompanyName" value=""><!-- CompanyName -->

</TD>
```

```
          </TR>
          <TR>
            <TD>Phone</TD>
            <TD><INPUT type="text" name="Phone" value=""><!-- Phone number -->

</TD>
          </TR>
          <TR><TD colspan="2"><input type="submit" name="accept" value="Accept"></TD></TR>
        </TABLE>
    </FORM>
    <!-- end of form -->

    <!-- list of entries -->

    <TABLE>
       <TR><TD>Company Name</TD><TD>Phone #</TD><TD colspan="2">Action</TD></TR>
       <!--TR><TD>CompanyName</TD><TD>Phone</TD><TD>Edit URI</TD><TD>Delete URI</TD></TR-->

    </TABLE>
    <!-- end of list -->

  </BODY>
</HTML>
```

2. **Filling a List of Entries.** We have added in part 'list of entries' one active part (for select ...). And using <?=...?> shortcut we have made for each row in database table one row in table defined in step 1. Also we have added two useful links 'Edit' and 'Delete' with URL parameter 'EDIT' and 'DELETE' whose value is equal to the primary key value of the row;

```
<HTML>
  <BODY>

    <!-- edit form -->

    <FORM name="ShippersForm" method="POST" action="shippers.vsp">
       <INPUT type="hidden" name="ShipperID" value="">
       <TABLE>
         <TR>
           <TD>Company Name</TD>
           <TD><input type="text" name="CompanyName" value=""><!-- CompanyName -->

</TD>
         </TR>
         <TR>
           <TD>Phone</TD>
           <TD><INPUT type="text" name="Phone" value=""><!-- Phone number -->

</TD>
         </TR>
         <TR><TD colspan="2"><input type="submit" name="accept" value="Accept"></TD></TR>
       </TABLE>
    </FORM>
    <!-- end of form -->

    <!-- list of entries -->

    <TABLE>
       <TR><TD>Company Name</TD><TD>Phone #</TD><TD colspan="2">Action</TD></TR>
       <!--TR><TD>CompanyName</TD><TD>Phone</TD><TD>Edit URI</TD><TD>Delete URI</TD></TR-->

       <?vsp
         for select ShipperID, CompanyName, Phone from Demo.demo.Shippers do
          {
       ?>
       <TR>
         <TD><?=CompanyName?></TD>
         <TD><?=Phone?></TD>
         <TD><a href="shippers.vsp?EDIT=<?=ShipperID?>">Edit</a></TD>
         <TD><a href="shippers.vsp?DELETE=<?=ShipperID?>">Delete</a></TD>
       </TR>
       <?vsp
          }
```

```
      ?>
    </TABLE>
    <!-- end of list -->

  </BODY>
</HTML>
```

3. **Retrieving Parameters and Filling in The Form.** We must retrieve the parameters 'Delete' and 'Edit' from the URL and fill in the form elements

```
<?vsp
declare company_name, phone_number varchar;
declare shipper_id, operation integer;
declare shipper_info
  cursor for select CompanyName, Phone
    from Demo.demo.Shippers where ShipperID = shipper_id;

company_name := '';
phone_number := '';
operation := 0;

shipper_id := atoi (get_keyword ('EDIT', params, '0'));

if (shipper_id > 0)
  {
    whenever not found goto not_found_any;
    open shipper_info (prefetch 1);
    fetch shipper_info into company_name, phone_number;
not_found_any:
    close shipper_info;
    goto display_page;
  }

shipper_id := atoi (get_keyword ('DELETE', params, '0'));

if (shipper_id > 0)
  {
    delete from Demo.demo.Shippers where ShipperID = shipper_id;
    goto display_page;
  }

display_page:

?>

<HTML>
  <BODY>

    <!-- edit form -->

    <FORM name="ShippersForm" method="POST" action="shippers.vsp">
       <INPUT type="hidden" name="ShipperID" value="<?=shipper_id?>">
       <TABLE>
         <TR>
           <TD>Company Name</TD>
           <TD><input type="text" name="CompanyName" value="<?=company_name?>"</TD>
         </TR>
         <TR>
           <TD>Phone</TD>
           <TD><INPUT type="text" name="Phone" value="<?=phone_number?>"</TD>
         </TR>
         <TR><TD colspan="2"><input type="submit" name="accept" value="Accept"></TD></TR>
       </TABLE>
    </FORM>
    <!-- end of form -->

    <!-- list of entries -->

    <TABLE>
       <TR><TD>Company Name</TD><TD>Phone #</TD><TD colspan="2">Action</TD></TR>
       <!--TR><TD>CompanyName</TD><TD>Phone</TD><TD>Edit URI</TD><TD>Delete URI</TD></TR-->

       <?vsp
```

```
          for select ShipperID, CompanyName, Phone from Demo.demo.Shippers do
           {
        ?>
        <TR>
          <TD><?=CompanyName?></TD>
          <TD><?=Phone?></TD>
          <TD><a href="shippers.vsp?EDIT=<?=ShipperID?>">Edit</a></TD>
          <TD><a href="shippers.vsp?DELETE=<?=ShipperID?>">Delete</a></TD>
        </TR>
        <?vsp
           }
        ?>
    </TABLE>
    <!-- end of list -->

  </BODY>
</HTML>
```

We have added a section to the top of the page that we use to recognize what operation to perform and do the appropriate action in the database. In the case of editing we use the values input on the form.

4. **Editing Form Logic.** We have a logic in the same initialization part of the page that retrieves a 'ShipperID' parameter, so if this parameter exists in the params array then the operation is to perform an insert, otherwise we must update a record.

```
<?vsp
declare company_name, phone_number varchar;
declare shipper_id integer;
declare shipper_info
  cursor for select CompanyName, Phone
    from Demo.demo.Shippers where ShipperID = shipper_id;

company_name := '';
phone_number := '';

shipper_id := atoi (get_keyword ('EDIT', params, '0'));
-- If the current operation is edit then we retrieve the Company Name and Phone
if (shipper_id > 0)
  {
    whenever not found goto not_found_any;
    open shipper_info (prefetch 1);
    fetch shipper_info into company_name, phone_number;
not_found_any:
    close shipper_info;
    goto display_page;
  }

shipper_id := atoi (get_keyword ('DELETE', params, '0'));

-- If the operation is delete then delete it (if any error
-- occurred then HTTP server will handle it)
if (shipper_id > 0)
  {
    delete from Demo.demo.Shippers where ShipperID = shipper_id;
    shipper_id := 0; -- reset it to prevent submission of wrong shipper id
    goto display_page;
  }

-- If pressed button Accept then we can recognize the operation
if ('' <> get_keyword ('accept', params, ''))
  {
    shipper_id := atoi (get_keyword ('ShipperID', params, '0'));
    company_name := get_keyword ('CompanyName', params, 'unknown');
    phone_number := get_keyword ('Phone', params, 'N/A');
    -- the old entry going edited
    if (shipper_id > 0)
        update Demo.demo.Shippers set CompanyName = company_name, Phone = phone_number
            where ShipperID = shipper_id;
    else -- this is an new entry
      {
        shipper_id := coalesce ((select max(ShipperID) from Demo.demo.Shippers), 0) + 1;
        insert into Demo.demo.Shippers (ShipperID,CompanyName,Phone)
```

```
            values (shipper_id,company_name,phone_number);
        }
      company_name := ''; -- we clear all entered data
      phone_number := '';
      shipper_id := 0; -- to prevent setting of wrong shipper id in form
    }

display_page:

?>

<HTML>
  <BODY>
    <H3>Editing a shipment companies</H3>
    <!-- edit form -->

    <FORM name="ShippersForm" method="POST" action="shippers.vsp">
       <INPUT type="hidden" name="ShipperID" value="<?=shipper_id?>">
       <TABLE>
         <TR>
           <TD>Company Name</TD>
           <TD><input type="text" name="CompanyName" value="<?=company_name?>"</TD>
         </TR>
         <TR>
           <TD>Phone</TD>
           <TD><INPUT type="text" name="Phone" value="<?=phone_number?>"</TD>
         </TR>
         <TR><TD colspan="2"><input type="submit" name="accept" value="Accept"></TD></TR>
       </TABLE>
    </FORM>
    <!-- end of form -->

    <!-- list of entries -->

    <TABLE>
       <TR><TD>Company Name</TD><TD>Phone #</TD><TD colspan="2">Action</TD></TR>
       <!--TR><TD>CompanyName</TD><TD>Phone</TD><TD>Edit URI</TD><TD>Delete URI</TD></TR-->

       <?vsp
         for select ShipperID, CompanyName, Phone from Demo.demo.Shippers do
          {
       ?>
       <TR>
         <TD><?=CompanyName?></TD>
         <TD><?=Phone?></TD>
         <TD><a href="shippers.vsp?EDIT=<?=ShipperID?>">Edit</a></TD>
         <TD><a href="shippers.vsp?DELETE=<?=ShipperID?>">Delete</a></TD>
       </TR>
       <?vsp
          }
       ?>
    </TABLE>
    <!-- end of list -->

  </BODY>
</HTML>
```

5. **Final Remarks.** The page for shippers does not have any specific error-handling. If there are any SQL errors produced the HTTP server will display the error number and the server error message. For example, if we trying to delete an record of a predefined shippers we would have a foreign key violation to which the server would respond:

```
'SQL Error S1000 DELETE statement conflicted with
  COLUMN REFERENCE constraint "Orders_Shippers_ShipVia_ShipperID"'
```

We can, however, add PL code to handle errors and display a different page.

14.3.9. Generation of non-HTML output

VSP pages are not restricted to generating only HTML markup. In order to generate non-HTML markup the VSP page MUST conform to the target format specification, for example if XML output is needed, the markup MUST follow XML syntax rules.

The output sent to the user agent is entirely controlled by the scripting on the page, no constant headers beyond the required HTTP ones are added by the serber.

Also it is important when output is not HTML that the HTTP header field 'Content-Type' be set to appropriate MIME type (for example 'text/xml', 'text/plain' etc) using `http_header()` .

Example 14.11. XML generation

The following example shows how an XML document can be generated using a VSP page

```
<?vsp
    -- source of the xmldemo.vsp
    declare txt varchar;
    http_header ('Content-Type: text/xml\r\n');
    txt := 'this is a XML test';
?><?xml version="1.0" ?>
<document>
    <para><?= txt ?></para>
</document>
```

Example 14.12. WML generation

The following example shows how a WML page can be generated using a VSP page

```
<?vsp
    -- source of the wmldemo.vsp
    declare txt varchar;
    http_header ('Content-type: text/vnd.wap.wml\r\n');
    txt := 'this is a WML test';
?><?xml version="1.0" ?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN" "http://www.wapforum.org/DTD/wml_1.1.xml"
<wml>
    <card>
        <p><?= txt ?></p>
    </card>
</wml>
```

See Also:
WML Tutorials

14.3.10. Post VSP XSLT Transformation Mode

The Virtuoso server can perform server-side XSLT transformations on the internal stream data. Suppose we have an XML document, we can tell the HTTP server to invoke XSLT processor before sending the reply to the user agent. We do this by making a call to the http_xslt() function inside VSP or any procedure called from it and pass the URL of the stylesheet and probably some parameters.

`http_xslt(in` *`style_sheet_URI`* `varchar [, in` *`parameters`* `any])`

The VSP page will produce an XML document, rather than HTML. The XSL stylesheet will then produce the required output for the user-agent, which could be more XML or HTML.

The XSL stylesheet could also be generated using VSP.

Example 14.13. In Server Transformation of XML

```
<?vsp
declare ses any;
ses := string_output ();
xml_auto ('select ShipperID, CompanyName, Phone from Demo.demo.Shippers
```

```
  for xml auto element', '', ses);

http ('<list>');
http (string_output_string (ses));
http ('</list>');
http_xslt ('file:/samples/xslt/shippers.xsl');
?>
```

Where shippers.xsl will have the following content:

```
<?xml version="1.0"?>

<xsl:stylesheet
    version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:template match="/">
<html>
<head>
<title>Shippers list</title>
</head>
<body link="#0000b4" vlink="#0000b4" bgcolor="#ffffff">
<xsl:apply-templates/>
</body>
</html>
</xsl:template>

  <xsl:template match="list">
<table>
<tr><td>ID</td><td>Name</td><td>Phone</td></tr>
<xsl:apply-templates/>
</table>
</xsl:template>

  <xsl:template match="Shippers">
<tr>
<td><xsl:value-of select="ShipperID"/></td>
<td><xsl:value-of select="CompanyName"/></td>
<td><xsl:value-of select="Phone"/></td>
</tr>
</xsl:template>
</xsl:stylesheet>
```

14.3.11. XML & XSLT Generated VSP Pages

The Virtuoso HTTP server provides a special case for producing VSP pages. For this purpose the extensions .vxml and .vxsl are reserved. If the requested document is named dummy.vxml then the HTTP server will try to locate dummy.vxsl. If the dummy.vxsl exists then it will perform an XSLT transformation using .vxml as a target XML document and .vxsl as a XSLT stylesheet. After successful transformation it will compile a result from it in the usual way and then perform execute.

The requirements are: XML document with extension .vxml, XSLT sheet with the same name, location but extension .vxsl. The location of these files must be set to allow execution (see Virtual Directories Support and Execution Privileges).

Example 14.14. Executable (V)XSL

Source of portfolio.vxml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="portfolio.xsl"?>
<portfolio xmlns:dt="urn:schemas-microsoft-com:datatypes" xml:space="preserve">
<stock exchange="nyse">
<name>zacx corp</name>
<symbol>ZCXM</symbol>
<price dt:dt="number">28.875</price>
</stock>
<stock exchange="nasdaq">
```

```
<name>zaffymat inc</name>
<symbol>ZFFX</symbol>
<price dt:dt="number">92.250</price>
</stock>
<stock exchange="nasdaq">
<name>zysmergy inc</name>
<symbol>ZYSZ</symbol>
<price dt:dt="number">20.313</price>
</stock>
</portfolio>
```

Source of portfolio.vxsl

```
<?xml version='1.0'?>

<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/TR/WD-xsl">

  <xsl:template match="/">
<HTML>
<BODY>
<TABLE BORDER="2">
<TR>
<TD>Symbol</TD>
<TD>Name</TD>
<TD>Price</TD>
</TR>
<xsl:for-each select="portfolio/stock">
<TR>
<TD><xsl:value-of select="symbol"/></TD>
<TD><xsl:value-of select="name"/></TD>
<TD><xsl:value-of select="price"/></TD>
</TR>
</xsl:for-each>
</TABLE>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```

The result of retrieving portfolio.vxml

```
<HTML>
<BODY>
<TABLE BORDER="2">
<TR><TD>Symbol</TD><TD>Name</TD><TD>Price</TD></TR>
<TR><TD>ZCXM</TD><TD>zacx corp</TD><TD>28.875</TD></TR>
<TR><TD>ZFFX</TD><TD>zaffymat inc</TD><TD>92.250</TD></TR>
<TR><TD>ZYSZ</TD><TD>zysmergy inc</TD><TD>20.313</TD></TR>
</TABLE>
</BODY>
</HTML>
```

14.4. Virtuoso Server Pages for XML (VSPX)

VSPX is an XML vocabulary for server generated HTML and XHTML pages that may or may not be bound to native or third-party data sources. VSPX is a framework for building and deploying dynamic web content atop Virtuoso. VSPX offers a widget set and event model which is similar to that of GUI tool kits, providing the developer with much automation for common web development tasks. Many controls are tightly integrated with the Virtuoso database, providing seamless data binding to local or remote relational and XML data. Server side scripting in VSPX is performed in Virtuoso/PL. The VSPX source code consists of HTML or XML with interspersed VSPX specific XML elements which describe the UI control hierarchy of the page.

VSPX covers the following areas:

Rich set of controls, covering all basic HTML plus complex composite controls like data grids and tree browsers.

Session management - Transparent session and session state handling using cookies, URL poisoning or digest authentication.

Form entry validation server and client side.

Single and multi-row controls for viewing and directly updating SQL tables, including scrolling through long result sets.

Repeating and conditional instantiation of UI elements based on run time choices.

Event model for HTTP request handling, providing distinct phases for creating the controls, retrieving data, processing posted data, preserving state and rendering for a user agent.

Object oriented architecture, allowing easy definition of subclasses of existing controls for new functionality. It suffices to implement a new SQL class with a few predefined methods and to implement an XSLT rule for generating the code based on the external XML syntax for the new control.

Easy automatic generation of VSPX pages. The pages being XML, it is simple to automate generating pages based on other data, such as automatically making table maintenance pages based on a SQL table definition.

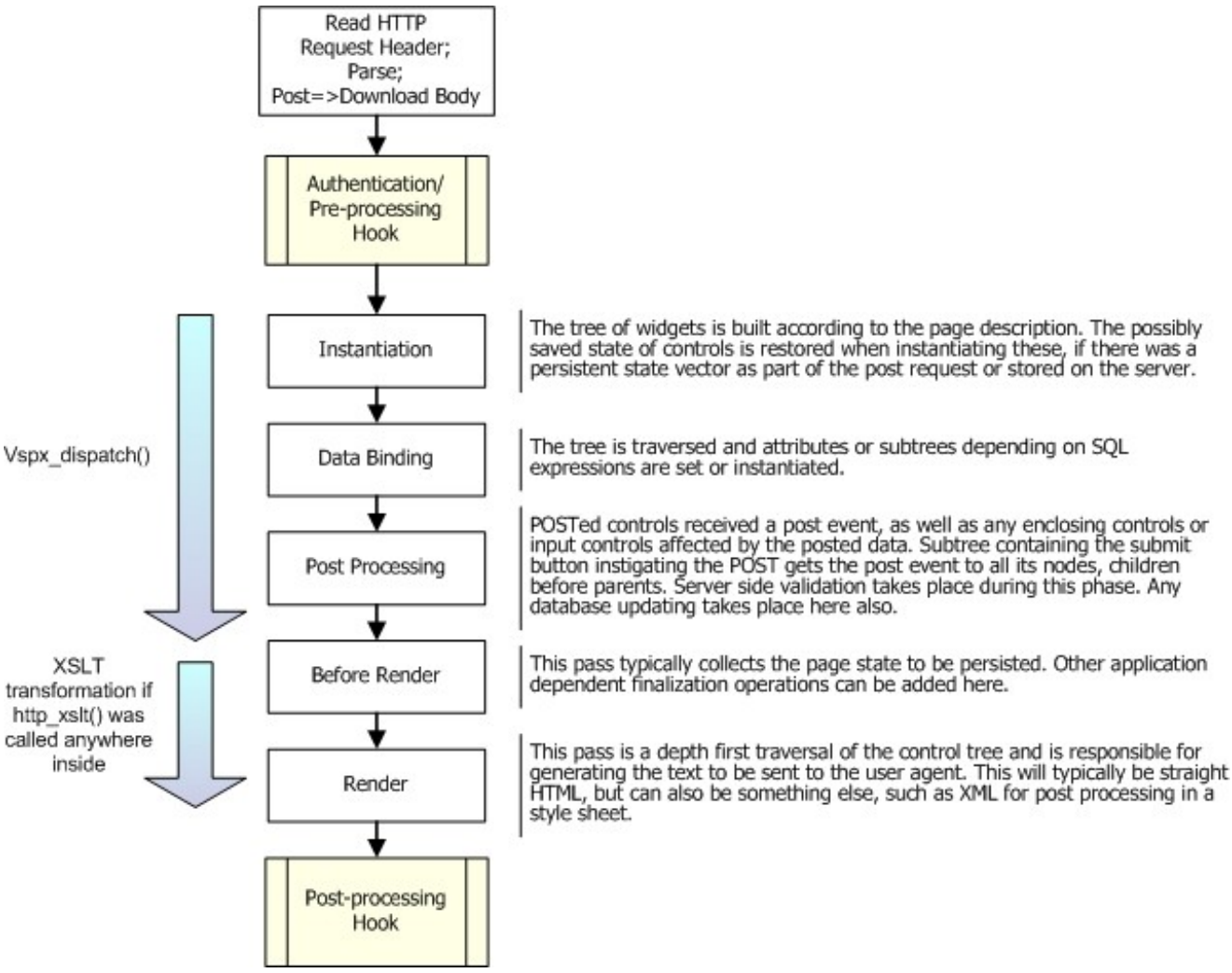
The VSPX development cycle consists of editing .vspx resources in the file system or Virtuoso DAV. The editing can take place using a regular text editor or a supporting HTML editor.

14.4.1. Processing Model

A VSPX page describes a web page in terms of static XHTML plus XML elements in the VSPX namespace, `"http://example.com/vspx/"` . This namespace is abbreviated as `v:` in the rest of this document.

Elements in the `v` namespace introduce VSPX elements, options or controls. Some of these may in turn have HTML children. VSPX elements with HTML children are called templates, as these will process their HTML contents at run time, typically modifying these based on run time data.

Figure 14.15. VSPX Conceptual Diagram



When the page is requested, the system checks whether it is already compiled and compiles it if the compilation is absent or older than the source. The VSPX compilation has two phases: pre-processing and compilation. The first phase expands included files

and applies the external macro XSL-T sheet. The result of which is a single page encapsulating all related components which will be stored in a .vspx-m intermediary file. The result of second phase, compilation is a single .vspx-sql file containing class and method definitions for a subclass of the generic VSPX page class. All code directly derived from the pre-processed page will be found in this file. The file can of course refer to outside Virtuoso/PL code.

The results of compilation process are stored usually in an OS dependent temporary directory. This would be the $TMPDIR for UNIXes or %TMP% for Windows platforms. If these environment variables are not available it will be some default system specific location, such as `/var/tmp` or `/tmp` on Unix's. Note that this temporary storage applies to the VSPX pages that are stored within the file system, for the WebDAV repository the product of compilation is stored as described below. For development purposes the use of temporary storage can be turned off by executing:

```
registry_set ('__no_vspx_temp', '1')
```

from ISQL. In this case both file-system and WebDAV repository will contain .vspx-m and .vspx-sql files in the same place and with the same name as the VSPX source file. VSPX temporary storage can be re-enabled in the same way but using the string value '0' instead of '1'. Note that this is a string rather than a number.

Any VSPX page invocation, whether through the GET or POST HTTP request, consists of the following steps:

- **Instantiation.** The tree of widgets is built according to the page description. The possibly saved state of controls is restored when instantiating these, if there was a persistent state vector as part of the post request or stored on the server.
- **Data Binding.** The tree is traversed and attributes or subtrees depending on SQL expressions are set or instantiated.
- **Post Processing.** If this was a POST request, the control that was mentioned in the POST gets a post event, as well as any enclosing controls or input controls affected by the posted data. The subtree containing the submit button originating the POST gets the post event to all its nodes, children before parents. Post data server side validation takes place during this phase. Any database updating takes place during this processing, typically inside the post handler of the form element, after the post handling of each individual field is complete.
- **Before Render.** The control tree is now assumed to be in a state reflecting the operation intended by the POST or GET. This pass typically collects the page state to be persisted. Other application dependent finalization operations can be added here.
- **Render.** This pass is a depth first traversal of the control tree and is responsible for generating the text to be sent to the user agent. This will typically be straight HTML, but can also be something else, such as XML for post processing in a style sheet.

Just as with VSP pages the code of the page make call http_xslt(), this has the effect of applying the specific stylesheet to the HTML text produced by the render phase. Since output contains HTML tags generated by VSPX controls, the style-sheet should have these as a general rule to leave these unchanged. The http_xslt () is more useful with VSP pages producing XML than with VSPX pages.

14.4.2. Object Model

VSPX controls are SQL user defined type instances, or objects. The SQL object system is substantially similar to any other single inheritance object system, such as that of Java or C#. The XML source code of the VSPX page is processed to generate SQL for instantiating the VSPX control tree. The XML elements and classes may have similar names but are not one to one identical.

All controls, including pages, are instances of a subclass of `vspx_control` . To each VSPX source file corresponds a class named after the file, which inherits the common superclass `vspx_page` . Normally, all code, regardless of which specific control on the page it pertains to, runs inside a method of the page, thus the *self* (like 'this' in Java or C++) refers to the page instance. By convention, the variable *control* references the object representing the control which declares the particular code snippet. The variable `e` represents the event which is at the root of the code being invoked. The event is normally an event object representing a GET or POST HTTP request, but it can also be a user defined event sent to the page by other code on the page or elsewhere.

The page has a data member `vc_children` which contains an array of all top level VSPX controls on the page. Each control in turn has this same variable for referring to subordinate controls, if any.

Each processing pass allows the developer to specify arbitrary SQL code to run in the context of the pass. The code runs in the context of a method of the page object, where the local variable *control* references the control that declares the code in question.

Controls may be enabled or disabled, thus supporting conditional activation and rendering of page parts. Parts may be enabled in function of a page state, for example a search result control is only on after the search text has been posted. Another example is that some controls may only be enabled if a user is logged in.

Code inside a control may locate other controls on the page by name, either starting from the control itself or at the page. Note that when repeating control groups are involved, it is best to start at the leaf and work upwards in the tree, since there can be several identically named children of the page. All controls have a name which is unique in terms of the VSPX source file. Additionally, all controls, including members of repeating groups, have a name which is generated to be unique at run time. The latter is not however constant. The `vc_parent` data member references the parent. `vc_name` is the name as appears in the name attribute of the corresponding XML element. `vc_instance_name` is a page-wide unique name, generated to be different for each possible repetition of the control. The `find_control` method of `vspx_control` can be used to look for a child with a specific `vc_name` .

14.4.3. Keeping Page and Session State

VSPX does not require any particular session management to be in effect to operate. It is in its default form entirely stateless on the server side. There is a mechanism called view state for keeping track of data between consecutive post backs of a single page. This is simply an automatically maintained hidden field which will keep the state of controls which have a state that may be persisted. The view state mechanism does not however safeguard data across different pages but is fine for remembering entered form data or a bookmark in a scrolled list across page reloads.

To keep real sessions and session variables, a `v:login` control is offered.

14.4.4. Application Code

Most VSPX controls will support XML children specifying SQL code to run at various points of the page processing cycle. The elements are:

v:before-data-bind

v:after-data-bind

v:before-post

v:before-render

These elements will have a script tag as unique child, most often containing a CDATA section for escaping the SQL text.

Additionally, most attributes of VSPX elements can have a SQL expression evaluated at initialization, pre data bind or after data bind time, as the case may be, depending on the attribute and its value. An attribute value beginning with `"--"` indicates evaluation on the initialization or the descending edge of the data bind pass. An attribute value indicating `"--after"` specifies that the value is calculated on the returning edge of recursion of the data bind pass.

Arbitrary HTML text may contain attributes with a data bound value. The attributes should be in the XHTML namespace and have a text beginning with `"--"` . The expression will be evaluated at render time, where the *control* variable refers to the enclosing VSPX template control, e.g. the page instance when at top level.

VSP Compatibility and In-Line SQL

Arbitrary HTML text may contain <?vsp ?>, <?V ?> (equivalent of <?= VSP notation) and <?U ?> (equivalent of <?/ VSP notation) processing instructions. The code in question will be evaluated during the render pass, with control set to the closest enclosing VSPX control. Please note that the <?= and <?/ VSP shortcuts cannot be used inside VSPX pages as they are not valid processing-instructions.

VSPX maintains backward compatibility with VSP through supporting the same processing instructions, but additionally requires the page source to be well formed XML. VSP pages which are well formed in XML terms will run as such under with the VSPX extension, with the addition of the `v:page` top level element.

The <?vsp ?> processing instruction expects a SQL statement. The statement can be a compound statement ending in another <?vsp ?> processing instruction at the same nesting level under the same parent tag. The other pi's <?V and <?U expect a SQL expression.

Note:

<?V ?> is not permitted inside attributes without quotation, as this is not well formed XML. When quoted, this notation in attributes has no special effect, hence the convention about the leading `"--"` and the `XHTML` namespace for HTML attributes with a computed value.

14.4.5. A Simple Example

Example 14.15. Simple VSPX Page

The fragment below shows us a VSPX page with a few labels. A label is a simple control that renders as HTML text, using a data bindable attribute to specify a value and a C printf style format string for the format. Note the use of the <?vsp ?> processing instruction. This could just as well have been an HTML literal.

```
<html>
  <body>
    <v:page name="demo_label" xmlns:v="http://example.com/vspx/">
      <p>
        <?vsp http ('<H3>Simple page to demonstrate simple VSPX controls</H3>'); ?>
      </p>
      <p> An integer <v:label name="label1" value="--(1 + 2)" format="--'%d'"/> </p>
      <p> A string <v:label name="label2" value="'123'" format="%s"/> </p>
      <p> A string with default format <v:label name="label3" value="String"/> </p>
      <p>
      An url <v:url name="url1" value="--'OpenLink Software Ltd.'" format="%s" url="--'http://openlinksw.
      <br/>
      An url (default format) <v:url name="url2" value="OpenLink Software Ltd." url="http://openlinksw.co
      </p>
      <v:include url="footer.xml"/>
    </v:page>
  </body>
</html>
```

14.4.6. VSPX Event Handler Parameters

The user-defined event handlers always accept a 'control' parameter whose type is the same as that of the control it belongs to. For example the on-post event of 'button' has a parameter 'control' of type vspx_button; after-data-bind of a 'label' has a parameter 'control' of type 'vspx_label' etc.

The event handlers for before-data-bind, after-data-bind and on-post have an additional parameter: 'e' which is of type vspx_event. The parameter ('e') keeps the HTTP request as post data, request header and path just like the global variables 'path', 'params' and 'lines' on a VSP page. In addition to that, the vspx_event (e) keeps a reference to the button pressed (if any) and a flag indicating the request type: POST or GET. It may thus be used in a complex form to detect when to trigger an event depending of event data or to directly access parameters of the HTTP request, etc.

```
create type vspx_event
as (
    ve_params any,              -- name value pairs of post data
    ve_lines  any,              -- HTTP header lines
    ve_path   any,              -- requested path , parsed as vector
    ve_button vspx_control,     -- which active control originated the event.
    ve_is_post int default 0    -- 0 for GET 1 for POST
   )  temporary self as ref
;
```

14.4.7. Registering a VSPX Event Callbacks

The event handlers mechanism can be extended with callbacks. This means registering a page method(s) which will be invoked after event scrips are processed. The page methods used in this case can be either defined using 'method' declaration inside VSPX page or using page subclass defined in code-behind script(s) (see 'code-file' below). To register a callback vc_add_handler (name, method_name) method must be invoked in any script preceding the handler stage. Parameters to the vc_add_handler are: name - name of event where callback to be executed (before-data-bind, after-data-bind, on-post, before-render), method_name - name of a page class method to be registered to the event.

```
<html>
    <body>
        <v:page name="handler_demo" xmlns:v="http://example.com/vspx/">
            <v:form name="sf" type="simple" action="form.vspx" method="POST">
                <v:text name="txt" />
                <v:button name="submit2" action="simple" value="OK" />
                <v:on-init>
```

```
                    control.vc_add_handler ('on-post', 'user_post');
                </v:on-init>
            </v:form>
            <v:method name="user_post" arglist="inout control vspx_control">
                dbg_printf ('Invoked: user post method');
            </v:method>
        </v:page>
    </body>
</html>
```

14.4.8. Commonly Used Types of Attributes of VSPX Controls

CalculateableValue

CalculateableValue — The type of attribute that contains an expression to be calculated at run time.

Description

The type of a string constant or an expression. If the value of this type is started with '--', the rest of string is the text of an expression to be calculated in order to get the effective value of the property. If the value of this type is not started with '--', the effective value is the original string itself.

The type identifier 'CalculateableValue' is introduced only for diagnostic purposes, you will never use it in VSPX code. When Virtuoso server tries to compile an invalid VSPX page, you might see a diagnostic messages like 'the value of attribute X of a control Y does not match pattern ... for type CalculateableValue'. If you see this then you should check the syntax of the value of the specified attribute.

ForcedCalculateableValue

ForcedCalculateableValue — The type of attribute that contains an expression to be calculated at run time.

Description

The type of an expression. The value of this type must be started with '--', the rest of string is the text of an expression to be calculated in order to get the effective value of the property.

The type identifier 'ForcedCalculateableValue' is introduced only for diagnostic purposes, you will never use it in VSPX code. When Virtuoso server tries to compile an invalid VSPX page, you might see a diagnostic messages like 'the value of attribute X of a control Y does not match pattern ... for type ForcedCalculateableValue'. If you see this then you should check the syntax of the value of the specified attribute.

SqlCode

SqlCode — The type of a string value that is a Text of a Virtuoso/PL procedure.

Description

Text of an event handler or other Virtuoso/PL procedure. This is expected to be a complete statement or sequence of statements, hence to be terminated like a statement, unlike the 'calculatable value' expression case.

The type identifier 'SqlCode' is introduced only for diagnostic purposes, you will never use it in VSPX code. When Virtuoso server tries to compile an invalid VSPX page, you might see a diagnostic messages like 'the value of attribute X of a control Y does not match pattern ... for type SqlCode'. If you see this then you should check the syntax of the value of the specified attribute.

SqlName

SqlName — The type of attribute that contains a name of the Virtuoso/PL UDT instance that should be created for the control.

Description

The type for a Virtuoso/PL name that should be used during code generation. KNOWN BUG: If double quotes are used in the value of this type then the generated code may be incorrect. This is why this schema restrictes the syntax of values of such type.

The type identifier 'SqlName' is introduced only for diagnostic purposes, you will never use it in VSPX code. When Virtuoso server tries to compile an invalid VSPX page, you might see a diagnostic messages like 'the value of attribute X of a control Y does not match pattern ... for type SqlName'. If you see this then you should check the syntax of the value of the specified attribute.

14.4.9. VSPX Controls

after-data-bind

after-data-bind — Container for code that should be executed after data bind.

Synopsis

```
<after-data-bind >(EventHandler)</after-data-bind >(    );
```

Description

The code contained in this element is executed after the default data bind processing on the returning edge of recursion. The context has self as the page, control as the control with this handler, event as the vspx_event which caused the page invocation. Any return value from the code is ignored.

after-data-bind-container

after-data-bind-container — This is identical to v:after-data-bind.

Synopsis

```
<after-data-bind-container >(EventHandler)</after-data-bind-container >(    );
```

Description

before-data-bind

before-data-bind — Container for code that should be executed before data bind.

Synopsis

```
<before-data-bind >(EventHandler)</before-data-bind >(    );
```

Description

The code contained in this element is executed before the default data bind processing on the descending edge of recursion. The context has self as the page, control as the control with this handler, event as the vspx_event which caused the page invocation. A return value from the code if equal to 1 (one) will stop further data-bind processing of the containing control and it's children if any.

before-data-bind-container

before-data-bind-container — This is identical to v:before-data-bind.

Synopsis

```
<before-data-bind-container >(EventHandler)</before-data-bind-container >(    );
```

Description

before-render

before-render — Container for code that should be executed before rendering.

Synopsis

```
<before-render >(EventHandler)</before-render >(    );
```

Description

The code contained in this element is executed on the descending edge of recursion when traversing the vspx control tree before rendering. This is expected to have side effects on vspx controls only, not to return anything or emit any output. Any return value from the code is ignored.

before-render-container

before-render-container — This is identical to v:before-render.

Synopsis

```
<before-render-container >(EventHandler)</before-render-container >(    );
```

Description

button

button — Scriptable button.

Synopsis

<**button** /> (
name (required) ,
annotation (optional) ,
initial-enable (optional) ,
enabled (optional) ,
instantiate (optional) ,
control-udt (optional) ,
xsd-stub-xhtml (optional) ,
width (optional) ,
height (optional) ,
column (optional) ,
null-value (optional) ,
action (required) ,
value (required) ,
style (optional) ,
active (optional) ,
initial-active (optional) ,
child-window-options (optional) ,
browser-current (optional) ,
browser-filter (optional) ,
browser-list (optional) ,
browser-mode (optional) ,
browser-type (optional) ,
browser-xfer (optional) ,
selector (optional) ,
format (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

Scriptable version of Submit Button of the HTML form. Depending of 'action' attribute it may have variants. In some of these variants the button will not have a submit function, as in select and browse buttons. In these cases the button will use client JavaScript to pop up new windows or for setting values in other windows.

Attributes

name = SqlName . A page level unique name identifying a control.

annotation. A human readable comment.

initial-enable = CalculateableValue . Determines whether a control is initially visible. True by default. Could be data-bound to an SQL expression.

enabled = CalculateableValue . Determines whether a control is visible. True by default. Could be data-bound to an SQL expression.

instantiate = CalculateableValue . Determines whether a control instantiate its children. It is true by default. It could be data-bound to an SQL expression. Unlike most of calcucateable attributes, the value of this attribute for a control is calcluated before calling 'on-init' event handler of the control; other values are calculated before calling 'before-data-bind' event handler.

control-udt = SqlName . At run time every control is represented as an instance of some user-defined type (UDT). VSPX compiles selects the UDT to use depending on name of the XML element that represents the control and maybe some of its attributes. In some specific cases the application developer may instruct VSPX compiler to use some other UDT, e.g. an application-specific UDT that is derived from the preset one. If specified, the value of 'control-udt' attribute should be equal to the name of the desired target UDT.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

column = SqlName . The name of the column bound. If nothing else is specified, the column meta data from the containing update form sets the field's attributes.

null-value. This value will be shown if value of the column is NULL, also value of the control will be set to null if this value is passed upon POST request.

action. This specifies the button subclass to use.

Table 14.1. Allowed values of the 'action' attribute

simple	is a simple submit button, no special functions
submit	an alias of 'simple' button, obsolete
delete	a button for deleting a row in a data-set or data-grid control. This is represented as a submit button with a special on-post script. The delete function is performed based on table, key attributes. (check implementation!!)
browse	Button that opens pup-up window using client-side javascript (browse-button). This control allows the designer to place some part of a form's input into a pop-up window. When the HTML page is rendered, this control puts a button and associated client-side javascript into the resulting page so clicking the button opens a child window.
return	This is to be used on a page invoked for selecting from a browse button. Specifies that the value of the fields designated by the field children are to be asigned to the corresponding fields of the invoking page.
logout	This will terminate the current session when clicked. Can only occur inside the login control.

value = CalculateableValue . Text of the label of the button.

style. A style of button, affects appearance.

Table 14.2. Allowed values of the 'style' attribute

submit	This is a default style used, no special handling. The button will be rendered as a usual submit button.
url	The button will be rendered as a link, furthermore client side JavaScript code will be produced to act as submit button.
image	The button will be rendered as an image on the browser under link button. In that case value of button control must be link to the image file.

active = CalculateableValue . A data bound value to enable or disable the button. The default is '1' meaning 'enable'.

initial-active = CalculateableValue . A data bound value to enable or disable the button. The default is '1' meaning 'enable'. Unlike 'active' attribute, this one is effective only when the page is displayed in the first time, not after user posts data back to the page.

child-window-options. Options for oppening a child pop-up window.This can be used only in conjuction with action browse.

browser-current. Current directory of browsing. Can be used only with browse button for WebDAV or File system resources.

browser-filter. Filter expression(s) for browsing. (see above)

browser-list. Type of browser list, 0 - short, 1 - long

browser-mode. COL, RES or STANDALONE , for a file system or DAV browse button, this specifies whether to return an path name or internal ID.

browser-type. This designates the type of browser: dav or OS (filesystem).

browser-xfer. For a browse button, this is the means of transferring the data between windows, DOM is default. No other options in current implementation.

selector. For a browse button this is the URL that will be loaded into the pop up window.

format = CalculateableValue . A sprintf format string for printing the value

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Declaration of type vspx_button

Button class, encapsulate all controls originating a event

```
create type vspx_button under vspx_field as
(
  bt_pressed int default 0,
  bt_style  varchar default 'submit',
  bt_close_img varchar,
  bt_open_img varchar,
  bt_url varchar default '',
  bt_l_pars varchar default '',
  bt_text varchar default '',
  bt_anchor int default 0
)  temporary self as ref
overriding method vc_render () returns any,
overriding method vc_xrender () returns any,
overriding method vc_set_model () returns any,
constructor method vspx_button (name varchar, parent vspx_control)
```

Examples

Example 14.16. Buton shown as an image

The button of the form will be shown as a picture plus.gif. Attributes whose names start with 'xhtml_' are placed into the resulting HTML with no changes.

```
<v:page name="button__0" xmlns:v="http://http://example.com/vspx/">
  <html>
    <head>
      <title>VSPX samples | v:button</title>
    </head>
    <body>
      <v:form name="form1" type="simple" action="" method="POST">
        <v:text name="txt1" default=""/>
        <v:button name="submit1" action="simple" style="image" xhtml_alt="A test button" value="--'plus.g
      </v:form>
    </body>
  </html>
</v:page>
```

Example 14.17. Simple 'Add-Number' example

The form accepts two numbers and calculates their sum.

```
<v:page name="button__1" xmlns:v="http://http://example.com/vspx/">
  <html>
    <head>
      <title>VSPX samples | v:button</title>
    </head>
    <body>
      <v:form name="f1" type="simple" action="" method="POST">
        <v:label name="l1" value="--0" format="%.2f" /><br/>
        <v:text name="t1" default="0"/>
        <v:text name="t2" default="0"/>
        <v:button name="b1" action="simple" value="Add" >
        <v:on-post>
         self.l1.ufl_value := atof (self.t1.ufl_value) + atof (self.t2.ufl_value);
        </v:on-post>
        </v:button>
      </v:form>
    </body>
  </html>
</v:page>
```

Example 14.18. Simple 'Browse File' dialog

The form allows to select a file from server's file system <html> <body>

```
<v:page name="browse__2" xmlns:v="http://http://example.com/vspx/">
  <v:form name="f1" type="simple" method="POST">
      <v:text name="floc" value="" xhtml_size="60%" />
      <v:button action='browse' name="cmb" selector="browser/dav_browser.vsp"
                child-window-options="resizable=yes, status=no, menubar=no, scrollbars=no, width=640, hei
                value="Browse"
                browser-type="os"
                browser-mode="RES"
                browser-xfer="DOM"
                browser-list="1"
                browser-current="1"
                browser-filter="*">
        <v:field name="floc" />
      </v:button>
  </v:form>
</v:page>
</body>
</html>
```

Example 14.19. Simple browse button example

The form allows to select a two values from a different page <html> <body>

```
<v:page name="button__3" xmlns:v="http://http://example.com/vspx/">
  <v:form name="frm" type="simple" method="POST">
      <v:text name="fld1" value="" />
      <v:text name="fld2" value="" />
      <v:button action='browse' name="br1" selector="button__4.vspx"
                child-window-options="scrollbars=yes, menubar=no, height=630, width=600"
                value="Browse">
        <v:field name="fld1" />
        <v:field name="fld2" />
      </v:button>
  </v:form>
</v:page>
```

```
</body>
</html>
```

Example 14.20. Simple select button example

The form returns back to caller form two walues from a text box and select list <html> <body>

```
<v:page name="button__4" xmlns:v="http://http://example.com/vspx/">
  <v:form name="f1" type="simple" method="POST">
     <v:text name="sl1" value="Test" />
     <v:select-list name="sl2">
       <v:item name="*default*" value="0" />
       <v:item name="FIRST"     value="1" />
       <v:item name="SECOND"    value="2" />
     </v:select-list>
        <v:button name="select1" action="return" value="Select">
          <v:field name="fld1" ref="sl1" />
          <v:field name="fld2" ref="sl2" />
        </v:button>
 </v:form>
</v:page>
</body>
</html>
```

See Also: Reference Material in the Tutorial:

VX-S-2

VX-S-7

calendar

calendar — Databound calendar control.

Synopsis

<**calendar** /> (
name (required) ,
annotation (optional) ,
initial-enable (optional) ,
enabled (optional) ,
instantiate (optional) ,
control-udt (optional) ,
xsd-stub-xhtml (optional) ,
width (optional) ,
height (optional) ,
initial-date (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

This is the generic calendar control.

Attributes

name = SqlName . A page level unique name identifying a control.

annotation. A human readable comment.

initial-enable = CalculateableValue . Determines whether a control is initially visible. True by default. Could be data-bound to an SQL expression.

enabled = CalculateableValue . Determines whether a control is visible. True by default. Could be data-bound to an SQL expression.

instantiate = CalculateableValue . Determines whether a control instantiate its children. It is true by default. It could be data-bound to an SQL expression. Unlike most of calcucateable attributes, the value of this attribute for a control is calcluated before calling 'on-init' event handler of the control; other values are calculated before calling 'before-data-bind' event handler.

control-udt = SqlName . At run time every control is represented as an instance of some user-defined type (UDT). VSPX compiles selects the UDT to use depending on name of the XML element that represents the control and maybe some of its attributes. In some specific cases the application developer may instruct VSPX compiler to use some other UDT, e.g. an application-specific UDT that is derived from the preset one. If specified, the value of 'control-udt' attribute should be equal to the name of the desired target UDT.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

initial-date = CalculateableValue . Expression which returns a date for setting initially the current date of the calendar.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Declaration of type vspx_calendar

```
create type vspx_calendar under vspx_control as
(
  cal_date date,
  cal_meta any,
  cal_selected datetime,
  cal_current_row vspx_row_template
) temporary self as ref
method vc_get_date_array () returns any,
overriding method vc_view_state (stream any, n int) returns any,
overriding method vc_set_view_state (e vspx_event) returns any,
constructor method vspx_calendar (name varchar, parent vspx_control)
```

Examples

Example 14.21. A calendar

This demo will show a calendar allowing to list dates by months:

```
<v:page name="pagecal1" xmlns:v="http://http://example.com/vspx/">
<html>
<body>
<v:form name="form1" type="simple" method="POST" >
    <v:calendar name="cal1" initial-date="--now()">
      <v:template type="simple" name="chead1" name-to-remove="table" set-to-remove="bottom">
      <table border="0" cellpadding="0" cellspacing="0">
        <tr>
         <td align="center">
          <v:button name="pmon" value="<" action="simple" style="url">
            <v:on-post>
            self.cal1.cal_date := dateadd ('month', -1, self.cal1.cal_date);
            self.cal1.vc_data_bind (e);
            </v:on-post>
          </v:button>
         </td>
         <td colspan="5" align="center">
           <?V monthname((control.vc_parent as vspx_calendar).cal_date) ?>
           <?V year((control.vc_parent as vspx_calendar).cal_date) ?>
         </td>
         <td align="center">
          <v:button name="nmon" value=">" action="simple" style="url">
            <v:on-post>
            self.cal1.cal_date := dateadd ('month', 1, self.cal1.cal_date);
            self.cal1.vc_data_bind (e);
            </v:on-post>
          </v:button>
         </td>
         </tr>
        <tr><td>Sun</td><td>|Mon</td><td>|Tue</td><td>|Wed</td><td>|Thu</td><td>|Fri</td><td>|Sat</td></t
      </table>
      </v:template>
      <v:template type="repeat" name="cbody1" name-to-remove="" set-to-remove="">
      <v:template type="browse" name="crow1" name-to-remove="table" set-to-remove="both">
      <table>
      <tr>
       <td align="center"><v:button style="url" action="simple" name="b1" value="--cal_icell(control, 0)"
       <td align="center"><v:button style="url" action="simple" name="b2" value="--cal_icell(control, 1)"
       <td align="center"><v:button style="url" action="simple" name="b3" value="--cal_icell(control, 2)"
       <td align="center"><v:button style="url" action="simple" name="b4" value="--cal_icell(control, 3)"
       <td align="center"><v:button style="url" action="simple" name="b5" value="--cal_icell(control, 4)"
       <td align="center"><v:button style="url" action="simple" name="b6" value="--cal_icell(control, 5)"
       <td align="center"><v:button style="url" action="simple" name="b7" value="--cal_icell(control, 6)"
      </tr>
```

```
        </table>
        </v:template>
        </v:template>
        <v:template type="simple" name="cbott1" name-to-remove="table" set-to-remove="top">
        <table>
        </table>
        </v:template>
      </v:calendar>
</v:form>
</body>
</html>
</v:page>
```

check-box

check-box — Scriptable check-box.

Synopsis

<**check-box** /> (
name (required) ,
annotation (optional) ,
initial-enable (optional) ,
enabled (optional) ,
instantiate (optional) ,
control-udt (optional) ,
xsd-stub-xhtml (optional) ,
width (optional) ,
height (optional) ,
column (optional) ,
null-value (optional) ,
error-glyph (optional) ,
auto-submit (optional) ,
value (optional) ,
element-value (optional) ,
element-place (optional) ,
element-path (optional) ,
element-params (optional) ,
element-update-path (optional) ,
element-update-params (optional) ,
fmt-function (optional) ,
cvt-function (optional) ,
group-name (optional) ,
initial-checked (optional) ,
is-boolean (optional) ,
true-value (optional) ,
false-value (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

is a represantation of HTML check box. Scriptable, databindable.

Attributes

name = SqlName . A page level unique name identifying a control.

annotation. A human readable comment.

initial-enable = CalculateableValue . Determines whether a control is initially visible. True by default. Could be data-bound to an SQL expression.

enabled = CalculateableValue . Determines whether a control is visible. True by default. Could be data-bound to an SQL expression.

instantiate = CalculateableValue . Determines whether a control instantiate its children. It is true by default. It could be data-bound to an SQL expression. Unlike most of calcucateable attributes, the value of this attribute for a control is calcluated before calling 'on-init' event handler of the control; other values are calculated before calling 'before-data-bind' event handler.

control-udt = SqlName . At run time every control is represented as an instance of some user-defined type (UDT). VSPX compiles selects the UDT to use depending on name of the XML element that represents the control and maybe some of its attributes. In some specific cases the application developer may instruct VSPX compiler to use some other UDT, e.g. an application-specific UDT that is derived from the preset one. If specified, the value of 'control-udt' attribute should be equal to the name of the desired target UDT.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

column = SqlName . The name of the column bound. If nothing else is specified, the column meta data from the containing update form sets the field's attributes.

null-value. This value will be shown if value of the column is NULL, also value of the control will be set to null if this value is passed upon POST request.

error-glyph. The character to be displayed near the resulting HTML input element if the test of v:validator of the element detects an error in the current value of the element.

auto-submit. Flag to auto submit the parent form if value of the control is changed.

value = CalculateableValue . Data bindable value of control, represents value of HTML control to be drawn. The calculated value of this attribute is stored in the control in ufl_value field.

element-value = CalculateableValue . An XML entity that contains a value to be displayed by an HTML control. The calculated value is stored in ufl_element_value field of the control. This value is used only if ufl_value is null (e.g. if the 'value' attribute is not set at all), and the method vc_get_value_from_element() will be used to calculate ufl_value based on ufl_element_value, ufl_element_path and ufl_element_place.

element-place = CalculateableValue . The place of actual data inside an XML element. The calculated value of this attribute is stored in ufl_element_place field of the control. This is to process XMLSchema-compatible data without writing extra code for handling NULL values. The XML element can contain the value to be bound in either attribute or in an inner text node. XMLSchema treats missing attributes as NULL values. missing text node as an empty string value and if 'xsi:nil' attribute is 'true' then inner text value is NULL no matter if there are any text nodes. These self-evident rules are convenient for XQuery data retrieval but not for data update: an result of some XQuery expression can point to data but not to a place where data should occur in the future. The use of 'element-place' allows to eliminate the problem. Attributes 'element-value', 'element-path' and 'element-update-path' may locate an element that is always present whereas 'element-place' describes where the desired datum will occur in the element. The value of this attribute should be a string of syntax '@attributename' or 'text()'. If this is '@attributename' then the effective bound value is a value of the specified attribute or NULL if the specified attribute is missing. On update, either the attribute is created/edited in order to set non-NULL value or the attribute is deleted in order to set NULL. If text() is specified then both text value of the element and the value of its 'xsi:nil' are properly handled on both data bind and update.

element-path = CalculateableValue . This is the path inside the entity that comes from 'element-value' attribute. The default path is 'self::node()'. This path is used to point precisely to an subentity that should actually be used for data binding. The calculated value of this attribute should be a string in XQuery syntax, and it is saved in ufl_element_path member of the control.

element-params = CalculateableValue . This is the vector of parameter values of the path specified by 'element-path' attribute. The default is NULL indicating no parameters. The attribute is calculated but not used if 'element-path' attribute is missing. The calculated value of this attribute should be either NULL or a vector of even length whose items are parameter names and parameter values; refer to the description of xquery_eval() function for more details. The value of the attribute is saved in

ufl_element_params member of the control.

element-update-path = CalculateableValue . Like the value of 'element-path', this is the path inside the entity that comes from 'element-value' attribute. The difference is that 'element-path' is used inside 'vc_get_value_from_element() during data-bind to set ufl_value based on ufl_element_value whereas 'element-update-path' is used during data update inside 'vc_put_value_to_element() in order to save data inside the document that is referenced by ufl_element_value. The default path for update is equal to the path for data bind, i.e. it is the value of the calculated 'element-path' attribute or 'self::node()'. The calculated value of 'element-update-path' should be a string in XQuery syntax, and it is saved in ufl_element_update_path member of the control.

element-update-params = CalculateableValue . This is the vector of parameter values of the path specified by 'element-update-path' attribute. The default is NULL indicating no parameters. The attribute is calculated but not used if 'element-update-path' attribute is missing (e.g. you should not try to specify 'element-path' without 'element-update-path' and both 'element-params' and 'element-update-params' in hope that values from 'element-update-params' will be passed to 'element-path' on update). The calculated value of this attribute should be either NULL or a vector of even length whose items are parameter names and parameter values; refer to the description of xquery_eval() function for more details. The value of the attribute is saved in ufl_element_update_params member of the control.

fmt-function = CalculateableValue . This is to convert the value that is bound to the control into a string to use during the rendering. The value of this attribute should be a name of function that takes a single argument of the type that matches the type of ufl_value and returns a string. If this is used then the value of 'cvt-function' should probably be a name of function that converts the string back to the desired data type. If a control supports 'format' attribute then the format is applied to the result of 'fmt-function', so the value returned by fmt-function may be e.g. an integer to be formatted by '%d' format. The calculated value of this attribute is stored in ufl_fmt_fn field of the control.

cvt-function = CalculateableValue . This is to convert a user input of type varchar into a value that should be stored in ufl_value (and e.g. placed into some database column by an update). The calculated value of this attribute is stored in ufl_cvt_fn field of the control. For more details, see the description of 'fmt_function' attribute above.

group-name = SqlName . When several check boxes are grouped, this is the group name. This will be submitted instead of the control name on post.

initial-checked = CalculateableValue . Specifies whether the control is initially checked.

is-boolean. If this attribute is set to '1' then the check-box is forced to ignore its 'initial-checked' status and work using the data-bound value as the only criterion for enabling or disabling its 'checked' property. This mode is convenient for editing two-state data values. Before render, the control will check if the ufl_value is equal to 'true-value' or 'false-value', and it is displayed as checked if the ufl_value is equal to 'true-value' (or it is not equal to any of these two but is not a logical 'false'). On post, a new status of the checkbox is inspected, and the ufl_value is set to 'true-value' if it is checked or 'false-value' otherwise. The default value of this attribute is '0' meaning that there is no system-level relation between the 'checked' status and the 'value'. When the form with the check-box submits data, the submitted value of the attribute depends on its 'is-boolean' property. If it's '0' then the result of cast (control.ufl_value as varchar) is submitted; otherwise a string '1' is submitted.

true-value = CalculateableValue . This attribute should be used if and only if 'is-boolean' attribute is set to '1'. The calculated value of this attribute is saved in ufl_true_value field of the control and is used to represent a logical 'true' for the control. The checkbox is displayed as checked if the bound value is equal to the ufl_true_value. If the checked checkbox is submitted then the bound value of the control is set to the ufl_true_value.

false-value = CalculateableValue . This attribute should be used if and only if 'is-boolean' attribute is set to '1'. The calculated value of this attribute is saved in ufl_false_value field of the control and is used to represent a logical 'false' for the control. The checkbox is displayed as not checked if the bound value is equal to the ufl_valuse_value. If no checked checkbox is submitted by a form then the bound value of the control is set to ufl_false_value.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Declaration of type vspx_check_box

```
create type vspx_check_box under vspx_field
temporary self as ref
constructor method vspx_check_box (name varchar, parent vspx_control),
overriding method vc_render () returns any,
overriding method vc_xrender () returns any,
overriding method vc_view_state (stream any, n int) returns any,
overriding method vc_set_view_state (e vspx_event) returns any,
overriding method vc_set_model () returns any
```

Examples

Example 14.22. Entry form with a checkbox

Depending on the state of the checkbox, the submitted value is either 'checked' or 'unchecked'.

```
<v:page name="check_box__0" xmlns:v="http://http://example.com/vspx/">
  <html>
    <head>
      <title>VSPX samples | v:check-box</title>
    </head>
    <body>
      <v:form name="form1" type="simple" action="" method="POST">
        <v:check-box name="cb1" initial-checked="0" value="-- case control.ufl_selected when 1 then 'chec
        <v:button name="submit1" action="simple" value="OK"/>
      </v:form>
    </body>
  </html>
</v:page>
```

See Also: Reference Material in the Tutorial:
VX-S-2

code-file

code-file — Code-behind file containing SQL script.

Synopsis

<**code-file** /> (*url* (required) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

This element is used to load an external SQL script after page class compilation and before page execution. This script may contain page subclass or custom control definitions. In this way SQL code can be separated from VSPX markup, or in other words to separate page design from application logic.

Attributes

url. URL of file to be loaded after VSPX page compilation. If relative then the base is the page where the control resides (i.e. same URL resolution rules applied as for includes).

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Examples

Example 14.23. Code-behind SQL script

The script of a button's post event is in separate SQL file.

The code_file__0.sql contents is:

```
drop type my_page_subclass
;

create type my_page_subclass under DB.dba.page__vspx_code__file____0_vspx
temporary self as ref
overriding method vc_post_b1 (control vspx_button, e vspx_event) returns any,
method button_change (control vspx_button) returns any
;

create method vc_post_b1 (inout control vspx_button, inout e vspx_event) for my_page_subclass
 {
   if (not control.vc_focus) return;
   self.button_change (control);
   return;
 }
;

create method button_change (inout control vspx_button) for my_page_subclass
 {
   self.var1 := self.var1 + 1;
   control.ufl_value := 'Activated';
 }
;

<v:page name="page1" xmlns:v="http://http://example.com/vspx/" page-subclass="my_page_subclass">
  <html>
```

```
    <body>
      <v:code-file  url="code_file__0.sql"/>
      <v:variable name="var1" type="int" default="0" param-name="id" persist="pagestate" />
      <v:form type="simple" name="f1">
        <v:label name="l1" value="--self.var1" format="%d" />
        <v:button action="simple" name="b1" value="Submit" />
      </v:form>
    </body>
  </html>
</v:page>
```

column

column — A column marker.

Synopsis

<**column** /> (*name* (required) ,
label (optional) ,
input-format (optional) ,
output-format (optional) ,
xsd-stub-xhtml (optional) ,
width (optional) ,
height (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

A column marker for use in v:data-set, v:data-grid and v:data-source. This element should exist for each selected column of the SQL expression of the data set. If this is specified under data-set, data-source or data-grid controls, this must be the exact same columns as in SQL statement to be evaluated. If those element is omitted in data-set or data-grid, the columns will be extracted from compilation of the SQL statement. However, this will always be required for calls of stored procedures that return resultsets and for the data-source control.

Attributes

name = SqlName . The name of column

label = CalculateableValue . Alternate text which could be used to display in column headings.

input-format. A sprintf format string for printing the input value

output-format. A sprintf format string for printing the output value

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

data-grid

data-grid — Scrollable databound multi-row control.

Synopsis

```
<data-grid /> (   name (required) ,
                  annotation (optional) ,
                  initial-enable (optional) ,
                  enabled (optional) ,
                  instantiate (optional) ,
                  control-udt (optional) ,
                  xsd-stub-xhtml (optional) ,
                  width (optional) ,
                  height (optional) ,
                  data (optional) ,
                  meta (optional) ,
                  nrows (optional) ,
                  sql (required) ,
                  scrollable (optional) ,
                  cursor-type (optional) ,
                  edit (optional) ,
                  debug-srcfile (optional) ,
                  debug-srcline (optional) ,
                  debug-log (optional) ) ;
```

Description

This is the generic multi-row database view control. It is used to show repeating data, as from tables or procedure views. Allows scrolling (paging) editing the data; adding a row or removal of existing rows. Usually this control generates a PL scrollable cursor of type as specified. The number of rows shown are configurable via nrows attribute. Note also that column children elements are optional; they can be omitted so then VSPX engine will compute them from specified SQL statement in 'sql' attribute. Specific to this control is to have row-template and frame-template, these are to present repeating and non-repeating groups of children elements. The frame-template will usually have a 'rowset' placeholder to designate the place where repeating group (row-template) will be instantiated. The control expects to find child controls with specific names. for scrolling buttons - [data-grid name]'_prev' and [data-grid name]'_next'; for editing buttons - [data-grid name]'_edit' and [data-grid name]'_delete'.

Attributes

name = SqlName . A page level unique name identifying a control.

annotation. A human readable comment.

initial-enable = CalculateableValue . Determines whether a control is initially visible. True by default. Could be data-bound to an SQL expression.

enabled = CalculateableValue . Determines whether a control is visible. True by default. Could be data-bound to an SQL expression.

instantiate = CalculateableValue . Determines whether a control instantiate its children. It is true by default. It could be data-bound to an SQL expression. Unlike most of calcucateable attributes, the value of this attribute for a control is calcluated before calling 'on-init' event handler of the control; other values are calculated before calling 'before-data-bind' event handler.

control-udt = SqlName . At run time every control is represented as an instance of some user-defined type (UDT). VSPX compiles selects the UDT to use depending on name of the XML element that represents the control and maybe some of its attributes. In some specific cases the application developer may instruct VSPX compiler to use some other UDT, e.g. an application-specific UDT that is derived from the preset one. If specified, the value of 'control-udt' attribute should be equal to the name of the desired target UDT.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

data = CalculateableValue . Expression which returns an array of rowset data, this is an alternative to specifying an SQL expression. The result then will be used to instantiate the repeating group. An example of such data is the result returned by 'exec' function:

```
(
  ("ALFKI" "Alfreds Futterkiste" "030-0074321" )
  ("ANATR" "Ana Trujillo Emparedados y helados" "(5) 555-4729" )
)
```

meta = CalculateableValue . This works with combination of data attribute, this expression must return an array of ro metadata. So when this is specified it needs to be an expression returning meta-data for columns. In practice this is the same as 1st element of metadata returned by 'exec()' :

```
(
  ("CustomerID" 182 0 5 0 1 1 "Demo" "CustomerID" "demo" "Customers" 0 )
  ("CompanyName" 182 0 40 1 1 1 "Demo" "CompanyName" "demo" "Customers" 0 )
  ("Phone" 182 0 24 1 1 1 "Demo" "Phone" "demo" "Customers" 0 )
)
```

for details of metadata see 'exec()' function description.

nrows = CalculateableValue . The maximum number of replicas of the row template to be made for rows selected. This is like the rowset size for a scrollable cursor.

sql = SqlCode . The select statement that gets the data. Parameters to the statement MUST be specified as SQL identifiers prefixed with a colon (named parameter). The values are given in the control's vspx:param children. These specify the parameter name and a data bound expression for the value.

scrollable. The setting controls whether next and previous page buttons are presented. This can be on even if the cursor is not scrollable as such.

cursor-type.

Table 14.3. Allowed values of the 'cursor-type' attribute

static	A static cursor is used.
dynamic	A dynamic cursor is used.
keyset	A keyset cursor is used.

edit = CalculateableValue . Flags whether editing features are enabled on data grid control

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Declaration of type vspx_data_grid

Scrollable, Multi-Row data grid Class

```
create type vspx_data_grid under vspx_form
as (
    dg_nrows int default -1,    -- how many rows to show on single page
    dg_scrollable int default 0,  -- scroll on form is enabled
    dg_editable int default 1,          -- disable edit/add on whole grid
    dg_row_meta any,      -- metadata
    dg_row_data any,      -- the data for procedure binding
    dg_current_row vspx_row_template,   -- current row template
    dg_rowno_edit int default null, -- last edited row in result set, to re-display the edit box on error
    dg_rows_fetched int default 0,  -- these are to keep state for scrolling
    dg_prev_bookmark any default null,
    dg_last_bookmark any default null
   )  temporary self as ref
  method vc_templates_clean () returns any,
  constructor method vspx_data_grid (name varchar, parent vspx_control),
  overriding method vc_set_view_state (e vspx_event) returns any,
  overriding method vc_view_state (stream any, n int) returns any
```

Examples

Example 14.24. Editable multi-row data grid based on data-grid control

This example will show a list of customers from the Northwind demo database. Allows editing, addtion and removal of existing records:

```
<v:page name="data_grid__0" xmlns:v="http://http://example.com/vspx/">
<html>
  <body>
      <v:data-grid name="dg" nrows="5" sql="select CustomerID, CompanyName, Phone from Demo.demo.Customer
        <v:param name="mask" value="'%'"/>
        <v:template name="t1" type="frame">
          <div>
            <div>
              Action
                <v:label name="l1" value="--(control.vc_parent as vspx_data_grid).dg_row_meta[0]"/>
              <v:label name="l2" value="--(control.vc_parent as vspx_data_grid).dg_row_meta[1]"/>
              <v:label name="l3" value="--(control.vc_parent as vspx_data_grid).dg_row_meta[2]"/>
            </div>
            <v:rowset/>
            <v:form type="add"/>
            <div>
              <v:button name="dg_prev" action="simple" value="&lt;&lt;"/>
              <v:button name="dg_next" action="simple" value="&gt;&gt;"/>
            </div>
          </div>
        </v:template>
        <v:template name="t2" type="row">
          <div>
            <v:button name="dg_edit" action="simple" value="Edit"/>
            <v:button name="dg_delete" action="simple" value="Delete">
              <v:on-post>
              delete from Demo.demo.Customers where CustomerID = self.dg.dg_current_row.te_rowset[0];
              self.dg.vc_data_bind (e);
                </v:on-post>
            </v:button>
            <v:label name="label1" value="--(control.vc_parent as vspx_row_template).te_rowset[0]"/>
            <v:label name="label2" value="--(control.vc_parent as vspx_row_template).te_rowset[1]"/>
            <v:label name="label3" value="--(control.vc_parent as vspx_row_template).te_rowset[2]"/>
          </div>
        </v:template>
        <v:template name="t3" type="if-not-exists">
          <div>No rows selected</div>
        </v:template>
        <v:template name="t4" type="edit">
          <v:form name="u1" type="update" table="Demo.demo.Customers" if-not-exists="insert">
            <v:key column="CustomerID" value="--self.dg.dg_current_row.te_rowset[0]" default="null"/>
            <v:template type="if-exists" name="t5">
```

```
            <div>
              <v:button name="b1" action="submit" value="Update"/>
              <input type="submit" name="b2" value="Cancel"/>
              <v:text name="c_id1" column="CustomerID"/>
              <v:text name="c_name1" column="CompanyName"/>
              <v:text name="c_phone1" column="Phone"/>
            </div>
          </v:template>
        </v:form>
      </v:template>
      <v:template name="t6" type="add">
        <v:form type="update" name="a1" table="Demo.demo.Customers" if-not-exists="insert">
          <v:key column="CustomerID" value="--''" default="null"/>
          <v:template name="t7" type="if-exists">
            <v:button name="b3" action="submit" value="Add"/>
            <v:text name="c_id2" column="CustomerID"/>
            <v:text name="c_name2" column="CompanyName"/>
            <v:text name="c_phone2" column="Phone"/>
          </v:template>
        </v:form>
      </v:template>
    </v:data-grid>
  </body>
</html>
</v:page>
```

data-list

data-list — Select list initialized from database table.

Synopsis

<**data-list** /> (
name (required) ,
annotation (optional) ,
initial-enable (optional) ,
enabled (optional) ,
instantiate (optional) ,
control-udt (optional) ,
xsd-stub-xhtml (optional) ,
width (optional) ,
height (optional) ,
error-glyph (optional) ,
auto-submit (optional) ,
column (optional) ,
null-value (optional) ,
value (optional) ,
element-value (optional) ,
element-place (optional) ,
element-path (optional) ,
element-params (optional) ,
element-update-path (optional) ,
element-update-params (optional) ,
fmt-function (optional) ,
cvt-function (optional) ,
table (optional) ,
key-column (optional) ,
value-column (optional) ,
sql (optional) ,
defvalue (optional) ,
multiple (optional) ,
list-document (optional) ,
list-match (optional) ,
list-key-path (optional) ,
list-value-path (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

This control is used to make a select list, based on a SQL expression. Also instead of SQL expression only table name could be given, so then control will compose apropriate select statement.

Attributes

name = SqlName . A page level unique name identifying a control.

annotation. A human readable comment.

initial-enable = CalculateableValue . Determines whether a control is initially visible. True by default. Could be data-bound to an SQL expression.

enabled = CalculateableValue . Determines whether a control is visible. True by default. Could be data-bound to an SQL expression.

instantiate = CalculateableValue . Determines whether a control instantiate its children. It is true by default. It could be data-bound to an SQL expression. Unlike most of calcucateable attributes, the value of this attribute for a control is calcluated before calling 'on-init' event handler of the control; other values are calculated before calling 'before-data-bind' event handler.

control-udt = SqlName . At run time every control is represented as an instance of some user-defined type (UDT). VSPX compiles selects the UDT to use depending on name of the XML element that represents the control and maybe some of its attributes. In some specific cases the application developer may instruct VSPX compiler to use some other UDT, e.g. an application-specific UDT that is derived from the preset one. If specified, the value of 'control-udt' attribute should be equal to the name of the desired target UDT.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

error-glyph. The character to be displayed near the resulting HTML input element if the test of v:validator of the element detects an error in the current value of the element.

auto-submit. Flag to auto submit the parent form if value of the control is changed.

column = SqlName . The name of the column bound. If nothing else is specified, the column meta data from the containing update form sets the field's attributes.

null-value. This value will be shown if value of the column is NULL, also value of the control will be set to null if this value is passed upon POST request.

value = CalculateableValue . Data bindable value of control, represents value of HTML control to be drawn. The calculated value of this attribute is stored in the control in ufl_value field.

element-value = CalculateableValue . An XML entity that contains a value to be displayed by an HTML control. The calculated value is stored in ufl_element_value field of the control. This value is used only if ufl_value is null (e.g. if the 'value' attribute is not set at all), and the method vc_get_value_from_element() will be used to calculate ufl_value based on ufl_element_value, ufl_element_path and ufl_element_place.

element-place = CalculateableValue . The place of actual data inside an XML element. The calculated value of this attribute is stored in ufl_element_place field of the control. This is to process XMLSchema-compatible data without writing extra code for handling NULL values. The XML element can contain the value to be bound in either attribute or in an inner text node. XMLSchema treats missing attributes as NULL values. missing text node as an empty string value and if 'xsi:nil' attribute is 'true' then inner text value is NULL no matter if there are any text nodes. These self-evident rules are convenient for XQuery data retrieval but not for data update: an result of some XQuery expression can point to data but not to a place where data should occur in the future. The use of 'element-place' allows to eliminate the problem. Attributes 'element-value', 'element-path' and 'element-update-path' may locate an element that is always present whereas 'element-place' describes where the desired datum will occur in the element. The value of this attribute should be a string of syntax '@attributename' or 'text()'. If this is '@attributename' then the effective bound value is a value of the specified attribute or NULL if the specified attribute is missing. On update, either the attribute is created/edited in order to set non-NULL value or the attribute is deleted in order to set NULL. If text() is specified then both text value of the element and the value of its 'xsi:nil' are properly handled on both data bind and update.

element-path = CalculateableValue . This is the path inside the entity that comes from 'element-value' attribute. The default path is 'self::node()'. This path is used to point precisely to an subentity that should actually be used for data binding. The calculated value of this attribute should be a string in XQuery syntax, and it is saved in ufl_element_path member of the control.

element-params = CalculateableValue . This is the vector of parameter values of the path specified by 'element-path' attribute. The default is NULL indicating no parameters. The attribute is calculated but not used if 'element-path' attribute is missing. The calculated value of this attribute should be either NULL or a vector of even length whose items are parameter names and parameter values; refer to the description of xquery_eval() function for more details. The value of the attribute is saved in ufl_element_params member of the control.

element-update-path = CalculateableValue . Like the value of 'element-path', this is the path inside the entity that comes from 'element-value' attribute. The difference is that 'element-path' is used inside 'vc_get_value_from_element() during data-bind to set ufl_value based on ufl_element_value whereas 'element-update-path' is used during data update inside 'vc_put_value_to_element() in order to save data inside the document that is referenced by ufl_element_value. The default path for update is equal to the path for data bind, i.e. it is the value of the calculated 'element-path' attribute or 'self::node()'. The calculated value of 'element-update-path' should be a string in XQuery syntax, and it is saved in ufl_element_update_path member of the control.

element-update-params = CalculateableValue . This is the vector of parameter values of the path specified by 'element-update-path' attribute. The default is NULL indicating no parameters. The attribute is calculated but not used if 'element-update-path' attribute is missing (e.g. you should not try to specify 'element-path' without 'element-update-path' and both 'element-params' and 'element-update-params' in hope that values from 'element-update-params' will be passed to 'element-path' on update). The calculated value of this attribute should be either NULL or a vector of even length whose items are parameter names and parameter values; refer to the description of xquery_eval() function for more details. The value of the attribute is saved in ufl_element_update_params member of the control.

fmt-function = CalculateableValue . This is to convert the value that is bound to the control into a string to use during the rendering. The value of this attribute should be a name of function that takes a single argument of the type that matches the type of ufl_value and returns a string. If this is used then the value of 'cvt-function' should probably be a name of function that converts the string back to the desired data type. If a control supports 'format' attribute then the format is applied to the result of 'fmt-function', so the value returned by fmt-function may be e.g. an integer to be formatted by '%d' format. The calculated value of this attribute is stored in ufl_fmt_fn field of the control.

cvt-function = CalculateableValue . This is to convert a user input of type varchar into a value that should be stored in ufl_value (and e.g. placed into some database column by an update). The calculated value of this attribute is stored in ufl_cvt_fn field of the control. For more details, see the description of 'fmt_function' attribute above.

table. A table name for select statement.

key-column = SqlName . A key column name.

value-column = SqlName . A value column name.

sql = CalculateableValue . An alterantive of 'table' attribute, when this attribute is specified it must contains a valid select statement. The selection list will be instantiated using a cursor as this expression is defined.

defvalue = CalculateableValue . A default value for the control.

multiple. Used to desigante a multiple selection list box. In this case the control's value will be an array of key data of selected items.

list-document = CalculateableValue . This is to obtain all or some of items of the select-list from an XML document. If this attribute is specified then the value of the calculated attribute should be an XML entity and this entity will be saved in vsl_list_document field of the control; values of 'list-match', 'list-key-path' and 'list-value-path' arguments should specify XQuery expressions that should be used to retrieve keys and displayable values of items. All selection items composed from the value of 'list-document' will be listed before all items from 'table' or 'sql', if both methods of data retrieval are used.

list-match = CalculateableValue . This is to calculate a string value that is an XQuery expression. The text of this expression will be saved in 'vsl_list_match' member of the control. This expression will be used during data binding to get array of displayable items from the value of 'list-document'. The expression should return a node-set using 'list-document' entity as a context mode; every item of the node-set will be converted into list item via 'list-key-path' and 'list-value-path' expressions.

list-key-path = CalculateableValue . This is to specify the XQuery expression that calculates key value of a list item, using a result of 'list-match' as a context node. The text of the expression will be saved in 'vsl_list_key_path' member of the control.

list-value-path = CalculateableValue . This is to specify the XQuery expression that calculates a displayable value of a list item, using a result of 'list-match' as a context node. The text of the expression will be saved in 'vsl_list_value_path' member of the control.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Declaration of type vspx_data_list

```
create type vspx_data_list under vspx_select_list temporary self as ref
constructor method vspx_data_list (name varchar, parent vspx_control)
```

Examples

Example 14.25. Select list based on table source

This example will render in a form a select list containing the customers from Northwind demo database with a option values their customer IDs.

```
<v:page name="data_list__0" xmlns:v="http://http://example.com/vspx/">
<html>
  <body>
      <v:form name="f1" type="simple" method="POST">
        <v:data-list name="dl1" table="Demo.demo.Customers" key-column="CustomerID" value-column="Company
      </v:form>
  </body>
</html>
</v:page>
```

Example 14.26. Select list based on SQL select statement

This example will render in a form a select list containing the customers from Northwind demo database with a option values their customer IDs.

```
<v:page name="data_list__1" xmlns:v="http://http://example.com/vspx/">
<html>
  <body>
      <v:form name="f1" type="simple" method="POST">
        <v:data-list name="dl1" sql="select * from Demo.demo.Customers" key-column="CustomerID" value-col
      </v:form>
  </body>
</html>
</v:page>
```

See Also: Reference Material in the Tutorial:
VX-S-3

data-set

data-set — Scrollable, multi-row data bound grid.

Synopsis

<**data-set** /> (
name (required) ,
annotation (optional) ,
initial-enable (optional) ,
enabled (optional) ,
instantiate (optional) ,
control-udt (optional) ,
xsd-stub-xhtml (optional) ,
width (optional) ,
height (optional) ,
sql (optional) ,
data-source (optional) ,
data (optional) ,
meta (optional) ,
nrows (optional) ,
scrollable (required) ,
cursor-type (optional) ,
edit (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

A container for displaying and/or editing the content of a resultset. This is the generic multi-row database view control. It is used to show repeating data, as from tables or procedure views. Allows scrolling (paging) editing the data; adding a row or removal of existing rows. Usually this control generates a PL scrollable cursor of type. The number of rows shown is configurable via nrows attribute. Note also that column children elements are optional; they can be omitted so then VSPX engine will compute them from the specified SQL statement in 'sql' attribute. This control has the same functionality as the data-grid control but has different syntax, better suited for editing via plugins for 3-d party WYSWYG HTML authoring tools (as Addobe GoLive and similar). Specific to this control is to have several templates to present repeating and non-repeating groups of children elements. These templates are as follows: two generic templates to represent header and footer sections and one repeating template to enclose edit, add, not-exists and row (browse) sections. All of those templates are optional and their usage can be seen in examples. Also following the convention for names of controls must be followed: for scrolling buttons - [data-set name]'_prev', [data-set name]'_next', [data-set name]'_first' and [data-set name]'_last'; for editing buttons - [data-set name]'_edit' and [data-set name]'_delete'.

Attributes

name = SqlName . A page level unique name identifying a control.

annotation. A human readable comment.

initial-enable = CalculateableValue . Determines whether a control is initially visible. True by default. Could be data-bound to an SQL expression.

enabled = CalculateableValue . Determines whether a control is visible. True by default. Could be data-bound to an SQL expression.

instantiate = CalculateableValue . Determines whether a control instantiate its children. It is true by default. It could be data-bound to an SQL expression. Unlike most of calcucateable attributes, the value of this attribute for a control is calcluated before calling 'on-init' event handler of the control; other values are calculated before calling 'before-data-bind' event handler.

control-udt = SqlName . At run time every control is represented as an instance of some user-defined type (UDT). VSPX compiles selects the UDT to use depending on name of the XML element that represents the control and maybe some of its attributes. In some specific cases the application developer may instruct VSPX compiler to use some other UDT, e.g. an application-specific UDT that is derived from the preset one. If specified, the value of 'control-udt' attribute should be equal to the name of the desired target UDT.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

sql = SqlCode . The select statement that gets the data. Parameters to the statement MUST be specified as SQL identifiers prefixed with a colon. The values are given in the control's vspx:param children. These specify the parameter name and a data bound expression for the value.

data-source = SqlName . A reference to a data-source, which wil lbe used to bind the data in data-set grid.

data = CalculateableValue . When this attribute is specified it must be an expression returning array of arrays. This is an alternative to specifying an SQL expression or data-source. The result then will be used to instantiate the repeating group. An example of such data is the result returned by 'exec' function:

```
(
  ("ALFKI" "Alfreds Futterkiste" "030-0074321" )
  ("ANATR" "Ana Trujillo Emparedados y helados" "(5) 555-4729" )
)
```

meta = CalculateableValue . This attribute is used together with 'data', so when is specified it needs to be an expression returning a meta-data for columns corresponding to the these returned by 'data' expression. In practice it's same as 1-st element of metadata returned by 'exec()' :

```
(
  ("CustomerID" 182 0 5 0 1 1 "Demo" "CustomerID" "demo" "Customers" 0 )
  ("CompanyName" 182 0 40 1 1 1 "Demo" "CompanyName" "demo" "Customers" 0 )
  ("Phone" 182 0 24 1 1 1 "Demo" "Phone" "demo" "Customers" 0 )
)
```

for details of metadata see 'exec()' function description.

nrows = CalculateableValue . The maximum number of replicas of the row template to be made for rows selected. It is like the rowset size for a scrollable cursor.

scrollable. The setting controls whether next and previous page buttons are presented. This can be on even if the cursor is not scrollable as such.

cursor-type.

Table 14.4. Allowed values of the 'cursor-type' attribute

static	A static cursor is used.
dynamic	A dynamic cursor is used.
keyset	A keyset cursor is used.

edit. Flags whether editing features are enabled on data grid control

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Declaration of type vspx_data_set

```
create type vspx_data_set under vspx_form
as (
    ds_nrows int,     -- how many rows to show on single page
    ds_scrollable int,      -- scroll on form is enabled
    ds_editable int default 1,          -- disable edit/add on whole grid
    ds_row_meta any,      -- metadata
    ds_row_data any,      -- data coming from a function, rowset
    ds_current_row vspx_row_template,   -- current row template
    ds_rowno_edit int default null, -- last edited row in result set, to re-display the edit box on error
    ds_rows_fetched int default 0,  -- how many rows are fetched for current page
    ds_rows_total int default 0,  -- how many data rows do we have in total (for vector)
    ds_rows_offs    int default 0,  -- this is the zero-based index of the first row of the current page
    ds_rows_offs_saved int default 0,  -- this is the value of ds_rows_offs that was saved in the page st
    ds_scrolled     int default 0,
    ds_has_next_page int default 0,     -- Flag if there are rows after the current page (so 'next page'
    ds_prev_bookmark any default null, -- Bookmark of the record that was at the beginning of previous re
    ds_last_bookmark any default null, -- Bookmark of the record that was at the end of previous retrieva
    ds_rows_cache any,
    ds_data_source vspx_data_source default null
   )  temporary self as ref
  method vc_templates_clean () returns any,
  method vc_reset () returns any,
  method ds_column_offset (name varchar) returns any,
  method ds_iterate_rows (inx int) returns any,
  constructor method vspx_data_set (name varchar, parent vspx_control),
  overriding method vc_set_view_state (e vspx_event) returns any,
  overriding method vc_view_state (stream any, n int) returns any
```

Examples

Example 14.27. Editable multi-row data grid based on data-set control

This example will show a list of customers from the Northwind demo database. Allows editing, addtion and removal of existing records.

```
<v:page name="data_set__0" xmlns:v="http://http://example.com/vspx/">
<html>
  <body>
      <v:data-set name="ds" sql="select CustomerID, CompanyName, Phone from Demo.demo.Customers" nrows="1
        <v:template name="t1" type="simple">
          <div>
            Action
            <v:label name="l1" value="CustomerID" format="%s" width="80"/>
            <v:label name="l2" value="CompanyName" format="%s" width="80"/>
            <v:label name="l3" value="Phone" format="%s" width="80"/>
          </div>
        </v:template>
        <v:template name="t2" type="repeat">
          <v:template name="t7" type="if-not-exists">
            <div>No rows selected</div>
          </v:template>
          <v:template name="t5" type="edit">
            <v:form name="u1" type="update" table="Demo.demo.Customers" if-not-exists="insert">
              <v:key column="CustomerID" value="--self.ds.ds_current_row.te_rowset[0]" default="null"/>
              <v:template name="template6" type="simple">
                <div>
                  <v:button name="upd_button" action="submit" value="Update"/>
                  <input type="submit" name="ds_cancel" value="Cancel"/>
                  <v:text name="c_id1" column="CustomerID"/>
                  <v:text name="c_name1" column="CompanyName"/>
                  <v:text name="c_phone1" column="Phone"/>
                </div>
```

```
            </v:template>
          </v:form>
        </v:template>
        <v:template name="t8" type="add">
          <v:form name="a1" type="update" table="Demo.demo.Customers" if-not-exists="insert">
            <v:key column="CustomerID" value="--''" default="null"/>
            <v:template name="template9" type="simple">
              <div>
                <v:button name="add_button" action="submit" value="Add"/>
                <v:text name="c_id2" column="CustomerID"/>
                <v:text name="c_name2" column="CompanyName"/>
                <v:text name="c_phone2" column="Phone"/>
              </div>
            </v:template>
          </v:form>
        </v:template>
        <v:template name="t4" type="browse">
          <div>
            <v:button name="ds_edit" action="simple" value="Edit"/>
            <v:button name="ds_delete" action="simple" value="Delete">
              <v:on-post>
                    delete from Demo.demo.Customers where CustomerID = self.ds.ds_current_row.te_rowset
                    self.ds.vc_data_bind(e);
                </v:on-post>
            </v:button>
            <v:label name="l5" value="--(control.vc_parent as vspx_row_template).te_rowset[0]"/>
            <v:label name="l6" value="--(control.vc_parent as vspx_row_template).te_rowset[1]"/>
            <v:label name="l7" value="--(control.vc_parent as vspx_row_template).te_rowset[2]"/>
          </div>
        </v:template>
      </v:template>
      <v:template name="t3" type="simple">
        <div>
          <v:button name="ds_prev" action="simple" value="&lt;&lt;"/>
          <v:button name="ds_next" action="simple" value="&gt;&gt;"/>
        </div>
      </v:template>
    </v:data-set>
  </body>
</html>
</v:page>
```

Example 14.28. Simple multi-row data grid based on data-set control

This example will show a list of customers from the Northwind demo database. Allows scrolling only of existing records.

```
<v:page name="data_set__0" xmlns:v="http://http://example.com/vspx/">
<html>
  <body>
      <v:data-set name="ds" sql="select CustomerID, CompanyName, Phone from Demo.demo.Customers" nrows="1
        <v:template name="t1" type="simple">
          <div>
            <v:label name="l1" value="CustomerID" format="%s" width="80"/>
            <v:label name="l2" value="CompanyName" format="%s" width="80"/>
            <v:label name="l3" value="Phone" format="%s" width="80"/>
          </div>
        </v:template>
        <v:template name="t2" type="repeat">
          <v:template name="t7" type="if-not-exists">
            <div>No rows selected</div>
          </v:template>
          <v:template name="t4" type="browse">
            <div>
              <v:label name="l5" value="--(control.vc_parent as vspx_row_template).te_rowset[0]"/>
              <v:label name="l6" value="--(control.vc_parent as vspx_row_template).te_rowset[1]"/>
              <v:label name="l7" value="--(control.vc_parent as vspx_row_template).te_rowset[2]"/>
            </div>
          </v:template>
        </v:template>
        <v:template name="t3" type="simple">
```

```
            <div>
              <v:button name="ds_prev" action="simple" value="&lt;&lt;"/>
              <v:button name="ds_next" action="simple" value="&gt;&gt;"/>
            </div>
          </v:template>
        </v:data-set>
  </body>
</html>
</v:page>
```

See Also: Reference Material in the Tutorial:
VX-S-3

data-source

data-source — Scrollable multi-row data bound source.

Synopsis

<**data-source** /> (
name (required) ,
annotation (optional) ,
initial-enable (optional) ,
enabled (optional) ,
instantiate (optional) ,
control-udt (optional) ,
xsd-stub-xhtml (optional) ,
width (optional) ,
height (optional) ,
expression-type (required) ,
nrows (optional) ,
initial-offset (optional) ,
data (optional) ,
meta (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

Invisible multi-row data source. This control is an invisible represantation of a result set. Also it offer a number of methods for accessing and updating the data. It can be used as a source in the data-set control or as a separate data source for various form controls (such as label, url etc.). This control is using 'SELECT TOP N,M ..' statements to bind the data into the resultset (when source is a table or SQL statement), hence this should be taken into account when using it.

Attributes

name = SqlName . A page level unique name identifying a control.

annotation. A human readable comment.

initial-enable = CalculateableValue . Determines whether a control is initially visible. True by default. Could be data-bound to an SQL expression.

enabled = CalculateableValue . Determines whether a control is visible. True by default. Could be data-bound to an SQL expression.

instantiate = CalculateableValue . Determines whether a control instantiate its children. It is true by default. It could be data-bound to an SQL expression. Unlike most of calcucateable attributes, the value of this attribute for a control is calcluated before calling 'on-init' event handler of the control; other values are calculated before calling 'before-data-bind' event handler.

control-udt = SqlName . At run time every control is represented as an instance of some user-defined type (UDT). VSPX compiles selects the UDT to use depending on name of the XML element that represents the control and maybe some of its attributes. In some specific cases the application developer may instruct VSPX compiler to use some other UDT, e.g. an application-specific UDT that is derived from the preset one. If specified, the value of 'control-udt' attribute should be equal to the name of the desired target UDT.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

expression-type. Denotes the type of 'expression'

Table 14.5. Allowed values of the 'expression-type' attribute

SQL	The expression is a SQL select statement
TABLE	The expression is a table reference
PROCEDURE	The expression is a procedure call
ARRAY	The rowset is already prepared and supplied as an array of arrays

nrows = CalculateableValue . The maximum number of rows to be selected. It is like the rowset size for a scrollable cursor.

initial-offset = CalculateableValue . Number of record to start data binding of rowset

data = CalculateableValue . When this attribute is specified it must be an expression returning array of arrays. This is an alternative to specifying an SQL expression or data-source. The result then will be used to instantiate the repeating group. An example of such data is the result returned by 'exec' function:

```
(
  ("ALFKI" "Alfreds Futterkiste" "030-0074321" )
  ("ANATR" "Ana Trujillo Emparedados y helados" "(5) 555-4729" )
)
```

meta = CalculateableValue . This attribute is used together with 'data', so when is specified it needs to be an expression returning a meta-data for columns corresponding to the these returned by 'data' expression. In practice it's same as 1-st element of metadata returned by 'exec()' :

```
(
  ("CustomerID" 182 0 5 0 1 1 "Demo" "CustomerID" "demo" "Customers" 0 )
  ("CompanyName" 182 0 40 1 1 1 "Demo" "CompanyName" "demo" "Customers" 0 )
  ("Phone" 182 0 24 1 1 1 "Demo" "Phone" "demo" "Customers" 0 )
)
```

for details of metadata see 'exec()' function description.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Declaration of type vspx_data_source

```
create type vspx_data_source under vspx_control
as    (
    ds_row_meta any,      -- metadata
    ds_row_data any,      -- data coming from a function, rowset
    ds_array_data any,    -- data coming from @data=... where @expression-type='array'
    ds_rows_fetched int default 0,  -- these are to keep state for scrolling
    ds_rows_offs    int default 0,  -- this is the pos of row at 0 offset
    ds_nrows int,     -- how many rows to show on single page
    ds_total_pages int default 0,
    ds_current_page int default 0,
    ds_current_pager_idx int default 0,
    ds_npages int default 10,
    ds_first_page int default 0,
    ds_last_page int default 0,
    ds_total_rows int default 0,
```

```
    ds_prev_bookmark any default null,
    ds_next_bookmark any default null,
    ds_parameters any default null,
    ds_columns any default null,
    ds_sql varchar default null,
    ds_sql_type varchar default 'sql',
    ds_current_inx int default 0,
    ds_update_inx int default -1,
    ds_tables any default null,
    ds_insert any default null,
    ds_update any default null,
    ds_delete any default null,
    ds_rb_data any default null,
    ds_have_more any default null
      )
temporary self as ref
method set_parameter (num any, value any) returns any,
method get_parameter (num any) returns any,
method add_parameter (value any) returns any,
method delete_parameter (num any) returns any,
method get_column_name (num any) returns any,
method set_column_label (num any, value any) returns any,
method get_column_label (num any) returns any,
method get_column_label (col varchar) returns any,
--method set_column_add_style (num int, style any) returns any,
--method get_column_add_style (num int) returns any,
--method set_column_edit_style () returns any,
--method get_column_edit_style () returns any,
--method set_column_browse_style () returns any,
--method get_column_browse_style () returns any,
--method set_column_add_format () returns any,
--method get_column_add_format () returns any,
--method set_column_edit_format () returns any,
--method get_column_edit_format () returns any,
--method set_column_browse_format () returns any,
--method get_column_browse_format () returns any,
method set_item_value (row any, col any, value any) returns any,
method set_item_value (col varchar, value any) returns any,
method get_item_value (row any, col any) returns any,
method get_rb_item_value (row any, col any) returns any,
method get_item_value (col any) returns any,
method get_item_value (col varchar) returns any,
method set_expression (expression varchar) returns any,
method get_expression () returns varchar,
method set_expression_type (type varchar) returns any,
method get_expression_type () returns varchar,
method reset () returns any,
method ds_data_bind (e vspx_event) returns any,
method ds_make_statistic () returns any,
method ds_insert (e vspx_event) returns any,
method ds_update (e vspx_event) returns any,
method ds_delete (e vspx_event) returns any,
method ds_key_params (tbl varchar) returns any,
method ds_tbl_params (tbl varchar) returns any,
method get_current_row () returns any,
constructor method vspx_data_source (name varchar, parent vspx_control)
```

Examples

Example 14.29. Simple navigation form using invisible data-source control

This example will show a cell of the customers table from the Northwind demo database. Allows to show different columns : name and phone.

```
<v:page name="data_source__0" xmlns:v="http://http://example.com/vspx/">
<html>
<body>
<v:variable name="offset" default="0" type="int"/>
   <v:form name="f1" method="POST" type="simple">
   <div>There is a hidden data-source on "<?V self.datasource1.ds_sql ?>"</div>
   <v:data-source name="datasource1" expression-type="sql" nrows="10" initial-offset="--self.offset">
     <v:expression>
```

```
        select * from Demo.demo.Customers where CustomerID like ? order by CompanyName desc
      </v:expression>
      <v:param name="mask" value="--('L%')"/>
      <v:column name="CompanyName" label="Company Name" input-format="%s" output-format="%s"/>
      <v:column name="Phone" label="Telephone" input-format="%s" output-format="%s" />
    </v:data-source>
    <div>
    row #: <v:text name="r1" value="0" /><br />
    col #: <v:text name="c1" value="0" /><br />
    Seek : <v:button action="simple" name="b1" value="OK" />
    </div>
    <div>
    "<v:label name="l1" value="--self.datasource1.get_column_label (atoi(self.c1.ufl_value))" />" =
    <v:label name="l2" value="--self.datasource1.get_item_value (atoi(self.r1.ufl_value),atoi(self.c1.ufl_
    </div>
     <v:on-post>
       self.l1.vc_data_bind (e);
       self.l2.vc_data_bind (e);
     </v:on-post>
    </v:form>
</body>
</html>
</v:page>
```

error-summary

error-summary — Placeholder for error messages.

Synopsis

<**error-summary** /> (*match* (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

Placeholder for form error messages. This is used on a form to mark where error messages resulting from field or form validation are to be placed. This will be rendered if the vc_is_valid member of the enclosing page is false. If rendered, this prints an error message generated by a validator or a catch handler. This control wiil not be instantiated. The error-summary may show errors for all controls or for a group of controls whose name matches a given regular expression. The vc_error_message members of all controls whose validation failed will be shown at the place marked by this control if attribute 'match' is not specified. Otherwise the vc_error_message of controls whose validation failed and 'name' matches the pattern specified will be shown.

Attributes

match. This attribute specifies a regular expression to be matched against names of controls with failed validation. The expression may match more than one control name, concatenating the messages in document order of controls. In this way an error summary may appear in different places of the page to print errors for different controls.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Examples

Example 14.30. Validation of text area input

The form contains two v:textarea controls with v:validator elements inside. When the OK button is pressed data are posted back to the same URI so the page is instantiated again. If any validator found a violation the message is shown to the user in the place specified by v:error-summary element.

```
<v:page name="error_summary__0" xmlns:v="http://http://example.com/vspx/">
  <html>
    <head>
      <title>VSPX samples | v:error-summary</title>
    </head>
    <body>
      <v:error-summary />
      <v:form name="form1" type="simple" action="" method="POST">
        <v:textarea name="ta1" default="enter your first text here" value="--coalesce ({?'ta1'}, control.
          <v:validator test="length" min="0" max="50" message="The length of the first input should not e
        </v:textarea>
        <v:textarea name="ta2" default="enter your second text here" value="--coalesce ({?'ta1'}, control
          <v:validator test="length" min="0" max="50" message="The length of the second input should not
        </v:textarea>
        <v:button name="submit1" action="simple" value="OK"/>
      </v:form>
    </body>
  </html>
</v:page>
```

Example 14.31. Validation of text area input with separate error summaries place

The form contains two v:textarea controls with v:validator elements inside. When the OK button is pressed data are posted back to the same URI so the page is instantiated again. If any validator found a violation the message is shown to the user in the place specified by v:error-summary element depending of a match attribute.

```
<v:page name="error_summary__0" xmlns:v="http://http://example.com/vspx/">
  <html>
    <head>
      <title>VSPX samples | v:error-summary</title>
    </head>
    <body>
      <v:error-summary match="ta1"/>
      <v:form name="form1" type="simple" action="" method="POST">
        <v:textarea name="ta1" default="enter your first text here" value="--coalesce ({?'ta1'}, control.
          <v:validator test="length" min="0" max="50" message="The length of the first input should not e
        </v:textarea>
      <v:error-summary match="ta2"/>
        <v:textarea name="ta2" default="enter your second text here" value="--coalesce ({?'ta1'}, control
          <v:validator test="length" min="0" max="50" message="The length of the second input should not
        </v:textarea>
        <v:button name="submit1" action="simple" value="OK"/>
      </v:form>
    </body>
  </html>
</v:page>
```

expression

expression — An SQL statement that generates a rowset for v:data-source.

Synopsis

<**expression** /> (*language* (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

A container for SQL statement generating a rowset or table name (see data-source).

Attributes

language = SqlName . Denotes the type of language, currently only SQL expressions are supported, see 'expression-type' attribute of the data-source control.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

field

field — Name of input to be sent to and back between pup-up and parent window.

Synopsis

<**field** /> (
name (required) ,
annotation (optional) ,
initial-enable (optional) ,
enabled (optional) ,
instantiate (optional) ,
control-udt (optional) ,
xsd-stub-xhtml (optional) ,
width (optional) ,
height (optional) ,
ref (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

This element may occur under browse-button or select button, it enumerates the names of inputs to be sent between pop-up and parent window.

Attributes

name = SqlName . A page level unique name identifying a control.

annotation. A human readable comment.

initial-enable = CalculateableValue . Determines whether a control is initially visible. True by default. Could be data-bound to an SQL expression.

enabled = CalculateableValue . Determines whether a control is visible. True by default. Could be data-bound to an SQL expression.

instantiate = CalculateableValue . Determines whether a control instantiate its children. It is true by default. It could be data-bound to an SQL expression. Unlike most of calcucateable attributes, the value of this attribute for a control is calcluated before calling 'on-init' event handler of the control; other values are calculated before calling 'before-data-bind' event handler.

control-udt = SqlName . At run time every control is represented as an instance of some user-defined type (UDT). VSPX compiles selects the UDT to use depending on name of the XML element that represents the control and maybe some of its attributes. In some specific cases the application developer may instruct VSPX compiler to use some other UDT, e.g. an application-specific UDT that is derived from the preset one. If specified, the value of 'control-udt' attribute should be equal to the name of the desired target UDT.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

ref = SqlName . When a field is used as child of and return button then this must contain valid reference to name of a input control (like text, select-list, data-list etc.) from current page. The referenced control's value will be used to set the value of control with name specified in attribute 'name' in the target page. The data will be set using automatically generated JavaScript function.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

form

form — Generic scriptable container for elements that may accept user's input.

Synopsis

<**form** /> (*name* (required) ,
annotation (optional) ,
initial-enable (optional) ,
enabled (optional) ,
instantiate (optional) ,
control-udt (optional) ,
xsd-stub-xhtml (optional) ,
width (optional) ,
height (optional) ,
type (optional) ,
action (optional) ,
method (optional) ,
table (optional) ,
if-not-exists (optional) ,
concurrency (optional) ,
triggers (optional) ,
data-source (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

This is a grouping element for controls that handle post data and validation of user's input. In case of type 'update', this is a single row update control for a database table. This retrieves the data from the row identified by the keys and shows the data in the column bound fields enclosed. If responding to a post for a submit inside this form, the form updates/inserts the data and shows the resulting state at the render pass.

Attributes

name = SqlName . A page level unique name identifying a control.

annotation. A human readable comment.

initial-enable = CalculateableValue . Determines whether a control is initially visible. True by default. Could be data-bound to an SQL expression.

enabled = CalculateableValue . Determines whether a control is visible. True by default. Could be data-bound to an SQL expression.

instantiate = CalculateableValue . Determines whether a control instantiate its children. It is true by default. It could be data-bound to an SQL expression. Unlike most of calcucateable attributes, the value of this attribute for a control is calcluated before calling 'on-init' event handler of the control; other values are calculated before calling 'before-data-bind' event handler.

control-udt = SqlName . At run time every control is represented as an instance of some user-defined type (UDT). VSPX compiles selects the UDT to use depending on name of the XML element that represents the control and maybe some of its attributes. In some specific cases the application developer may instruct VSPX compiler to use some other UDT, e.g. an application-specific UDT that is derived from the preset one. If specified, the value of 'control-udt' attribute should be equal to the

name of the desired target UDT.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

type. Type of form, can be 'simple' or 'update'. When this is 'simple' the vspx_form class will be instantiated, otherwise in case of 'update' vspx_update_form subclass will be used.

Table 14.6. Allowed values of the 'type' attribute

simple	Form with no special data binding.
update	Form that may be used in templates of type 'update' or 'add'.

action. A URL for processing. The data filled-in the form will be submitted to the 'action' page. Therefore this is a page where the content of that form will be processed. If it is empty, the target is a current page.

method. A method of transferring the form data, can be POST or GET. The same as 'method' of HTML forms.

The use of 'GET' is not recommended because the length of the URI may easily exceed internal limits of the browser or an intermediate proxy server.

table. If the value of 'type' attribute is 'update', the form acts as a single row update control for a database table; and the value of 'table' attribute specifies the name of that table. It retrieves the data from the row identified by the keys and shows the data in the column bound fields enclosed. If responding to a post for a submit inside this form, the form updates/inserts the data and shows the resulting state at the render pass. This attribute is required if 'type' is 'update' and prohibited otherwise.

if-not-exists. If the keys do not select any row, and the value of this attribute is 'insert', the update form will be filled in with defaults filled in and will do an insert into the table when the form is submitted. In this case of course the keys will have to be defaulted after the post or will have to be supplied in the post as fields. Any defaulting should take place in the v:on-post handler which is before the insert or update in the event of the post.

concurrency. If true and a previous value was shown in the form before the post being processed, this causes the system to check whether any of the data has been updated between getting it and the post at hand. This is a sort of optimistic concurrency control at the row level. If an update is detected, the updated_meanwhile message is set as the error message of the form and no update is made. The render pass may choose to process this message as it will, the default is to show it as any other error message for form level validation failure.

triggers. This is applicable for update form only, when this is OFF no PL triggers will be fired when inserting or updating.

data-source = SqlName . This is applicable to the update form and is An alternative of 'table'. When is specified the value is suppsed to be valid reference to existing data-source control. Further the given data-source will be used to bind/update the data.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Declaration of type vspx_form

```
create type vspx_form under vspx_control
   as
     (
       uf_action varchar default '',
       uf_method varchar default 'post',
```

```
        --uf_inside_form int default 0,
        uf_validators any,
        uf_xmodel any default null,
        uf_xsubmit any default null,
        uf_xschema any default null
      )  temporary self as ref
    constructor method vspx_form (name varchar, parent vspx_control),
    method prologue_render (sid varchar, realm varchar, nonce varchar) returns any,
    method epilogue_render () returns any,
    overriding method vc_set_model () returns any
```

Examples

Example 14.32. Simple entry form

The OK button of the form submits data from the form back to the same page.

```
<v:page name="form__0" xmlns:v="http://http://example.com/vspx/">
  <html>
    <head>
      <title>VSPX samples | v:form</title>
    </head>
    <body>
      <v:form name="form1" type="simple" action="" method="POST">
        <v:label name="label1" value="--'Simple form'"/>
        <v:text name="txt1" default=""/>
        <v:text name="txt2" default=""/>
        <v:button name="submit1" action="simple" value="OK"/>
      </v:form>
    </body>
  </html>
</v:page>
```

Example 14.33. Simple entry form containing various controls

The OK button of the form submits data from the form back to the same page. The check-box is validated against ragio group:

```
<v:page name="form__1" xmlns:v="http://http://example.com/vspx/">
  <html>
    <head>
      <title>VSPX samples | v:form | v:validator</title>
    </head>
    <body>
      <v:form name="f1" type="simple" method="POST">
        <v:error-summary/><br/>
        <v:label name="l1" value="--'Simple form'"/><br/>
        <v:url name="u1" value="--'This page'" url="--self.f1.uf_action" /><br/>
        <v:text name="t1" value="--self.rb1.ufl_value"/><br/>
        <v:text name="t2" value="--'my'||'password'" type="password"/><br/>
        <v:text name="t3" value="--'something'||'hidden'" type="hidden"/>
        <v:textarea name="ta1" value="--'some text'" /><br/>
        <v:check-box name="cb1" value="check-box" /><br/>
        <v:radio-group name="rg1">
         <v:radio-button name="rb1" value="one" />
         <v:radio-button name="rb2" value="two" />
        </v:radio-group><br/>
        <v:button name="b1" action="simple" value="--'OK'"/><br/>
        <v:validator test="sql" message="only when radio is on position one check-box could be checked">
        if (self.rb2.ufl_selected and self.cb1.ufl_selected)
          {
            return 1;
          }
        </v:validator>
      </v:form>
    </body>
  </html>
</v:page>
```

See Also: Reference Material in the Tutorial:

VX-S-2

VX-S-3

hidden

hidden — A wrapper that prevents WYSIWYG editors from displaying its content.

Synopsis

<**hidden** /> (
xsd-stub-xhtml (optional) ,
width (optional) ,
height (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

This control does not affect the resulting HTML and Virtuoso/PL code. It is used by some WYSIWYG editors in order to temporarily hide details of the page fragment from the editor's drawing area.

Attributes

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

horizontal-template

horizontal-template

Synopsis

<**horizontal-template** /> (
name (required) ,
annotation (optional) ,
initial-enable (optional) ,
enabled (optional) ,
instantiate (optional) ,
control-udt (optional) ,
xsd-stub-xhtml (optional) ,
width (optional) ,
height (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

Attributes

name = SqlName . A page level unique name identifying a control.

annotation. A human readable comment.

initial-enable = CalculateableValue . Determines whether a control is initially visible. True by default. Could be data-bound to an SQL expression.

enabled = CalculateableValue . Determines whether a control is visible. True by default. Could be data-bound to an SQL expression.

instantiate = CalculateableValue . Determines whether a control instantiate its children. It is true by default. It could be data-bound to an SQL expression. Unlike most of calcucateable attributes, the value of this attribute for a control is calcluated before calling 'on-init' event handler of the control; other values are calculated before calling 'before-data-bind' event handler.

control-udt = SqlName . At run time every control is represented as an instance of some user-defined type (UDT). VSPX compiles selects the UDT to use depending on name of the XML element that represents the control and maybe some of its attributes. In some specific cases the application developer may instruct VSPX compiler to use some other UDT, e.g. an application-specific UDT that is derived from the preset one. If specified, the value of 'control-udt' attribute should be equal to the name of the desired target UDT.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Declaration of type vspx_horizontal_template

```
create type vspx_horizontal_template under vspx_control
as ( vc_stub any ) temporary self as ref
```

include

include — A place where the source code of other page should be inserted.

Synopsis

<include /> (
url (required) ,
active (optional) ,
initial-active (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

The VSPX compiler replaces this control with the content of another page. This will include the content from the specified url at this point of the page. The url is relative to the page. The inclusion will be done before compilation of VSPX into Virtuoso/PL, hence the include can contain either vspx or static HTML content. The only requirement is that the included file be well formed. Note also that if you are using vspx namespace in the include file, it must be declared as such. This element will not be instantiated. instead of instantiating it will be replaced with content of the specified document. The included page may or may not contain a 'page' element. If a 'page' element is found in the included file this will be skipped in the resulting page. Also names of controls in the included file MUST NOT conflict with names of controls in the top level page or other included content.

Attributes

url. URL of file to be included. If relative then the base is the page where the control resides. E.g. If '/home/pageA' includes 'subdir1/pageB' and the included page includes 'subdir2/pageC' then 'pageC' should reside in '/home/subdir1/subdir2', not in '/home/subdir2'.

active = CalculateableValue . A data bound value to enable or disable the link. The default is '1' meaning 'enable'.

initial-active = CalculateableValue . A data bound value to enable or disable the link. The default is '1' meaning 'enable'. Unlike 'active' attribute, this one is effective only when the page is displayed in the first time, not after user posts data back to the page.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Examples

Example 14.34. Include a common back-link into the end of page

If the file footer.xml is placed in the same directory where this sample page is located and contains the paragraph <P>Back to index</P> then the resulting page will have additional paragraph at the end.

```
<v:page name="include__0" xmlns:v="http://http://example.com/vspx/">
  <html>
    <head><title>VSPX samples | v:include</title></head>
    <body>
      <p>The link after the horizontal ruler is included from footer.xml</p>
      <hr/>
      <v:include url="footer.xml"/> <!-- this will include the file described above -->

    </body>
  </html>
</v:page>
```

isql

isql — An interactive SQL control that allows the user to type an SQL statement and to see the result of its execution.

Synopsis

<**isql** /> (
name (required) ,
annotation (optional) ,
initial-enable (optional) ,
enabled (optional) ,
instantiate (optional) ,
control-udt (optional) ,
xsd-stub-xhtml (optional) ,
width (optional) ,
height (optional) ,
isolation (optional) ,
timeout (optional) ,
maxrows (optional) ,
user (optional) ,
password (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

Attributes

name = SqlName . A page level unique name identifying a control.

annotation. A human readable comment.

initial-enable = CalculateableValue . Determines whether a control is initially visible. True by default. Could be data-bound to an SQL expression.

enabled = CalculateableValue . Determines whether a control is visible. True by default. Could be data-bound to an SQL expression.

instantiate = CalculateableValue . Determines whether a control instantiate its children. It is true by default. It could be data-bound to an SQL expression. Unlike most of calcucateable attributes, the value of this attribute for a control is calcluated before calling 'on-init' event handler of the control; other values are calculated before calling 'before-data-bind' event handler.

control-udt = SqlName . At run time every control is represented as an instance of some user-defined type (UDT). VSPX compiles selects the UDT to use depending on name of the XML element that represents the control and maybe some of its attributes. In some specific cases the application developer may instruct VSPX compiler to use some other UDT, e.g. an application-specific UDT that is derived from the preset one. If specified, the value of 'control-udt' attribute should be equal to the name of the desired target UDT.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of

this attribute will not be used when the resulting HTML is rendered.

isolation. Transaction isolation level to be used.

Table 14.7. Allowed values of the 'isolation' attribute

uncomitted	'uncomitted' transaction isolation level
committed	'committed' transaction isolation level
repeatable	'repeatable' transaction isolation level
serializable	'serializable' transaction isolation level

timeout = CalculateableValue . Query time-out in seconds (currently not supported)

maxrows = CalculateableValue . Maximum number of rows to be displayed.

user = CalculateableValue . SQL user account to perform the query, if not specified the SQL account used to run the VSPX page will be used.

password = CalculateableValue . When the 'user' attribute is supplied, this is the user's password.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Declaration of type vspx_isql

A isql control

```
create type vspx_isql under vspx_form
as
 (
   -- parameters
   isql_custom_exec integer default 0,  -- do nothing to allow vspx code to perform actual exec (i.e. dat
   isql_explain integer default 0,  -- explain instead of execute
   isql_maxrows integer default 20,
   isql_chunked integer default 0,
   isql_current_stmt varchar default null,
   isql_current_state any default null,
   isql_current_meta any default null,
   isql_current_pos int default 0,
   isql_current_row int default 0,
   isql_rows_fetched int default 0,
   isql_user varchar default null,
   isql_password varchar default null,
   isql_isolation varchar default 'committed',
   isql_timeout integer default 60,
   isql_text varchar default '', -- sql text to execute
   --results
   --also used vc_error_message as vector(vector('sqlstate','sqlmessage'), ... ) for multiple statements
   isql_mtd any default null, --as vector(mtd, ... ) for multiple statements
   isql_res any default null,  --as vector(res, ... ) for multiple statements
   isql_stmts any default null -- array of statements to be executed on render as cursors
 )
   temporary self as ref
  constructor method vspx_isql (name varchar, parent vspx_control),
  method isql_exec () returns any
```

item

item — Item representing a selection inside a select list.

Synopsis

<item /> (*name* (required) ,
value (required) ,
xsd-stub-xhtml (optional) ,
width (optional) ,
height (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

Item representing a selection inside a select list.

Attributes

name = SqlCode . The visible value of the option, it will be shown in the option list.

value. The key value of the option, it will be assigned to the select control ufl_value is given option is seelcted.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

key

key — A key value of the vspx:form of type 'update'.

Synopsis

<**key** /> (*name* (optional) ,
column (required) ,
value (required) ,
default (optional) ,
xsd-stub-xhtml (optional) ,
width (optional) ,
height (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

This control defines a key value of the vspx:form of type 'update'. All the vspx:key children together should select one or zero rows from the table. Controls inside the form will process fields of this selected row.

Attributes

name = SqlName . Not used

column = SqlName . Name of the column in the table to be updated. This may be either the name of a primary key column or an other column, as long as the selection specified by all vspx:key children of the vspx:form is unambiguous.

value = CalculateableValue . The value of the key field referred to by the 'column' attribute.

default = CalculateableValue . The default value to be used when the 'value' expression returns NULL.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

label

label — Generic scriptable text.

Synopsis

<label /> (
name (required) ,
annotation (optional) ,
initial-enable (optional) ,
enabled (optional) ,
instantiate (optional) ,
control-udt (optional) ,
xsd-stub-xhtml (optional) ,
width (optional) ,
height (optional) ,
column (optional) ,
null-value (optional) ,
value (optional) ,
element-value (optional) ,
element-place (optional) ,
element-path (optional) ,
element-params (optional) ,
element-update-path (optional) ,
element-update-params (optional) ,
fmt-function (optional) ,
cvt-function (optional) ,
format (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

This is for displaying a value as plain text. The underlying class is derived from VSPX_FIELD so the value to be displayed is accessible as a value of any VSPX field.

Attributes

name = SqlName . A page level unique name identifying a control.

annotation. A human readable comment.

initial-enable = CalculateableValue . Determines whether a control is initially visible. True by default. Could be data-bound to an SQL expression.

enabled = CalculateableValue . Determines whether a control is visible. True by default. Could be data-bound to an SQL expression.

instantiate = CalculateableValue . Determines whether a control instantiate its children. It is true by default. It could be data-bound to an SQL expression. Unlike most of calcucateable attributes, the value of this attribute for a control is calcluated before calling 'on-init' event handler of the control; other values are calculated before calling 'before-data-bind' event handler.

control-udt = SqlName . At run time every control is represented as an instance of some user-defined type (UDT). VSPX compiles selects the UDT to use depending on name of the XML element that represents the control and maybe some of its attributes. In some specific cases the application developer may instruct VSPX compiler to use some other UDT, e.g. an application-specific UDT that is derived from the preset one. If specified, the value of 'control-udt' attribute should be equal to the name of the desired target UDT.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

column = SqlName . The name of the column bound. If nothing else is specified, the column meta data from the containing update form sets the field's attributes.

null-value. This value will be shown if value of the column is NULL, also value of the control will be set to null if this value is passed upon POST request.

value = CalculateableValue . Data bindable value of control, represents value of HTML control to be drawn. The calculated value of this attribute is stored in the control in ufl_value field.

element-value = CalculateableValue . An XML entity that contains a value to be displayed by an HTML control. The calculated value is stored in ufl_element_value field of the control. This value is used only if ufl_value is null (e.g. if the 'value' attribute is not set at all), and the method vc_get_value_from_element() will be used to calculate ufl_value based on ufl_element_value, ufl_element_path and ufl_element_place.

element-place = CalculateableValue . The place of actual data inside an XML element. The calculated value of this attribute is stored in ufl_element_place field of the control. This is to process XMLSchema-compatible data without writing extra code for handling NULL values. The XML element can contain the value to be bound in either attribute or in an inner text node. XMLSchema treats missing attributes as NULL values. missing text node as an empty string value and if 'xsi:nil' attribute is 'true' then inner text value is NULL no matter if there are any text nodes. These self-evident rules are convenient for XQuery data retrieval but not for data update: an result of some XQuery expression can point to data but not to a place where data should occur in the future. The use of 'element-place' allows to eliminate the problem. Attributes 'element-value', 'element-path' and 'element-update-path' may locate an element that is always present whereas 'element-place' describes where the desired datum will occur in the element. The value of this attribute should be a string of syntax '@attributename' or 'text()'. If this is '@attributename' then the effective bound value is a value of the specified attribute or NULL if the specified attribute is missing. On update, either the attribute is created/edited in order to set non-NULL value or the attribute is deleted in order to set NULL. If text() is specified then both text value of the element and the value of its 'xsi:nil' are properly handled on both data bind and update.

element-path = CalculateableValue . This is the path inside the entity that comes from 'element-value' attribute. The default path is 'self::node()'. This path is used to point precisely to an subentity that should actually be used for data binding. The calculated value of this attribute should be a string in XQuery syntax, and it is saved in ufl_element_path member of the control.

element-params = CalculateableValue . This is the vector of parameter values of the path specified by 'element-path' attribute. The default is NULL indicating no parameters. The attribute is calculated but not used if 'element-path' attribute is missing. The calculated value of this attribute should be either NULL or a vector of even length whose items are parameter names and parameter values; refer to the description of xquery_eval() function for more details. The value of the attribute is saved in ufl_element_params member of the control.

element-update-path = CalculateableValue . Like the value of 'element-path', this is the path inside the entity that comes from 'element-value' attribute. The difference is that 'element-path' is used inside 'vc_get_value_from_element() during data-bind to set ufl_value based on ufl_element_value whereas 'element-update-path' is used during data update inside 'vc_put_value_to_element() in order to save data inside the document that is referenced by ufl_element_value. The default path for update is equal to the path for data bind, i.e. it is the value of the calculated 'element-path' attribute or 'self::node()'. The calculated value of 'element-update-path' should be a string in XQuery syntax, and it is saved in ufl_element_update_path member of the control.

element-update-params = CalculateableValue . This is the vector of parameter values of the path specified by 'element-update-path' attribute. The default is NULL indicating no parameters. The attribute is calculated but not used if

'element-update-path' attribute is missing (e.g. you should not try to specify 'element-path' without 'element-update-path' and both 'element-params' and 'element-update-params' in hope that values from 'element-update-params' will be passed to 'element-path' on update). The calculated value of this attribute should be either NULL or a vector of even length whose items are parameter names and parameter values; refer to the description of xquery_eval() function for more details. The value of the attribute is saved in ufl_element_update_params member of the control.

fmt-function = CalculateableValue . This is to convert the value that is bound to the control into a string to use during the rendering. The value of this attribute should be a name of function that takes a single argument of the type that matches the type of ufl_value and returns a string. If this is used then the value of 'cvt-function' should probably be a name of function that converts the string back to the desired data type. If a control supports 'format' attribute then the format is applied to the result of 'fmt-function', so the value returned by fmt-function may be e.g. an integer to be formatted by '%d' format. The calculated value of this attribute is stored in ufl_fmt_fn field of the control.

cvt-function = CalculateableValue . This is to convert a user input of type varchar into a value that should be stored in ufl_value (and e.g. placed into some database column by an update). The calculated value of this attribute is stored in ufl_cvt_fn field of the control. For more details, see the description of 'fmt_function' attribute above.

format. A sprintf format string for printing the value

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Declaration of type vspx_label

```
create type vspx_label under vspx_field
as
(
  vl_format varchar default '%s'
) temporary self as ref
constructor method vspx_label (name varchar, parent vspx_control),
overriding method vc_render () returns any
```

Examples

Example 14.35. Simple dynamic text

This page demonstrates output of data values of various types.

```
<v:page name="label__0" xmlns:v="http://http://example.com/vspx/">
  <html>
    <head><title>VSPX samples | v:label</title></head>
    <body>
        Float: <v:label name="flo" value="--3.14" format="%d" /><br/>
        Integer: <v:label name="intgr" value="--2+2" format="%d" /><br/>
        String: <v:label name="strg" value="--'The quick brown fox has stopped his jumping over the lazy
    </body>
  </html>
</v:page>
```

leaf-template

leaf-template

Synopsis

```
<leaf-template /> (   name (required) ,
                      annotation (optional) ,
                      initial-enable (optional) ,
                      enabled (optional) ,
                      instantiate (optional) ,
                      control-udt (optional) ,
                      xsd-stub-xhtml (optional) ,
                      width (optional) ,
                      height (optional) ,
                      debug-srcfile (optional) ,
                      debug-srcline (optional) ,
                      debug-log (optional) ) ;
```

Description

Attributes

name = SqlName . A page level unique name identifying a control.

annotation. A human readable comment.

initial-enable = CalculateableValue . Determines whether a control is initially visible. True by default. Could be data-bound to an SQL expression.

enabled = CalculateableValue . Determines whether a control is visible. True by default. Could be data-bound to an SQL expression.

instantiate = CalculateableValue . Determines whether a control instantiate its children. It is true by default. It could be data-bound to an SQL expression. Unlike most of calcucateable attributes, the value of this attribute for a control is calcluated before calling 'on-init' event handler of the control; other values are calculated before calling 'before-data-bind' event handler.

control-udt = SqlName . At run time every control is represented as an instance of some user-defined type (UDT). VSPX compiles selects the UDT to use depending on name of the XML element that represents the control and maybe some of its attributes. In some specific cases the application developer may instruct VSPX compiler to use some other UDT, e.g. an application-specific UDT that is derived from the preset one. If specified, the value of 'control-udt' attribute should be equal to the name of the desired target UDT.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Declaration of type vspx_leaf_template

```
create type vspx_leaf_template under vspx_control
as ( vc_stub any ) temporary self as ref
```

local-variable

local-variable — Local variable - an invisible control to store a temporary value.

Synopsis

<**local-variable** /> (
name (required) ,
annotation (optional) ,
initial-enable (optional) ,
enabled (optional) ,
instantiate (optional) ,
control-udt (optional) ,
xsd-stub-xhtml (optional) ,
width (optional) ,
height (optional) ,
value (optional) ,
element-value (optional) ,
element-place (optional) ,
element-path (optional) ,
element-params (optional) ,
element-update-path (optional) ,
element-update-params (optional) ,
fmt-function (optional) ,
cvt-function (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

This tag declares a control that can store a temporary value that can be accessed from other controls. Usually it is an intermediate value that is used by a group of controls of the same form. E.g. if controls of a form display various data stored in a complex object then it may be convenient to obtain this object once and place it into value of a local-variable control that is the first child control of a form. The rest of child controls may access the value from the first child to calculate their values.

Attributes

name = SqlName . A page level unique name identifying a control.

annotation. A human readable comment.

initial-enable = CalculateableValue . Determines whether a control is initially visible. True by default. Could be data-bound to an SQL expression.

enabled = CalculateableValue . Determines whether a control is visible. True by default. Could be data-bound to an SQL expression.

instantiate = CalculateableValue . Determines whether a control instantiate its children. It is true by default. It could be data-bound to an SQL expression. Unlike most of calcucateable attributes, the value of this attribute for a control is calcluated before calling 'on-init' event handler of the control; other values are calculated before calling 'before-data-bind' event handler.

control-udt = SqlName . At run time every control is represented as an instance of some user-defined type (UDT). VSPX compiles selects the UDT to use depending on name of the XML element that represents the control and maybe some of its

attributes. In some specific cases the application developer may instruct VSPX compiler to use some other UDT, e.g. an application-specific UDT that is derived from the preset one. If specified, the value of 'control-udt' attribute should be equal to the name of the desired target UDT.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

value = CalculateableValue . Data bindable value of control, represents value of HTML control to be drawn. The calculated value of this attribute is stored in the control in ufl_value field.

element-value = CalculateableValue . An XML entity that contains a value to be displayed by an HTML control. The calculated value is stored in ufl_element_value field of the control. This value is used only if ufl_value is null (e.g. if the 'value' attribute is not set at all), and the method vc_get_value_from_element() will be used to calculate ufl_value based on ufl_element_value, ufl_element_path and ufl_element_place.

element-place = CalculateableValue . The place of actual data inside an XML element. The calculated value of this attribute is stored in ufl_element_place field of the control. This is to process XMLSchema-compatible data without writing extra code for handling NULL values. The XML element can contain the value to be bound in either attribute or in an inner text node. XMLSchema treats missing attributes as NULL values. missing text node as an empty string value and if 'xsi:nil' attribute is 'true' then inner text value is NULL no matter if there are any text nodes. These self-evident rules are convenient for XQuery data retrieval but not for data update: an result of some XQuery expression can point to data but not to a place where data should occur in the future. The use of 'element-place' allows to eliminate the problem. Attributes 'element-value', 'element-path' and 'element-update-path' may locate an element that is always present whereas 'element-place' describes where the desired datum will occur in the element. The value of this attribute should be a string of syntax '@attributename' or 'text()'. If this is '@attributename' then the effective bound value is a value of the specified attribute or NULL if the specified attribute is missing. On update, either the attribute is created/edited in order to set non-NULL value or the attribute is deleted in order to set NULL. If text() is specified then both text value of the element and the value of its 'xsi:nil' are properly handled on both data bind and update.

element-path = CalculateableValue . This is the path inside the entity that comes from 'element-value' attribute. The default path is 'self::node()'. This path is used to point precisely to an subentity that should actually be used for data binding. The calculated value of this attribute should be a string in XQuery syntax, and it is saved in ufl_element_path member of the control.

element-params = CalculateableValue . This is the vector of parameter values of the path specified by 'element-path' attribute. The default is NULL indicating no parameters. The attribute is calculated but not used if 'element-path' attribute is missing. The calculated value of this attribute should be either NULL or a vector of even length whose items are parameter names and parameter values; refer to the description of xquery_eval() function for more details. The value of the attribute is saved in ufl_element_params member of the control.

element-update-path = CalculateableValue . Like the value of 'element-path', this is the path inside the entity that comes from 'element-value' attribute. The difference is that 'element-path' is used inside 'vc_get_value_from_element() during data-bind to set ufl_value based on ufl_element_value whereas 'element-update-path' is used during data update inside 'vc_put_value_to_element() in order to save data inside the document that is referenced by ufl_element_value. The default path for update is equal to the path for data bind, i.e. it is the value of the calculated 'element-path' attribute or 'self::node()'. The calculated value of 'element-update-path' should be a string in XQuery syntax, and it is saved in ufl_element_update_path member of the control.

element-update-params = CalculateableValue . This is the vector of parameter values of the path specified by 'element-update-path' attribute. The default is NULL indicating no parameters. The attribute is calculated but not used if 'element-update-path' attribute is missing (e.g. you should not try to specify 'element-path' without 'element-update-path' and both 'element-params' and 'element-update-params' in hope that values from 'element-update-params' will be passed to 'element-path' on update). The calculated value of this attribute should be either NULL or a vector of even length whose items are parameter names and parameter values; refer to the description of xquery_eval() function for more details. The value of the attribute is saved in ufl_element_update_params member of the control.

fmt-function = CalculateableValue . This is to convert the value that is bound to the control into a string to use during the rendering. The value of this attribute should be a name of function that takes a single argument of the type that matches the type of

ufl_value and returns a string. If this is used then the value of 'cvt-function' should probably be a name of function that converts the string back to the desired data type. If a control supports 'format' attribute then the format is applied to the result of 'fmt-function', so the value returned by fmt-function may be e.g. an integer to be formatted by '%d' format. The calculated value of this attribute is stored in ufl_fmt_fn field of the control.

cvt-function = CalculateableValue . This is to convert a user input of type varchar into a value that should be stored in ufl_value (and e.g. placed into some database column by an update). The calculated value of this attribute is stored in ufl_cvt_fn field of the control. For more details, see the description of 'fmt_function' attribute above.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

login

login — Authentication parameters of a page.

Synopsis

<login /> (
name (required) ,
annotation (optional) ,
initial-enable (optional) ,
enabled (optional) ,
instantiate (optional) ,
control-udt (optional) ,
xsd-stub-xhtml (optional) ,
width (optional) ,
height (optional) ,
realm (required) ,
mode (required) ,
user-password (optional) ,
user-password-check (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

The login control controls authentication for its page. Depending on options this may or may not be visible. This works together with a login-form and logout-button controls. The storage of passwords and user names is left to the application, which can specify functions called by these widgets. The server keeps login information in the following table:

```
create table VSPX_SESSION (VS_REALM varchar, VS_SID varchar, VS_UID varchar, VS_STATE long varchar,
  primary key (VS_REALM, VS_SID));
```

This table is shared between all vspx applications, each with their login differentiated by realm. The state is an array of name value pairs which will be accessed through connection_get and connection_set inside the page code. There are three methods of keeping session state:URL-poisoning, digest authentication and Cookies.The URL-poisoning scheme presents a login dialog form and sets a session ID as hidden form element. Digest authentication uses an opaque value as session id and is available only for browsers that support standard HTTP 1.1 digest authentication. The Cookie is an analogue of URL poisoning, but in that case session ID is kept as a Cookie, thus cookies must be enabled on the browser. The login control is mandatory when using a persistent page variables (see persist="session" attribute of 'variable' control). This is because HTTP protocol is state-less; therefore value of such variables have to be stored in a table and restored on next hit with same session ID (and realm). Hence as login control maintains a session state (via some mechanism, see notes above), this capability is used to maintain persistent page variables. Note also that persistent page variables can be used between different pages, in that case all of those pages need to have login control (in most cases it is invisible).

Attributes

name = SqlName . A page level unique name identifying a control.

annotation. A human readable comment.

initial-enable = CalculateableValue . Determines whether a control is initially visible. True by default. Could be data-bound to an SQL expression.

enabled = CalculateableValue . Determines whether a control is visible. True by default. Could be data-bound to an SQL expression.

instantiate = CalculateableValue . Determines whether a control instantiate its children. It is true by default. It could be data-bound to an SQL expression. Unlike most of calcucateable attributes, the value of this attribute for a control is calcluated before calling 'on-init' event handler of the control; other values are calculated before calling 'before-data-bind' event handler.

control-udt = SqlName . At run time every control is represented as an instance of some user-defined type (UDT). VSPX compiles selects the UDT to use depending on name of the XML element that represents the control and maybe some of its attributes. In some specific cases the application developer may instruct VSPX compiler to use some other UDT, e.g. an application-specific UDT that is derived from the preset one. If specified, the value of 'control-udt' attribute should be equal to the name of the desired target UDT.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

realm. This is the authentication realm name.

mode. This specifies the preferred mode of keeping session and login information. In the case of url and cookie the name and password will have to be supplied in a form submit. The vspx:login-form control provides a convenient way of doing this. If passwords are transmitted in a form submit, it is best for security to have the login page accessed via SSL only. Many values may be separated by spaces. This is the order of preference. The system will use the first available depending on the user agent.

Table 14.8. Allowed values of the 'mode' attribute

digest	HTTP digest authentication is used and the session id will travel as the opaque data in the digest headers.
url	The url mode means that URL poisoning is used and that the session id passes with the links, posts etc. Some automation is offered for this by the vspx:url and vspx:form and derived controls.
cookie	The session id is set in a cookie on browser.

user-password. The name of a SQL function which will retrieve the password given a user name. If digest authentication is to be possible, this function must be specified. Example:

```
create procedure
sql_user_password (in name varchar)
{
  declare pass varchar;
  pass := NULL;
  whenever not found goto none;
  select pwd_magic_calc (U_NAME, U_PASSWORD, 1) into pass
          from SYS_USERS where U_NAME = name and U_SQL_ENABLE = 1 and U_IS_ROLE = 0;
none:
  return pass;
}
;
```

user-password-check. This is a function which takes the user name and password (unencrypted) and returns true if these match, false otherwise.

```
create procedure
sql_user_password_check (in name varchar, in pass varchar)
{
  if (exists (select 1 from SYS_USERS where U_NAME = name and U_SQL_ENABLE = 1 and U_IS_ROLE = 0 and
        pwd_magic_calc (U_NAME, U_PASSWORD, 1) = pass))
    return 1;
  return 0;
}
```

```
;
```

If the application will keep a session state which will automatically be extracted from the application's user repository upon login, then either of these functions may set this using connection_set. This will persist in the session if the login is successful and will be discarded otherwise. An example of such information is a user privilege class, real name, email or such. These functions may be called one or more times during the session, but no more than one call at the start is guaranteed.

Example:

Note: when the mode is digest the login control will only call user_password and in other modes it will only call user_password_check.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Declaration of type vspx_login

```
create type vspx_login under vspx_form
as (
    vl_realm varchar,
    vl_mode varchar,
    vl_pwd_get varchar,
    vl_usr_check varchar,
    vl_authenticated int default 0,
    vl_user varchar,
    vl_sid varchar,
    vl_no_login_redirect varchar,
    vl_logout_in_progress int default 0
   )
 temporary self as ref
overriding method vc_view_state (stream any, n int) returns any,
overriding method vc_set_view_state (e vspx_event) returns any,
constructor method vspx_login (name varchar, parent vspx_control)
```

Examples

Example 14.36. Simple login dialog

This example will show simltaniously 'not-autneticated' message and a simple post counter.

```
<v:page name="login__0" xmlns:v="http://http://example.com/vspx/">
<html>
  <body>
      <v:variable name="ctr" type="int" persist="session" default="0"/>
      <v:login name="lc1" realm="vspx" mode="url" user-password-check="sql_user_password_check">
        <v:template type="if-no-login">You are not logged in</v:template>
        <v:login-form name="f1" required="1" title="Login" user-title="User Name" password-title="Passwor
        <v:template type="if-login">
    Posted #: <v:label name="l1" value="--self.ctr" format="%d"/>
          <v:button name="b1" action="simple" value="Reload">
            <v:on-post>
                self.ctr := self.ctr + 1;
                self.l1.vc_data_bind (e);
                </v:on-post>
          </v:button>
          <v:button name="b2" action="logout" value="Logout"/>
        </v:template>
      </v:login>
  </body>
</html>
</v:page>
```

See Also: Reference Material in the Tutorial:
VX-S-6

login-form

login-form — A form to be shown if the user is not logged in.

Synopsis

<login-form /> (
name (required) ,
annotation (optional) ,
initial-enable (optional) ,
enabled (optional) ,
instantiate (optional) ,
control-udt (optional) ,
xsd-stub-xhtml (optional) ,
width (optional) ,
height (optional) ,
required (required) ,
title (optional) ,
user-title (optional) ,
password-title (optional) ,
submit-title (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

login-form is the control for getting login information. This may only appear inside a login. Its function will depend on the type of login being attempted. If the type is digest, this is a button which, when submitted, will send the digest challenge.

If the url or cookie session mode is chosen, this will be a form of 2 fields with user name and password and a submit button.

if the standard form is not suitable, this can be an arbitrary form. This must post buttons values named "username" and "password" and a submit button named "login". No children need be specified if the default form is OK.

If the redirect attribute of template[@type=if-no-login] is not specified, the contents of this child are instantiated and shown in the place of the login control. This can be arbitrary content.

Attributes

name = SqlName . A page level unique name identifying a control.

annotation. A human readable comment.

initial-enable = CalculateableValue . Determines whether a control is initially visible. True by default. Could be data-bound to an SQL expression.

enabled = CalculateableValue . Determines whether a control is visible. True by default. Could be data-bound to an SQL expression.

instantiate = CalculateableValue . Determines whether a control instantiate its children. It is true by default. It could be data-bound to an SQL expression. Unlike most of calcucateable attributes, the value of this attribute for a control is calcluated before calling 'on-init' event handler of the control; other values are calculated before calling 'before-data-bind' event handler.

control-udt = SqlName . At run time every control is represented as an instance of some user-defined type (UDT). VSPX

compiles selects the UDT to use depending on name of the XML element that represents the control and maybe some of its attributes. In some specific cases the application developer may instruct VSPX compiler to use some other UDT, e.g. an application-specific UDT that is derived from the preset one. If specified, the value of 'control-udt' attribute should be equal to the name of the desired target UDT.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

required. If digest is the mode, then this means that the login is mandatory and that instead of displaying the login button which will send the challenge if pressed, the challenge will be sent automatically.

title. Title of login button if digest is attempted.

user-title. Title of user name field

password-title. Title of password field

submit-title. Title of submit button

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Declaration of type vspx_login_form

```
create type vspx_login_form under vspx_form
as
  (
   vlf_title varchar,
   vlf_user_title varchar,
   vlf_password_title varchar,
   vlf_submit_title varchar,
   vlf_login vspx_login
  ) temporary self as ref
constructor method vspx_login_form (name varchar,  parent vspx_control),
constructor method vspx_login_form (name varchar, title varchar, user_title varchar, password_title varch
overriding method vc_render () returns any
```

Examples

Example 14.37. A simple default login-form

This example will render just a login form:

```
<v:page name="login_form__0" xmlns:v="http://http://example.com/vspx/">
  <html>
    <head><title>VSPX samples | v:login-form</title></head>
    <body>
      <v:login name="login_0" mode="url" user-password-check="sql_user_password_check" realm="vspx">
        <v:login-form name="login_form_0"
          required="1"
          title="Login"
          user-title="User Name"
          password-title="Password"
          submit-title="Login"/>
        </v:login>
    </body>
  </html>
```

```
</v:page>
```

method

method — Page method - a user defined member of page class.

Synopsis

<**method** /> (*name* (required) ,
returns (optional) ,
arglist (optional)) ;

Description

This creates a custom method of the page class.

Attributes

name = SqlName . The name of method to be created

returns = SqlName . Optional return datatype type

arglist. comma-separated list of arguments. For example : "in arg1 varchar, out arg1 integer ..."

node

node — A place holder for rendering a tree node.

Synopsis

<node /> (
void (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

This control indicates the place of a subtree inside a node template.

Attributes

void. Not used.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

node-template

node-template

Synopsis

<node-template /> (
name (required) ,
annotation (optional) ,
initial-enable (optional) ,
enabled (optional) ,
instantiate (optional) ,
control-udt (optional) ,
xsd-stub-xhtml (optional) ,
width (optional) ,
height (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

Attributes

name = SqlName . A page level unique name identifying a control.

annotation. A human readable comment.

initial-enable = CalculateableValue . Determines whether a control is initially visible. True by default. Could be data-bound to an SQL expression.

enabled = CalculateableValue . Determines whether a control is visible. True by default. Could be data-bound to an SQL expression.

instantiate = CalculateableValue . Determines whether a control instantiate its children. It is true by default. It could be data-bound to an SQL expression. Unlike most of calcucateable attributes, the value of this attribute for a control is calcluated before calling 'on-init' event handler of the control; other values are calculated before calling 'before-data-bind' event handler.

control-udt = SqlName . At run time every control is represented as an instance of some user-defined type (UDT). VSPX compiles selects the UDT to use depending on name of the XML element that represents the control and maybe some of its attributes. In some specific cases the application developer may instruct VSPX compiler to use some other UDT, e.g. an application-specific UDT that is derived from the preset one. If specified, the value of 'control-udt' attribute should be equal to the name of the desired target UDT.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Declaration of type vspx_node_template

```
create type vspx_node_template under vspx_control
as ( vc_stub any ) temporary self as ref
```

on-init

on-init — Container for code that should be executed at the end of control instantiation.

Synopsis

```
<on-init >(EventHandler)</on-init >(    );
```

Description

The code contained in this element is executed at the end of the control's constructor. This is expected to have additional initialization checking etc.

on-init-container

on-init-container — This is identical to v:on-init.

Synopsis

```
<on-init-container >(EventHandler)</on-init-container >(    );
```

Description

on-post

on-post — Container for code that should be executed when the element gets a post.

Synopsis

```
<on-post >(EventHandler)</on-post >(   );
```

Description

The code contained in this element is executed when the element gets a post. For fields and forms, this is a context for an a SQL compound statement that will be executed. The statement should end with a return 1 or 0. The implicit return is 0. A return of 1 means that the post is fully handled here and not to be passed on to the parent chain. A 0 means that the post was not handled and the next control up the chain will try.

The context has a self which is the page, a vspx_control which is the control which defines this handler, and a origin_control, which is the control that first got the post event. This will be a data field or button. The vspx_event with the post's data will be in the variable event.

The interpretation of the return value is the same for other event handlers. SQL conditions signalled will go to the page, these condition handlers are invoked in no special context, hence the declare handler for SQL construct should not be used to pass control between handlers, but only within handlers.

on-post-container

on-post-container — This is identical to v:on-post.

Synopsis

```
<on-post-container >(EventHandler)</on-post-container >(    );
```

Description

page

page — Container of VSPX code.

Synopsis

<**page** /> (
name (required) ,
decor (optional) ,
style (optional) ,
on-error-redirect (optional) ,
on-deadlock-retry (optional) ,
doctype (optional) ,
doctype-system (optional) ,
page-subclass (optional) ,
file-name (optional) ,
no-script-function (optional) ,
xsd-stub-xhtml (optional) ,
width (optional) ,
height (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

The container for the rest of the vspx code. The page or a subclass of it will be the outermost VSPX element on any VSPX page. Each .vspx file defines explicitly one custom page class derived from vspx_page. This element can enclose all HTML elements or can be incorporated in it's children on place apropriate for rendering the VSPX controls, but one VSPX page must contain only one vspx:page element. and noe VSPX-specific element may appear outside the vspx:page. It can be placed in other HTML stuff or around it, doesnt' matter, but VSP & VSPX code must be inside it (the same applies to customized HTML attributes, i.e. containing <?V & <?U shortcuts). Note that when using 'include' or 'decoration' features the top level page element will be : in case of include - 'this' page, in case of 'decoration' - the decoration page. Also page variables, controls will be combined in all cases of inclusion or macro-expansion, so their names must not be duplicated; this means that names of variables introduced by macro expansion or inclusion must not conflict with cnames of variables or controls in the top level page or any included or macro content.

Attributes

name = SqlName . The name of subclass of vspx_page to be generated. The name of class will be prefixed with 'page_' plus that name.

decor. This attribute specifies a 'decoration' page URL. The decoration page is a VSPX page that is wrapped around the including page. The content of the including page is then put in the place indicated by a placeholder element in the decor page. This is useful for defining enclosing tables, headers, footers and other repeating content.

style. This designates an external XSL-T style sheet to be apply over the page, before page compilation. This is useful as a 'macro' feature. The XSLT sheet may define rules for macroexpanding things in the page body.

on-error-redirect. This specifies where to redirect the HTTP client in the event of an an unhandled error. This will pass a __SQL_STATE, __SQL_MESSAGE and __PAGE as parameters to the error page.

on-deadlock-retry. This specifies how many times to retry page execution upon deadlock. When the number of retries is reached, the behaviour will be as for other SQL errors.

doctype. This is to specify the resulting document type declaration (DTD) public URI.

doctype-system. This is to specify the resulting document type declaration (DTD) system URI.

page-subclass = SqlName . The name of page subclass to be instantiated for processing. The subclass with this name MUST already be defined. The subclass definition can be kept in a code-behind file (see code-file element).

file-name. The name of page to be generated. Used by wizards to designate where generated page will be stored. This attribute do not affect VSPX processing.

no-script-function. A function to enable or disable automatic NOSCRIPT element generation.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Declaration of type vspx_page

VSPX Page Class , from it must be derived all VSPX pages, no subcalsses for others

```
create type vspx_page under vspx_control
as (
    vc_view_state any,
    vc_is_postback int default 0,
    vc_persisted_vars any,
    vc_event vspx_event,
    vc_is_valid int default 1,          -- set to 0 when first validator fails
    vc_authenticated int default 0,     -- is true if login control in the page suceeded
    vc_current_id int default 0,
    vc_browser_caps any default 0,
    vc_authentication_mode int default 1, -- authentication mode 0 - cookie, 1 - url, 2 - digest
    vc_debug_log_acc any default null   -- The accumulator in xte_nodebld_... style for keeping debugging
    )  temporary self as ref
  method vc_state_deserialize (stream any, n int) returns any,
  method vc_get_debug_log (title varchar) returns any
```

Examples

Example 14.38. Sample VSPX page

```
<v:page name="page__0" xmlns:v="http://http://example.com/vspx/">
  <html>
    <head><title>VSPX samples | v:page</title></head>
    <body>
      <p>This VSPX page does nothing.<br/>
It does not even print traditional 'Hello world' phrase.</p>
    </body>
  </html>
</v:page>
```

See Also: Reference Material in the Tutorial:

VX-S-1

VX-S-8

param

param — Named parameter for the cursor.

Synopsis

<**param** /> (
name (required) ,
value (required) ,
xsd-stub-xhtml (optional) ,
width (optional) ,
height (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

Named parameter for execution of the cursor select statement. This must be the name of a parameter for SQL statement specified, but without leading colon.

Attributes

name = SqlName . Name of the parameter.

value = CalculateableValue . Data bound value of the parameter.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

placeholder

placeholder — Placeholder for original page in the decoration page.

Synopsis

<**placeholder** /> (
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

This marks the place in the decoration page where the content of the original page should be placed.

Attributes

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

radio-button

radio-button — Scriptable radio button.

Synopsis

<**radio-button** /> (
name (required) ,
annotation (optional) ,
initial-enable (optional) ,
enabled (optional) ,
instantiate (optional) ,
control-udt (optional) ,
xsd-stub-xhtml (optional) ,
width (optional) ,
height (optional) ,
column (optional) ,
null-value (optional) ,
error-glyph (optional) ,
auto-submit (optional) ,
group-name (optional) ,
value (optional) ,
element-value (optional) ,
element-place (optional) ,
element-path (optional) ,
element-params (optional) ,
element-update-path (optional) ,
element-update-params (optional) ,
fmt-function (optional) ,
cvt-function (optional) ,
initial-checked (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

is a scriptable version of HTML radio button.

Attributes

name = SqlName . A page level unique name identifying a control.

annotation. A human readable comment.

initial-enable = CalculateableValue . Determines whether a control is initially visible. True by default. Could be data-bound to an SQL expression.

enabled = CalculateableValue . Determines whether a control is visible. True by default. Could be data-bound to an SQL expression.

instantiate = CalculateableValue . Determines whether a control instantiate its children. It is true by default. It could be data-bound to an SQL expression. Unlike most of calcucateable attributes, the value of this attribute for a control is calcluated

before calling 'on-init' event handler of the control; other values are calculated before calling 'before-data-bind' event handler.

control-udt = SqlName . At run time every control is represented as an instance of some user-defined type (UDT). VSPX compiles selects the UDT to use depending on name of the XML element that represents the control and maybe some of its attributes. In some specific cases the application developer may instruct VSPX compiler to use some other UDT, e.g. an application-specific UDT that is derived from the preset one. If specified, the value of 'control-udt' attribute should be equal to the name of the desired target UDT.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

column = SqlName . The name of the column bound. If nothing else is specified, the column meta data from the containing update form sets the field's attributes.

null-value. This value will be shown if value of the column is NULL, also value of the control will be set to null if this value is passed upon POST request.

error-glyph. The character to be displayed near the resulting HTML input element if the test of v:validator of the element detects an error in the current value of the element.

auto-submit. Flag to auto submit the parent form if value of the control is changed.

group-name. When several check boxes are to be grouped, this is the name of the group. Upon post the name of the group will be submitted instead of the than name of the control. This also will restrict in the group to have more than one button on

value = CalculateableValue . Data bindable value of control, represents value of HTML control to be drawn. The calculated value of this attribute is stored in the control in ufl_value field.

element-value = CalculateableValue . An XML entity that contains a value to be displayed by an HTML control. The calculated value is stored in ufl_element_value field of the control. This value is used only if ufl_value is null (e.g. if the 'value' attribute is not set at all), and the method vc_get_value_from_element() will be used to calculate ufl_value based on ufl_element_value, ufl_element_path and ufl_element_place.

element-place = CalculateableValue . The place of actual data inside an XML element. The calculated value of this attribute is stored in ufl_element_place field of the control. This is to process XMLSchema-compatible data without writing extra code for handling NULL values. The XML element can contain the value to be bound in either attribute or in an inner text node. XMLSchema treats missing attributes as NULL values. missing text node as an empty string value and if 'xsi:nil' attribute is 'true' then inner text value is NULL no matter if there are any text nodes. These self-evident rules are convenient for XQuery data retrieval but not for data update: an result of some XQuery expression can point to data but not to a place where data should occur in the future. The use of 'element-place' allows to eliminate the problem. Attributes 'element-value', 'element-path' and 'element-update-path' may locate an element that is always present whereas 'element-place' describes where the desired datum will occur in the element. The value of this attribute should be a string of syntax '@attributename' or 'text()'. If this is '@attributename' then the effective bound value is a value of the specified attribute or NULL if the specified attribute is missing. On update, either the attribute is created/edited in order to set non-NULL value or the attribute is deleted in order to set NULL. If text() is specified then both text value of the element and the value of its 'xsi:nil' are properly handled on both data bind and update.

element-path = CalculateableValue . This is the path inside the entity that comes from 'element-value' attribute. The default path is 'self::node()'. This path is used to point precisely to an subentity that should actually be used for data binding. The calculated value of this attribute should be a string in XQuery syntax, and it is saved in ufl_element_path member of the control.

element-params = CalculateableValue . This is the vector of parameter values of the path specified by 'element-path' attribute. The default is NULL indicating no parameters. The attribute is calculated but not used if 'element-path' attribute is missing. The calculated value of this attribute should be either NULL or a vector of even length whose items are parameter names and parameter values; refer to the description of xquery_eval() function for more details. The value of the attribute is saved in ufl_element_params member of the control.

element-update-path = CalculateableValue . Like the value of 'element-path', this is the path inside the entity that comes from 'element-value' attribute. The difference is that 'element-path' is used inside 'vc_get_value_from_element() during data-bind to set ufl_value based on ufl_element_value whereas 'element-update-path' is used during data update inside 'vc_put_value_to_element() in order to save data inside the document that is referenced by ufl_element_value. The default path for update is equal to the path for data bind, i.e. it is the value of the calculated 'element-path' attribute or 'self::node()'. The calculated value of 'element-update-path' should be a string in XQuery syntax, and it is saved in ufl_element_update_path member of the control.

element-update-params = CalculateableValue . This is the vector of parameter values of the path specified by 'element-update-path' attribute. The default is NULL indicating no parameters. The attribute is calculated but not used if 'element-update-path' attribute is missing (e.g. you should not try to specify 'element-path' without 'element-update-path' and both 'element-params' and 'element-update-params' in hope that values from 'element-update-params' will be passed to 'element-path' on update). The calculated value of this attribute should be either NULL or a vector of even length whose items are parameter names and parameter values; refer to the description of xquery_eval() function for more details. The value of the attribute is saved in ufl_element_update_params member of the control.

fmt-function = CalculateableValue . This is to convert the value that is bound to the control into a string to use during the rendering. The value of this attribute should be a name of function that takes a single argument of the type that matches the type of ufl_value and returns a string. If this is used then the value of 'cvt-function' should probably be a name of function that converts the string back to the desired data type. If a control supports 'format' attribute then the format is applied to the result of 'fmt-function', so the value returned by fmt-function may be e.g. an integer to be formatted by '%d' format. The calculated value of this attribute is stored in ufl_fmt_fn field of the control.

cvt-function = CalculateableValue . This is to convert a user input of type varchar into a value that should be stored in ufl_value (and e.g. placed into some database column by an update). The calculated value of this attribute is stored in ufl_cvt_fn field of the control. For more details, see the description of 'fmt_function' attribute above.

initial-checked. This flag specifies whther this control is initially checked.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Declaration of type vspx_radio_button

```
create type vspx_radio_button under vspx_field
temporary self as ref
constructor method vspx_radio_button (name varchar, parent vspx_control),
overriding method vc_view_state (stream any, n int) returns any,
overriding method vc_set_view_state (e vspx_event) returns any,
overriding method vc_render () returns any,
overriding method vc_xrender () returns any,
overriding method vc_set_model () returns any
```

Examples

Example 14.39. Groups of radio buttons.

The form contains two groups of radio buttons, three button in each group.

```
<v:page name="radio_button__0" xmlns:v="http://http://example.com/vspx/">
  <html>
    <head>
      <title>VSPX samples | v:radio-button</title>
    </head>
    <body>
      <v:form name="form1" type="simple" action="" method="GET">
        <input type="text" name="qq"/>
        <table>
          <tr>
            <td>
              <v:radio-button name="radio1" group-name="radio_group_A" value="--'A-one'" initial-checked=
```

```
            <v:radio-button name="radio2" group-name="radio_group_A" value="--'A-two'"/>
            <v:radio-button name="radio3" group-name="radio_group_A" value="--'A-three'"/>
          </td>
        </tr>
        <tr>
          <td>
            <v:radio-button name="radio4" group-name="radio_group_B" value="--'B-one'" initial-checked=
            <v:radio-button name="radio5" group-name="radio_group_B" value="--'B-two'"/>
            <v:radio-button name="radio6" group-name="radio_group_B" value="--'B-three'"/>
          </td>
        </tr>
      </table>
      <v:button name="submit1" action="simple" style="image" xhtml:alt="A test button" value="--'plus.g
    </v:form>
  </body>
 </html>
</v:page>
```

See Also: Reference Material in the Tutorial:
VX-S-2

radio-group

radio-group — A group of radio-buttons.

Synopsis

<**radio-group** /> (
name (required) ,
annotation (optional) ,
initial-enable (optional) ,
enabled (optional) ,
instantiate (optional) ,
control-udt (optional) ,
xsd-stub-xhtml (optional) ,
width (optional) ,
height (optional) ,
error-glyph (optional) ,
auto-submit (optional) ,
column (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

This control is used to group containing radio-buttons in a group. This ensures that only one button is selected at a time. Note: This control is not mandatory for making such group of buttons, another option is to use 'group-name' attribute of the radio-button control.

Attributes

name = SqlName . A page level unique name identifying a control.

annotation. A human readable comment.

initial-enable = CalculateableValue . Determines whether a control is initially visible. True by default. Could be data-bound to an SQL expression.

enabled = CalculateableValue . Determines whether a control is visible. True by default. Could be data-bound to an SQL expression.

instantiate = CalculateableValue . Determines whether a control instantiate its children. It is true by default. It could be data-bound to an SQL expression. Unlike most of calcucateable attributes, the value of this attribute for a control is calcluated before calling 'on-init' event handler of the control; other values are calculated before calling 'before-data-bind' event handler.

control-udt = SqlName . At run time every control is represented as an instance of some user-defined type (UDT). VSPX compiles selects the UDT to use depending on name of the XML element that represents the control and maybe some of its attributes. In some specific cases the application developer may instruct VSPX compiler to use some other UDT, e.g. an application-specific UDT that is derived from the preset one. If specified, the value of 'control-udt' attribute should be equal to the name of the desired target UDT.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

error-glyph. The character to be displayed near the resulting HTML input element if the test of v:validator of the element detects an error in the current value of the element.

auto-submit. Flag to auto submit the parent form if value of the control is changed.

column = SqlName . The name of the column bound. If nothing else is specified, the column meta data from the containing update form sets the field's attributes.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Declaration of type vspx_radio_group

```
create type vspx_radio_group under vspx_field
temporary self as ref
constructor method vspx_radio_group (name varchar, parent vspx_control),
method vc_choose_selected () returns any,
overriding method vc_xrender () returns any,
overriding method vc_set_model () returns any
```

Examples

Example 14.40. Groups of radio buttons.

The form contains a groups of radio buttons, grouped with radio-group control.

```
<v:page name="radio_group_demo" xmlns:v="http://http://example.com/vspx/">
<html>
<body>
  <v:form name="sf" type="simple" method="POST">
     <v:radio-group name="radio00">
       <v:radio-button name="radiob01" value="--'one'" />
       <v:radio-button name="radiob02" value="--'two'" />
     </v:radio-group>
  </v:form>
</body>
</html>
</v:page>
```

See Also: Reference Material in the Tutorial:

VX-S-2

VX-S-3

script

script — A (redundant) wrapper for SQL code, can be omitted.

Synopsis

<**script** /> (*language* (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

This control can appear only inside event handling controls and it always contain SQL code to be executed when the event occurs. There is no difference for VSPX compiler whether SQL code is enclosed in 'script' control or not. Some WYSIWYG tools can display the enclosed SQL code to the application developer according to custom attributes of this control but it does not affect the generated Virtuoso/PL code of the page.

Attributes

language = SqlName . Denotes the type of language, currently only Virtuoso/PL expressions are supported.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

select-list

select-list — Fixed initialized select list.

Synopsis

<**select-list** /> (
name (required) ,
annotation (optional) ,
initial-enable (optional) ,
enabled (optional) ,
instantiate (optional) ,
control-udt (optional) ,
xsd-stub-xhtml (optional) ,
width (optional) ,
height (optional) ,
column (optional) ,
null-value (optional) ,
error-glyph (optional) ,
auto-submit (optional) ,
multiple (optional) ,
value (optional) ,
element-value (optional) ,
element-place (optional) ,
element-path (optional) ,
element-params (optional) ,
element-update-path (optional) ,
element-update-params (optional) ,
fmt-function (optional) ,
cvt-function (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

This is a scriptable version of HTML select control. It shows a static list of items (see item element). Databind and on-post scripts are allowed.

Attributes

name = SqlName . A page level unique name identifying a control.

annotation. A human readable comment.

initial-enable = CalculateableValue . Determines whether a control is initially visible. True by default. Could be data-bound to an SQL expression.

enabled = CalculateableValue . Determines whether a control is visible. True by default. Could be data-bound to an SQL expression.

instantiate = CalculateableValue . Determines whether a control instantiate its children. It is true by default. It could be data-bound to an SQL expression. Unlike most of calcucateable attributes, the value of this attribute for a control is calcluated

before calling 'on-init' event handler of the control; other values are calculated before calling 'before-data-bind' event handler.

control-udt = SqlName . At run time every control is represented as an instance of some user-defined type (UDT). VSPX compiles selects the UDT to use depending on name of the XML element that represents the control and maybe some of its attributes. In some specific cases the application developer may instruct VSPX compiler to use some other UDT, e.g. an application-specific UDT that is derived from the preset one. If specified, the value of 'control-udt' attribute should be equal to the name of the desired target UDT.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

column = SqlName . The name of the column bound. If nothing else is specified, the column meta data from the containing update form sets the field's attributes.

null-value. This value will be shown if value of the column is NULL, also value of the control will be set to null if this value is passed upon POST request.

error-glyph. The character to be displayed near the resulting HTML input element if the test of v:validator of the element detects an error in the current value of the element.

auto-submit. Flag to auto submit the parent form if value of the control is changed.

multiple. Used to designate a multiple selection list box. In this case the control's value will be an array of key data of selected items.

value = CalculateableValue . Data bindable value of control, represents value of HTML control to be drawn. The calculated value of this attribute is stored in the control in ufl_value field.

element-value = CalculateableValue . An XML entity that contains a value to be displayed by an HTML control. The calculated value is stored in ufl_element_value field of the control. This value is used only if ufl_value is null (e.g. if the 'value' attribute is not set at all), and the method vc_get_value_from_element() will be used to calculate ufl_value based on ufl_element_value, ufl_element_path and ufl_element_place.

element-place = CalculateableValue . The place of actual data inside an XML element. The calculated value of this attribute is stored in ufl_element_place field of the control. This is to process XMLSchema-compatible data without writing extra code for handling NULL values. The XML element can contain the value to be bound in either attribute or in an inner text node. XMLSchema treats missing attributes as NULL values. missing text node as an empty string value and if 'xsi:nil' attribute is 'true' then inner text value is NULL no matter if there are any text nodes. These self-evident rules are convenient for XQuery data retrieval but not for data update: an result of some XQuery expression can point to data but not to a place where data should occur in the future. The use of 'element-place' allows to eliminate the problem. Attributes 'element-value', 'element-path' and 'element-update-path' may locate an element that is always present whereas 'element-place' describes where the desired datum will occur in the element. The value of this attribute should be a string of syntax '@attributename' or 'text()'. If this is '@attributename' then the effective bound value is a value of the specified attribute or NULL if the specified attribute is missing. On update, either the attribute is created/edited in order to set non-NULL value or the attribute is deleted in order to set NULL. If text() is specified then both text value of the element and the value of its 'xsi:nil' are properly handled on both data bind and update.

element-path = CalculateableValue . This is the path inside the entity that comes from 'element-value' attribute. The default path is 'self::node()'. This path is used to point precisely to an subentity that should actually be used for data binding. The calculated value of this attribute should be a string in XQuery syntax, and it is saved in ufl_element_path member of the control.

element-params = CalculateableValue . This is the vector of parameter values of the path specified by 'element-path' attribute. The default is NULL indicating no parameters. The attribute is calculated but not used if 'element-path' attribute is missing. The calculated value of this attribute should be either NULL or a vector of even length whose items are parameter names and parameter values; refer to the description of xquery_eval() function for more details. The value of the attribute is saved in ufl_element_params member of the control.

element-update-path = CalculateableValue . Like the value of 'element-path', this is the path inside the entity that comes from 'element-value' attribute. The difference is that 'element-path' is used inside 'vc_get_value_from_element() during data-bind to set ufl_value based on ufl_element_value whereas 'element-update-path' is used during data update inside 'vc_put_value_to_element() in order to save data inside the document that is referenced by ufl_element_value. The default path for update is equal to the path for data bind, i.e. it is the value of the calculated 'element-path' attribute or 'self::node()'. The calculated value of 'element-update-path' should be a string in XQuery syntax, and it is saved in ufl_element_update_path member of the control.

element-update-params = CalculateableValue . This is the vector of parameter values of the path specified by 'element-update-path' attribute. The default is NULL indicating no parameters. The attribute is calculated but not used if 'element-update-path' attribute is missing (e.g. you should not try to specify 'element-path' without 'element-update-path' and both 'element-params' and 'element-update-params' in hope that values from 'element-update-params' will be passed to 'element-path' on update). The calculated value of this attribute should be either NULL or a vector of even length whose items are parameter names and parameter values; refer to the description of xquery_eval() function for more details. The value of the attribute is saved in ufl_element_update_params member of the control.

fmt-function = CalculateableValue . This is to convert the value that is bound to the control into a string to use during the rendering. The value of this attribute should be a name of function that takes a single argument of the type that matches the type of ufl_value and returns a string. If this is used then the value of 'cvt-function' should probably be a name of function that converts the string back to the desired data type. If a control supports 'format' attribute then the format is applied to the result of 'fmt-function', so the value returned by fmt-function may be e.g. an integer to be formatted by '%d' format. The calculated value of this attribute is stored in ufl_fmt_fn field of the control.

cvt-function = CalculateableValue . This is to convert a user input of type varchar into a value that should be stored in ufl_value (and e.g. placed into some database column by an update). The calculated value of this attribute is stored in ufl_cvt_fn field of the control. For more details, see the description of 'fmt_function' attribute above.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Declaration of type vspx_select_list

```
create type vspx_select_list under vspx_field
as (
     vsl_items any,
     vsl_item_values any,
     vsl_selected_inx any default null,
     vsl_change_script int default 0,
     vsl_list_document any default null,
     vsl_list_match varchar default null,
     vsl_list_key_path varchar default null,
     vsl_list_value_path varchar default null,
     vsl_multiple int default 0
   )
  temporary self as ref
 overriding method vc_render () returns any,
 overriding method vc_view_state (stream any, n int) returns any,
 overriding method vc_set_view_state (e vspx_event) returns any,
 overriding method vc_set_model () returns any,
 overriding method vc_xrender () returns any,
 method vs_set_selected () returns any,
constructor method vspx_select_list (name varchar, parent vspx_control)
```

Examples

Example 14.41. A simple static select list

The form of the sample page contains a list of three items.

```
<v:page name="select_list__0" xmlns:v="http://http://example.com/vspx/">
  <html>
    <head>
```

```
      <title>VSPX samples | v:select-list</title>
    </head>
    <body>
      <v:form name="form1" type="simple" action="" method="POST">
        <v:select-list name="sel_list">
          <v:item name="*default*" value="0"/>
          <v:item name="FIRST" value="1"/>
          <v:item name="SECOND" value="2"/>
        </v:select-list>
        <v:button name="submit1" action="simple" value="OK"/>
      </v:form>
    </body>
  </html>
</v:page>
```

See Also: Reference Material in the Tutorial:
VX-S-2

style

style — A markup tag for use by 'macro stylesheets'.

Synopsis

<**style** /> (*name* (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

This control marks the enclosed content for special processing in the 'macro stylesheet' whose name is specified by 'style' attribute of v:page element.

Attributes

name. Optional name of the control which could be used in the macro expansion stylesheet assigned.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

tab

tab — Selects one of its children to be active at any one time.

Synopsis

<tab /> (
name (required) ,
annotation (optional) ,
initial-enable (optional) ,
enabled (optional) ,
instantiate (optional) ,
control-udt (optional) ,
xsd-stub-xhtml (optional) ,
width (optional) ,
height (optional) ,
initial-active (optional) ,
style (optional) ,
active (optional) ,
is-input (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

A container that contains some number of pages and displays them one by one. This can be used for multi-page forms or Windows style tabbed decks or making multi-part forms, alternative visualizations of the same data etc.

Attributes

name = SqlName . A page level unique name identifying a control.

annotation. A human readable comment.

initial-enable = CalculateableValue . Determines whether a control is initially visible. True by default. Could be data-bound to an SQL expression.

enabled = CalculateableValue . Determines whether a control is visible. True by default. Could be data-bound to an SQL expression.

instantiate = CalculateableValue . Determines whether a control instantiate its children. It is true by default. It could be data-bound to an SQL expression. Unlike most of calcucateable attributes, the value of this attribute for a control is calcluated before calling 'on-init' event handler of the control; other values are calculated before calling 'before-data-bind' event handler.

control-udt = SqlName . At run time every control is represented as an instance of some user-defined type (UDT). VSPX compiles selects the UDT to use depending on name of the XML element that represents the control and maybe some of its attributes. In some specific cases the application developer may instruct VSPX compiler to use some other UDT, e.g. an application-specific UDT that is derived from the preset one. If specified, the value of 'control-udt' attribute should be equal to the name of the desired target UDT.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

initial-active = SqlName . This is the name of the child template which is active at the time and which thus will be rendered. Whether inactive templates are instantiated and keep state is controlled by is-input. The active attribute initializes the tb_active member of the vspx_tab instance. This is preserved in the view state. Data bindable.

style. This determines what HTML control is used to select which child is shown. If unspecified, there will be no such control and the program logic is responsible for flipping the pages. Otherwise "list" means there is a HTML select control showing the titles of the tabs. "radio" means the titles of the tabs are shown as a radio group. If there is a tab selector widget, it is always above the tabs.

active = CalculateableValue . A data bound value to enable or disable the tab switch. The default is '1' meaning 'enable'.

is-input. If true, all the children are considered to be collectively a multipart form and each will be instantiated when the page is made and will get to retain a view state. Although only one page of the form is shown at a time, the state of all is kept. Not data bindable. The default is '1' meaning 'template contains input'.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Declaration of type vspx_tab

Tab Deck

```
create type vspx_tab under vspx_form
as
   (
     tb_active vspx_template,
     tb_is_input int default 0,
     tb_style varchar
   )
    temporary self as ref
   constructor method vspx_tab (name varchar, parent vspx_control),
   --overriding method vc_render () returns any,
   overriding method vc_set_view_state (e vspx_event) returns any,
   overriding method vc_view_state (stream any, n int) returns any
```

Examples

Example 14.42. Switching of page content

The sample page contains three templates; only one of three is shown and user can switch them to change the visible content of the page.

```
<v:page name="tab__0" xmlns:v="http://http://example.com/vspx/">
  <html>
    <head><title>VSPX samples | v:tab</title></head>
    <body>
      <v:tab name="tab" initial-active="template1" style="list" is-input="0">
        <v:template name="template1" type="simple">This is a first template</v:template>
        <v:template name="template2" type="simple">This is a second template</v:template>
        <v:template name="template3" type="simple">This is a third template</v:template>
      </v:tab>
    </body>
  </html>
</v:page>
```

See Also: Reference Material in the Tutorial:

See Also: Reference Material in the Tutorial:

VX-S-5

template

template — Container for a group of controls and/or HTML code.

Synopsis

```
<template /> (   name (required) ,
                 annotation (optional) ,
                 initial-enable (optional) ,
                 enabled (optional) ,
                 instantiate (optional) ,
                 control-udt (optional) ,
                 xsd-stub-xhtml (optional) ,
                 width (optional) ,
                 height (optional) ,
                 type (optional) ,
                 redirect (optional) ,
                 condition (optional) ,
                 name-to-remove (optional) ,
                 set-to-remove (optional) ,
                 title (optional) ,
                 debug-srcfile (optional) ,
                 debug-srcline (optional) ,
                 debug-log (optional) ) ;
```

Description

The container for any optional, repeatable or otherwise grouped controls or code. The type modifier is used to specify special kind of templates (i.e. repeatable content or tree node representation)

Attributes

name = SqlName . A page level unique name identifying a control.

annotation. A human readable comment.

initial-enable = CalculateableValue . Determines whether a control is initially visible. True by default. Could be data-bound to an SQL expression.

enabled = CalculateableValue . Determines whether a control is visible. True by default. Could be data-bound to an SQL expression.

instantiate = CalculateableValue . Determines whether a control instantiate its children. It is true by default. It could be data-bound to an SQL expression. Unlike most of calcucateable attributes, the value of this attribute for a control is calcluated before calling 'on-init' event handler of the control; other values are calculated before calling 'before-data-bind' event handler.

control-udt = SqlName . At run time every control is represented as an instance of some user-defined type (UDT). VSPX compiles selects the UDT to use depending on name of the XML element that represents the control and maybe some of its attributes. In some specific cases the application developer may instruct VSPX compiler to use some other UDT, e.g. an application-specific UDT that is derived from the preset one. If specified, the value of 'control-udt' attribute should be equal to the name of the desired target UDT.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

type. The behaviour of the template.

Table 14.9. Allowed values of the 'type' attribute

simple	The template contains an arbitrary number of controls and HTML tags and groups them together to control their processing, e.g. to enable or disable them altogether depending on some condition.
repeat	A repeating row of v:data-set. The template of this type will be repeated for each row in the window of the data-set. It will be instantiated, data bound, rendered. If the template contains forms and submits, they get the post method called as one would expect. On data binding, the parent of the template has its dg_current_row data member set to an array representing the selected row from left to right.
row	A repeating row of v:data-grid. The template of this type will be repeated for each row in the window of the v:data-grid. It will be instantiated, data bound, rendered. If the template contains forms and submits, they get the post method called as one would expect. On data binding, the parent of the template has its dg_current_row data member set to an array representing the selected row from left to right. The controls under this template (such as text, label, button etc.) can access data in current row in order as columns are given by referencing te_rowset member of row template.
frame	A non-repeating static content of v:data-grid. Content of this template represents view in page when rows are selected, position of scroll buttons, rowset and form for adding a record.
if-exists	Template that is enabled only if the resultset is not empty in v:data-set.
if-not-exists	Template that is enabled only if the resultset is empty in v:data-set.
add	Template that is enabled when user adds new record in the resultset in v:data-set (if the resultset is not read-only). Similarly to templates of type 'edit', the 'add' template contains a vspx:form of type 'update' that is used to allow adding of rows in table as specified. Usually this form is a vspx_form + vspx_text constrained to columns and table selected in SQL expression of the parent vspx:data-set control with value of key attributes set to null. Please remember to set the 'if-not-exists' attribute of the form to 'insert'.
browse	This type is meaningful only for templates right inside templates of type 'repeat'. When the template of type 'repeat' is used multiple times to display every row of a data-set, the template of type 'browse' expands only for those rows that are not currently editable.
edit	Template that is enabled when user edits an existing record in the resultset in v:data-set (if the resultset isnot read-only and if the 'edit' attribute of v:data-set is set to 'true'). The template of this type usually contains a vspx:form of type 'update'; the 'table' attribute of the form is the name of the table from 'sql' attribute of the v:data-set where the 'edit' template is located; controls of the form are usually constrained to columns of the SQL expression of that 'sql' attribute.
if-login	Template that is enabled when user is logged in (e.g. for use in v:login. If the login is valid, the contents of this child will be instantiated. This can be for example a button with 'action' attribute set to 'logout', a welcome message or such. When using the vspx:login system, the user name is is obtained by connection_get ('vspx_user').
if-not-login	Template that is enabled when user is not yet logged in or is already logged out (e.g. for use in v:login. The template specifies what to do if there are no credentials with the page. If the 'redirect' attribute is given, then the entire page containing this is not processed at all but instead the page specified in the url is processed instead, with the context being that of the invocation of this page. The redirect page can thus ask for the login and having checked it return to this page, since it knows the url for this. In this way it is possible to bookmark places of which the uri's may expire, and accessing an expired place will just prompt for the login before going to the page. If the redirect is not specified, the content of the template is instantiated and shown. The content can be arbitrary, e.g. the vspx:login-form control can appear here, providing a standard login form that prompts for a user name and password and has a submit button.
tree-node	Template for internal nodes of the tree in v:tree (i.e. for nodes with children).
tree-leaf	Template for terminal nodes of the tree in v:tree (i.e. for nodes without children).
input	Template for input the SQL statement to execute it interactively in v:isql element.
result	Template to show if v:isql returns result without an error.
error	Template to be shown by v:isql if the user entered the SQL query and the execution of this query caused an error.

redirect. The URL to which the user agent is redirected if not authenticated, applicable only when type is 'if-no-login'.

condition = SqlCode . This is a SQL expression to be tested for rendering the template

name-to-remove. This works together with set-to-remove, see next.

set-to-remove. This combined with name-to-remove gives posibility to remove a HTML elements from output.

Table 14.10. Allowed values of the 'set-to-remove' attribute

none	The rendered output of the enclosing template will not be affected by the 'name-to-remove' attribute of the template.
top	The rendered output of the enclosing template should not contain the first opening tag whose name is equal to the value of 'name-to-remove' attribute of the template.
bottom	The rendered output of the enclosing template should not contain the last closing tag whose name is equal to the value of 'name-to-remove' attribute of the template.
both	The rendered output of the enclosing template should not contain the both the first opening tag and the last closing tag whose name is equal to the value of 'name-to-remove' attribute of the template.

title. When used inside tab control, this is used to show as label of the selector

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Declaration of type vspx_template

```
create type vspx_template under vspx_control
  temporary self as ref
 constructor method vspx_template (name varchar, parent vspx_control)
```

text

text — Scriptable, data-bindable input control.

Synopsis

```
<text /> (  name (required) ,
            annotation (optional) ,
            initial-enable (optional) ,
            enabled (optional) ,
            instantiate (optional) ,
            control-udt (optional) ,
            xsd-stub-xhtml (optional) ,
            width (optional) ,
            height (optional) ,
            column (optional) ,
            null-value (optional) ,
            error-glyph (optional) ,
            auto-submit (optional) ,
            type (optional) ,
            default (optional) ,
            value (optional) ,
            element-value (optional) ,
            element-place (optional) ,
            element-path (optional) ,
            element-params (optional) ,
            element-update-path (optional) ,
            element-update-params (optional) ,
            fmt-function (optional) ,
            cvt-function (optional) ,
            default-value (optional) ,
            default_value (optional) ,
            format (optional) ,
            debug-srcfile (optional) ,
            debug-srcline (optional) ,
            debug-log (optional) ) ;
```

Description

Text input, with scripts and validation options but no implied database binding. String input of the HTML form.

Attributes

name = SqlName . A page level unique name identifying a control.

annotation. A human readable comment.

initial-enable = CalculateableValue . Determines whether a control is initially visible. True by default. Could be data-bound to an SQL expression.

enabled = CalculateableValue . Determines whether a control is visible. True by default. Could be data-bound to an SQL expression.

instantiate = CalculateableValue . Determines whether a control instantiate its children. It is true by default. It could be data-bound to an SQL expression. Unlike most of calcucateable attributes, the value of this attribute for a control is calcluated before calling 'on-init' event handler of the control; other values are calculated before calling 'before-data-bind' event handler.

control-udt = SqlName . At run time every control is represented as an instance of some user-defined type (UDT). VSPX compiles selects the UDT to use depending on name of the XML element that represents the control and maybe some of its attributes. In some specific cases the application developer may instruct VSPX compiler to use some other UDT, e.g. an application-specific UDT that is derived from the preset one. If specified, the value of 'control-udt' attribute should be equal to the name of the desired target UDT.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

column = SqlName . The name of the column bound. If nothing else is specified, the column meta data from the containing update form sets the field's attributes.

null-value. This value will be shown if value of the column is NULL, also value of the control will be set to null if this value is passed upon POST request.

error-glyph. The character to be displayed near the resulting HTML input element if the test of v:validator of the element detects an error in the current value of the element.

auto-submit. Flag to auto submit the parent form if value of the control is changed.

type.

Table 14.11. Allowed values of the 'type' attribute

plain	The value is displayed in a usual way and user can edit it. This is the default value.
password	The value is not shown on the screen to prevent occasional reading, only astericks are shown in the input field.
hidden	The text is passed to the target page without displaying it to the user.

default = CalculateableValue . The default value of input field.

value = CalculateableValue . Data bindable value of control, represents value of HTML control to be drawn. The calculated value of this attribute is stored in the control in ufl_value field.

element-value = CalculateableValue . An XML entity that contains a value to be displayed by an HTML control. The calculated value is stored in ufl_element_value field of the control. This value is used only if ufl_value is null (e.g. if the 'value' attribute is not set at all), and the method vc_get_value_from_element() will be used to calculate ufl_value based on ufl_element_value, ufl_element_path and ufl_element_place.

element-place = CalculateableValue . The place of actual data inside an XML element. The calculated value of this attribute is stored in ufl_element_place field of the control. This is to process XMLSchema-compatible data without writing extra code for handling NULL values. The XML element can contain the value to be bound in either attribute or in an inner text node. XMLSchema treats missing attributes as NULL values. missing text node as an empty string value and if 'xsi:nil' attribute is 'true' then inner text value is NULL no matter if there are any text nodes. These self-evident rules are convenient for XQuery data retrieval but not for data update: an result of some XQuery expression can point to data but not to a place where data should occur in the future. The use of 'element-place' allows to eliminate the problem. Attributes 'element-value', 'element-path' and 'element-update-path' may locate an element that is always present whereas 'element-place' describes where the desired datum will occur in the element. The value of this attribute should be a string of syntax '@attributename' or 'text()'. If this is '@attributename'

then the effective bound value is a value of the specified attribute or NULL if the specified attribute is missing. On update, either the attribute is created/edited in order to set non-NULL value or the attribute is deleted in order to set NULL. If text() is specified then both text value of the element and the value of its 'xsi:nil' are properly handled on both data bind and update.

element-path = CalculateableValue . This is the path inside the entity that comes from 'element-value' attribute. The default path is 'self::node()'. This path is used to point precisely to an subentity that should actually be used for data binding. The calculated value of this attribute should be a string in XQuery syntax, and it is saved in ufl_element_path member of the control.

element-params = CalculateableValue . This is the vector of parameter values of the path specified by 'element-path' attribute. The default is NULL indicating no parameters. The attribute is calculated but not used if 'element-path' attribute is missing. The calculated value of this attribute should be either NULL or a vector of even length whose items are parameter names and parameter values; refer to the description of xquery_eval() function for more details. The value of the attribute is saved in ufl_element_params member of the control.

element-update-path = CalculateableValue . Like the value of 'element-path', this is the path inside the entity that comes from 'element-value' attribute. The difference is that 'element-path' is used inside 'vc_get_value_from_element() during data-bind to set ufl_value based on ufl_element_value whereas 'element-update-path' is used during data update inside 'vc_put_value_to_element() in order to save data inside the document that is referenced by ufl_element_value. The default path for update is equal to the path for data bind, i.e. it is the value of the calculated 'element-path' attribute or 'self::node()'. The calculated value of 'element-update-path' should be a string in XQuery syntax, and it is saved in ufl_element_update_path member of the control.

element-update-params = CalculateableValue . This is the vector of parameter values of the path specified by 'element-update-path' attribute. The default is NULL indicating no parameters. The attribute is calculated but not used if 'element-update-path' attribute is missing (e.g. you should not try to specify 'element-path' without 'element-update-path' and both 'element-params' and 'element-update-params' in hope that values from 'element-update-params' will be passed to 'element-path' on update). The calculated value of this attribute should be either NULL or a vector of even length whose items are parameter names and parameter values; refer to the description of xquery_eval() function for more details. The value of the attribute is saved in ufl_element_update_params member of the control.

fmt-function = CalculateableValue . This is to convert the value that is bound to the control into a string to use during the rendering. The value of this attribute should be a name of function that takes a single argument of the type that matches the type of ufl_value and returns a string. If this is used then the value of 'cvt-function' should probably be a name of function that converts the string back to the desired data type. If a control supports 'format' attribute then the format is applied to the result of 'fmt-function', so the value returned by fmt-function may be e.g. an integer to be formatted by '%d' format. The calculated value of this attribute is stored in ufl_fmt_fn field of the control.

cvt-function = CalculateableValue . This is to convert a user input of type varchar into a value that should be stored in ufl_value (and e.g. placed into some database column by an update). The calculated value of this attribute is stored in ufl_cvt_fn field of the control. For more details, see the description of 'fmt_function' attribute above.

default-value = CalculateableValue . This is an expression for setting the default value of the text.

default_value = CalculateableValue . This is deprecated alias for 'default-value' attribute (note the difference between minus sign and underscore).

format =

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Declaration of type vspx_text

```
create type vspx_text under vspx_field
as
 (
   tf_default varchar default '',
   tf_style any default 0 -- 0 is text, 1 is password, 2 is hidden
 )
```

```
 temporary self as ref
  overriding method vc_render () returns any,
  overriding method vc_set_view_state (e vspx_event) returns any,
  overriding method vc_view_state (stream any, n int) returns any,
  overriding method vc_set_model () returns any,
  overriding method vc_xrender () returns any,
  constructor method vspx_text (name varchar, parent vspx_control)
```

Examples

Example 14.43. Simple text input

The form contains text input control with 'Hello' string inside. User can edit the string and submit the changed value:

```
<v:page name="text__0" xmlns:v="http://http://example.com/vspx/">
  <html>
    <head>
      <title>VSPX samples | v:text</title>
    </head>
    <body>
      <v:form name="form1" type="simple" action="" method="POST">
        <v:text name="txt01" default="Hello"/>
        <v:button name="submit1" action="simple" value="OK"/>
      </v:form>
    </body>
  </html>
</v:page>
```

textarea

textarea — Scriptable text-area input.

Synopsis

<**textarea** /> (
name (required) ,
annotation (optional) ,
initial-enable (optional) ,
enabled (optional) ,
instantiate (optional) ,
control-udt (optional) ,
xsd-stub-xhtml (optional) ,
width (optional) ,
height (optional) ,
column (optional) ,
null-value (optional) ,
error-glyph (optional) ,
auto-submit (optional) ,
type (optional) ,
default_value (optional) ,
default (optional) ,
value (optional) ,
element-value (optional) ,
element-place (optional) ,
element-path (optional) ,
element-params (optional) ,
element-update-path (optional) ,
element-update-params (optional) ,
fmt-function (optional) ,
cvt-function (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

Scriptable, databindable HTML text area.

Attributes

name = SqlName . A page level unique name identifying a control.

annotation. A human readable comment.

initial-enable = CalculateableValue . Determines whether a control is initially visible. True by default. Could be data-bound to an SQL expression.

enabled = CalculateableValue . Determines whether a control is visible. True by default. Could be data-bound to an SQL expression.

instantiate = CalculateableValue . Determines whether a control instantiate its children. It is true by default. It could be data-bound to an SQL expression. Unlike most of calcucateable attributes, the value of this attribute for a control is calcluated before calling 'on-init' event handler of the control; other values are calculated before calling 'before-data-bind' event handler.

control-udt = SqlName . At run time every control is represented as an instance of some user-defined type (UDT). VSPX compiles selects the UDT to use depending on name of the XML element that represents the control and maybe some of its attributes. In some specific cases the application developer may instruct VSPX compiler to use some other UDT, e.g. an application-specific UDT that is derived from the preset one. If specified, the value of 'control-udt' attribute should be equal to the name of the desired target UDT.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

column = SqlName . The name of the column bound. If nothing else is specified, the column meta data from the containing update form sets the field's attributes.

null-value. This value will be shown if value of the column is NULL, also value of the control will be set to null if this value is passed upon POST request.

error-glyph. The character to be displayed near the resulting HTML input element if the test of v:validator of the element detects an error in the current value of the element.

auto-submit. Flag to auto submit the parent form if value of the control is changed.

type.

Table 14.12. Allowed values of the 'type' attribute

plain	The value is displayed in a usual way and user can edit it. This is the default value.
password	The value is not shown on the screen to prevent occasional reading, only astericks are shown in the input field.
hidden	The text is passed to the target page without displaying it to the user.

default_value = CalculateableValue . The default value of the field. Can be accessed via control.ufl_value n data bind stage.

default = CalculateableValue . Deprecated alias of 'default_value'.

value = CalculateableValue . Data bindable value of control, represents value of HTML control to be drawn. The calculated value of this attribute is stored in the control in ufl_value field.

element-value = CalculateableValue . An XML entity that contains a value to be displayed by an HTML control. The calculated value is stored in ufl_element_value field of the control. This value is used only if ufl_value is null (e.g. if the 'value' attribute is not set at all), and the method vc_get_value_from_element() will be used to calculate ufl_value based on ufl_element_value, ufl_element_path and ufl_element_place.

element-place = CalculateableValue . The place of actual data inside an XML element. The calculated value of this attribute is stored in ufl_element_place field of the control. This is to process XMLSchema-compatible data without writing extra code for handling NULL values. The XML element can contain the value to be bound in either attribute or in an inner text node. XMLSchema treats missing attributes as NULL values. missing text node as an empty string value and if 'xsi:nil' attribute is 'true' then inner text value is NULL no matter if there are any text nodes. These self-evident rules are convenient for XQuery data retrieval but not for data update: an result of some XQuery expression can point to data but not to a place where data should occur in the future. The use of 'element-place' allows to eliminate the problem. Attributes 'element-value', 'element-path' and 'element-update-path' may locate an element that is always present whereas 'element-place' describes where the desired datum will occur in the element. The value of this attribute should be a string of syntax '@attributename' or 'text()'. If this is '@attributename' then the effective bound value is a value of the specified attribute or NULL if the specified attribute is missing. On update, either

the attribute is created/edited in order to set non-NULL value or the attribute is deleted in order to set NULL. If text() is specified then both text value of the element and the value of its 'xsi:nil' are properly handled on both data bind and update.

element-path = CalculateableValue . This is the path inside the entity that comes from 'element-value' attribute. The default path is 'self::node()'. This path is used to point precisely to an subentity that should actually be used for data binding. The calculated value of this attribute should be a string in XQuery syntax, and it is saved in ufl_element_path member of the control.

element-params = CalculateableValue . This is the vector of parameter values of the path specified by 'element-path' attribute. The default is NULL indicating no parameters. The attribute is calculated but not used if 'element-path' attribute is missing. The calculated value of this attribute should be either NULL or a vector of even length whose items are parameter names and parameter values; refer to the description of xquery_eval() function for more details. The value of the attribute is saved in ufl_element_params member of the control.

element-update-path = CalculateableValue . Like the value of 'element-path', this is the path inside the entity that comes from 'element-value' attribute. The difference is that 'element-path' is used inside 'vc_get_value_from_element() during data-bind to set ufl_value based on ufl_element_value whereas 'element-update-path' is used during data update inside 'vc_put_value_to_element() in order to save data inside the document that is referenced by ufl_element_value. The default path for update is equal to the path for data bind, i.e. it is the value of the calculated 'element-path' attribute or 'self::node()'. The calculated value of 'element-update-path' should be a string in XQuery syntax, and it is saved in ufl_element_update_path member of the control.

element-update-params = CalculateableValue . This is the vector of parameter values of the path specified by 'element-update-path' attribute. The default is NULL indicating no parameters. The attribute is calculated but not used if 'element-update-path' attribute is missing (e.g. you should not try to specify 'element-path' without 'element-update-path' and both 'element-params' and 'element-update-params' in hope that values from 'element-update-params' will be passed to 'element-path' on update). The calculated value of this attribute should be either NULL or a vector of even length whose items are parameter names and parameter values; refer to the description of xquery_eval() function for more details. The value of the attribute is saved in ufl_element_update_params member of the control.

fmt-function = CalculateableValue . This is to convert the value that is bound to the control into a string to use during the rendering. The value of this attribute should be a name of function that takes a single argument of the type that matches the type of ufl_value and returns a string. If this is used then the value of 'cvt-function' should probably be a name of function that converts the string back to the desired data type. If a control supports 'format' attribute then the format is applied to the result of 'fmt-function', so the value returned by fmt-function may be e.g. an integer to be formatted by '%d' format. The calculated value of this attribute is stored in ufl_fmt_fn field of the control.

cvt-function = CalculateableValue . This is to convert a user input of type varchar into a value that should be stored in ufl_value (and e.g. placed into some database column by an update). The calculated value of this attribute is stored in ufl_cvt_fn field of the control. For more details, see the description of 'fmt_function' attribute above.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Declaration of type vspx_textarea

```
create type vspx_textarea under vspx_text  temporary self as ref
  constructor method vspx_textarea (name varchar, parent vspx_control),
  overriding method vc_xrender () returns any,
  overriding method vc_render () returns any
```

See Also: Reference Material in the Tutorial:
VX-S-2

tree

tree — A container for displaying a tree of nested nodes.

Synopsis

```
<tree /> (  name (required) ,
            annotation (optional) ,
            initial-enable (optional) ,
            enabled (optional) ,
            instantiate (optional) ,
            control-udt (optional) ,
            xsd-stub-xhtml (optional) ,
            width (optional) ,
            height (optional) ,
            show-root (optional) ,
            multi-branch (required) ,
            orientation (required) ,
            root (required) ,
            child-function (required) ,
            start-path (required) ,
            open-at (optional) ,
            debug-srcfile (optional) ,
            debug-srcline (optional) ,
            debug-log (optional) ) ;
```

Description

This can be used for hierarchical tables of contents, directory browsing, hierarchical menus and such. The tree can have either one or more branches open at any time, up to the leaves. The tree has a root, which may or may not be visible. The contents of the tree can either be fixed, in which case these are an XML tree, or dynamic, in which case these are generated level by level by SQL functions attached to the tree. The tree can have various external appearences. The tree is represented at run time by a vspx_tree instance. Nodes of the tree are represented by vspx_tree_node objects, which are childs of the vspx_tree. The nodes hold an identifier (vc_instance_name) of the corresponding tree branch, which is used to retrieve children of the node. These also hold a flag (tn_open member) indicating if the node is open or not.

Attributes

name = SqlName . A page level unique name identifying a control.

annotation. A human readable comment.

initial-enable = CalculateableValue . Determines whether a control is initially visible. True by default. Could be data-bound to an SQL expression.

enabled = CalculateableValue . Determines whether a control is visible. True by default. Could be data-bound to an SQL expression.

instantiate = CalculateableValue . Determines whether a control instantiate its children. It is true by default. It could be data-bound to an SQL expression. Unlike most of calcucateable attributes, the value of this attribute for a control is calcluated before calling 'on-init' event handler of the control; other values are calculated before calling 'before-data-bind' event handler.

control-udt = SqlName . At run time every control is represented as an instance of some user-defined type (UDT). VSPX compiles selects the UDT to use depending on name of the XML element that represents the control and maybe some of its

attributes. In some specific cases the application developer may instruct VSPX compiler to use some other UDT, e.g. an application-specific UDT that is derived from the preset one. If specified, the value of 'control-udt' attribute should be equal to the name of the desired target UDT.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

show-root. This attribute is obsolete and has no effect.

multi-branch. This determines whether more than one branches can be open at one time. The values are 0 and 1 (true and false). The default is 0 (false).

orientation. This can be 'horizontal' or 'vertical', the default is 'vertical'. The tree can have different styles. The default is a vertical arrangement with open levels indented, the children under the parent node. Each node template is a single line. The horizontal style shows each level on a separate line. In this case it is not allowed multiple open branches. The children of the root will be shown on one line. When one is opened, its children will fill the next line.

root = SqlName . This is a SQL expression which produces the root object. This can be of any data type, a file system path is an example.

```
create procedure root_node (in path varchar)
{
  declare i, l int;
  declare ret, arr any;
  arr :=
    vector_concat (sys_dirlist (path, 0), sys_dirlist (path, 1));

  return arr;
};
```

child-function = SqlName . Given the result of the root expression, this must generate an array of similar elements corresponing to the children of the node in question. This same function should be applicable to each element of the array it returns. If the array is empty then the node in question is a leaf.

```
-- Example of this function

create procedure child_node (in node_name varchar, in node varchar)
{
  declare i, l int;
  declare ret, arr any;
  declare exit handler for sqlstate '*'
    {
      return vector ();
    };
  if (isstring (file_stat (node_name, 3)))
    return vector ();

  arr :=
    vector_concat (sys_dirlist (node_name, 0), sys_dirlist (node_name, 1));

  return arr;
}
;
```

start-path = CalculateableValue . This is an expression which will be evaluated and passed to the 'root' function as argument.

open-at = CalculateableValue . This is an expression which will be used as XPath expression to designate which branches of the tree are open and which are not initially.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Declaration of type vspx_tree

```
create type vspx_tree under vspx_control
as
  (
    vt_current_node int default -1,
    vt_node any default null,
    vt_open_at varchar default null,
    vt_xpath_id varchar default null
  )
temporary self as ref
overriding method vc_view_state (stream any, n int) returns any,
method vc_get_state () returns any,
method vc_open_at (path varchar) returns any,
constructor method vspx_tree (name varchar, parent vspx_control)
```

Examples

Example 14.44.

```
<v:page name="demo_tree" xmlns:v="http://http://example.com/vspx/">
<html>
 <body>
 <v:form name="f1" method="POST" type="simple">
 <div>
 <v:tree name="tree1" multi-branch="1" orientation="vertical"
         root="root_node" start-path="." child-function="child_node" >
  <v:node-template name="node_tmpl">
    <div STYLE="margin-left:1em;">
        <v:button name="tree1_toggle" action="simple" style="image" value="--case (control.vc_parent as v
        <b><v:label name="label1" value="--(control.vc_parent as vspx_tree_node).tn_value" /></b>
        <v:node />
    </div>
  </v:node-template>
  <v:leaf-template name="leaf_tmpl">
    <div STYLE="margin-left:1em;">
        <v:label name="label2" value="--(control.vc_parent as vspx_tree_node).tn_value" />
    </div>
  </v:leaf-template>
 </v:tree>
 </div>
 </v:form>
 </body>
</html>
</v:page>
```

See Also: Reference Material in the Tutorial:
VX-S-4

url

url — Generic scriptable hypertext link.

Synopsis

<**url** /> (
name (required) ,
annotation (optional) ,
initial-enable (optional) ,
enabled (optional) ,
instantiate (optional) ,
control-udt (optional) ,
xsd-stub-xhtml (optional) ,
width (optional) ,
height (optional) ,
column (optional) ,
null-value (optional) ,
value (optional) ,
element-value (optional) ,
element-place (optional) ,
element-path (optional) ,
element-params (optional) ,
element-update-path (optional) ,
element-update-params (optional) ,
fmt-function (optional) ,
cvt-function (optional) ,
format (optional) ,
url (required) ,
active (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

Dynamic data bindable hypertext link.

Attributes

name = SqlName . A page level unique name identifying a control.

annotation. A human readable comment.

initial-enable = CalculateableValue . Determines whether a control is initially visible. True by default. Could be data-bound to an SQL expression.

enabled = CalculateableValue . Determines whether a control is visible. True by default. Could be data-bound to an SQL expression.

instantiate = CalculateableValue . Determines whether a control instantiate its children. It is true by default. It could be data-bound to an SQL expression. Unlike most of calcucateable attributes, the value of this attribute for a control is calcluated before calling 'on-init' event handler of the control; other values are calculated before calling 'before-data-bind' event handler.

control-udt = SqlName . At run time every control is represented as an instance of some user-defined type (UDT). VSPX compiles selects the UDT to use depending on name of the XML element that represents the control and maybe some of its attributes. In some specific cases the application developer may instruct VSPX compiler to use some other UDT, e.g. an application-specific UDT that is derived from the preset one. If specified, the value of 'control-udt' attribute should be equal to the name of the desired target UDT.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

column = SqlName . The name of the column bound. If nothing else is specified, the column meta data from the containing update form sets the field's attributes.

null-value. This value will be shown if value of the column is NULL, also value of the control will be set to null if this value is passed upon POST request.

value = CalculateableValue . Data bindable value of control, represents value of HTML control to be drawn. The calculated value of this attribute is stored in the control in ufl_value field.

element-value = CalculateableValue . An XML entity that contains a value to be displayed by an HTML control. The calculated value is stored in ufl_element_value field of the control. This value is used only if ufl_value is null (e.g. if the 'value' attribute is not set at all), and the method vc_get_value_from_element() will be used to calculate ufl_value based on ufl_element_value, ufl_element_path and ufl_element_place.

element-place = CalculateableValue . The place of actual data inside an XML element. The calculated value of this attribute is stored in ufl_element_place field of the control. This is to process XMLSchema-compatible data without writing extra code for handling NULL values. The XML element can contain the value to be bound in either attribute or in an inner text node. XMLSchema treats missing attributes as NULL values. missing text node as an empty string value and if 'xsi:nil' attribute is 'true' then inner text value is NULL no matter if there are any text nodes. These self-evident rules are convenient for XQuery data retrieval but not for data update: an result of some XQuery expression can point to data but not to a place where data should occur in the future. The use of 'element-place' allows to eliminate the problem. Attributes 'element-value', 'element-path' and 'element-update-path' may locate an element that is always present whereas 'element-place' describes where the desired datum will occur in the element. The value of this attribute should be a string of syntax '@attributename' or 'text()'. If this is '@attributename' then the effective bound value is a value of the specified attribute or NULL if the specified attribute is missing. On update, either the attribute is created/edited in order to set non-NULL value or the attribute is deleted in order to set NULL. If text() is specified then both text value of the element and the value of its 'xsi:nil' are properly handled on both data bind and update.

element-path = CalculateableValue . This is the path inside the entity that comes from 'element-value' attribute. The default path is 'self::node()'. This path is used to point precisely to an subentity that should actually be used for data binding. The calculated value of this attribute should be a string in XQuery syntax, and it is saved in ufl_element_path member of the control.

element-params = CalculateableValue . This is the vector of parameter values of the path specified by 'element-path' attribute. The default is NULL indicating no parameters. The attribute is calculated but not used if 'element-path' attribute is missing. The calculated value of this attribute should be either NULL or a vector of even length whose items are parameter names and parameter values; refer to the description of xquery_eval() function for more details. The value of the attribute is saved in ufl_element_params member of the control.

element-update-path = CalculateableValue . Like the value of 'element-path', this is the path inside the entity that comes from 'element-value' attribute. The difference is that 'element-path' is used inside 'vc_get_value_from_element() during data-bind to set ufl_value based on ufl_element_value whereas 'element-update-path' is used during data update inside 'vc_put_value_to_element() in order to save data inside the document that is referenced by ufl_element_value. The default path for update is equal to the path for data bind, i.e. it is the value of the calculated 'element-path' attribute or 'self::node()'. The calculated value of 'element-update-path' should be a string in XQuery syntax, and it is saved in ufl_element_update_path member of the control.

element-update-params = CalculateableValue . This is the vector of parameter values of the path specified by 'element-update-path' attribute. The default is NULL indicating no parameters. The attribute is calculated but not used if

'element-update-path' attribute is missing (e.g. you should not try to specify 'element-path' without 'element-update-path' and both 'element-params' and 'element-update-params' in hope that values from 'element-update-params' will be passed to 'element-path' on update). The calculated value of this attribute should be either NULL or a vector of even length whose items are parameter names and parameter values; refer to the description of xquery_eval() function for more details. The value of the attribute is saved in ufl_element_update_params member of the control.

fmt-function = CalculateableValue . This is to convert the value that is bound to the control into a string to use during the rendering. The value of this attribute should be a name of function that takes a single argument of the type that matches the type of ufl_value and returns a string. If this is used then the value of 'cvt-function' should probably be a name of function that converts the string back to the desired data type. If a control supports 'format' attribute then the format is applied to the result of 'fmt-function', so the value returned by fmt-function may be e.g. an integer to be formatted by '%d' format. The calculated value of this attribute is stored in ufl_fmt_fn field of the control.

cvt-function = CalculateableValue . This is to convert a user input of type varchar into a value that should be stored in ufl_value (and e.g. placed into some database column by an update). The calculated value of this attribute is stored in ufl_cvt_fn field of the control. For more details, see the description of 'fmt_function' attribute above.

format. A sprintf format string for printing the value

url = CalculateableValue . A data bound value to be printed in place of href attribute.

active = CalculateableValue . A data bound value to enable or disable the url. The default is '1' meaning 'enable'.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Declaration of type vspx_url

```
create type vspx_url under vspx_field
as
  (
    vu_format varchar default '%s',
    vu_url varchar default '',
    vu_l_pars varchar default '',
    vu_is_local int default 0
  )
temporary self as ref
overriding method vc_render () returns any,
constructor method vspx_url (name varchar, parent vspx_control)
```

Examples

Example 14.45. Simple dynamic link

This draws an anchor in browser screen.

```
<v:page name="url__0" xmlns:v="http://http://example.com/vspx/">
  <html>
    <head><title>VSPX samples | v:url</title></head>
    <body>
        <v:url name="url0" value="--'index'" url="--'index.vspx'" format="A link to page with name %s and
    </body>
  </html>
</v:page>
```

validator

validator — A validator that is applied to user input.

Synopsis

<**validator** /> (
name (required) ,
annotation (optional) ,
initial-enable (optional) ,
enabled (optional) ,
instantiate (optional) ,
control-udt (optional) ,
xsd-stub-xhtml (optional) ,
width (optional) ,
height (optional) ,
test (required) ,
min (optional) ,
max (optional) ,
regexp (optional) ,
expression (optional) ,
empty-allowed (optional) ,
message (optional) ,
runat (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

The validator objects are invoked when the element's control gets posted. the validator elements only make sense inside field or form types of controls. the validators are invoked in the order given, and the first one to fail stops the invocation chain, so that no later ones are attempted. furthermore the vc_is_valid member of the containing page class instance will be reset to false (0) to stop further processing. see also error-summary element and error-glyph attribute of field element.

Attributes

name = SqlName . A page level unique name identifying a control.

annotation. A human readable comment.

initial-enable = CalculateableValue . Determines whether a control is initially visible. True by default. Could be data-bound to an SQL expression.

enabled = CalculateableValue . Determines whether a control is visible. True by default. Could be data-bound to an SQL expression.

instantiate = CalculateableValue . Determines whether a control instantiate its children. It is true by default. It could be data-bound to an SQL expression. Unlike most of calcucateable attributes, the value of this attribute for a control is calcluated before calling 'on-init' event handler of the control; other values are calculated before calling 'before-data-bind' event handler.

control-udt = SqlName . At run time every control is represented as an instance of some user-defined type (UDT). VSPX compiles selects the UDT to use depending on name of the XML element that represents the control and maybe some of its attributes. In some specific cases the application developer may instruct VSPX compiler to use some other UDT, e.g. an application-specific UDT that is derived from the preset one. If specified, the value of 'control-udt' attribute should be equal to the

name of the desired target UDT.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

test. The type of test to be performed, can be 'length', 'value', 'regexp' or 'sql'.

min. A lower limit in value and length tests

max. The upper limit for value and length tests

regexp. The REGEXP pattern to match field value

expression. A SQL expression for validation

empty-allowed. If specified as true (1) this will allow submitting an empty field

message. The error message to be associated to parent control when the test represented by this validator fails.

runat. Where to perform validation, at server side after posting, or at browser side when entering the values. Note that client side validators can be assigned only to input controls, hence client side form validators for inter-field integrity testing are not allowed.

Client side validators will generate client side JavaScript.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Declaration of type vspx_validator

Examples

Example 14.46. Validation of text area input

The form contains a v:textarea control with v:validator assigned. When the OK button is pressed, data are posted back to the same URI so the page is instantiated again. If validator found a violation, the message is shown right after the text input. (The error reporting may be changed by using v:error-summary VSPX control and by adding 'error-glyph' attribute to the v:textarea.)

```
<v:page name="validator__0" xmlns:v="http://http://example.com/vspx/">
  <html>
    <head>
      <title>VSPX samples | v:validator</title>
    </head>
    <body>
      <v:form name="form1" type="simple" action="" method="POST">
        <v:textarea name="ta1" default="enter your text here" value="--coalesce ({?'ta1'}, control.ufl_va
          <v:validator test="length" min="0" max="50" message="The input length should not exceed 50 char
        </v:textarea>
        <v:button name="submit1" action="simple" value="OK"/>
      </v:form>
    </body>
  </html>
</v:page>
```

See Also: Reference Material in the Tutorial:
VX-S-2

variable

variable — Page variable - a user defined member of page class.

Synopsis

<variable /> (
name (required) ,
type (required) ,
default (optional) ,
persist (optional) ,
param-name (optional) ,
xsd-stub-xhtml (optional) ,
width (optional) ,
height (optional) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

This element declares a data member for the page subclass corresponding to the containing page. The value of this data member can be preserved between consecutive postbacks. The attribute "persist" determine how to save the variable's value: to session table for inter-page usage, keep in page state only or do not keep it at all. Note that inter-page variable storage is available only when a vspx:login control is included on the page and authentication is performed; also the name of page variables in corresponding pages needs to be same. There is no special class for this control because only a data member will be added. Also the page defines two special variables 'sid' and 'realm' for login control. So when an existing login control is is in authenticated state, these will contain values for session id and application realm. Also every non-repeating control of the page will be represented as a page variable and thus be acessible as 'self.<name_of_control>' anywhere in the VSPX page after page initialization.

Attributes

name = SqlName . The name of page class member.

type = SqlName . The SQL data type of the variable.

default = SqlCode . The default value. Must be a literal scalar value suitable for the default clause of a user defined type member.

persist.

Table 14.13. Allowed values of the 'persist' attribute

session	keep the variable in the session; needs a login control to be present
pagestate	keep the variable in page view state
temp	do not keep variable
0	same as 'pagestate'
1	same as 'session'

param-name = SqlName . This specifies an optional name of a URL parameter for setting the page variable value. When this is specified and there is a parameter matching the name, the variable is set after it. This is a shorthand for an explicit call of keyword_get.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Examples

Example 14.47. Declaration of page variable

The v:variable element defines a new member of page class and hance we can access it with self.mask in rest of VSPX code.

```
<v:page name="variable__0" xmlns:v="http://http://example.com/vspx/">
  <html>
    <head>
      <title>VSPX samples | v:variable</title>
    </head>
    <body>
      <v:variable name="mask" type="varchar" default="null"/>
      <?vsp self.mask := 'first value'; ?>
      <p>The variable value is set to '<?V self.mask ?>'</p>
      <?vsp self.mask := 'second value'; ?>
      <p>The variable value is set to '<?V self.mask ?>'</p>
    </body>
  </html>
</v:page>
```

vscx

vscx

Synopsis

<**vscx** /> (
name (required) ,
annotation (optional) ,
initial-enable (optional) ,
enabled (optional) ,
instantiate (optional) ,
control-udt (optional) ,
xsd-stub-xhtml (optional) ,
width (optional) ,
height (optional) ,
url (required) ,
debug-srcfile (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

Custom control encapsulated in a separate VSPX page. This is a wrapper for custom control instantiation. The target URL MUST contain a valid VSPX page reference. In this way repeatable pieces of code can be reused in form of instantiation of their page class, this is different from inclusion. The target page variables can be initialized as attributes in this control, where name of attribute is a name of variable of target page and value is an expression to be assigned.

Attributes

name = SqlName . A page level unique name identifying a control.

annotation. A human readable comment.

initial-enable = CalculateableValue . Determines whether a control is initially visible. True by default. Could be data-bound to an SQL expression.

enabled = CalculateableValue . Determines whether a control is visible. True by default. Could be data-bound to an SQL expression.

instantiate = CalculateableValue . Determines whether a control instantiate its children. It is true by default. It could be data-bound to an SQL expression. Unlike most of calcucateable attributes, the value of this attribute for a control is calcluated before calling 'on-init' event handler of the control; other values are calculated before calling 'before-data-bind' event handler.

control-udt = SqlName . At run time every control is represented as an instance of some user-defined type (UDT). VSPX compiles selects the UDT to use depending on name of the XML element that represents the control and maybe some of its attributes. In some specific cases the application developer may instruct VSPX compiler to use some other UDT, e.g. an application-specific UDT that is derived from the preset one. If specified, the value of 'control-udt' attribute should be equal to the name of the desired target UDT.

xsd-stub-xhtml. This attribute is for internal use only. It has no effect if added to the source VSPX file.

width. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of this attribute will not be used when the resulting HTML is rendered.

height. Visible width of the control when it is displayed in WYSIWYG tools when the source VSPX text is edited. The value of

this attribute will not be used when the resulting HTML is rendered.

url. Reference to a VSPX page to be instantiated as a control.

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Declaration of type vspx_vscx

```
create type vspx_vscx under vspx_form
self as ref temporary
constructor method vspx_vscx (name varchar, parent vspx_control, uri varchar),
overriding method vc_pre_render (stream any, n int) returns any
```

Examples

Example 14.48. Outer page, including a navigation bar via vscx__1.vspx

This page will embed another page twice, but not as simple inclusion. The second page is treated as a control and it's page class will be instantiated twice as as a child of outer page.

```
<v:page name="outer_page" xmlns:v="http://http://example.com/vspx/">
  <html>
    <body>
      <v:variable name="var1" type="int" default="0" />
      <h3>Embeeding a page as a custom control</h3>
      <v:form type="simple" name="f1">
        <div>
          <div>control1</div>
          <v:vscx name="bar1" url="vscx__1.vspx"/>
        </div>
        <div>
          <div>control2</div>
          <v:vscx name="bar2" url="vscx__1.vspx"/>
        </div>
        <div>
          <v:text name="txt2" />
          <v:button action="simple" value="Outer" name="b2" />
        </div>
      </v:form>
    </body>
  </html>
</v:page>
```

Example 14.49. Simple navigation bar used as custom control

This page is used inside vscx__0.vspx one as a custom control.

```
<v:page name="custom_bar" xmlns:v="http://http://example.com/vspx/">
  <v:variable name="test" type="int" default="12" />
  <table border="1" cellspacing="1" cellpadding="0">
    <tr>
      <td>
        <v:url name="u1" value="Home" url="--http_path ()" /> |
      </td>
      <td>
        <v:url name="u2" value="Settings" url="--http_path ()||'?settings'" /> |
      </td>
      <td>
        <v:url name="u3" value="Posts" url="--http_path ()||'?posts'" /> |
      </td>
      <td>
        <v:url name="u4" value="Maintenance" url="--http_path ()||'?maint'" />
```

```
        </td>
      </tr>
    </table>
    <div>
      <v:form type="simple" name="f1">
        <v:text name="txt1" />
        <v:button name="b1" action="simple" value="Inner"/>
      </v:form>
    </div>
  </v:page>
```

xsd-stub

xsd-stub — This is for internal use only.

Synopsis

<**xsd-stub** /> (*debug-srcfile* (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

This control should never appear in the VSPX source. It is for internal use only. Before applying XML schema validation to the page, Virtuoso replaces non-VSPX tags with this one when they reside inside v:page.

Attributes

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Declaration of type vspx_xsd_stub

```
create type vspx_xsd_stub under vspx_control
as ( vc_stub any ) temporary self as ref
```

xsd-stub-script

xsd-stub-script — This is for internal use only.

Synopsis

<**xsd-stub-script** /> (*debug-srcfile* (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

This control should never appear in the VSPX source. It is for internal use only. Before applying XML schema validation to the page, Virtuoso replaces non-VSPX tags with this one when they reside inside event script tag such as v:on-init.

Attributes

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Declaration of type vspx_xsd_stub_script

```
create type vspx_xsd_stub_script under vspx_control
as ( vc_stub any ) temporary self as ref
```

xsd-stub-top

xsd-stub-top — This is for internal use only.

Synopsis

<**xsd-stub-top** /> (*debug-srcfile* (optional) ,
debug-srcline (optional) ,
debug-log (optional)) ;

Description

This control should never appear in the VSPX source. It is for internal use only. Before applying XML schema validation to the page, Virtuoso replaces non-VSPX tags with this one when they reside outside v:page.

Attributes

debug-srcfile. URI of the source document where the tag comes from.

debug-srcline. Line number in the source document where the tag comes from.

debug-log. This defines what sort of data are saved to the debugging log.

Declaration of type vspx_xsd_stub_top

```
create type vspx_xsd_stub_top under vspx_control
as ( vc_stub any ) temporary self as ref
```

14.4.10. XForms rendering

The VSPX form controls such as form, button, select-list, text etc. can be rendered as XForms analogues using a special connection variable 'RenderXForms' set to true (integer 1). Also XML post data coming from XForms capable agents will be parsed and provided to the VSPX page methods in usual form (name/value array). The HTML form elements substitution to the XForms is as follows: input type="text" - xforms:input; input type="password" - xforms:secret; textarea - xforms:textarea; select - xforms:select1; select multiple - xforms:select; checkbox - xforms:input (of datatype boolean); input type="radio" - xforms:select1 appearance="full"; input type="submit" - xforms:submit; form - xforms:model. Furthermore validators which belongs to a input elements (not to a containers) will enforce XMLSchema types of the XForms model. Note that some XForms agents may need special object registering or a special Content-Type reported, so as this varies from agent to agent the application logic is responsible for setting them properly.

```
<v:page xmlns:v="http://example.com/vspx/" name="xform_demo">
    <v:on-init>
        -- enable XForms rendering
        connection_set ('RenderXForms', 1);
    </v:on-init>
    ... page content follows ...
</v:page>
```

14.4.11. XMLSchema for VSPX page

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://example.com/vspx/" xmlns:v="http://example.com/vspx/" xmlns:xs="http:/
  <xs:simpleType name="CalculateableValue" final="restriction">
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="preserve"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="ButtonStyle" final="restriction">
    <xs:restriction base="xs:string">
      <xs:enumeration value="submit"/>
      <xs:enumeration value="url"/>
      <xs:enumeration value="image"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="ButtonAction" final="restriction">
    <xs:restriction base="xs:string">
      <xs:enumeration value="simple"/>
      <xs:enumeration value="submit"/>
      <xs:enumeration value="delete"/>
      <xs:enumeration value="browse"/>
      <xs:enumeration value="return"/>
      <xs:enumeration value="logout"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="SqlName" final="restriction">
    <xs:restriction base="xs:string">
      <xs:minLength value="1" fixed="false"/>
      <xs:maxLength value="32" fixed="false"/>
      <xs:pattern value="[A-Za-z0-9_]{1,32}"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="SqlTableQname" final="restriction">
    <xs:restriction base="xs:string">
      <xs:pattern value="[A-Za-z0-9_]{1,32}\.[A-Za-z0-9_]{0,32}\.[A-Za-z0-9_]{1,32}"/>
      <xs:pattern value="[A-Za-z0-9_]{1,32}"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="SqlCode" final="restriction">
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="preserve"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="LoginMode" final="restriction">
    <xs:restriction base="xs:string">
      <xs:enumeration value="digest"/>
      <xs:enumeration value="url"/>
      <xs:enumeration value="cookie"/>
    </xs:restriction>
```

```
</xs:simpleType>
<xs:simpleType name="Unused" final="restriction">
  <xs:restriction base="xs:string"/>
</xs:simpleType>
<xs:simpleType name="TreeOrientation" final="restriction">
  <xs:restriction base="xs:string">
    <xs:pattern value="horizontal"/>
    <xs:pattern value="vertical"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="PlCursorType" final="restriction">
  <xs:restriction base="xs:string">
    <xs:enumeration value="static"/>
    <xs:enumeration value="dynamic"/>
    <xs:enumeration value="keyset"/>
  </xs:restriction>
</xs:simpleType>
<xs:attributeGroup name="HtmlGenAttributes">
  <xs:anyAttribute processContents="strict"/>
</xs:attributeGroup>
<xs:attributeGroup name="SqlGenAttributes">
  <xs:attribute name="name" type="v:SqlName" use="required"/>
  <xs:attribute name="annotation" type="xs:string" use="optional"/>
  <xs:attribute name="initial-enable" type="v:CalculateableValue" use="optional"/>
  <xs:attribute name="enabled" type="v:CalculateableValue" use="optional"/>
  <xs:attributeGroup ref="v:HtmlGenAttributes"/>
  <xs:anyAttribute namespace="##other" processContents="skip"/>
</xs:attributeGroup>
<xs:attributeGroup name="SqlColumn">
  <xs:attribute name="column" type="v:SqlName" use="optional"/>
</xs:attributeGroup>
<xs:attributeGroup name="UserInputAttributes">
  <xs:attribute name="error-glyph" type="xs:string" use="optional"/>
</xs:attributeGroup>
<xs:attributeGroup name="LoginParams">
  <xs:attribute name="realm" type="xs:NMTOKEN" use="required"/>
  <xs:attribute name="mode" type="v:LoginMode" use="required"/>
  <xs:attribute name="user-password" type="xs:NMTOKEN" use="required"/>
  <xs:attribute name="user-password-check" type="xs:NMTOKEN" use="required"/>
</xs:attributeGroup>
<xs:attributeGroup name="BrowseButtonParams">
  <xs:attribute name="child-window-options" type="xs:string" use="optional"/>
  <xs:attribute name="browser-current" type="xs:int" use="optional" default="0"/>
  <xs:attribute name="browser-filter" type="xs:string" use="optional" default="*"/>
  <xs:attribute name="browser-list" type="xs:string" use="optional" default="1"/>
  <xs:attribute name="browser-mode" type="xs:string" use="optional" default="RES"/>
  <xs:attribute name="browser-type" type="xs:string" use="optional"/>
  <xs:attribute name="browser-xfer" type="xs:string" use="optional" fixed="DOM"/>
  <xs:attribute name="selector" type="xs:anyURI" use="optional"/>
</xs:attributeGroup>
<xs:group name="AnyHtmlContent">
  <xs:choice>
    <xs:element ref="v:style"/>
    <xs:element ref="v:placeholder"/>
    <xs:any namespace="##other" processContents="skip"/>
  </xs:choice>
</xs:group>
<xs:complexType name="EventHandler" mixed="true">
  <xs:choice minOccurs="0">
    <xs:element ref="v:script"/>
  </xs:choice>
</xs:complexType>
<xs:element name="after-data-bind" type="v:EventHandler"/>
<xs:element name="before-data-bind" type="v:EventHandler"/>
<xs:element name="on-post" type="v:EventHandler"/>
<xs:element name="before-render" type="v:EventHandler"/>
<xs:element name="on-init" type="v:EventHandler"/>
<xs:group name="EventTarget">
  <xs:choice>
    <xs:element ref="v:after-data-bind"/>
    <xs:element ref="v:before-data-bind"/>
    <xs:element ref="v:on-post"/>
    <xs:element ref="v:before-render"/>
```

```
      <xs:element ref="v:on-init"/>
    </xs:choice>
  </xs:group>
  <xs:element name="page">
    <xs:complexType mixed="true">
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:group ref="v:EventTarget"/>
        <xs:group ref="v:AnyVspxPageContent"/>
        <xs:any/>
      </xs:choice>
      <xs:attribute name="name" type="v:SqlName" use="required"/>
      <xs:attribute name="decor" type="xs:anyURI" use="optional"/>
      <xs:attribute name="style" type="xs:anyURI" use="optional"/>
      <xs:attribute name="on-error-redirect" type="xs:anyURI" use="optional"/>
      <xs:attribute name="on-deadlock-retry" type="xs:integer" use="optional" default="0"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="include">
    <xs:complexType>
      <xs:attribute name="url" type="xs:anyURI" use="required"/>
      <xs:attribute name="name" type="v:Unused" use="optional"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="template">
    <xs:complexType mixed="true">
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:group ref="v:EventTarget"/>
        <xs:group ref="v:AnyVspxPageContent"/>
      </xs:choice>
      <xs:attributeGroup ref="v:SqlGenAttributes"/>
      <xs:attribute name="type" type="v:TemplateType" use="optional"/>
      <xs:attribute name="redirect" type="xs:anyURI" use="optional"/>
      <xs:attribute name="condition" type="v:SqlCode" use="optional"/>
      <xs:attribute name="name-to-remove" type="xs:QName" use="optional"/>
      <xs:attribute name="set-to-remove" type="v:TemplateSetToRemove" use="optional"/>
    </xs:complexType>
  </xs:element>
  <xs:simpleType name="TemplateType" final="restriction">
    <xs:restriction base="xs:string">
      <xs:enumeration value="simple"/>
      <xs:enumeration value="repeat"/>
      <xs:enumeration value="row"/>
      <xs:enumeration value="frame"/>
      <xs:enumeration value="if-exists"/>
      <xs:enumeration value="if-not-exists"/>
      <xs:enumeration value="add"/>
      <xs:enumeration value="browse"/>
      <xs:enumeration value="edit"/>
      <xs:enumeration value="if-login"/>
      <xs:enumeration value="if-not-login"/>
      <xs:enumeration value="tree-node"/>
      <xs:enumeration value="tree-leaf"/>
      <xs:enumeration value="input"/>
      <xs:enumeration value="result"/>
      <xs:enumeration value="error"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="TemplateSetToRemove" final="restriction">
    <xs:restriction base="xs:string">
      <xs:enumeration value="none"/>
      <xs:enumeration value="top"/>
      <xs:enumeration value="bottom"/>
      <xs:enumeration value="both"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="form">
    <xs:complexType mixed="true">
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:group ref="v:EventTarget"/>
        <xs:group ref="v:FormSpecificContent"/>
        <xs:group ref="v:FormNonSpecificContent"/>
        <xs:group ref="v:UserInputTarget"/>
        <xs:group ref="v:AnyHtmlContent"/>
```

```
      <xs:element ref="v:template"/>
    </xs:choice>
    <xs:attributeGroup ref="v:SqlGenAttributes"/>
    <xs:attribute name="type" type="v:FormType" use="optional"/>
    <xs:attribute name="action" type="xs:string" use="optional"/>
    <xs:attribute name="method" type="v:FormMethod" use="optional" default="POST"/>
    <xs:attribute name="table" type="v:SqlTableQname" use="optional"/>
    <xs:attribute name="if-not-exists" type="v:FormUpdateIfNotExists" use="optional"/>
    <xs:attribute name="concurrency" type="xs:boolean" use="optional"/>
  </xs:complexType>
</xs:element>
<xs:simpleType name="FormType" final="restriction">
  <xs:restriction base="xs:string">
    <xs:enumeration value="simple"/>
    <xs:enumeration value="update"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="FormMethod" final="restriction">
  <xs:restriction base="xs:string">
    <xs:enumeration value="GET"/>
    <xs:enumeration value="POST"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="VariableStorage" final="restriction">
  <xs:restriction base="xs:string">
    <xs:enumeration value="session"/>
    <xs:enumeration value="pagestate"/>
    <xs:enumeration value="temp"/>
    <xs:enumeration value="0"/>
    <xs:enumeration value="1"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="FormUpdateIfNotExists" final="restriction">
  <xs:restriction base="xs:string">
    <xs:enumeration value="insert"/>
    <xs:enumeration value="nothing"/>
  </xs:restriction>
</xs:simpleType>
<xs:element name="tab">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:group ref="v:EventTarget"/>
      <xs:group ref="v:AnyHtmlContent"/>
      <xs:element ref="v:template"/>
    </xs:choice>
    <xs:attributeGroup ref="v:SqlGenAttributes"/>
    <xs:attribute name="initial-active" type="v:SqlName" use="optional"/>
    <xs:attribute name="style" type="xs:string" use="optional"/>
  </xs:complexType>
</xs:element>
<xs:element name="script">
  <xs:complexType mixed="true">
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:any namespace="##any"/>
    </xs:sequence>
    <xs:attribute name="language" type="v:SqlName" use="optional"/>
  </xs:complexType>
</xs:element>
<xs:element name="variable">
  <xs:complexType mixed="true">
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:any namespace="##any"/>
    </xs:sequence>
    <xs:attribute name="name" type="v:SqlName" use="required"/>
    <xs:attribute name="type" type="v:SqlName" use="required"/>
    <xs:attribute name="default" type="v:SqlCode" use="optional"/>
    <xs:attribute name="persist" type="v:VariableStorage" use="optional" default="pagestate"/>
  </xs:complexType>
</xs:element>
<xs:element name="validator">
  <xs:complexType>
    <xs:attributeGroup ref="v:SqlGenAttributes"/>
    <xs:attribute name="test" type="xs:string" use="required"/>
```

```
    <xs:attribute name="min" type="xs:string" use="optional"/>
    <xs:attribute name="max" type="xs:string" use="optional"/>
    <xs:attribute name="regexp" type="xs:string" use="optional"/>
    <xs:attribute name="empty-allowed" type="xs:boolean" use="optional"/>
    <xs:attribute name="message" type="xs:string" use="required"/>
    <xs:attribute name="runat" type="v:ValidatorType" use="optional" default="server"/>
  </xs:complexType>
</xs:element>
<xs:simpleType name="ValidatorType" final="restriction">
  <xs:restriction base="xs:string">
    <xs:enumeration value="server"/>
    <xs:enumeration value="client"/>
  </xs:restriction>
</xs:simpleType>
<xs:group name="UserInputTarget">
  <xs:choice>
    <xs:element ref="v:validator"/>
  </xs:choice>
</xs:group>
<xs:element name="field">
  <xs:complexType>
    <xs:attributeGroup ref="v:SqlGenAttributes"/>
  </xs:complexType>
</xs:element>
<xs:element name="button">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:group ref="v:EventTarget"/>
      <xs:element ref="v:field"/>
    </xs:choice>
    <xs:attributeGroup ref="v:SqlGenAttributes"/>
    <xs:attributeGroup ref="v:SqlColumn"/>
    <xs:attribute name="action" type="v:ButtonAction" use="required"/>
    <xs:attribute name="value" type="v:CalculateableValue" use="required"/>
    <xs:attribute name="style" type="v:ButtonStyle" use="optional"/>
    <xs:attribute name="active" type="v:CalculateableValue" use="optional"/>
    <xs:attributeGroup ref="v:BrowseButtonParams"/>
  </xs:complexType>
</xs:element>
<xs:element name="radio-button">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:group ref="v:EventTarget"/>
    </xs:choice>
    <xs:attributeGroup ref="v:SqlGenAttributes"/>
    <xs:attributeGroup ref="v:SqlColumn"/>
    <xs:attributeGroup ref="v:UserInputAttributes"/>
    <xs:attribute name="group-name" type="xs:NCName" use="required"/>
    <xs:attribute name="value" type="v:CalculateableValue" use="required"/>
    <xs:attribute name="initial-checked" type="xs:integer" use="optional" default="0"/>
  </xs:complexType>
</xs:element>
<xs:element name="check-box">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:group ref="v:EventTarget"/>
    </xs:choice>
    <xs:attributeGroup ref="v:SqlGenAttributes"/>
    <xs:attributeGroup ref="v:SqlColumn"/>
    <xs:attributeGroup ref="v:UserInputAttributes"/>
    <xs:attribute name="value" type="v:CalculateableValue" use="required"/>
    <xs:attribute name="group-name" type="v:SqlName" use="optional"/>
    <xs:attribute name="initial-checked" type="xs:integer" use="optional" default="0"/>
  </xs:complexType>
</xs:element>
<xs:element name="text">
  <xs:complexType mixed="false">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:group ref="v:EventTarget"/>
      <xs:group ref="v:UserInputTarget"/>
    </xs:choice>
    <xs:attributeGroup ref="v:SqlGenAttributes"/>
    <xs:attributeGroup ref="v:SqlColumn"/>
```

```
    <xs:attributeGroup ref="v:UserInputAttributes"/>
    <xs:attribute name="type" type="v:TextInputType" use="optional" default="plain"/>
    <xs:attribute name="default" type="v:CalculateableValue" use="optional"/>
    <xs:attribute name="value" type="v:CalculateableValue" use="optional"/>
  </xs:complexType>
</xs:element>
<xs:element name="textarea">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:group ref="v:EventTarget"/>
      <xs:group ref="v:UserInputTarget"/>
    </xs:choice>
    <xs:attributeGroup ref="v:SqlGenAttributes"/>
    <xs:attributeGroup ref="v:SqlColumn"/>
    <xs:attributeGroup ref="v:UserInputAttributes"/>
    <xs:attribute name="type" type="v:TextInputType" use="optional" default="plain"/>
    <xs:attribute name="default" type="v:CalculateableValue" use="optional"/>
    <xs:attribute name="value" type="v:CalculateableValue" use="optional"/>
  </xs:complexType>
</xs:element>
<xs:simpleType name="TextInputType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="plain"/>
    <xs:enumeration value="password"/>
    <xs:enumeration value="hidden"/>
  </xs:restriction>
</xs:simpleType>
<xs:element name="item">
  <xs:complexType>
    <xs:attribute name="name" type="v:SqlCode" use="required"/>
    <xs:attribute name="value" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="select-list">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:group ref="v:EventTarget"/>
      <xs:element ref="v:item"/>
    </xs:choice>
    <xs:attributeGroup ref="v:SqlGenAttributes"/>
    <xs:attributeGroup ref="v:SqlColumn"/>
    <xs:attributeGroup ref="v:UserInputAttributes"/>
  </xs:complexType>
</xs:element>
<xs:element name="key">
  <xs:complexType>
    <xs:attribute name="column" type="v:SqlName" use="required"/>
    <xs:attribute name="value" type="v:CalculateableValue" use="required"/>
    <xs:attribute name="default" type="v:CalculateableValue" use="optional" default="null"/>
  </xs:complexType>
</xs:element>
<xs:group name="FormSpecificContent">
  <xs:choice>
    <xs:element ref="v:button"/>
    <xs:element ref="v:radio-group"/>
    <xs:element ref="v:radio-button"/>
    <xs:element ref="v:check-box"/>
    <xs:element ref="v:select-list"/>
    <xs:element ref="v:data-list"/>
    <xs:element ref="v:textarea"/>
    <xs:element ref="v:text"/>
    <xs:element ref="v:key"/>
    <xs:element ref="v:error-summary"/>
    <xs:element ref="v:calendar"/>
  </xs:choice>
</xs:group>
<xs:element name="label">
  <xs:complexType>
    <xs:attributeGroup ref="v:SqlGenAttributes"/>
    <xs:attributeGroup ref="v:SqlColumn"/>
    <xs:attribute name="value" type="v:CalculateableValue" use="required"/>
    <xs:attribute name="format" type="xs:string" use="optional"/>
  </xs:complexType>
```

```
</xs:element>
<xs:element name="url">
  <xs:complexType>
    <xs:attributeGroup ref="v:SqlGenAttributes"/>
    <xs:attributeGroup ref="v:SqlColumn"/>
    <xs:attribute name="value" type="v:CalculateableValue" use="required"/>
    <xs:attribute name="format" type="xs:string" use="optional"/>
    <xs:attribute name="url" type="v:CalculateableValue" use="required"/>
    <xs:attribute name="active" type="v:CalculateableValue" use="optional"/>
  </xs:complexType>
</xs:element>
<xs:element name="data-list">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:group ref="v:EventTarget"/>
    </xs:choice>
    <xs:attributeGroup ref="v:SqlGenAttributes"/>
    <xs:attributeGroup ref="v:SqlColumn"/>
    <xs:attribute name="table" type="v:SqlTableQname" use="required"/>
    <xs:attribute name="key" type="v:SqlName" use="required"/>
    <xs:attribute name="value" type="v:SqlName" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="node">
  <xs:complexType>
    <xs:attribute name="void" type="xs:string" use="optional"/>
  </xs:complexType>
</xs:element>
<xs:element name="tree">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:group ref="v:EventTarget"/>
      <xs:element ref="v:template"/>
    </xs:choice>
    <xs:attributeGroup ref="v:SqlGenAttributes"/>
    <xs:attribute name="show-root" type="xs:boolean" use="required"/>
    <xs:attribute name="multi-branch" type="xs:boolean" use="required"/>
    <xs:attribute name="orientation" type="v:TreeOrientation" use="required"/>
    <xs:attribute name="root-function" type="v:SqlName" use="required"/>
    <xs:attribute name="child-function" type="v:SqlName" use="required"/>
    <xs:attribute name="start-path" type="v:CalculateableValue" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="error-summary">
  <xs:complexType>
    <xs:attribute name="match" type="xs:string" use="optional"/>
  </xs:complexType>
</xs:element>
<xs:group name="FormNonSpecificContent">
  <xs:choice>
    <xs:element ref="v:variable"/>
    <xs:element ref="v:label"/>
    <xs:element ref="v:url"/>
    <xs:element ref="v:data-grid"/>
    <xs:element ref="v:data-set"/>
    <xs:element ref="v:tab"/>
    <xs:element ref="v:tree"/>
    <xs:element ref="v:include"/>
    <xs:element ref="v:isql"/>
  </xs:choice>
</xs:group>
<xs:element name="login-form">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:group ref="v:EventTarget"/>
      <xs:group ref="v:AnyVspxPageContent"/>
    </xs:choice>
    <xs:attributeGroup ref="v:SqlGenAttributes"/>
    <xs:attribute name="required" type="xs:boolean" use="required"/>
    <xs:attribute name="title" type="xs:string" use="required"/>
    <xs:attribute name="user-title" type="xs:string" use="required"/>
    <xs:attribute name="password-title" type="xs:string" use="required"/>
    <xs:attribute name="submit-title" type="xs:string" use="required"/>
```

```
  </xs:complexType>
</xs:element>
<xs:element name="login">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:group ref="v:EventTarget"/>
      <xs:group ref="v:FormNonSpecificContent"/>
      <xs:group ref="v:FormSpecificContent"/>
      <xs:element ref="v:template"/>
      <xs:element ref="v:login-form"/>
    </xs:choice>
    <xs:attributeGroup ref="v:SqlGenAttributes"/>
    <xs:attribute name="realm" type="xs:string" use="required"/>
    <xs:attribute name="mode" type="xs:string" use="required"/>
    <xs:attribute name="user-password" type="xs:string" use="optional"/>
    <xs:attribute name="user-password-check" type="xs:string" use="optional"/>
  </xs:complexType>
</xs:element>
<xs:group name="AnyVspxPageContent">
  <xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:group ref="v:FormSpecificContent"/>
    <xs:group ref="v:FormNonSpecificContent"/>
    <xs:group ref="v:AnyHtmlContent"/>
    <xs:element ref="v:node"/>
    <xs:element ref="v:form"/>
    <xs:element ref="v:template"/>
    <xs:element ref="v:hidden"/>
    <xs:element ref="v:login"/>
    <xs:element ref="v:login-form"/>
  </xs:choice>
</xs:group>
<xs:element name="column">
  <xs:complexType>
    <xs:attribute name="name" type="v:SqlName" use="required"/>
    <xs:attribute name="label" type="v:CalculateableValue" use="optional"/>
    <xs:attribute name="input-format" type="xs:string" use="optional"/>
    <xs:attribute name="output-format" type="xs:string" use="optional"/>
    <xs:attributeGroup ref="v:HtmlGenAttributes"/>
  </xs:complexType>
</xs:element>
<xs:element name="data-set">
  <xs:complexType mixed="true">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:group ref="v:EventTarget"/>
      <xs:group ref="v:AnyVspxPageContent"/>
      <xs:element ref="v:column"/>
      <xs:element ref="v:param"/>
      <xs:element ref="v:key"/>
    </xs:choice>
    <xs:attributeGroup ref="v:SqlGenAttributes"/>
    <xs:attribute name="sql" type="v:SqlCode" use="optional"/>
    <xs:attribute name="data-source" type="v:CalculateableValue" use="optional"/>
    <xs:attribute name="nrows" type="v:CalculateableValue" use="required"/>
    <xs:attribute name="scrollable" type="xs:boolean" use="required"/>
    <xs:attribute name="cursor-type" type="v:PlCursorType" use="optional" default="dynamic"/>
    <xs:attribute name="edit" type="xs:boolean" use="optional" default="false"/>
  </xs:complexType>
</xs:element>
<xs:element name="param">
  <xs:complexType>
    <xs:attribute name="name" type="v:SqlName" use="required"/>
    <xs:attribute name="value" type="v:CalculateableValue" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="data-grid">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:group ref="v:EventTarget"/>
      <xs:group ref="v:AnyVspxPageContent"/>
      <xs:element ref="v:column"/>
      <xs:element ref="v:param"/>
      <xs:element ref="v:key"/>
    </xs:choice>
```

```
    <xs:attributeGroup ref="v:SqlGenAttributes"/>
    <xs:attribute name="data" type="v:CalculateableValue"/>
    <xs:attribute name="meta" type="v:CalculateableValue"/>
    <xs:attribute name="nrows" type="v:CalculateableValue"/>
    <xs:attribute name="sql" type="v:SqlCode" use="required"/>
    <xs:attribute name="scrollable" type="xs:boolean"/>
    <xs:attribute name="cursor-type" type="v:PlCursorType"/>
    <xs:attribute name="edit" type="xs:boolean"/>
    <xs:anyAttribute namespace="##any"/>
  </xs:complexType>
</xs:element>
<xs:element name="isql">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:group ref="v:EventTarget"/>
      <xs:element ref="v:template"/>
    </xs:choice>
    <xs:attributeGroup ref="v:SqlGenAttributes"/>
    <xs:attribute name="isolation" type="v:IsqlIsolation" use="optional"/>
    <xs:attribute name="timeout" type="v:CalculateableValue" use="optional"/>
    <xs:attribute name="maxrows" type="v:CalculateableValue" use="optional"/>
    <xs:attribute name="user" type="v:CalculateableValue" use="optional"/>
    <xs:attribute name="password" type="v:CalculateableValue" use="optional"/>
  </xs:complexType>
</xs:element>
<xs:simpleType name="IsqlIsolation">
  <xs:restriction base="xs:string">
    <xs:enumeration value="committed"/>
  </xs:restriction>
</xs:simpleType>
<xs:element name="radio-group">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:group ref="v:EventTarget"/>
      <xs:group ref="v:AnyVspxPageContent"/>
    </xs:choice>
    <xs:attributeGroup ref="v:SqlGenAttributes"/>
    <xs:attributeGroup ref="v:UserInputAttributes"/>
  </xs:complexType>
</xs:element>
<xs:element name="row-template">
  <xs:complexType>
    <xs:attributeGroup ref="v:SqlGenAttributes"/>
  </xs:complexType>
</xs:element>
<xs:element name="tree-node">
  <xs:complexType>
    <xs:attributeGroup ref="v:SqlGenAttributes"/>
  </xs:complexType>
</xs:element>
<xs:element name="hidden">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:group ref="v:AnyVspxPageContent"/>
    </xs:choice>
    <xs:attributeGroup ref="v:HtmlGenAttributes"/>
  </xs:complexType>
</xs:element>
<xs:element name="style">
  <xs:complexType mixed="true">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:group ref="v:AnyVspxPageContent"/>
    </xs:choice>
    <xs:attribute name="name" type="xs:NMTOKEN" use="optional"/>
  </xs:complexType>
</xs:element>
<xs:element name="placeholder">
  <xs:complexType/>
</xs:element>
<xs:element name="calendar">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:group ref="v:EventTarget"/>
```

```
        <xs:group ref="v:AnyVspxPageContent"/>
        <xs:element ref="v:param"/>
      </xs:choice>
      <xs:attributeGroup ref="v:SqlGenAttributes"/>
      <xs:attribute name="inital-date" type="v:CalculateableValue"/>
      <xs:anyAttribute namespace="##any"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="expression">
    <xs:complexType mixed="true">
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:any namespace="##any"/>
      </xs:sequence>
      <xs:attribute name="language" type="v:SqlName" use="optional"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="data-source">
    <xs:complexType mixed="true">
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:group ref="v:EventTarget"/>
        <xs:group ref="v:AnyVspxPageContent"/>
        <xs:element ref="v:column"/>
        <xs:element ref="v:param"/>
        <xs:element ref="v:expression"/>
      </xs:choice>
      <xs:attributeGroup ref="v:SqlGenAttributes"/>
      <xs:attribute name="expression-type" type="xs:string" use="required"/>
      <xs:attribute name="nrows" type="v:CalculateableValue" use="required"/>
      <xs:attribute name="initial-offset" type="v:CalculateableValue" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

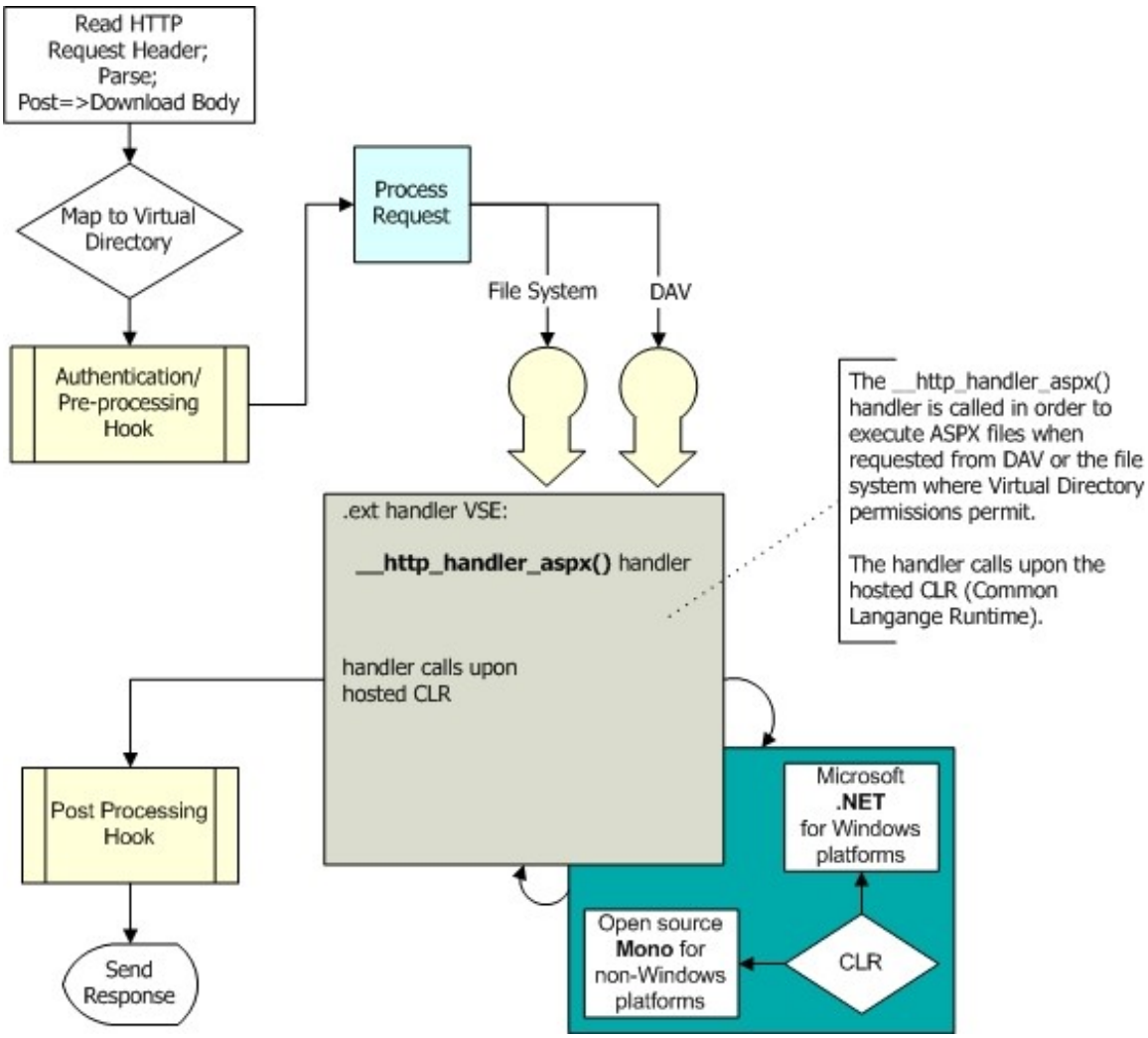
14.5. Deploying ASP.Net Web Applications

Virtuoso is a CLR host. It is responsible for initializing the runtime, defining the application domain (its runtime boundaries, security settings, etc), and executing user code within such domains. Windows can be used as the development platform, a very developer friendly environment with a rich set of tools, but you will not be restricted to Windows, .NET and IIS to run the assemblies produced. Where .NET is not readily available or desired Virtuoso contains Mono, an open source re-implementation of Microsoft .NET, a vehicle taking .NET cross-platform.

The CLR run-time is a part of the Virtuoso process. The Mono run time may be run either as in-process or as out-of-process. Hosted applications can make use of the regular Virtuoso .Net data provider to access Virtuoso SQL data and procedures. Microsoft ASPX files can also be run directly from Virtuoso either from the file system or WebDAV. Each of these capabilities releases you from the Microsoft platforms without compromising your development platform.

The Virtuoso CLR hosting is implemented using the VSEI.

Figure 14.16. The HTTP ASP.Net handler



See Also:

CLR Host Environment Setup

VSEI

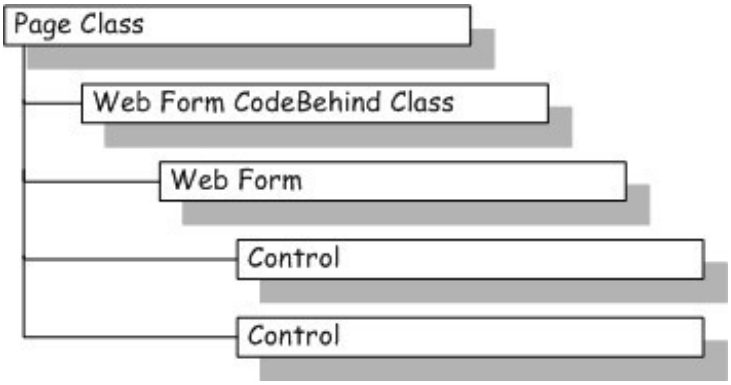
In-Process Data Access Client

14.5.1. Programming Concepts

ASP.Net Web Forms are divided into two sections: the user interface and the application logic. The user interface comprises HTML markup and ASP.Net web controls whereas the application logic contains all the programming code that allows the controls to interact with themselves and the server back-end. This provides the level of abstractions required for dynamic efficient Web application design. The interface can be altered without any backward or forward dependence on the code (logic). The Web Form interface should be created with the .aspx extension. The application logic can be contained in-line within the ASPX application, but developers should strive to keep the code in a separate location known as the "CodeBehind". This is a file that contains the logic (code) for the Web Form which should end in an extension appropriate for the programming language used, such as .vb for Visual Basic or .cs for C#. The CodeBehind can be written in any language for which there is a .Net compiler for. The compiler generates bytecode that can be deployed anywhere the .Net runtime exists.

The ASPX Web Form is compiled into an object that takes its place on a tree of controls and classes. The CodeBehind is compiled into an object on this tree when the page is requested. The Web Form must inherit from a "base-class" defined in the CodeBehind class file.

Figure 14.17. ASPX Web Form class hierarchy



Now we will create a new Web Form containing an HTML heading and an ASP.Net DataGrid control that will display results from the local Virtuoso server.

Example 14.50. Sample .Net Web Application: VirtTest.aspx

```
<%@ Page Language="vb" Inherits="VirtTest" Src="VirtTest.aspx.vb" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
        <head>
                <title>Virtuoso Provider to DataGrid in VB</title>
        </head>
        <body>
        <h1>Simple VB Virtuoso DataBinding Demo</h1>
                <form
xml:id="form1" method="post" runat="server">
                        <asp:DataGrid id=DataGrid1 runat="server" DataMember="Customers"
                                BorderColor="silver"
                                BorderWidth="1"
                                CellPadding="2"
                                Font-Name="Tahoma"
                                Font-Size="10pt">
                          <HeaderStyle BackColor="#2222ff" ForeColor="yellow"/>
                          <PagerStyle Mode="NextPrev" />
                        </asp:DataGrid>
                </form>
        <hr>
        </body>
</html>
```

On the first line of the file we use the `@ page` directive to inherit the CodeBehind class we will create in the CodeBehind file. The source of which can be optionally declared here with the `Src=` attribute. Visual Studio makes use of `CodeBehind=` for tracking associated source code instead. If either of these two attributes are specified then the .Net runtime will attempt to compile the code into an assembly (DLL) upon first execution if it does not already exist or seems out-of-date. The following code fragment is the CodeBehind, VirtTest.aspx.vb.

Now we want some logic behind the Web Form. We have placed a DataGrid control on a web page but we need to supply data to it now:

Example 14.51. Sample .Net Web Application CodeBehind: VirtTest.aspx.vb

```
imports System
imports System.Web
imports System.Web.UI
imports System.Web.UI.WebControls
imports System.Web.UI.HtmlControls
imports System.Data

imports OpenLink.Data.VirtuosoClient

Public Class VirtTest
    Inherits Page
```

```
    Protected WithEvents myConnection As OpenLink.Data.VirtuosoClient.VirtuosoConnection
    Protected WithEvents myCommand As OpenLink.Data.VirtuosoClient.VirtuosoDataAdapter
    Protected WithEvents DataGrid1 As System.Web.UI.WebControls.DataGrid
    Protected WithEvents ds As System.Data.DataSet

    Private Sub Page_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        Dim myConnection As new VirtuosoConnection("HOST=noodle:1112;UID=dba;PWD=dba;Database=Demo")
        Dim myCommand As new VirtuosoDataAdapter("select * from Demo..Customers", myConnection)

            Dim ds As new DataSet()
        myCommand.Fill(ds, "Customers")

        DataGrid1.DataSource=ds.Tables("Customers")
        DataGrid1.DataBind()

    End Sub

End Class
```

This file is compiled using the following command (split across lines for readability and would otherwise all be on one line):

```
E:\myweb\myapp>
  vbc /target:library
    /r:System.dll
    /r: System.Web.dll
    /r:System.Data.dll
    /r:OpenLink.Data.VirtuosoClient.dll
    /r:System.Xml.dll
    /out:bin\VirtTest.dll
    VirtTest.aspx.vb
```

Once compiled, the resulting DLL library should be in the bin subdirectory of the application root. When the assembly is available in this way the .aspx file does not need to contain the `Src=` attribute since there will be no need to compile the CodeBehind. If the `Src=` attribute is specified in the .aspx file but the CodeBehind file is not located on the server then an error will be signalled indicating that a required resource cannot be found, because the CodeBehind class is compiled on demand.

14.5.2. ASP.Net Deployment & Configuration

ASP.Net allows multiple Web applications to run on the same machine independently of each other. Web applications have their own directory for components (the ".\bin" directory) and its own XML-based configuration file (the ".config" file). This allows us to develop robust applications using custom configurations and components (or different versions of components) for each one.

Note:

The ASP.Net machine-wide configuration is held in machine.config found in the `(C:\Winnt\Microsoft.NET\)Framework\[version number]\CONFIG directory` depending on your installation. Applications hosting or making use of .Net may have an application configuration [App Name].config. Web applications store their individual configurations in Web.config stored in the Web application root directory.

Each ASP.Net application uses a local assembly cache to hold application specific logic (assemblies, contained in compiled DLLs). The \bin directory located in the application root is the local assembly cache for Web applications. This directory is denied access directly from browsers. This prevents users from downloading or executing any DLLs contained within.

When a Web application is started the .Net Framework constructs a new instance of the `System.AppDomain` class. When the instance is constructed it creates in-memory shadow copies of the DLLs in the \bin directory. The .Net Framework keeps an in-memory cache (shadow) of all assemblies used so that the actual files (.DLLs) are not locked. The .Net Framework will monitor the original DLL for changes. When changes occur the in-memory cache is updated. Applications already loaded and running with calls in-progress to the old library will continue using the old version so there is no interruption. All new calls to library will be effected, and use the new library. This means that you are free to replace the DLL with new versions as required. Previously with ASP you were required to stop IIS, register the DLL with RegServ32.exe and restart the Web server.

When new libraries are added to the .Net Framework for use in any application system wide, such as the Virtuoso Data Provider, these need to be added to the Global Assembly Cache (GAC). This is achieved using the `gacutil.exe` utility found in the `Framework\[version number]` directory. The `gacutil.exe` tool registers the DLL along with its version. One of .Net's strengths is its ability to maintain multiple versions of DLLs for multiple applications. This prevents "DLL-hell"; new DLLs

breaking old applications. Applications can also maintain their own local versions of DLLs in their \bin directory.

```
gacutil.exe /i <full path and file name>.dll
```

Registers assemblies with the Global Assembly Cache.

```
gacutil.exe /l
```

Can be used to list registered assemblies.

After registering assemblies using the `gacutil.exe` tool you must add an `<add assembly="..." />` entry into the `<configuration><system.web><compilation><assemblies>` section of the machine.config.

The application given in the previous section can be deployed to Virtuoso in the WebDAV repository or directly on file system under the VSPRoot directory. You should copy the directory structure as defined, applications in the root of the new directory and DLLs in a bin subdirectory. A virtual directory must be configured with execute permissions enabled. The application can be tested with a Web browser.

14.5.3. The Mono Project

The Mono Project is an open source version of the Microsoft.NET development platform. Incorporating key .NET compliant components, including a C# compiler, a Common Language Runtime just-in-time compiler, and a full suite of class libraries, the Mono Project will enable developers to create .NET applications and run them on Windows or any Mono-supported platform. Besides greatly improving the efficiency of development in the open source world, the Mono Project will allow the creation of operating-system-independent programs. Although primarily developed on Linux, Mono is being ported to as many operating systems as possible.

Microsoft.NET development tools, which include the C# compiler and Common Language Infrastructure (CLI), let programs written in C# and other languages run on non-Windows operating systems. The Mono Project development platform provides open source developers with a true "build once, deploy anywhere" tool-set taking advantage of the myriad of services enabled by Microsoft.NET.

The Mono Project will provide three key elements in a development framework designed to allow developers to quickly create, deploy and run .NET compatible applications on non-Windows platform. A C# compiler extending the GNOME development platform will allow Linux developers to create .NET compatible applications. These developers will also be able to build upon a complete implementation of class libraries compatible with the Microsoft CLI, enabling developers to create end-user applications as well as powerful web services using the database functionality available on open source systems. Portation of Mono yields versions of the Microsoft Common Language Run-Time (CLR) just-in-time (JIT) run-time engine will allow non-Windows systems to run .NET applications built on Windows, Linux or UNIX platforms.

The example in the previous section can be run using the CLR hosting ability of Virtuoso, but only with the precompiled assembly which has to be deployed into the Mono/lib directory of the Mono installation.

The Virtuoso installer on non-Windows platforms for which there is a port of Mono available installs the latest Mono CLR. This is required for the Virtuoso server to host .Net applications on non-Windows platforms.

Only a compiler for C#. Inline ASPX code using C# is a safe way to write and experience ASPX hosted from Virtuoso. Although this goes completely against the programming practice guidelines for .Net

The only permissible Application Domain is the Mono application. ASP.Net applications should have their own self containable Application Domain - the ./bin directory. Mono does not support this as of yet. The work around is to place all assemblies into the Mono domain.

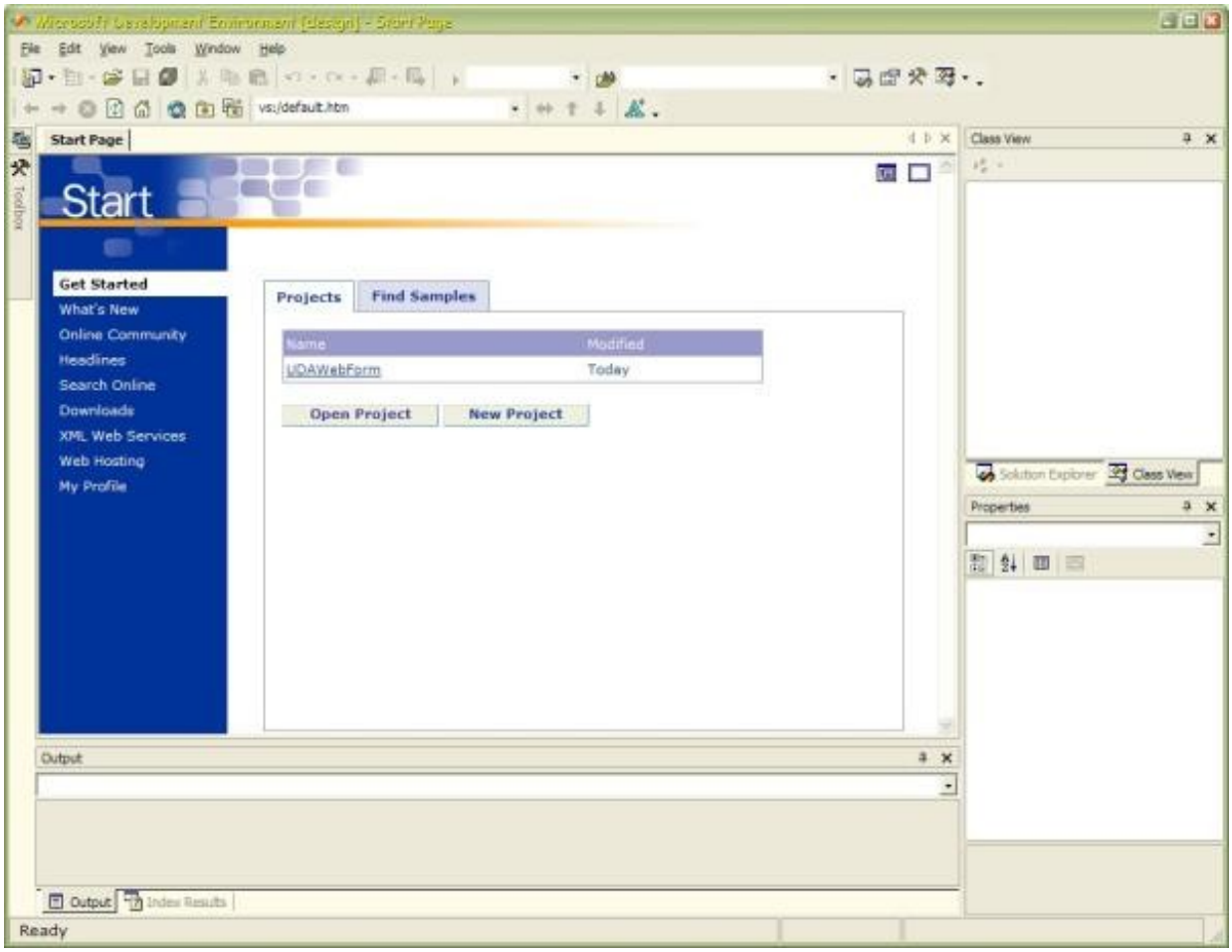
14.5.4. Migrating ASP.Net Applications to Virtuoso

Creating a Simple Data Bound Application

This section will describe how to use the MS Visual Studio to create an application using as much drag and drop as possible. We want to create a table of data in a web page from a database.

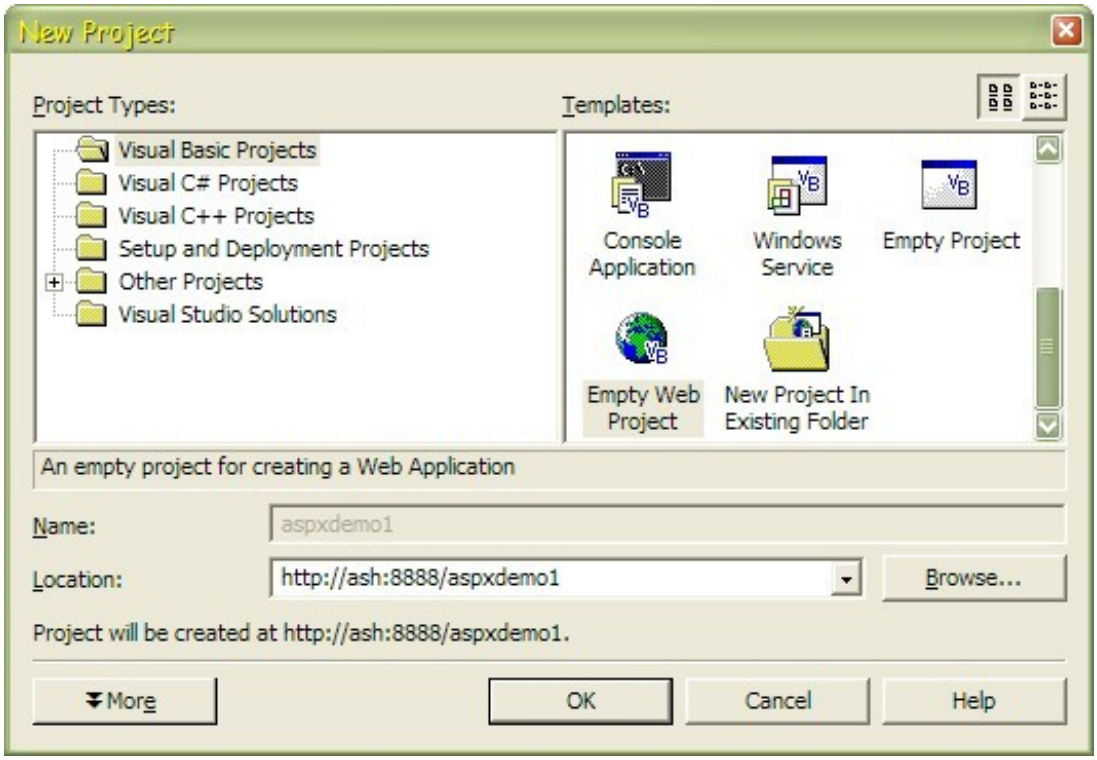
1. **Launch Microsoft's Visual Studio.** Once Visual Studio has been launch it present you with the start page that lets you create a New Project.

Figure 14.18. Databound Examples using MS Visual Studio



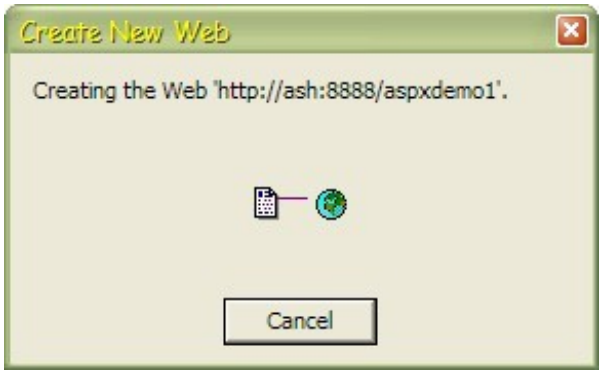
2. **Create a new Empty Web Project .** From the templates select a new Empty Web Project found in the Visual Basic Projects type. You must also select a location on an IIS web server.

Figure 14.19. Databound Examples using MS Visual Studio



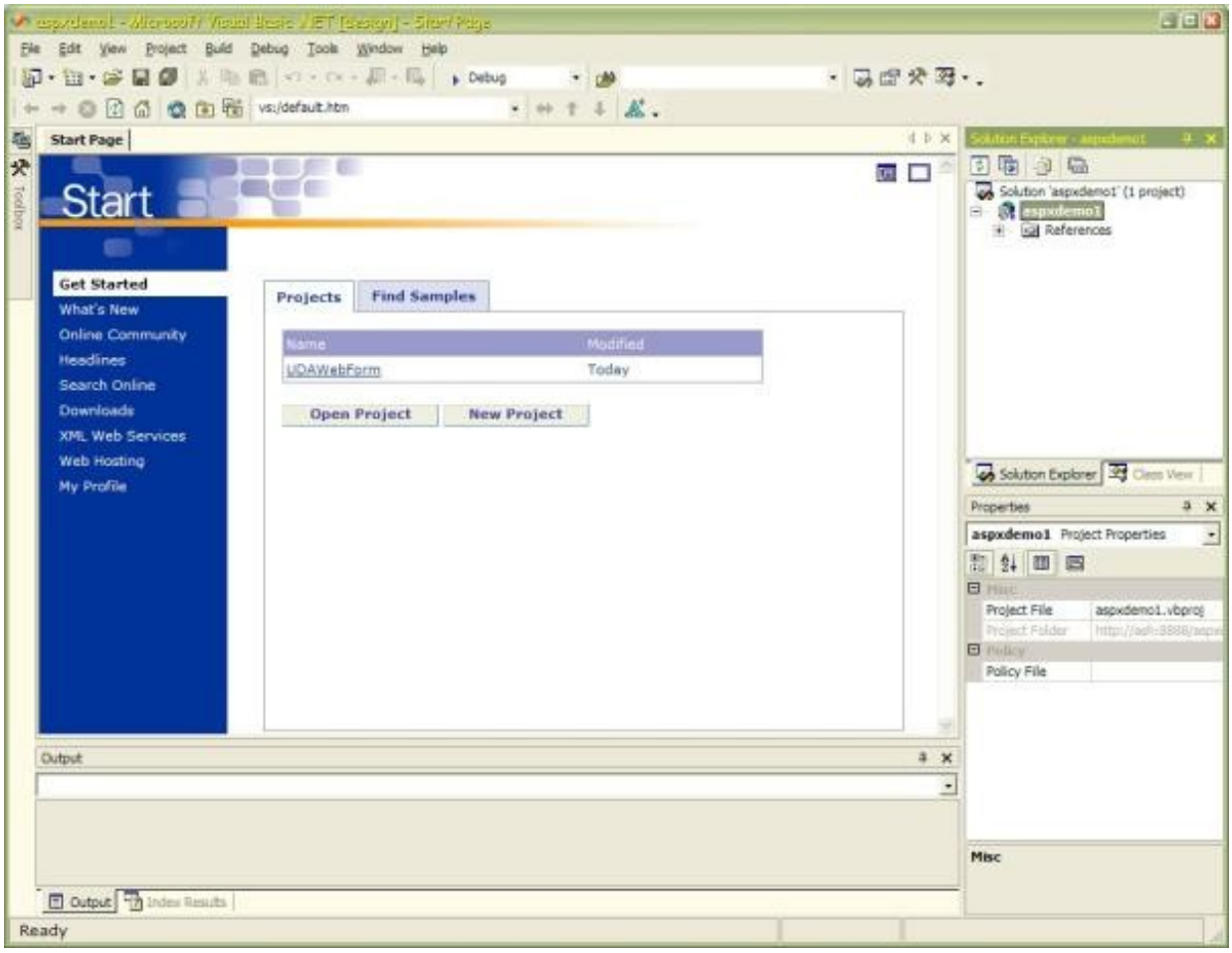
3. **Wait for IIS application to be set-up.** You will be shown a progress dialogue as Visual Studio contacts your web server creating a new web application there.

Figure 14.20. Databound Examples using MS Visual Studio



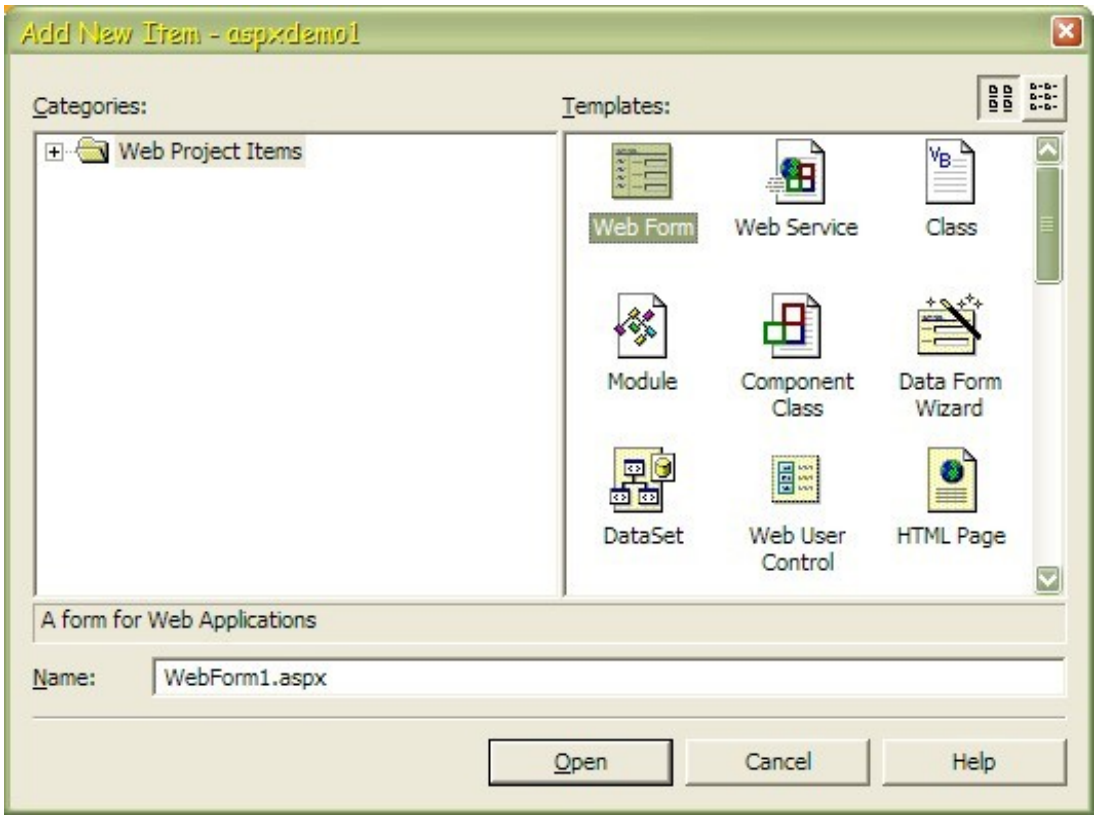
4. **Add New Web Form.** When the application-space has been configured on the web server you will be returned to the main Visual Studio windows. Now we must create our page. Right-click on the project name in the Solution Explorer and select Add New Web Form.

Figure 14.21. Databound Examples using MS Visual Studio



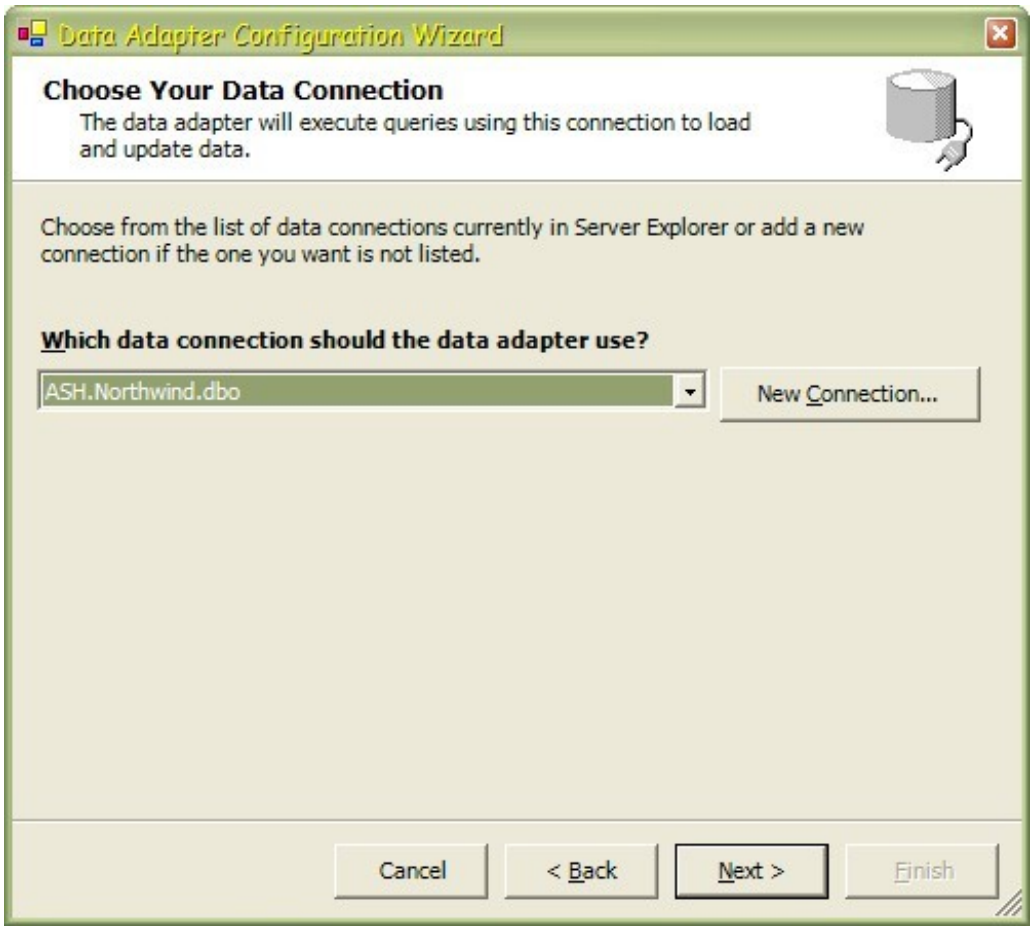
5. **New Web Form.** The Add New Item option will appear, automatically selecting the "Web Form" item to add. Supply a name for the page and click Open to begin.

Figure 14.22. Databound Examples using MS Visual Studio



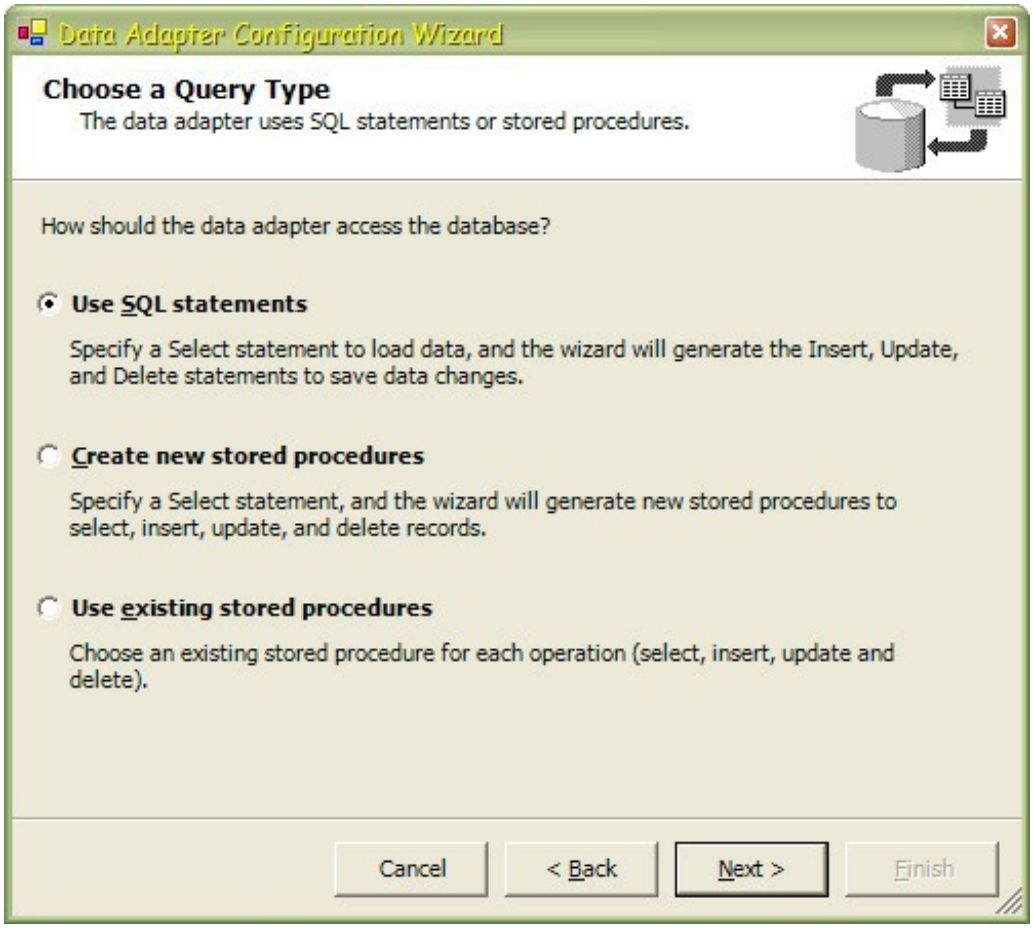
6. **Add a SqlDataAdapter Control.** With the fresh "Web Form" we can start. From the ToolBox double-click on the SqlDataAdapter in the Data panel. This will start the Data Adapter Configuration Wizard . After reading the description on the first panel click on the Next button. You will be asked to choose your connection. Assuming that you have access to a Northwind database hosted in MS SQLServer you can either make use of an existing connection to it or configure a new one using the New Connection button. Pressing the New Connection button will present the normal OLEDB dialogues for creating a new datasource. Once a data connection has been selected press Next to continue.

Figure 14.23. Databound Examples using MS Visual Studio



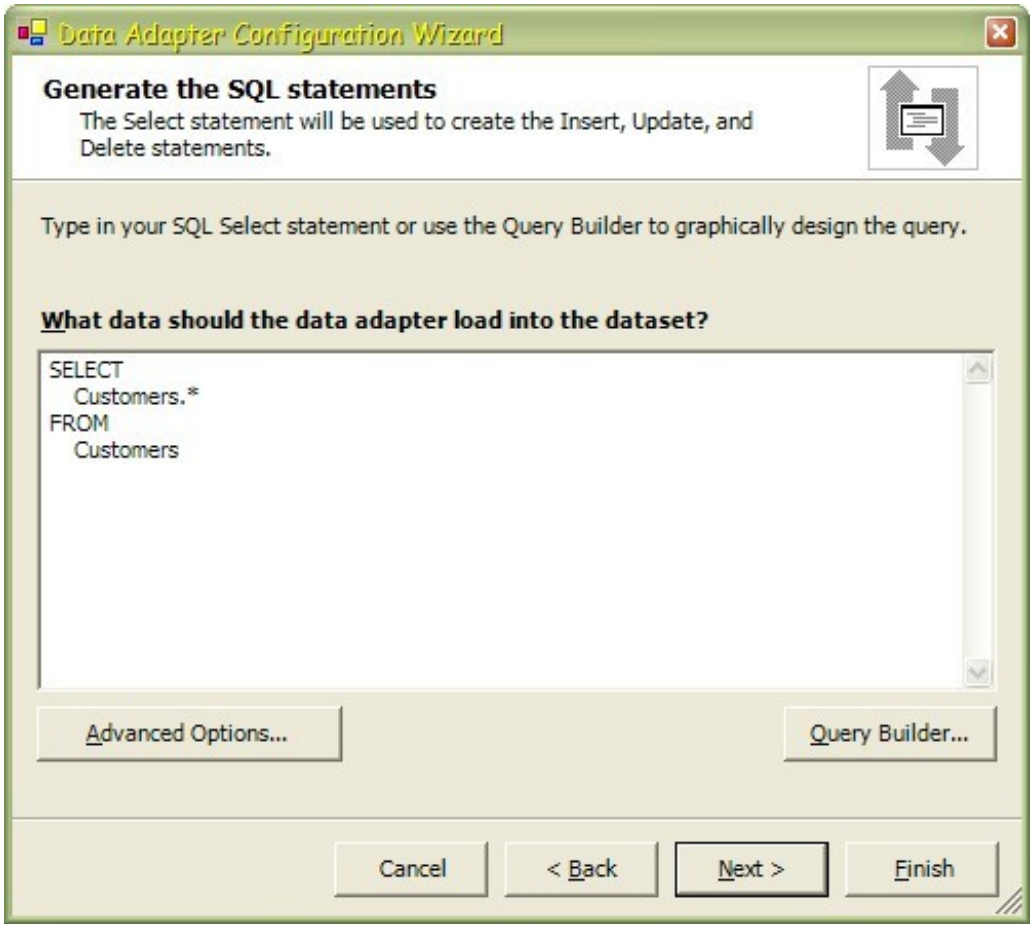
7. **Choose a Query Type.** The next panel will ask how the data adapter should access the database. For simplicity we will stick with the default of Use SQL Statements. Press Next.

Figure 14.24. Databound Examples using MS Visual Studio



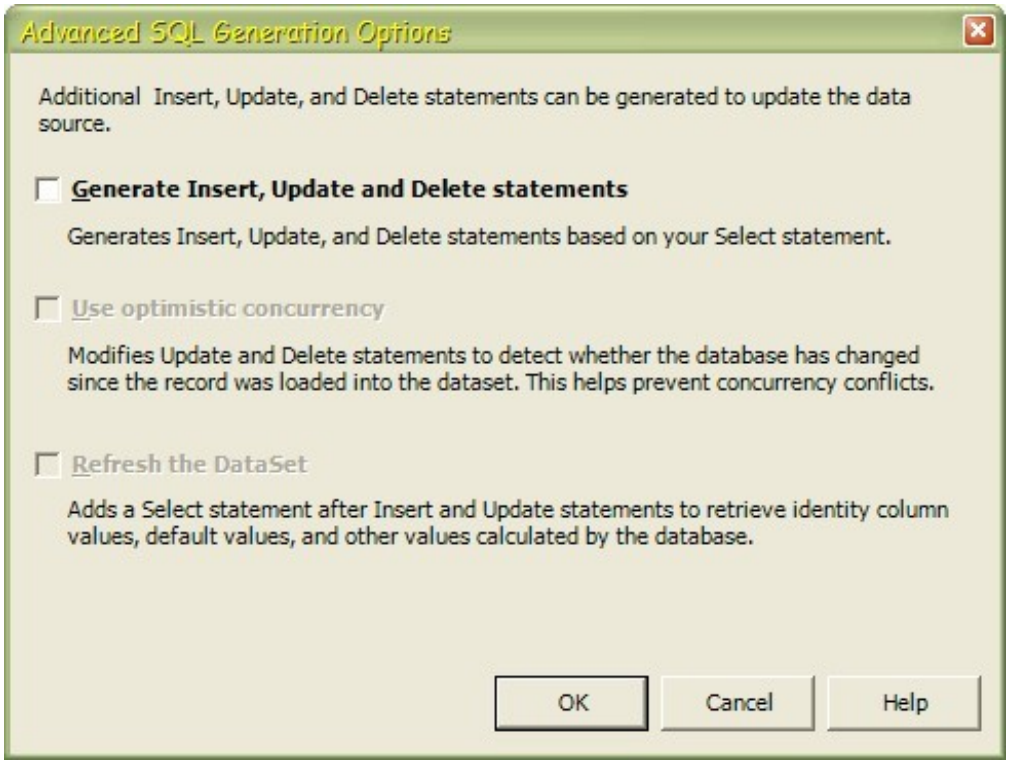
8. **Generate SQL Statements.** At this point you can either type a SQL statement if you know exactly what you are looking to get back from the database, or you can use the Query Builder to point-and-click to your data. In our case we will retrieve everything from the Customers table using a simple query:`select * from Customers`.

Figure 14.25. Databound Examples using MS Visual Studio



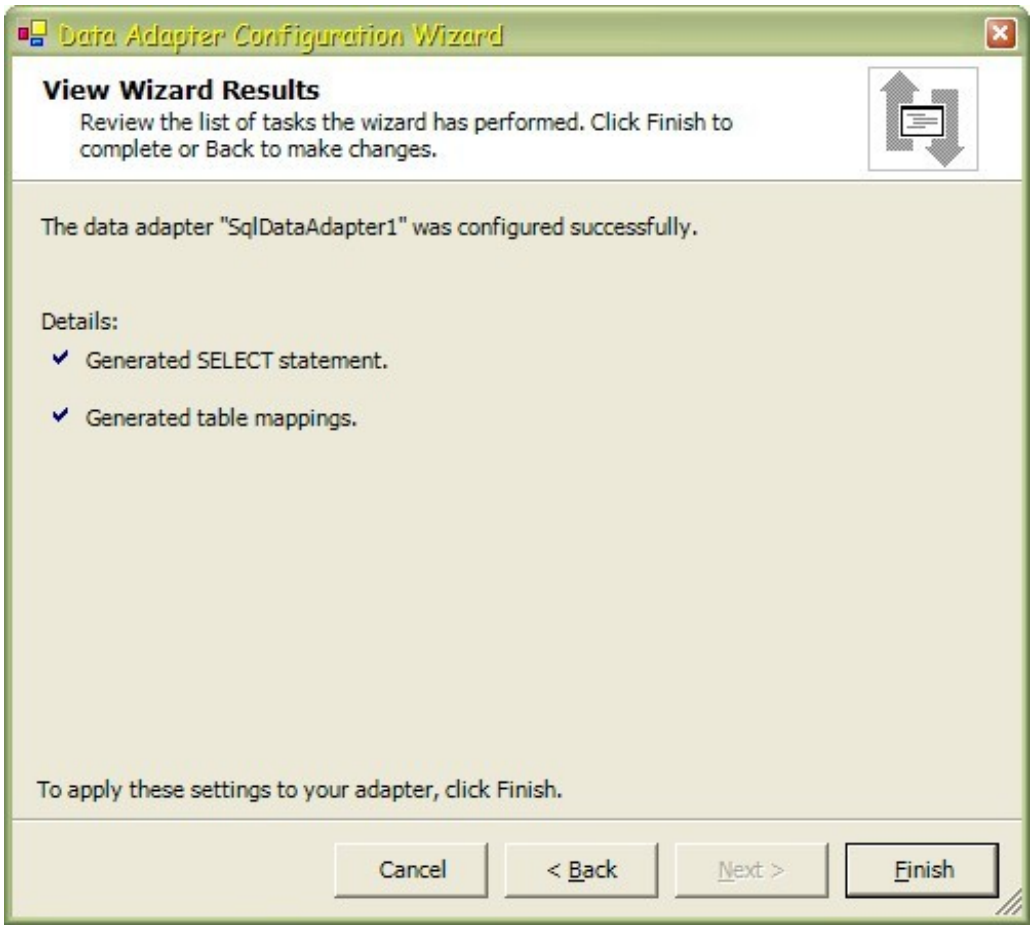
9. **Advanced Options.** Click on the Advanced Options button. Unselect the top checkbox. This will also unselect the other two automatically. We will not be performing any updates in this example. OK the advanced options, upon return to the main dialogue press the Next button to continue onwards.

Figure 14.26. Databound Examples using MS Visual Studio



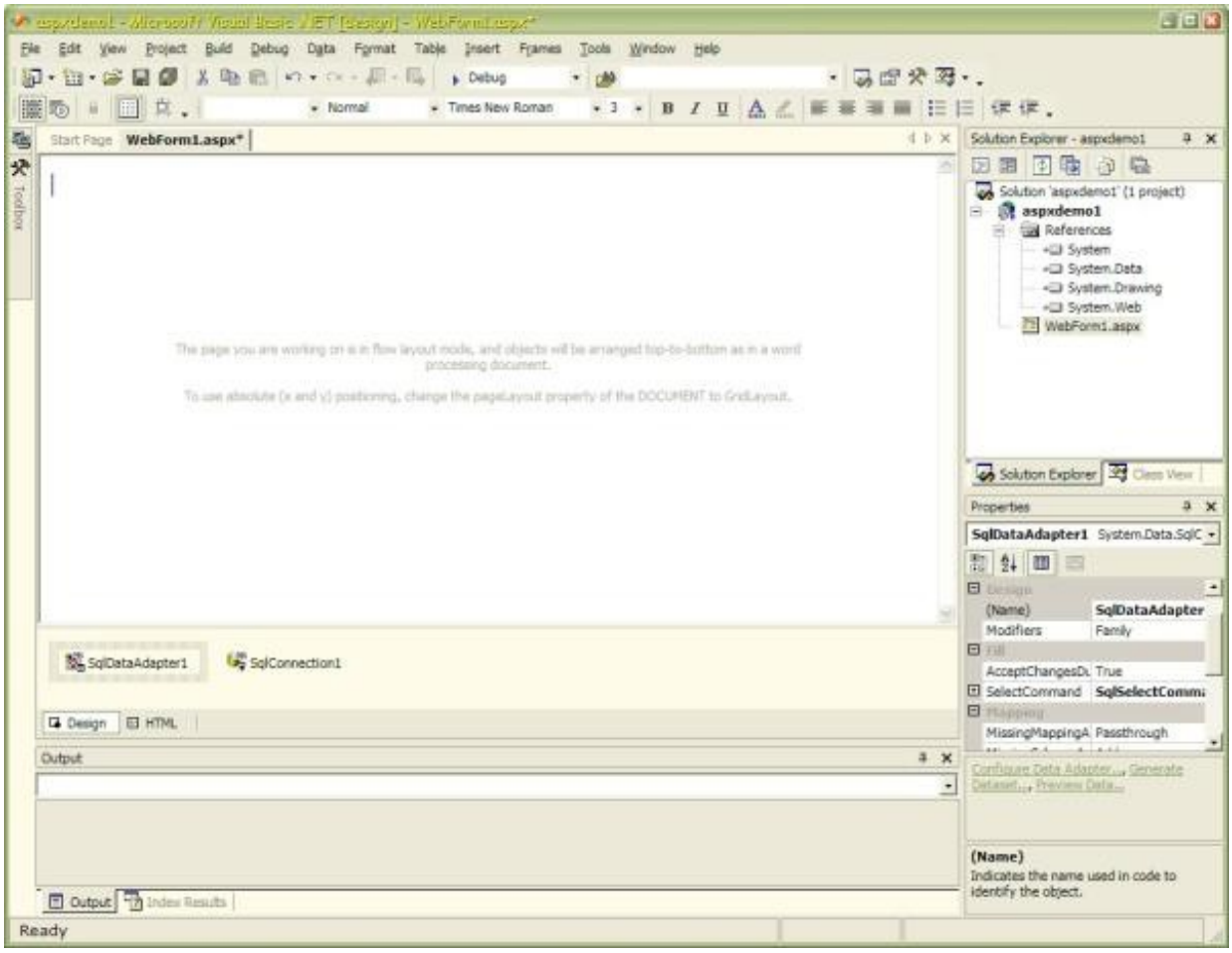
10. **View Wizard Results.** The next panel confirms all the options selected leaving us to simply complete the wizard.

Figure 14.27. Databound Examples using MS Visual Studio



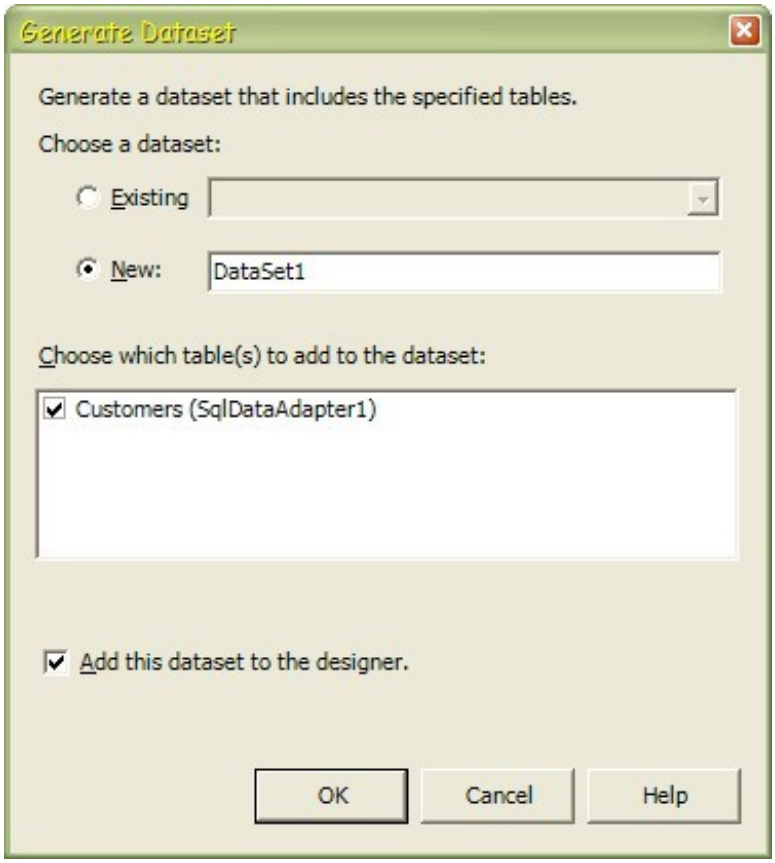
11. **Back to the main window...** When the wizard finishes and returns you to the main Visual Studio window you will see that two controls have been added to the Web Form. The connection control has been automatically generated to support the Data Adapter.

Figure 14.28. Databound Examples using MS Visual Studio



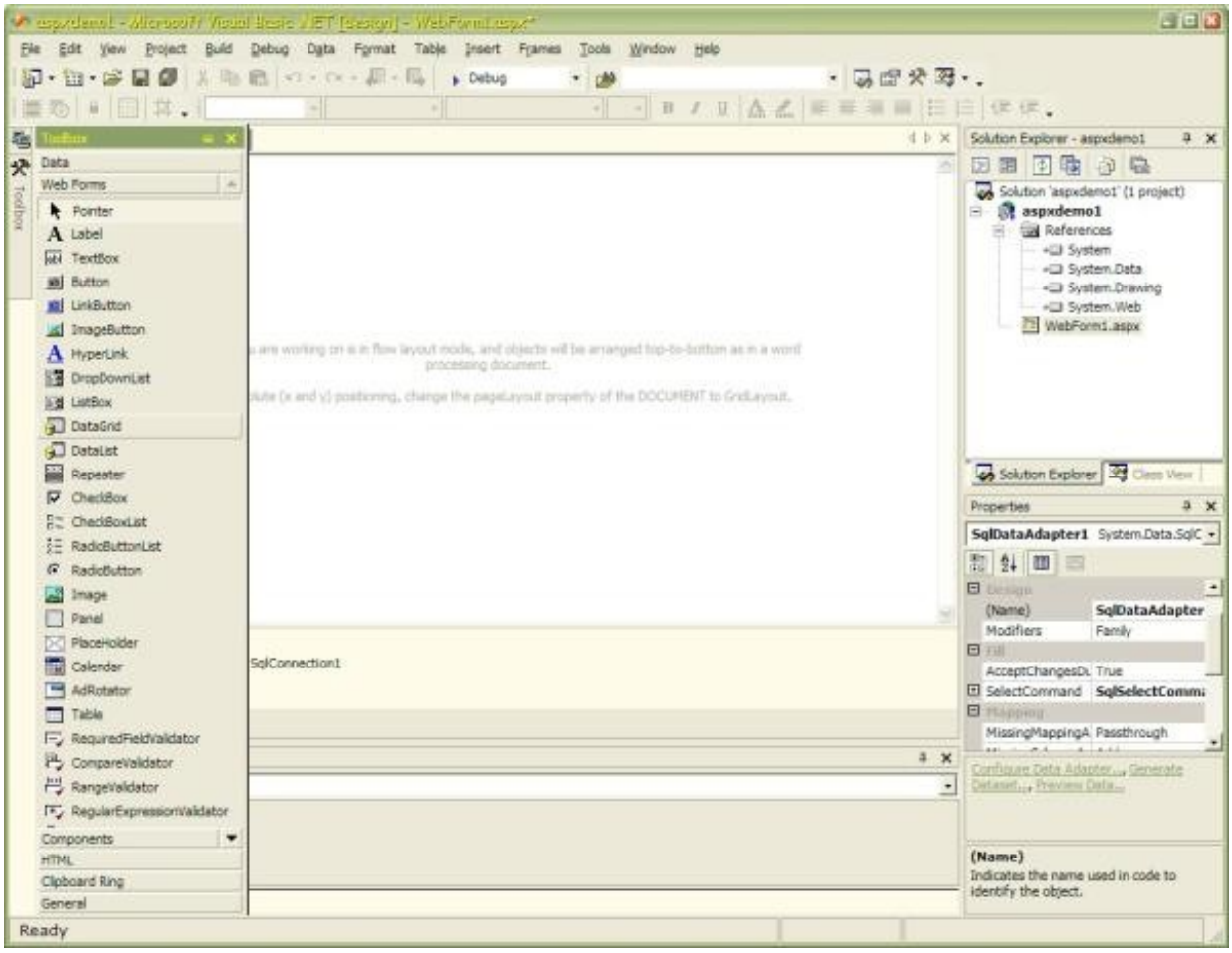
12. **Generate Dataset.** Before we can display anything on the page we need to form a Dataset. Right-click on the SQLDataAdapter1 control you previously added and select Generate Dataset. Defaults on the displayed dialog are all sufficient. After verifying them click the OK button to generate the dataset.

Figure 14.29. Databound Examples using MS Visual Studio



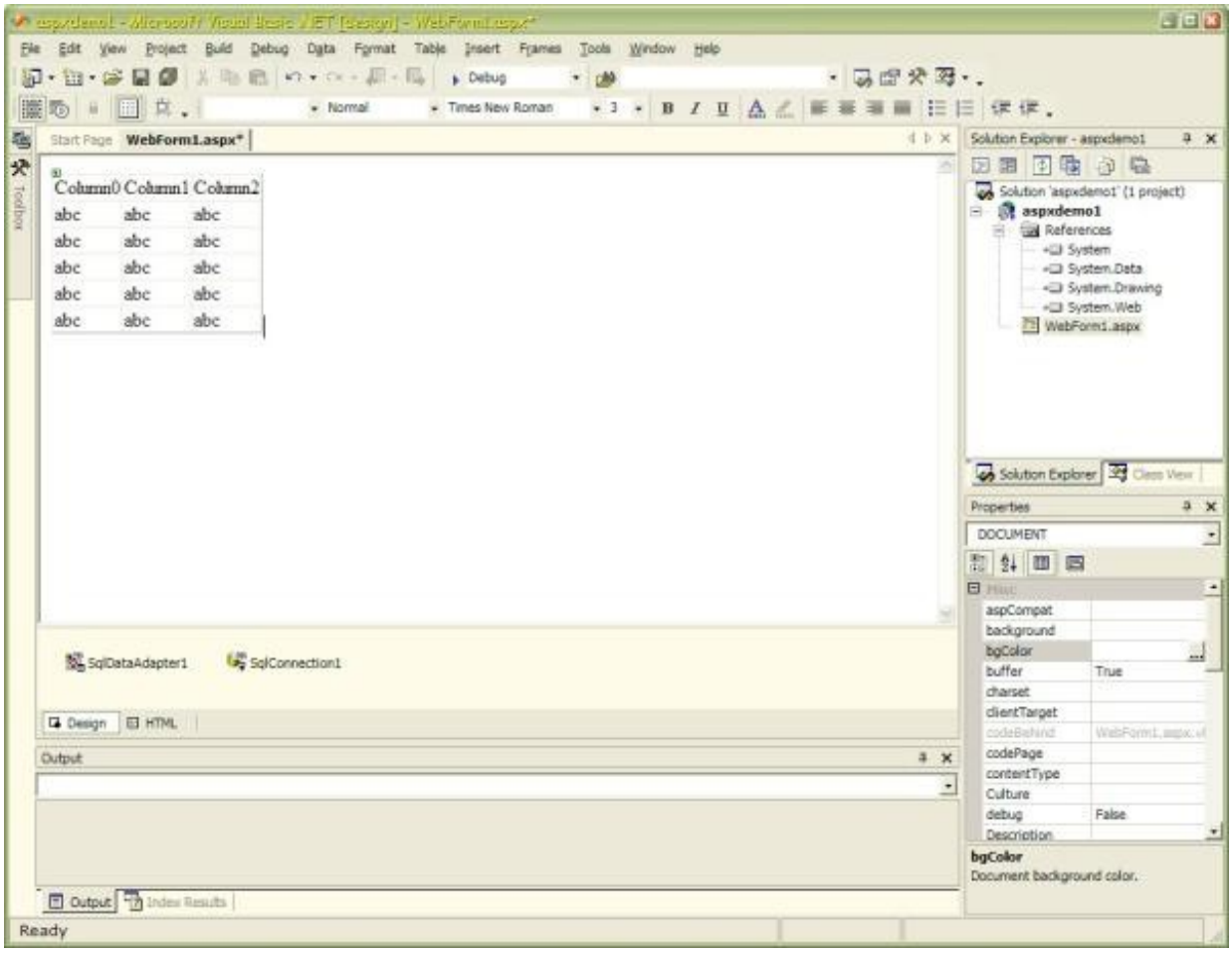
13. **Add a DataGrid Control.** Now we need to show the dataset that we are fetching from the database. We will use a simple DataGrid for this. Open the toolbox in the Web Forms panel and double-click on the DataGrid control. This add the control to the page and will display a table on the web page view.

Figure 14.30. Databound Examples using MS Visual Studio



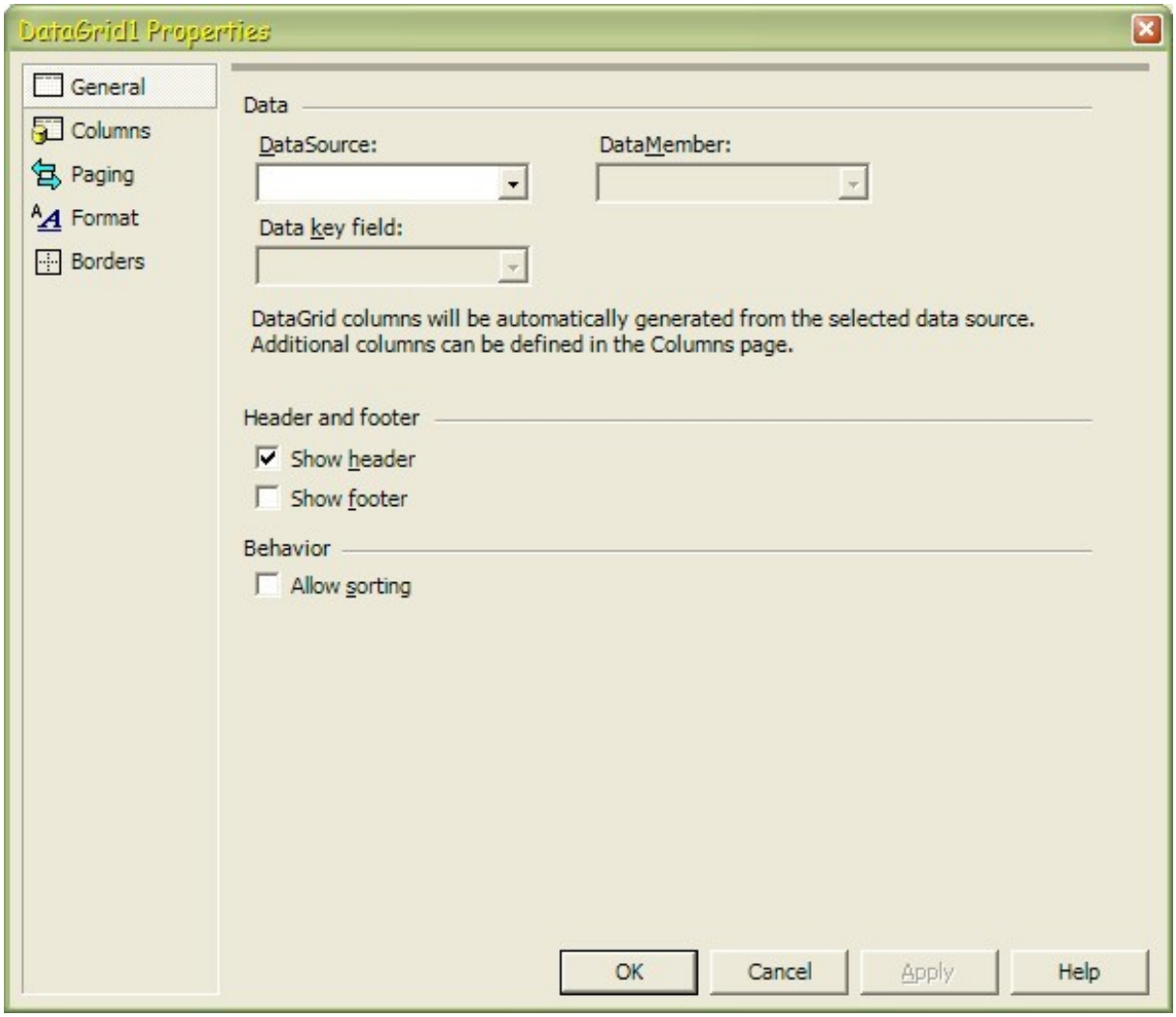
14. **Table Properties.** Configure the DataGrid control by right-clicking on the table and selecting Property Builder .

Figure 14.31. Databound Examples using MS Visual Studio



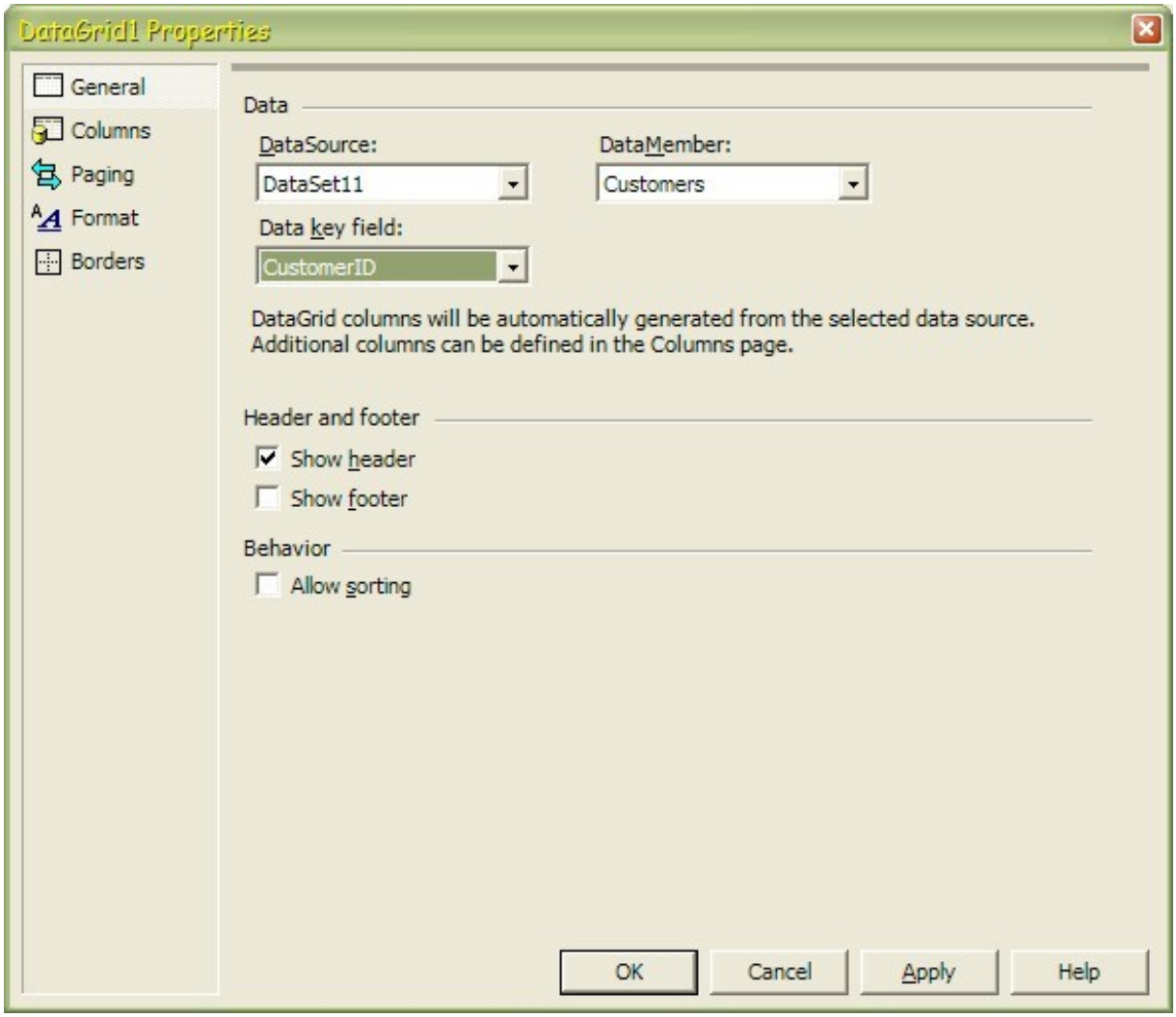
15. **DataGrid Properties.** You can adjust most properties of the table such as colours, fonts, borders, etc. You can control paging aspects that will be taken care of automatically by the control, you only need specify the number of results per page. For now we only want to adjust the most important aspects of the table, where the data comes from.

Figure 14.32. Databound Examples using MS Visual Studio



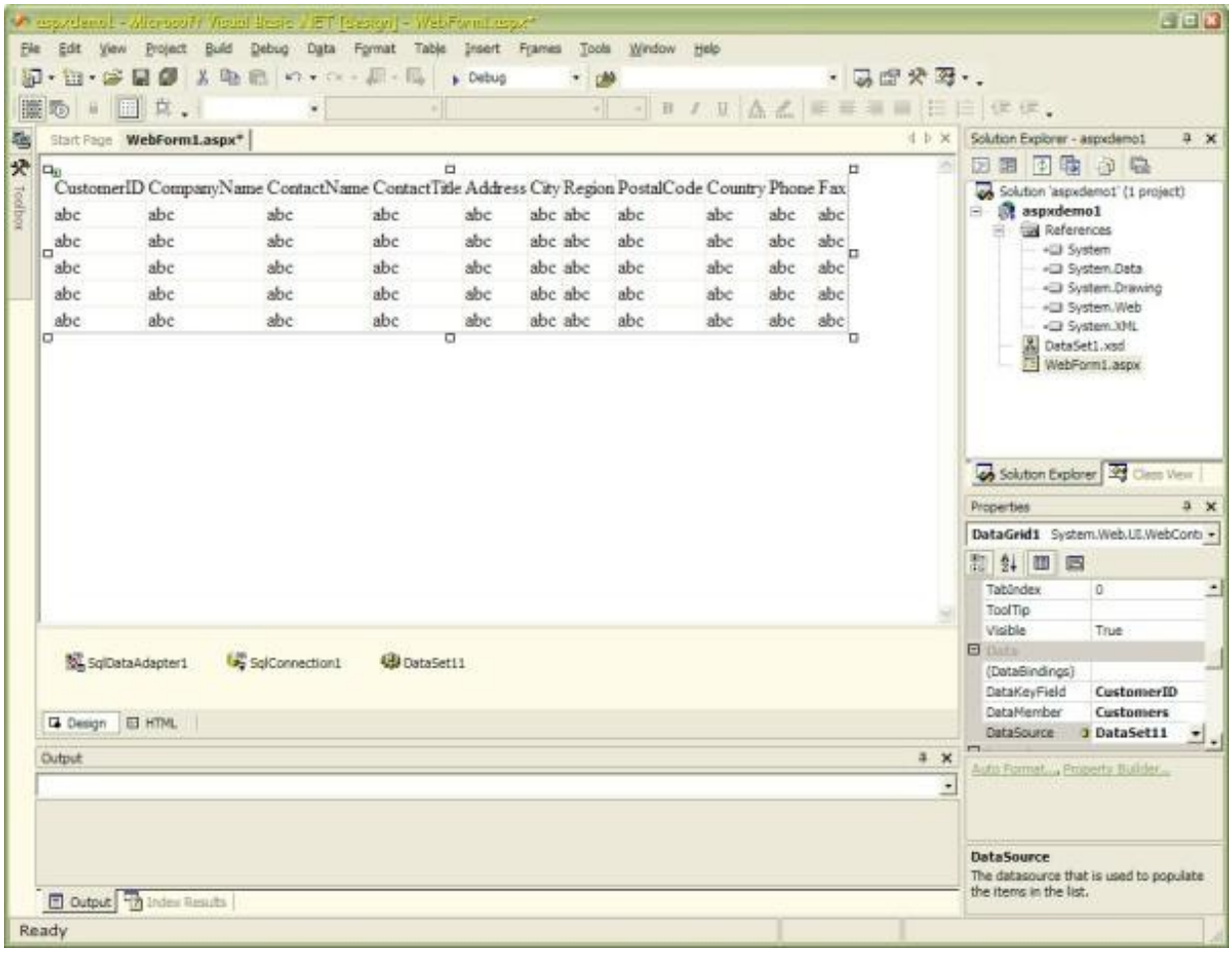
16. **DataGrid Bindings.** From the first General properties panel you must select the DataSource, DataMember, and select the Data key field (especially important for updatable tables). These properties may be selected via the drop-downs on the panel, allowing you to select the now-familiar controls configured earlier. Once configured, press the OK button to save the changes.

Figure 14.33. Databound Examples using MS Visual Studio



17. **Preview.** Once the details have been saved the view of the table in the page will be updated to reflect the dataset details.

Figure 14.34. Databound Examples using MS Visual Studio



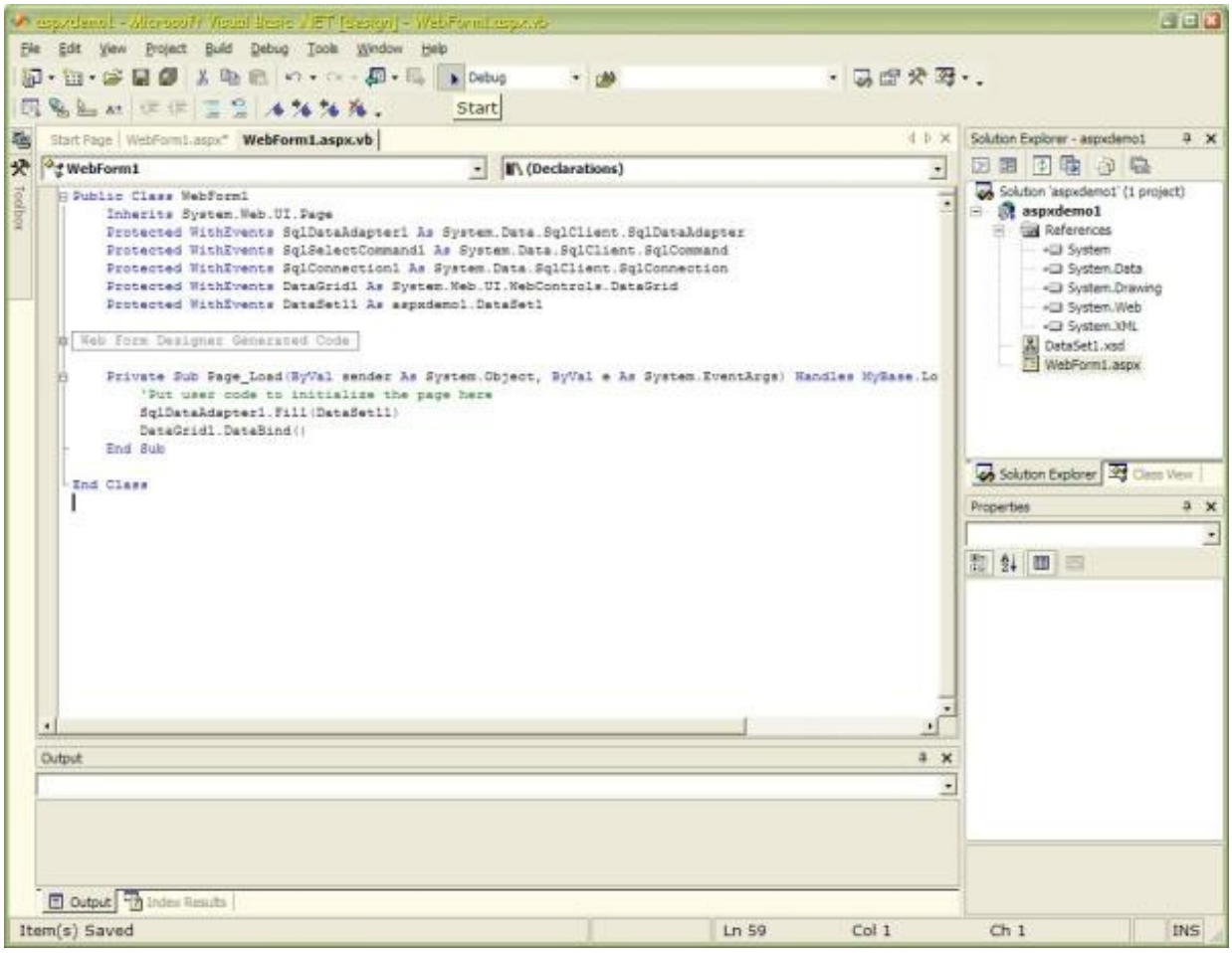
18. **Page Initialization.** We are very nearly finished. The controls are all ready to be used, however, the final touch is to edit our Page_Load function. Double-click on the empty part of the web page view in Visual Studio and the associated `.vb` file will be opened. This is the file containing the actual code. You will be automatically placed at the Page_Load function. This is the page initialization section where will need to instruct the DataAdapter to fill with data, and then instruct the DataGrid to bind to that data.

    These lines of code need to be added by hand:

    ```
    SqlDataAdapter1.Fill(DataSet11)
    DataGrid1.DataBind()
    ```

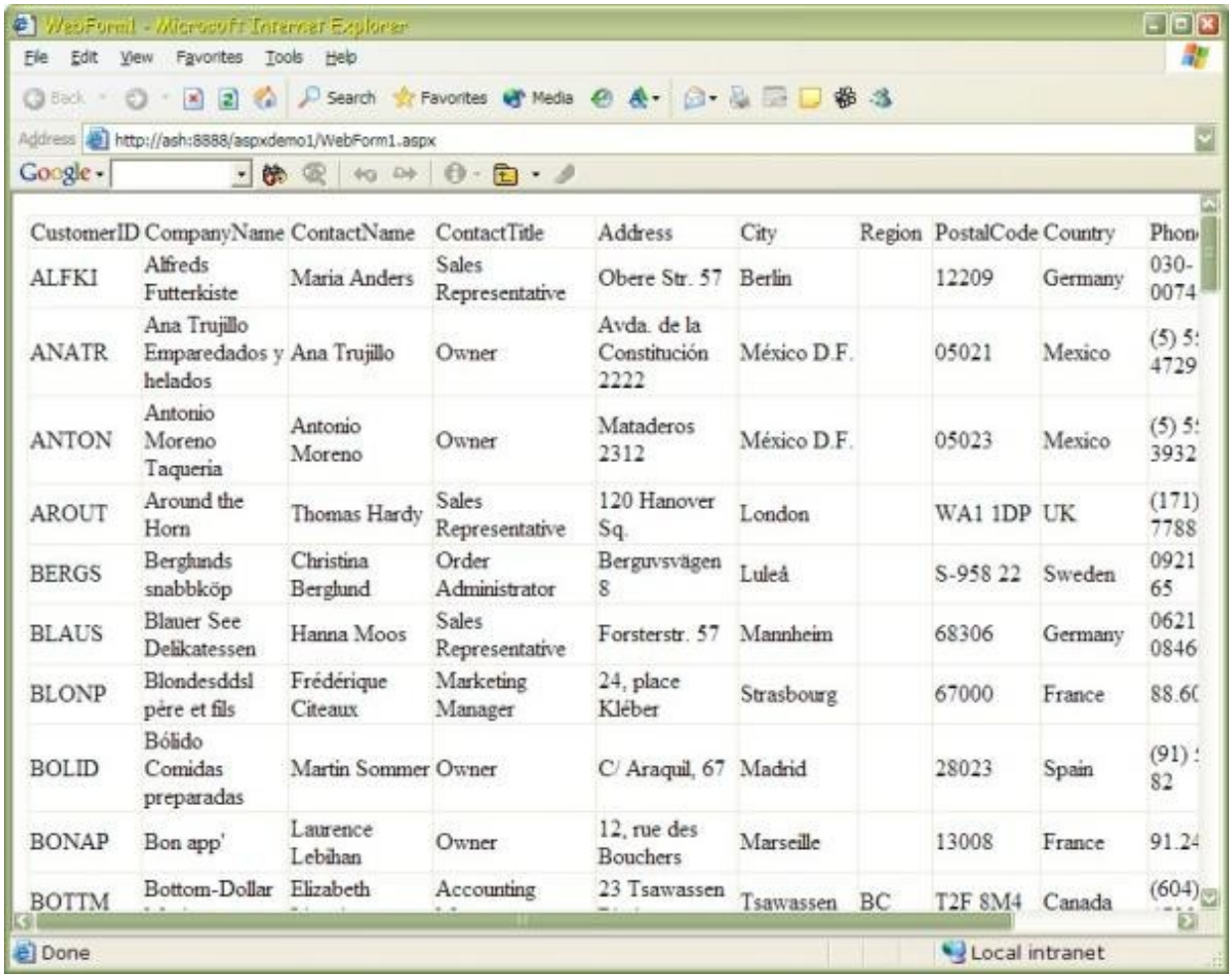
    You will find that Visual Studio will offer some assistance in the form of intellisense command-completion while you are typing.

Figure 14.35. Databound Examples using MS Visual Studio



19. **The End Result.** Next we will set the project's initial start-page by right-clicking on the form name in solution explorer and selecting Set as Start Page option. Finally we can run the project using the start button at the top. The project will automatically be built and a browser window launched automatically. Admire your results. When finished, on closing the browser windows you will automatically return to Visual Studio.

Figure 14.36. Databound Examples using MS Visual Studio



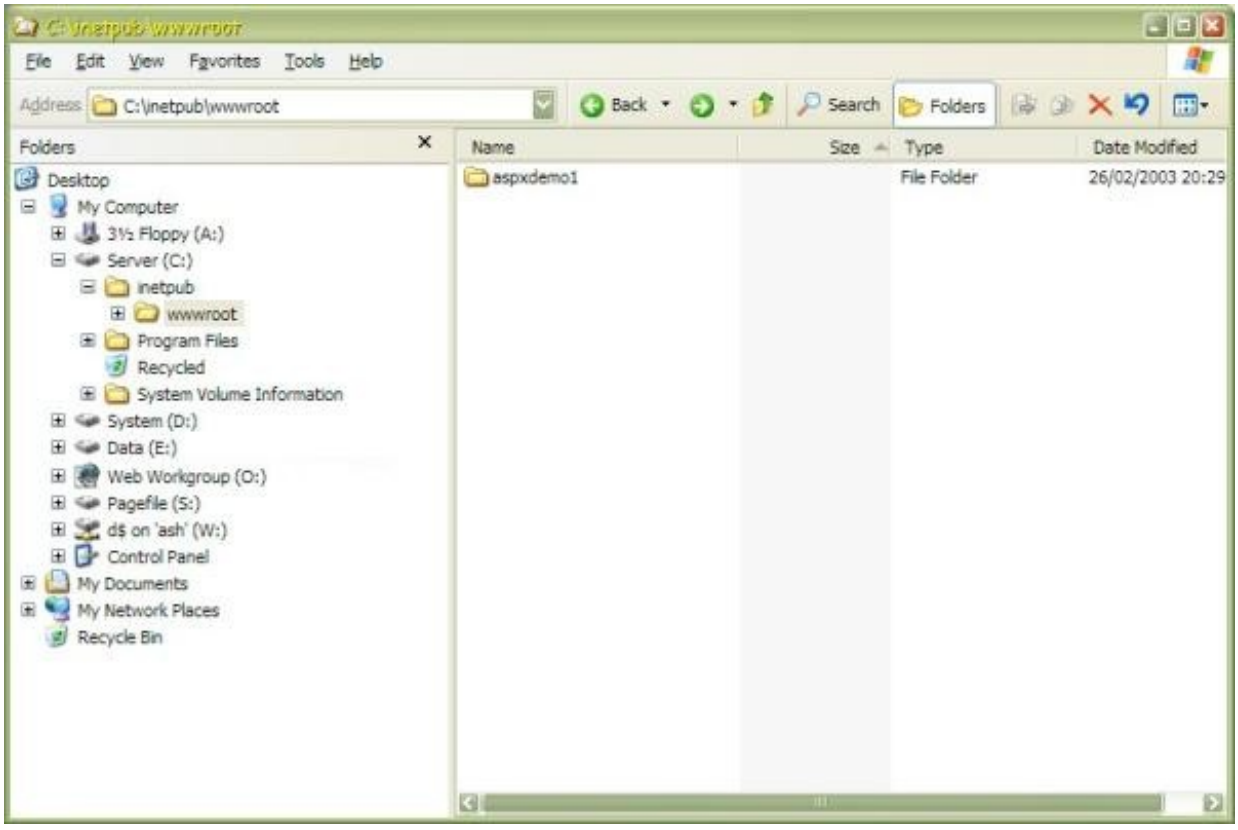
CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phon
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germany	030-0074
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.		05021	Mexico	(5) 5: 4729
ANTON	Antonio Moreno Taqueria	Antonio Moreno	Owner	Mataderos 2312	México D.F.		05023	Mexico	(5) 5: 3932
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London		WA1 1DP	UK	(171) 7788
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå		S-958 22	Sweden	0921 65
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim		68306	Germany	0621 0846
BLONP	Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg		67000	France	88.6(
BOLID	Bólido Comidas preparadas	Martin Sommer	Owner	C/ Araquil, 67	Madrid		28023	Spain	(91) : 82
BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Bouchers	Marseille		13008	France	91.24
BOTTM	Bottom-Dollar	Elizabeth	Accounting	23 Tsawassen	Tsawassen	BC	T2F 8M4	Canada	(604)

Migrating ASP.Net Applications to Virtuoso

The previous section showed us how to build a very basic web application using an ASPX data-bound control. Now we will host this same application in Virtuoso demonstrating that IIS is no longer required for ASPX application deployment.

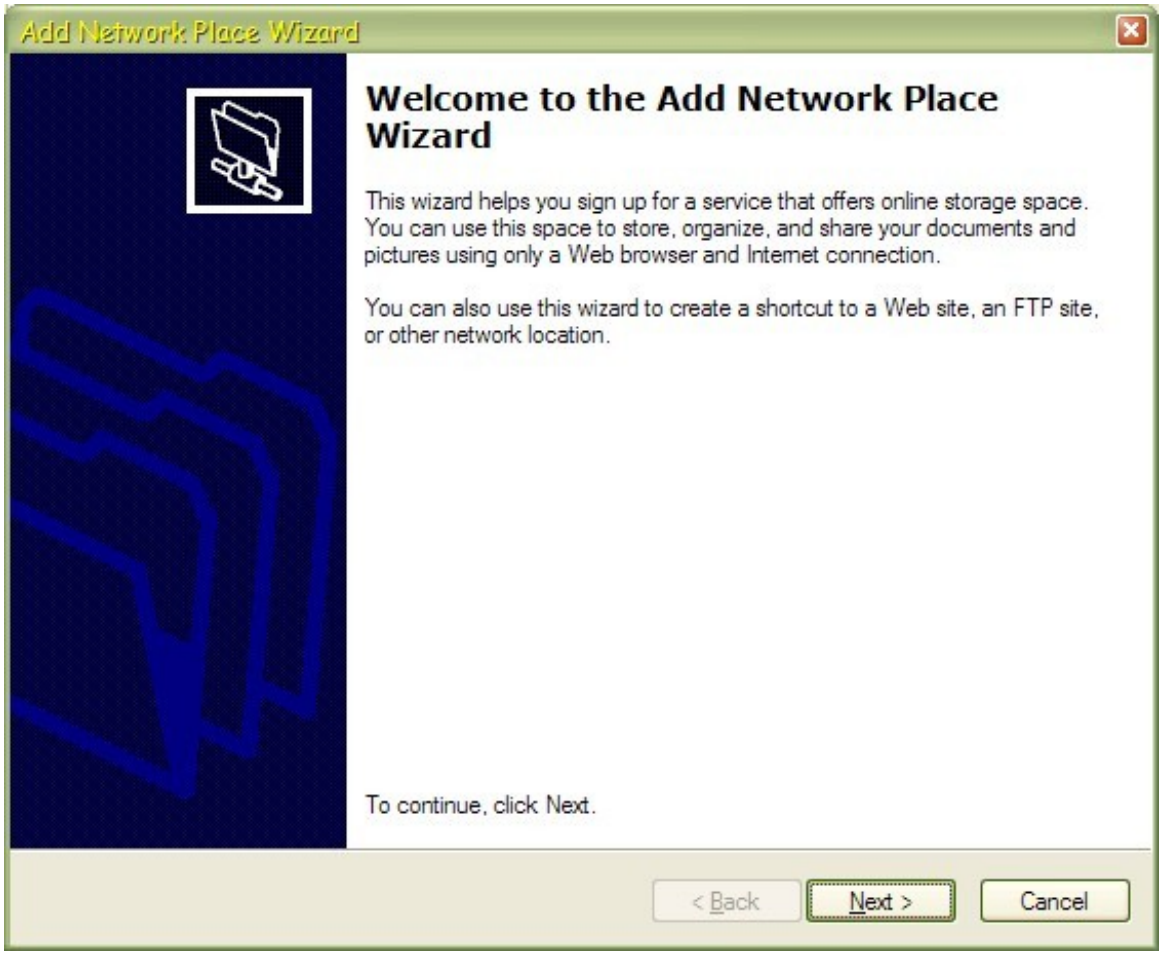
1. **Locating your ASPX application.** Use explorer to locate the ASPX application you want to move away from IIS and re-deploy under Virtuoso. The previous section created the application in the IIS virtual directory represented by `http://ash:8888/aspxdemo1/` which was physically located in `c:\inetpub\wwwroot\aspxdemo1\`. We can simply copy this directory and place it somewhere under either the VSP root directory or WebDAV. We will copy the application to a location in WebDAV for a local Virtuoso server.

Figure 14.37. Migrating ASP.Net Applications to Virtuoso



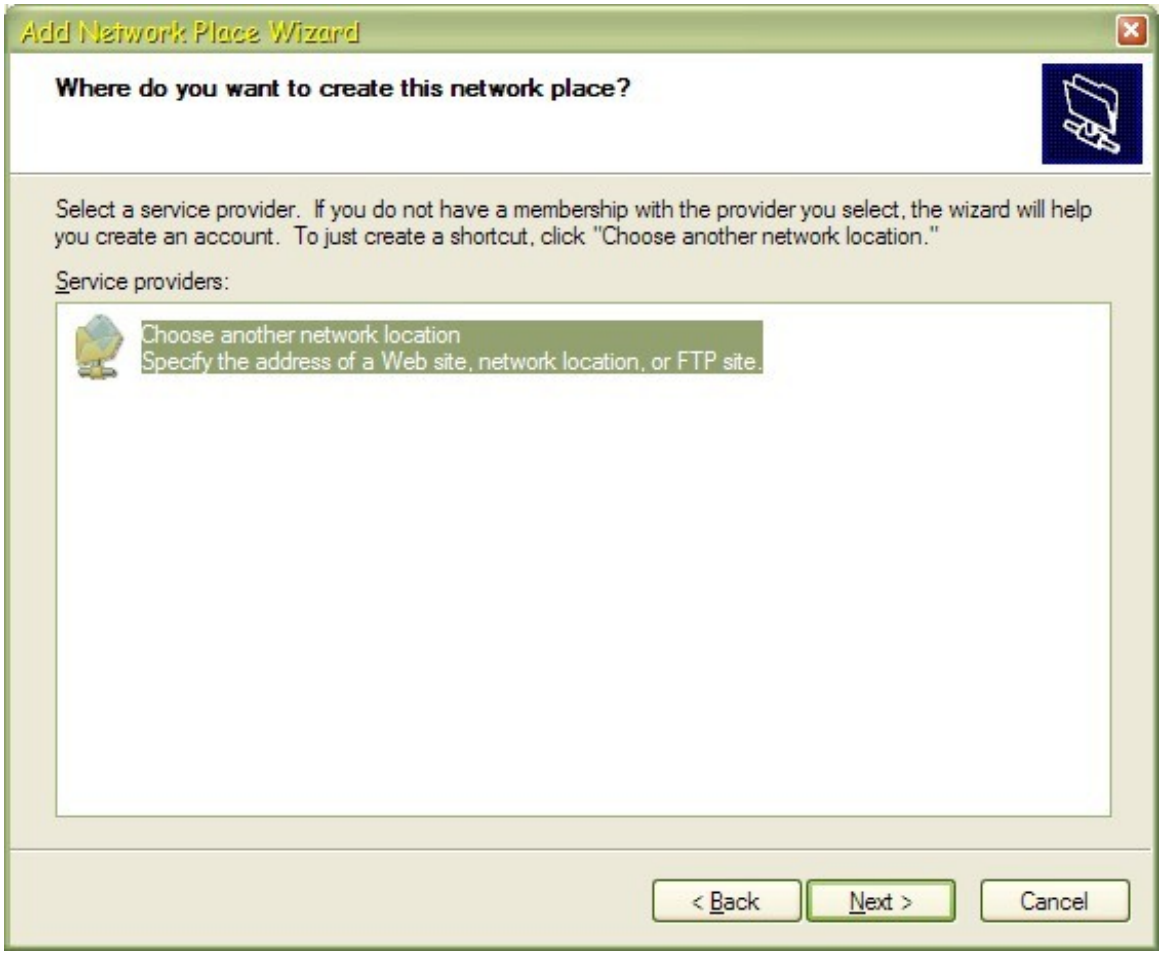
2. **Add a Network Place to DAV.** From My Network Places in explorer or Network Neighborhood double-click on the Add Network Place shortcut to start the wizard.

Figure 14.38. Migrating ASP.Net Applications to Virtuoso



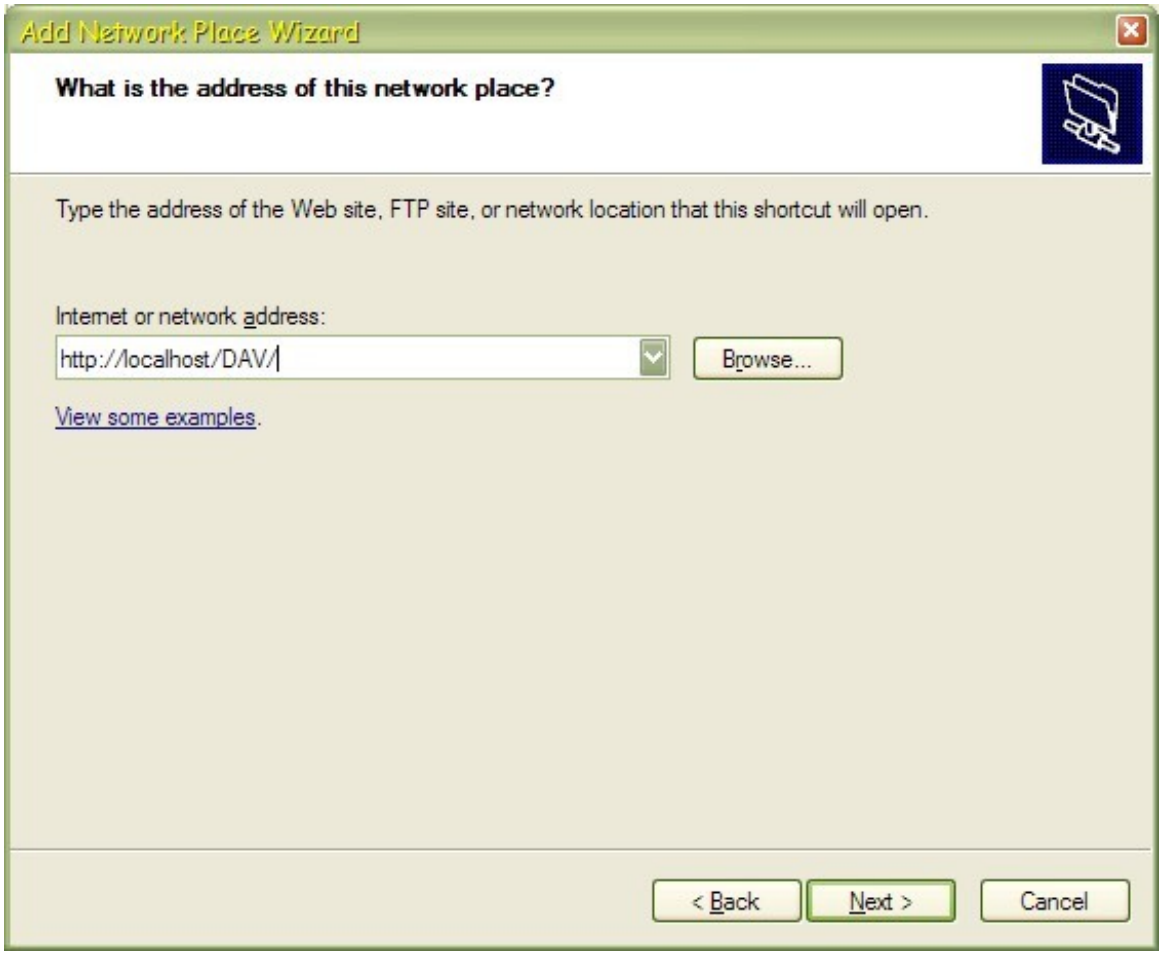
3. **Network service provider.** Select the default option. Click next to continue.

Figure 14.39. Migrating ASP.Net Applications to Virtuoso



4. **Internet or Network Address.** Here you specify the URL of DAV on the Virtuoso server. We are using Virtuoso on the local machine but this could be anywhere on the Internet. Click next to continue.

Figure 14.40. Migrating ASP.Net Applications to Virtuoso

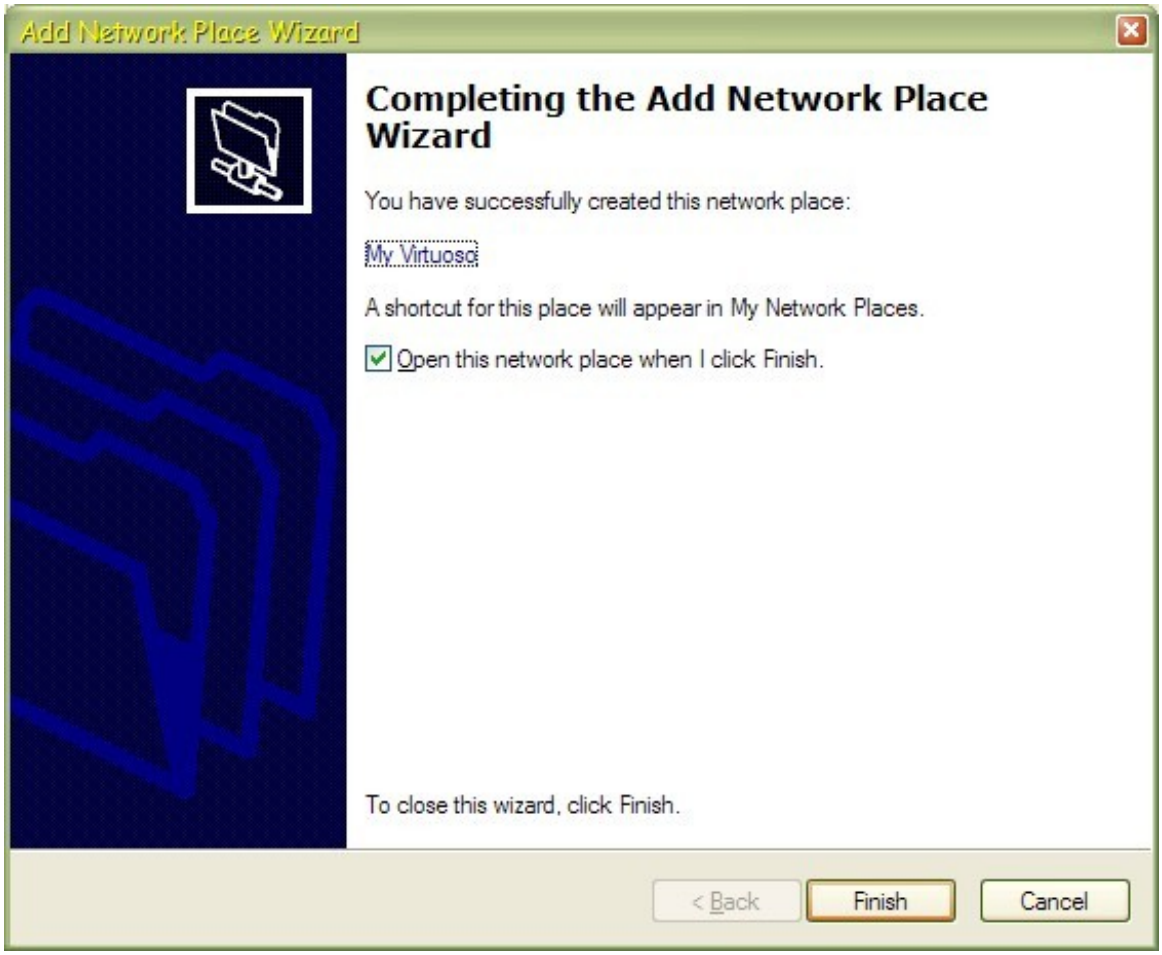


5. **Name the Network Place.** Provide the network place with a meaningful name so we can easily find this location in the future. Click on next to continue.

Figure 14.41. Migrating ASP.Net Applications to Virtuoso

6. **Open the Network Place.** On the final panel of the wizard there is a checkbox to "Open this network place when I click Finish" that we will leave checked. Click Finish to continue and open the DAV directory.

Figure 14.42. Migrating ASP.Net Applications to Virtuoso

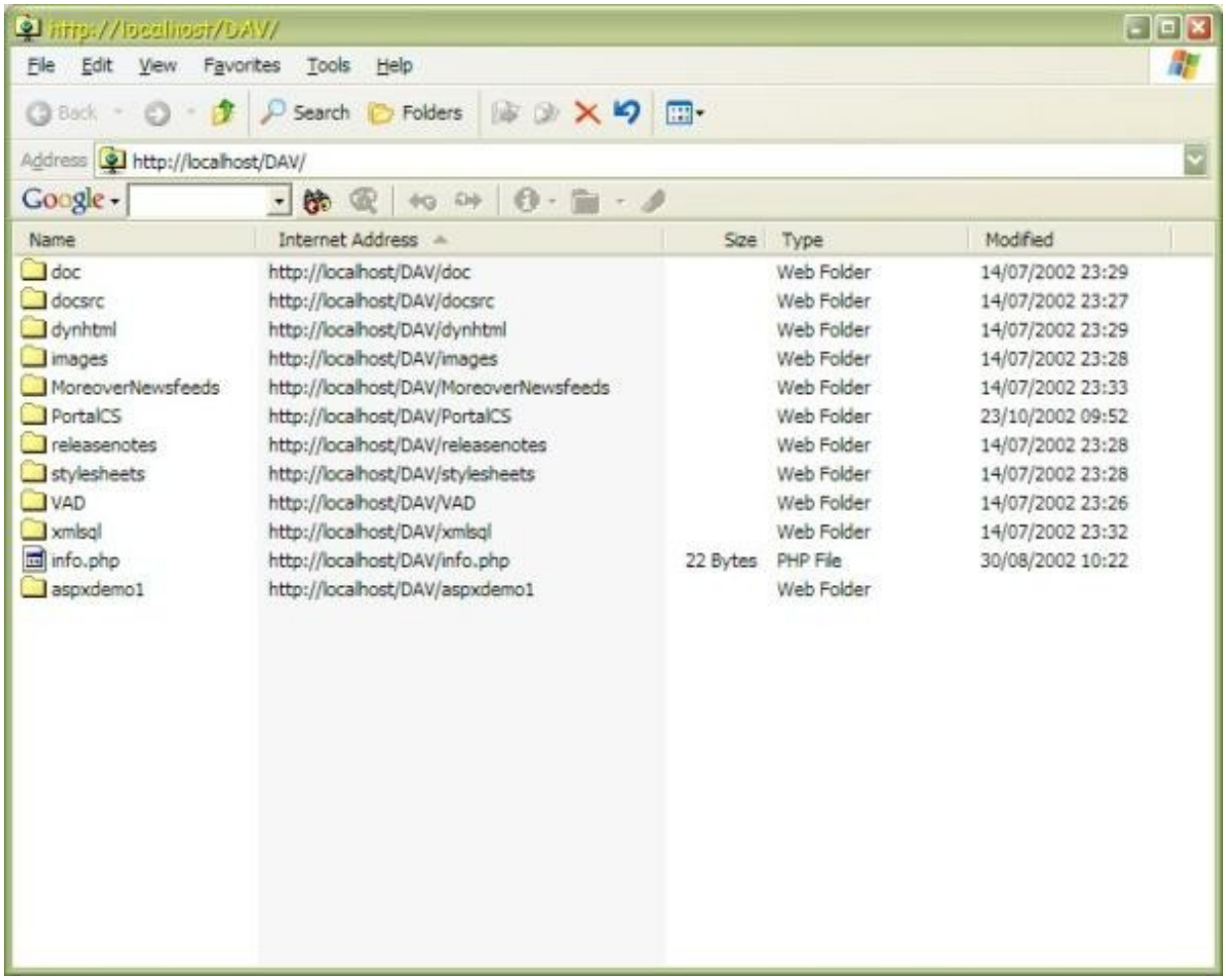


7. **Connect to DAV.** When you attempt to connect to DAV you will be prompted for a User name and Password. These will be whatever you entered during the installation, possibly both dav, in which case type dav in both fields and press the OK button to continue.

Figure 14.43. Migrating ASP.Net Applications to Virtuoso

8. **Copy the ASPX application to DAV.** Once the explorer opens up on the DAV network place we can copy the `aspxdemo1` application directory to it.

Figure 14.44. Migrating ASP.Net Applications to Virtuoso

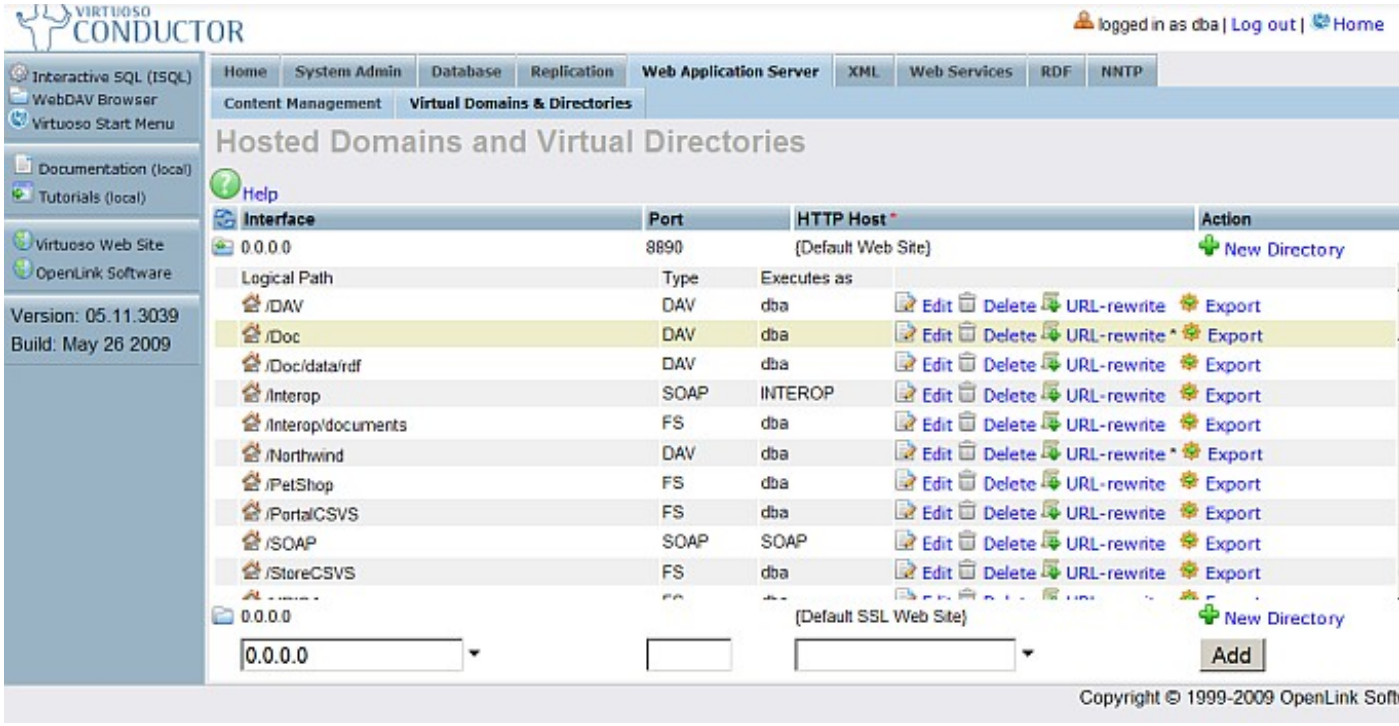


9. **Configure Virtual Directory.** Now we have to configure a Virtuoso virtual directory. Go to Conductor / Web Application Server / HTTP Hosts & Directories.

Figure 14.45. Migrating ASP.Net Applications to Virtuoso

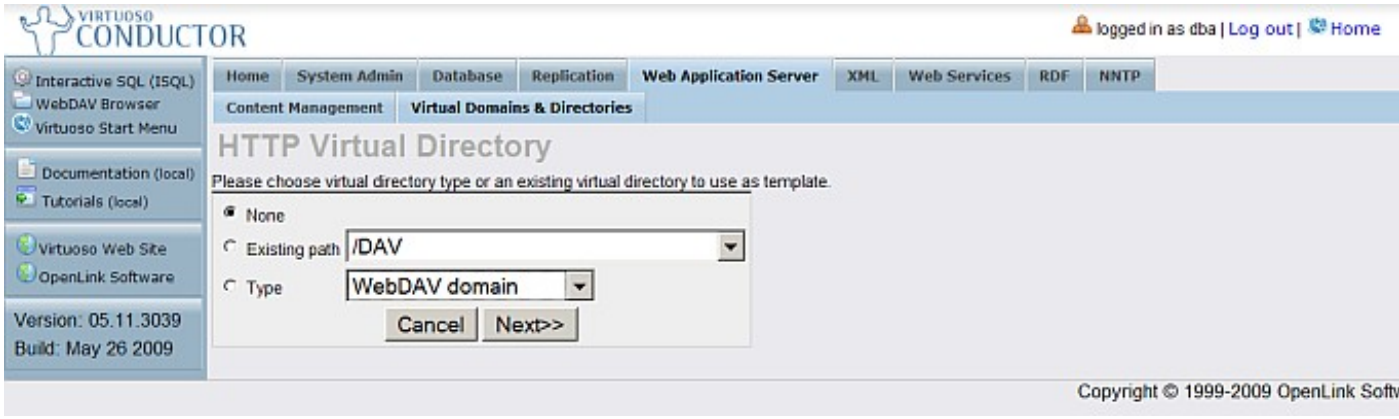
10. **Add Virtual Directory.** The "folder" icon for the hosts defined will list all mappings currently present for the Virtuoso server. Click on the "New Directory" link to continue.

Figure 14.46. Migrating ASP.Net Applications to Virtuoso



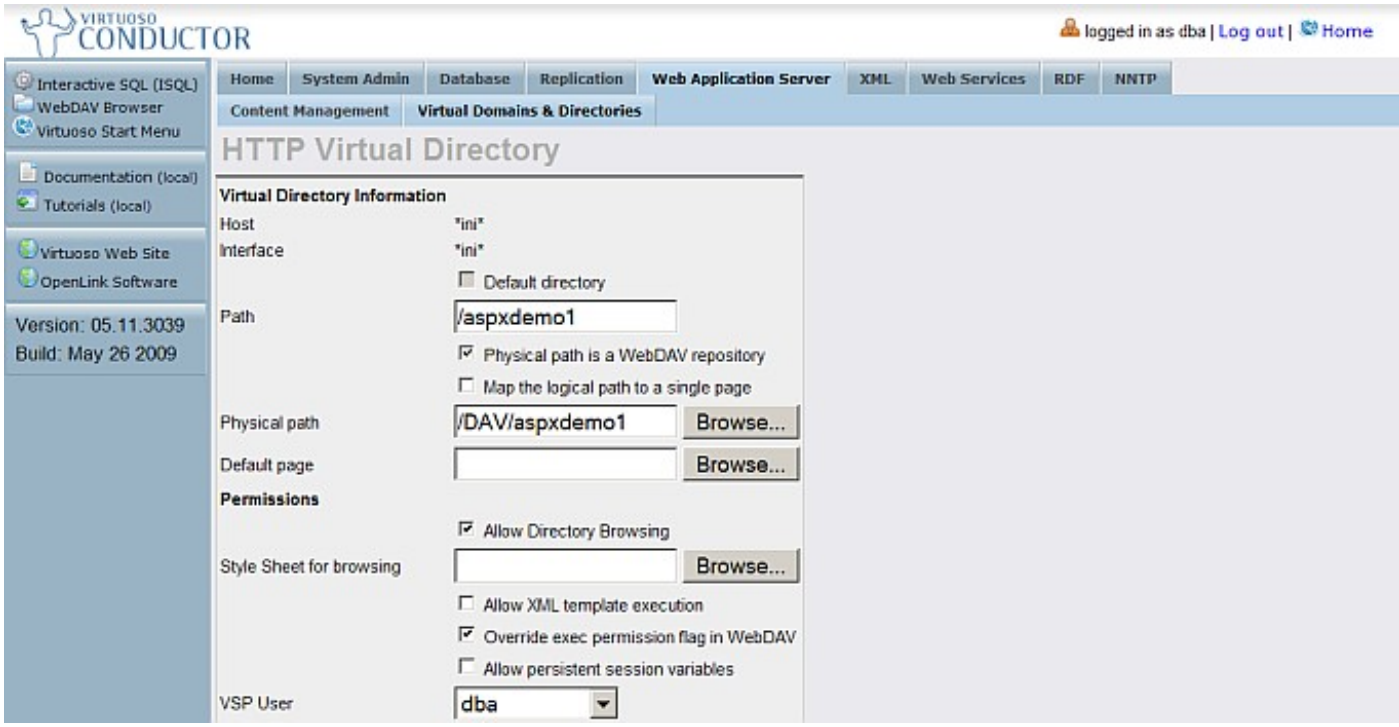
11. **Select DAV Domain template.** On the next page select for "Type" the DAV Domain template and click "Next".

Figure 14.47. Migrating ASP.Net Applications to Virtuoso



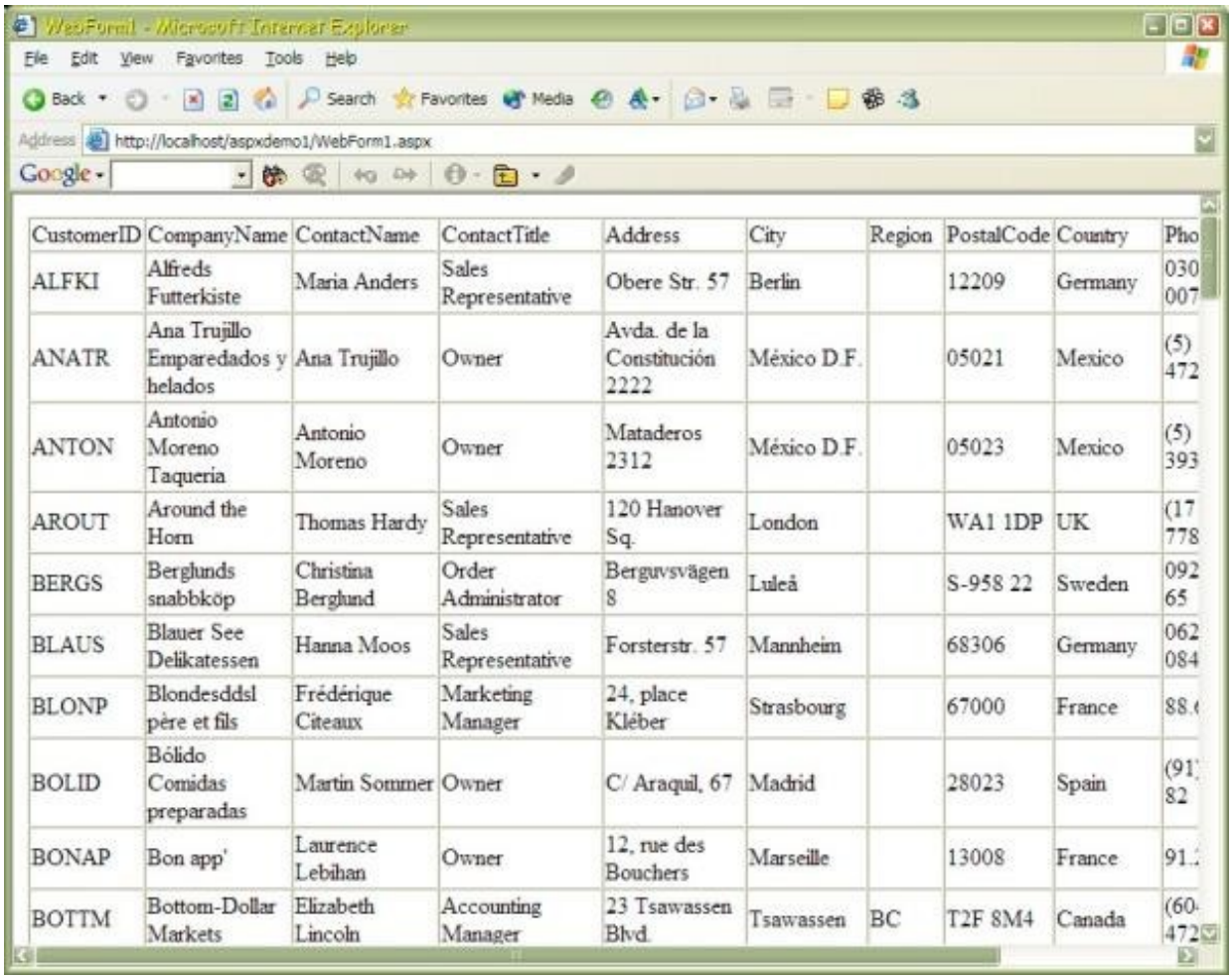
12. **Configure the Virtual Directory parameters.** We must supply a logical path, `/aspxdemo1` , in this case. We must also supply the physical path that this represents, we can either type in the DAV location or use the Browse button to the find the directory under DAV. The WebDAV checkbox will correctly already be checked. For now we will also set to Allow Directory Browsing, and Override execution permissions in WebDAV. Finally we must set the VSP Execution user to `dba` using the drop down. To save the changes click on the "Save Changes" button at the bottom of the page.

Figure 14.48. Migrating ASP.Net Applications to Virtuoso



13. **View the ASPX page hosted in Virtuoso DAV.** With the virtual directory configured we have only to test that it works as expected. On opening a browser at: `http://example.com/aspxdemo1/WebForm1.aspx`, we will see the same output as before.

Figure 14.49. Migrating ASP.Net Applications to Virtuoso



Note:
If ASPX hosting is attempted on a machine that never had IIS installed it is possible that ASP.NET is not configured, although required. The Virtuoso installer will take care of this for you in most cases but if you run into problems you may want to confirm that ASP.NET is configured correctly. You will have to use the regedit utility to edit the registry, changing a search-path so System.Web.dll can find aspnet_asapi.dll. Always exercise caution whenever editing the registry as invalid data can prevent your machine from operating properly.

You will first need to find out what version of the `aspnet_isapi.dll` library you have. The simplest way to determine this is by locating the file and right-clicking properties on it from explorer.

Assuming the version is 1.0.3705.288, we should enter the following details into the registry if the `HKLM\Microsoft\ASP.NET` key is missing (if your version differs change it accordingly):

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\ASP.NET]
     "RootVer"="1.0.3705.288"
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\ASP.NET\1.0.3705.288]
     "Path"="C:\WINDOWS\Microsoft.NET\Framework\v1.0.3705"
     "DllFullPath"="C:\WINDOWS\Microsoft.NET\Framework\v1.0.3705\aspnet_isapi.dll"
```

14.6. ASMX Web Service Hosting

Microsoft ASMX Web Services can be hosted in exactly the same way as ASPX Applications can be. A ASMX Web Services Project created in Visual Studio can be copied to either the file system (under the VSPROOT directory) or the Virtuoso WebDAV repository. You must copy the project directory to the required location and then make a Virtual Directory pointing to it. If the location is in WebDAV then ensure that execute permissions are set for any .asmx files.

See Also:
ASP.Net Web Application Hosting

14.7. Blogging & Weblogs

Weblogs, or Blog, are web pages or sites organized by date. The content of a typical blog page contains short messages listed in chronological order. The blog messages are typically excerpts of HTML markup, since they typically appear on web sites and expect web browsers as clients.

In the rest of this document we will use interchangeable the terms "blog" and "weblog", and refer to the users that have a blog as "bloggers".

The following terms are also commonly used in this section:

RSS/OCS/OPML/RDF/Atom - file formats, XML based to represent data or list of files containing a data

Channels - a public source containing data in any of XML/text/HTML format, usually in RSS format.

Post/Article(message) - this is a piece of information describing something in a human readable format, in some language. This can be have satellite parts as title, excerpt, publication data etc.

PermaLink - a URL that uniquely designates a single article in blog-space and time.

The Virtuoso Blog system comprises a native Web-based interface for publishing and blog administration, but can also be interacted with using an XML-RPC API. The Virtuoso server supports the following XML-RPC APIs for blogging purposes:

Blogger API

MetaWeblog API

Movable Type

In addition to the blogger APIs Virtuoso supports:

xmlStorageSystem - this allows blogging clients to upload static (ready to go) content onto a Weblog server.

Weblogs ping API - to allow cross-server notification for the latest updated sites/blogs

Pingback and Trackback API - to allow bloggers to notify each other.

Subscription Harmonizer - to allow bloggers to keep their subscriptions in-sync.

See Also:
Blogger System Tables Blogger API

See Also:
MetaWeblog API

14.7.1. The Virtuoso Blogging Application

Blogs provide their authors with a location on the Internet or Intranet to receive their thoughts, experiences, idea, or simply anything they have a desire to write about. For this most part, audience is neither expected nor required, however comments and feedback are common place.

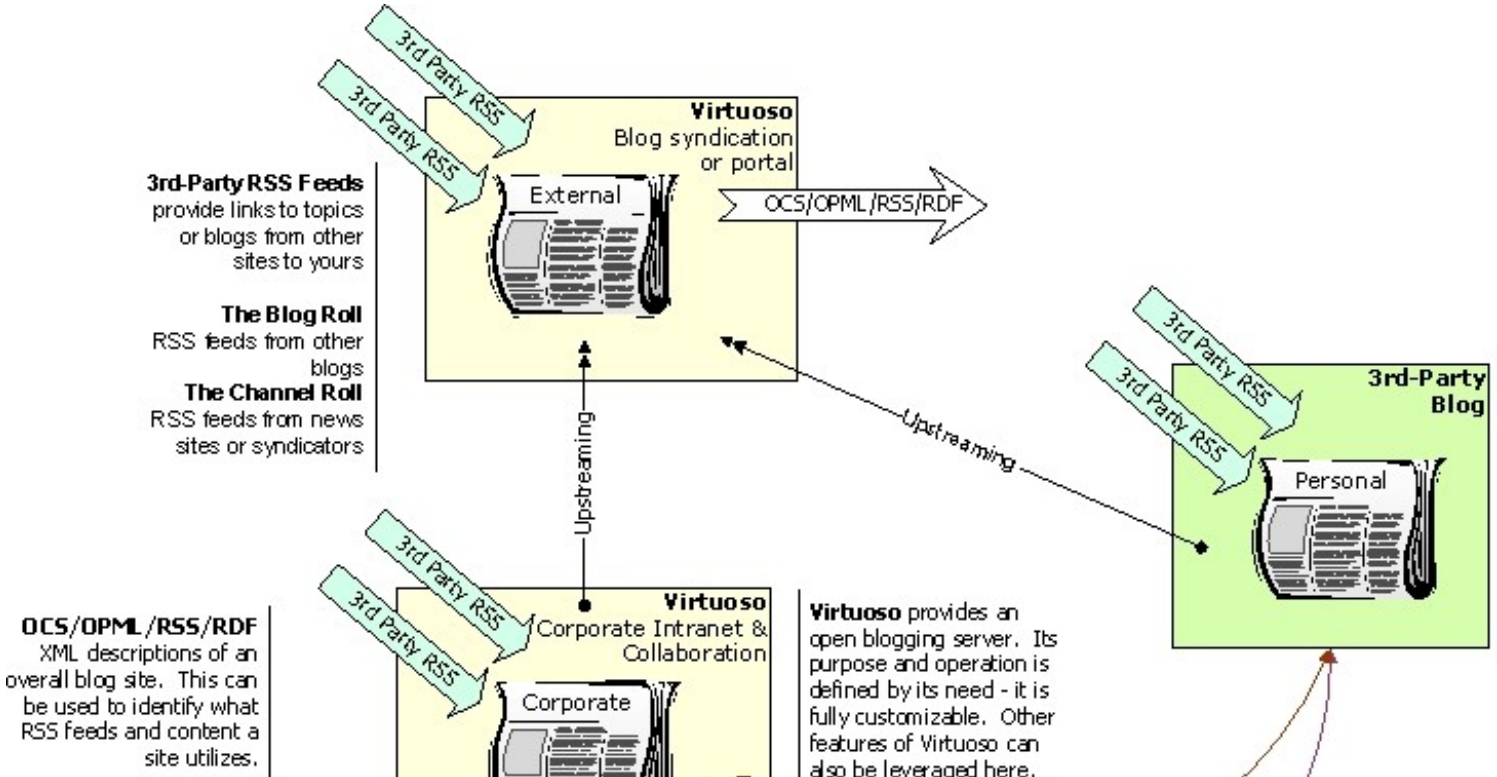
Blogs originated as a form of public diary. Their exploitation has seen them as a useful tool for finding personal reviews to products, situations or general news as guidance for their readers. These can be viewed quicker and more informative than regular journalism.

Corporate blogs, or use of blogging can be a valuable source of company information propagation. Members of the company can blog about the activities or findings with no particular audience in mind. This provides a continual journal or account of the company's undertakings that is really self-documenting for every other employee how they fit into the overall picture. Over time a searchable knowledge base will unravel itself, people may need information about a product or some aspect of internal support that took place possibly years prior, otherwise forgotten, stored within the blog.

Virtuoso implements the three major APIs for blogging which makes instantly attractive to any standard blog application. Virtuoso provides blog authoring applications of its own. On a personal level, your blogs can be stored or exposed using the most appropriate server, authored using the most appealing client and everything held in right places by Virtuoso. On a community level you get the same but scaled upwards for each member individually, and further more, centrally. Virtuoso can generate and automatically maintain a blog site specially for blog collaboration within a community suitable for a corporate need. All members are listed on the front page and have the opportunity to contribute (publish) to the main blog view. All blogs are fully free text searchable - a simple exploit of the Virtuoso free-text engine.

Virtuoso can provide the blog client, server or relay for blogs or any RSS or XML-feed based channel or information.

Figure 14.50. Virtuoso Blogging Conceptual Diagram



Syndication to your Blog - RSS & RDF

The default Virtuoso blog implementation maintains a "blog/gems" subdirectory for each blog account. Within this is an RDF file and an RSS file. These provide direct access to the blog information in XML. Other bloggers or news syndicators can read just the site contents, apart from the web interface that Virtuoso provides, potentially applying an XSLT transformations suitable to their application or circumstance, for example reading the blog from a PDA or mobile device.

Rich Site Summary (RSS) is a lightweight XML format designed for sharing headlines and other Web content. Think of it as a distributable "What's New" for your site. Each RSS text file contains both static information about your site, plus dynamic information about your new blog entries. Each entry is defined by an <item> tag, which contains a headline TITLE, URL, and DESCRIPTION. For example:

```
...
<item>
  <title>RSS Resources</title>
  <link>http://www.webreference.com/authoring/languages/xml/rss/</link>
  <description>Defined in XML, the Rich Site Summary (RSS) format has
  quietly become a dominant format for distributing headlines on the Web.
  Our list of links gives you the tools, tips and tutorials you need to get
  started using RSS. 0323</description>
</item>
...
```

The Resource Description Framework (RDF) integrates a variety of applications from library catalogs and world-wide directories to syndication and aggregation of news, software, and content to personal collections of music, photos, and events using XML as an interchange syntax. The RDF specifications provide a lightweight ontology system to support the exchange of knowledge on the Web.

Channels - OCS & OPML

The default Virtuoso blog implementation maintains a "blog/gems" subdirectory for each blog account. Within this are an OCS file and an OPML file. Both files provide an outline or hierarchical list of RSS feeds linked to the blog account in these two popular formats designed for channel/list exchange. The files are variably accessible following:

```
http://virtserv/DAV/myaccount/blog/gems/index.ocs
http://virtserv/DAV/myaccount/blog/gems/index.opml
```

Open Content Syndication (OCS), an application of XML, is an RDF description of all external RSS feeds linked to the blog site.

The OCS Directory format is designed to enable channel listings to be constructed for use by portal sites, client based headline software and other similar applications.

Outline Processor Markup Language (OPML) is a very simple XML format for storing information in outline format. XML being inherently hierarchical, OPML constrains XML so that a wide variety of applications can build in OPML support with the comfort of knowing it will work with any other OPML tool. Type of information stored in such hierarchies are web browser bookmarks, web directories, collaborative outlines, song playlists, and even web-site content, OPML is a great balance between the wide open freedom of raw XML and the feeling of security of a formal vocabulary.

The OCS RDF file is an XML of nested <rdf:descriptions> tags loosely defined as follows:

```
<rdf:RDF>
  <rdf:description>
      This contains a description of this RDF originator
    <rdf:description>
        Description of a link RSS blog or news feed
       <rdf:description>
         Description of the URL containing the above described
         RSS feed and details about its format, update frequency
         and when the link for established.
       </rdf:description>
    </rdf:description>
      more RSS descriptions may follow, one for each
      linked.
  </rdf:description>
</rdf:RDF>
```

Here is an example of the contents of the index.ocs file describing a single linked RSS news feed.

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:ocs="http://InternetAlchemy.org/ocs/directory#"
    xmlns:dc="http://purl.org/metadata/dublin_core#">
  <rdf:description about="">
  <dc:title>My Blog</dc:title>
  <dc:creator>blog@openlinksw.com</dc:creator>
  <dc:description />
  <rdf:description about="http://msdn.microsoft.com/">
    <dc:title>MSDN Just Published</dc:title>
    <dc:creator />
    <dc:description />
    <dc:subject />
    <dc:image />
    <rdf:description about="http://msdn.microsoft.com/rss.xml">
      <dc:language>us-en</dc:language>
      <ocs:format />
      <ocs:updatePeriod>hourly</ocs:updatePeriod>
      <ocs:updateFrequency>1</ocs:updateFrequency>
      <ocs:updateBase>1999-05-30T00:00</ocs:updateBase>
    </rdf:description>
  </rdf:description>
</rdf:RDF>
```

The OPML file is more simple, the corresponding index.opml would be as follows:

```
<opml>
  <body>
```

```
      <outline
        title="MSDN Just Published"
        htmlUrl="http://msdn.microsoft.com/"
        xmlUrl="http://msdn.microsoft.com/rss.xml" />
      more outline tags for each description...
    </body>
  </opml>
```

Personal Blog Sites

Every WebDAV account with a home directory has the option to create a default personal blog site. The default blog pages contains a call to a function that produces XML data containing a list of messages, calendar for archive navigation and channels that the owner has subscribed to. This blog directory and pages are generated upon initial sign-in of the user. Each WebDAV account with a blog directory will thus have the following:

```
http://<host:port>/DAV/<uname>/blog/
```

containing the following:

index.vspx - The default blog home page using VSPX controls.

gems/rss.xml - The RSS v2.0 file presenting the user's blog data.

gems/rss92.xml - The RSS v0.92 file presenting the user's blog data.

gems/rss91.xml - The RSS v0.91 file presenting the user's blog data.

gems/index.rdf - the user's blog data in RDF format.

gems/index.ocs - the user's channels represented in OCS format.

gems/rsd.xml - Really Simple Discovery file for the blog. This helps client software to find the services needed to read, edit, post etc.

gems/atom.xml - Atom feed file. It's similar to the RSS file but in different format.

gems/backlog.xml - RSS v2.0 file containing all posts in user's blog.

gems/foaf.xml - 'Friend Of Friend' file. This is to provide in a machine-processable format, about blogger, his/she interests, location, friends etc.

gems/index.opml - the user's channels represented in OPML format.

gems/ocs.xml - the user's channels represented in OCS format.

gems/opml.xml - the user's channels represented in OPML format.

comments.vsp - a VSP page for posting a comment to the user's blog.

default.xsl - an XSL-T style sheet providing the default HTML view of the blog XML data.

comments.vsp - pop-up for blog comments (obsoleted)

default.css - CSS for pages

main.xsl - VSPX controls for index.vspx

calendar.xsl - calendar control

trackback.xsl - pop-up for trackbacks (obsoleted)

rss_feeds/ - folder containing channel RSS feeds in XML format

The `rss.xml` , `index.rdf` , `index.ocs` files are dynamic SQL-XML resources, they are not static files and must not be deleted. Their content is generated in real-time per request.

Blog users may edit the `default.xsl` in order to change the appearance of the default page of their Blog.

Example 14.52. Default blog home (index.vspx) page

The following is source of default blog home page. It's built using a VSPX and macro expansion. All page components are represented with <vm:* /> this makes page more simpler and allows users to change it's appearance using custom VSPX components.

```
<v:page xmlns:vm="http://example.com/vspx/weblog/"
  xmlns:v="http://example.com/vspx/"
  style="main.xsl" name="blog-home-page"
  doctype="-//W3C//DTD XHTML 1.0 Transitional//EN"
  doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
  <vm:page
xml:id="<uid>">
    <vm:header>
      <vm:title/>
      <vm:disco-rss-link/>
      <vm:disco-pingback-link/>
      <vm:meta-author/>
      <vm:meta-description/>
      <vm:meta-keywords/>
      <vm:style url="default.css" />
    </vm:header>
    <vm:body>
      <div
xml:id="header">
        <img src="/blog/blogs/images/bloglogo.jpg" alt="OpenLink Virtuoso Blog"/>
        <h1><vm:blog-title/></h1>
        <h2><vm:blog-about/></h2>
      </div>
      <div
xml:id="navbartop">
        <div>Entries: [ <vm:entry-list /> ]</div>
      </div>
      <table
xml:id="pagecontainer" cellspacing="10" cellpadding="0" border="0" width="100%">
        <tr>
          <td class="box">
             <div class="box">
                  <div class="roll" style="border: none; border">
                    <div align="center" style="margin-bottom: 3px;">
                      <b>Personal Details</b>
                    </div>
                    <div style="margin-bottom: 3px;"><vm:site-home>Home</vm:site-home></div>
                    <vm:if test="contact">
                    <div>
                      <vm:e-mail> Contact</vm:e-mail>
                    </div>
                    </vm:if>
                    <div>
                      <vm:photo width="64" />
                    </div>
                  </div>
                  <vm:if test="login">
                  <div class="roll" >
                    <div align="center" style="margin-bottom: 3px;">
                      <b>Configuration</b>
                    </div>
                    <vm:configpages />
                  </div>
                  </vm:if>
                  <div class="roll">
                  <div align="center" style="margin-bottom: 3px;">
                    <b>Syndicate This Blog</b>
                  </div>
                    <vm:rss-link/>
                    <vm:rdf-link/>
                    <vm:ocs-link/>
                    <vm:opml-link/>
                  </div>
                  <div align="left" class="roll">
                    <p class="caption">Keyword search:</p>
                    <vm:search/>
                  </div>
                  <div class="roll">
                    <div align="center" style="margin-bottom: 3px;">
                      <b>Post Categories</b>
                    </div>
                    <vm:login />
                    <vm:categories/>
                  </div>
                <vm:if test="subscribe">
                <div align="left" class="roll">
                  <p class="caption">Subscribe</p>
                  <vm:subscribe/>
```

```
                </div>
                </vm:if>
                <div align="left" class="roll">
                   <vm:last-messages />
                </div>
              </div>
            </td>
            <td
xml:id="texttd">
              <div
xml:id="text">
                <vm:post-form />
                <vm:posts trackback="discovery" />
                <vm:if test="comments">
                <vm:comments-view>
                  <vm:trackbacks />
                  <vm:comments />
                  <vm:post-comment />
                </vm:comments-view>
                </vm:if>
              </div>
            </td>
            <td class="box">
              <div class="box">
                <vm:calendar/>
                <vm:if test="blog">
                <div class="roll">
                  <div align="center">
                    <b>Blog Roll</b>
                  </div>
                  <vm:rss/>
                </div>
                </vm:if>
                <vm:if test="channels">
                <div class="roll">
                  <div align="center">
                    <b>Channel Roll</b>
                  </div>
                  <vm:channels/>
                </div>
                </vm:if>
                <vm:if test="ocs">
                <div class="roll">
                  <div align="center" style="margin-bottom: 3px;">
                    <b>OCS Links</b>
                  </div>
                  <vm:ocs/>
                </div>
                </vm:if>
                <vm:if test="opml">
                <div class="roll">
                  <div align="center" style="margin-bottom: 3px;">
                    <b>OPML Links</b>
                  </div>
                  <vm:opml/>
                </div>
                </vm:if>
              </div>
            </td>
          </tr>
        </table>
        <div
xml:id="powered">
          <vm:powered-by/>
        </div>
        <div class="disclaimer">
        <vm:disclaimer/>
          </div>
        <div class="copy">
          <vm:copyright/>
        </div>
      </vm:body>
    </vm:page>
```

```
</v:page>
```

Example 14.53. Blog home (index.vsp) page

The XML Schema representing the XML data for blog home page is represented below. Following this fragment is the `default.xsl` XSL style sheet that would be used to transform XML document valid to this into the default web page for public consumption on the Internet or Intranet.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="about">
    <xs:complexType/>
  </xs:element>
  <xs:element name="blog">
    <xs:complexType mixed="true">
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="title"/>
        <xs:element ref="about"/>
        <xs:element ref="email"/>
        <xs:element ref="items"/>
        <xs:element ref="navigation"/>
        <xs:element ref="copy"/>
        <xs:element ref="disclaimer"/>
      </xs:choice>
      <xs:attribute name="base" type="xs:anyURI"/>
      <xs:attribute name="category" type="xs:string"/>
      <xs:attribute name="category-name" type="xs:string"/>
      <xs:attribute name="post" type="xs:string"/>
      <xs:attribute name="trackback-url" type="xs:anyURI"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="blogroll">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="link"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:boolean" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="calendar">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="week" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="monthname" type="xs:string" use="required"/>
      <xs:attribute name="year" type="xs:short" use="required"/>
      <xs:attribute name="month" type="xs:byte" use="required"/>
      <xs:attribute name="day" type="xs:byte" use="required"/>
      <xs:attribute name="prev" type="xs:string" use="required"/>
      <xs:attribute name="prev-label" type="xs:string" use="required"/>
      <xs:attribute name="next" type="xs:string" use="required"/>
      <xs:attribute name="next-label" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="categories">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="category" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="category">
    <xs:complexType>
      <xs:attribute name="id" use="required" type="xs:NMTOKEN" />
      <xs:attribute name="name" use="required" type="xs:string" />
    </xs:complexType>
  </xs:element>
  <xs:element name="comments" type="xs:boolean"/>
  <xs:element name="copy" type="xs:string"/>
  <xs:element name="day">
```

```
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute name="active" use="required">
            <xs:simpleType>
              <xs:restriction base="xs:NMTOKEN">
                <xs:enumeration value="0"/>
                <xs:enumeration value="1"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="description" type="xs:string"/>
  <xs:element name="disclaimer" type="xs:string"/>
  <xs:element name="email" type="xs:string"/>
  <xs:element name="id" type="xs:string" />
  <xs:element name="item">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="description"/>
        <xs:element ref="pubDate"/>
        <xs:element ref="id"/>
        <xs:element ref="comments"/>
        <xs:element ref="trackbacks"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="items">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="item" maxOccurs="unbounded"/>
        <xs:element ref="last"/>
      </xs:sequence>
      <xs:attribute name="search" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="last">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="message" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="link">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="blog"/>
      </xs:sequence>
      <xs:attribute name="href" type="xs:anyURI" use="required"/>
      <xs:attribute name="rss" type="xs:anyURI" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="message">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="id"/>
        <xs:element ref="title"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="navigation">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="calendar"/>
        <xs:element ref="blogroll"/>
        <xs:element ref="categories"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="pubDate" type="xs:string"/>
```

```
  <xs:element name="title">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute name="blogid" type="xs:byte"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="trackbacks" type="xs:boolean"/>
  <xs:element name="week">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="day" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The XML data as following XML Schema above will be transformed using the following XSL-T style sheet, default.xsl:

```
<?xml version="1.0"?>

<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0"
    xmlns:vb="http://example.com/weblog/"
    xmlns:vtb="http://example.com/weblog/tb/" >

  <xsl:output method="xml" doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
    doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
        indent="yes" encoding="UTF-8" omit-xml-declaration="yes" media-type="text/html"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>
          <xsl:value-of select="//title[1]"/>
        </title>
        <link rel="alternate" type="application/rss+xml" title="RSS" href="{/blog/@base}gems/rss.xml"/>
        <link rel="stylesheet" type="text/css" href="default.css"/>
      </head>
      <body>
        <div
xml:id="header_01">
          <img src="/blog/blogs/images/bloglogo.jpg" alt="OpenLink Virtuoso Blog"/>
          <h1>
            <xsl:value-of select="/blog/title"/>
          </h1>
          <xsl:if test="/blog/about != ''">
          <h2><xsl:value-of select="/blog/about"/></h2>
          </xsl:if>
        </div>
        <div
xml:id="navbartop_01">
          <div>Entries: [ <xsl:call-template name="entrylist"/> ]</div>
        </div>
        <table
xml:id="pagecontainer_01" cellspacing="10" cellpadding="0" border="0" width="100%">
          <tr>
            <td class="box">
              <div class="box">
                <div align="left" class="roll" style="border: none; border">
                    <div align="center" style="margin-bottom: 3px;">
                      <b>Personal Details</b>
                    </div>
                  <div>
                  <a>
                    <xsl:attribute name="href">mailto:<xsl:value-of select="/blog/email" /></xsl:attribut
                  <img src="/blog/blogs/images/mailto.gif" border="0" alt="mailto"/>
                  </a> Contact
                  </div>
                </div>
```

```
                  <div class="roll">
                    <div align="center" style="margin-bottom: 3px;">
                      <b>Syndicate This Blog</b>
                    </div>
                    <div>
                      <a href="gems/rss.xml">
                        <img src="/blog/blogs/images/xml.gif" border="0" alt="rss"/>
                        RSS</a>
                    </div>
                    <div>
                      <a href="gems/index.rdf">
                        <img src="/blog/blogs/images/rdf48.gif" border="0" alt="rdf"/>
                        RDF</a>
                    </div>
                    <div>
                      <a href="gems/index.ocs"><img src="/blog/blogs/images/xml.gif" border="0" alt="ocs"/>
                    </div>
                    <div>
                      <a href="gems/index.opml"><img src="/blog/blogs/images/xml.gif" border="0" alt="opml"
                    </div>
                  </div>
                  <div align="left" class="roll">
                    <p class="caption">Keyword search</p>
                    <form method="post" action="index.vsp">
                      <div>
                        <input type="text" name="txt" value="" size="10"/>
                        <input type="submit" name="GO" value="GO"/>
                      </div>
                      <div>
                        <input type="radio" name="srch_where" value="blog" checked="checked"/> My Blog
                      <div>
                        <input type="radio" name="srch_where" value="web"/> The Web</div>
                    </form>
                  </div>
                  <xsl:apply-templates select="/blog/navigation/categories"/>
                  <div align="left" class="roll">
                    <xsl:for-each select="/blog/items/last/message">
                      <div>
                        <a>
                          <xsl:attribute name="href">index.vsp?id=<xsl:value-of select="id" /></xsl:attribu
                          <xsl:value-of select="title" />
                        </a>
                      </div>
                    </xsl:for-each>
                  </div>
                </div>
              </td>
              <td
xml:id="texttd_01">
                <xsl:apply-templates select="/blog/items"/>
              </td>
              <td class="box">
                <div class="box">
                  <xsl:apply-templates select="/blog/navigation/calendar"/>
                  <xsl:apply-templates select="/blog/navigation/blogroll"/>
                  <xsl:apply-templates select="/blog/navigation/channelroll"/>
                  <xsl:apply-templates select="/blog/navigation/ocs"/>
                  <xsl:apply-templates select="/blog/navigation/opml"/>
                </div>
              </td>
            </tr>
          </table>
        </body>
      </html>
    </xsl:template>

    <xsl:template match="title|navigation"/>

    <xsl:template name="entrylist">
      <xsl:for-each select="/blog/items/item">
        <a href="#{id}">
          <xsl:number level="multiple" format=" 1 " count="item"/>
        </a>
```

```
      <xsl:if test="following-sibling::item"> | </xsl:if>
    </xsl:for-each>
  </xsl:template>

  <xsl:template match="items">
    <div
xml:id="text_01">
      <xsl:if test="/blog/@category-name != ''">
        <h3>Category: "<xsl:value-of select="/blog/@category-name"/>"</h3>
      </xsl:if>
      <xsl:apply-templates select="item"/>
      <xsl:if test="/blog/@post != ''">
      <div><a name="tb" /><h3>TrackBacks</h3>

<div class="tb-url">TrackBack URL for this entry:
<br/>
<b>
<xsl:value-of select="/blog/@trackback-url"/>
<xsl:value-of select="/blog/@post"/>
</b>
<br />
      PingBack URL for this entry:<br/>
      <b><xsl:value-of select="/blog/@pingback-url"/></b>
      </div>
      <xsl:apply-templates select="trackbacks"/>
      </div>
      <div><a name="comments" /><h3>Comments</h3>
      <xsl:if test="not post-comments">
      <div>No comments posted yet</div>
      </xsl:if>
      <xsl:apply-templates select="post-comments"/>
      <div>
        <form method="post" action="index.vsp">
        <input type="hidden" name="id">
            <xsl:attribute name="value"><xsl:value-of select="/blog/@post" /></xsl:attribute>
        </input>
        <table border="0">
        <tr><td>Name</td><td><input type="text" name="name" value="" size="50"/></td></tr>
        <tr><td>Email</td><td><input type="text" name="email" value=""  size="50"/></td></tr>
        <tr><td>Web Site</td><td><input type="text" name="url" value=""  size="50"/></td></tr>
        <tr><td colspan="2">Comment</td></tr>
        <tr><td colspan="2"><textarea name="comment" rows="15" cols="50"> </textarea></td></tr>
        <tr><td colspan="2"><input type="submit" name="submit" value="Submit" /></td></tr>
        </table>
        </form>
      </div>
      </div>
      </xsl:if>
      <xsl:choose>
        <xsl:when test="not item and @search != ''">
  No messages found containing "<xsl:value-of select="@search"/>".
  </xsl:when>
      <xsl:when test="not item and /blog/@category != ''">
  No messages found for category  "<xsl:value-of select="/blog/@category-name"/>".
  </xsl:when>
      <xsl:when test="not item and @search = '' and /blog/@cat = ''">
        <div class="message">
  This is a placeholder for your new weblog.
  There are no posts currently.
  </div>
      </xsl:when>
    </xsl:choose>
    <div
xml:id="powered_01">
      <a href="http://example.com/virtuoso/">
        <img src="/blog/blogs/images/PoweredByVirtuoso.gif" border="0" alt="powered by" />
      </a>
    </div>
    <div class="disclaimer">
      <xsl:value-of  select="/blog/disclaimer" disable-output-escaping="yes" />
    </div>
    <div class="copy">
      <xsl:value-of select="/blog/copy" disable-output-escaping="yes" />
```

```
      </div>
    </div>
  </xsl:template>

  <xsl:template match="item">
    <a name="{id}"/>
    <div class="message">
      <xsl:apply-templates select="pubDate"/>
      <xsl:apply-templates select="description"/>
      <xsl:if test="function-available('vb:mt_track_back_discovery')">
        <xsl:value-of select="vb:mt_track_back_discovery (id)" disable-output-escaping="yes" />
      </xsl:if>
      <div class="comment">
        <a href="#">
          <xsl:attribute name="href">index.vsp?id=<xsl:value-of select="id"/>#comments</xsl:attribute>
        Comments [<xsl:value-of select="comments"/>]
   </a> | <a href="#">
          <xsl:attribute name="href">index.vsp?id=<xsl:value-of select="id"/>#tb</xsl:attribute>
        TrackBack [<xsl:value-of select="trackbacks"/>]
        </a>
      </div>
    </div>
  </xsl:template>

  <xsl:template match="post-comments">
    <div class="message">
      <xsl:apply-templates select="posted"/>

<a>
<xsl:attribute name="href">
<xsl:value-of select="home"/>
</xsl:attribute>
<b>
<xsl:apply-templates select="name"/>
</b>
</a>
      <div class="desc">
        <xsl:value-of select="vb:tidy_xhtml (comment, '*default*')" disable-output-escaping="yes"/>
      </div>
    </div>
  </xsl:template>

  <xsl:template match="trackbacks">
    <div class="message">
      <xsl:apply-templates select="posted"/>

<a>
<xsl:attribute name="href">
<xsl:value-of select="url"/>
</xsl:attribute>
<b>
<xsl:apply-templates select="blog-name"/>
</b>
</a>
      <div class="desc">
      <b><xsl:apply-templates select="title"/></b>
        <xsl:value-of select="vb:tidy_xhtml (excerpt, '*default*')" disable-output-escaping="yes"/>
      </div>
    </div>
  </xsl:template>

  <xsl:template match="description">
    <div class="desc">
      <xsl:choose>
      <xsl:when test="function-available('vb:tidy_xhtml')">
        <xsl:value-of select="vb:tidy_xhtml (., '*default*')" disable-output-escaping="yes"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="." disable-output-escaping="yes"/>
      </xsl:otherwise>
      </xsl:choose>
    </div>
  </xsl:template>
```

```
  <xsl:template match="pubDate">
    <div class="pubdate">
      <xsl:value-of select="."/>
    </div>
  </xsl:template>

  <xsl:template match="posted">
    <div class="pubdate">
      <xsl:value-of select="."/>
    </div>
  </xsl:template>

  <xsl:template match="blogroll">
    <div class="roll">
      <div align="center">
        <b>Blog Roll</b>
      </div>
      <xsl:apply-templates/>
    </div>
  </xsl:template>

  <xsl:template match="channelroll">
    <div class="roll">
      <div align="center">
        <b>Channel Roll</b>
      </div>
      <xsl:apply-templates/>
    </div>
</xsl:template>

  <xsl:template match="opml">
    <div class="roll">
      <div align="center" style="margin-bottom: 3px;">
        <b>OPML Links</b>
      </div>
      <xsl:for-each select="link">
        <a>
          <xsl:attribute name="href"><xsl:value-of select="@rss"/></xsl:attribute>
          <b>
            <xsl:value-of select="blog"/>
          </b>
        </a>
        <div style="margin-left:1em;">
          <xsl:apply-templates select="link"/>
        </div>
      </xsl:for-each>
    </div>
  </xsl:template>

  <xsl:template match="ocs">
    <div class="roll">
      <div align="center" style="margin-bottom: 3px;">
        <b>OCS Links</b>
      </div>
      <xsl:for-each select="link">
        <a>
          <xsl:attribute name="href"><xsl:value-of select="@rss"/></xsl:attribute>
          <b>
            <xsl:value-of select="blog"/>
          </b>
        </a>
        <div style="margin-left:1em;">
          <xsl:apply-templates select="link"/>
        </div>
      </xsl:for-each>
    </div>
  </xsl:template>

  <xsl:template match="categories[category]">

  <xsl:variable name="dt" select="concat(//calendar/@year, '-', //calendar/@month, '-', //calendar/@day)"
    <div class="roll">
```

```
      <div align="center" style="margin-bottom: 3px;">
        <b>Post Categories</b>
      </div>
      <div>
        <a>
          <xsl:attribute name="href">index.vsp?date=<xsl:value-of select="$dt"/>&amp;cat=</xsl:attribute>
          <b>All</b>
        </a>
      </div>
      <xsl:for-each select="category">
        <div>
          <a >
            <xsl:attribute name="href">/blog/rss_cat.xml?:cid=<xsl:value-of select="@id"/>&amp;:bid=<xsl:
            <img src="/blog/blogs/images/mxml.gif" border="0" alt="rss"/>
          </a>
          <a>
            <xsl:attribute name="href">index.vsp?date=<xsl:value-of select="$dt"/>&amp;cat=<xsl:value-of
            <b>
              <xsl:value-of select="@name"/>
            </b>
          </a>
        </div>
      </xsl:for-each>
    </div>
  </xsl:template>

  <xsl:template match="link">
    <div>
      <xsl:if test="@rss != ''">
        <a>
          <xsl:attribute name="href"><xsl:value-of select="@rss"/></xsl:attribute>
          <img src="/blog/blogs/images/mxml.gif" border="0" alt="rss"/>
        </a>
      </xsl:if>
      <a>
        <xsl:attribute name="href"><xsl:value-of select="@href"/></xsl:attribute>
        <xsl:value-of select="." disable-output-escaping="yes" />
      </a>
    </div>
  </xsl:template>

  <xsl:template match="calendar">
    <table
xml:id="calendar">
      <caption>
        <xsl:value-of select="@monthname"/>
        <xsl:text> </xsl:text>
        <xsl:value-of select="@year"/>
      </caption>
      <tr>
        <th>Sun</th>
        <th>Mon</th>
        <th>Tue</th>
        <th>Wed</th>
        <th>Thu</th>
        <th>Fri</th>
        <th>Sat</th>
      </tr>
      <xsl:apply-templates/>
      <tr>
        <td colspan="3">
          <xsl:if test="@prev != ''">
            <a>
              <xsl:attribute name="href">index.vsp?date=<xsl:value-of select="@prev"/>&amp;cat=<xsl:value
              <xsl:value-of select="@prev-label"/>
            </a>
          </xsl:if> 
        </td>
        <td> </td>
        <td colspan="3">
          <xsl:if test="@next != ''">
         
        <a>
```

```
            <xsl:attribute name="href">index.vsp?date=<xsl:value-of select="@next"/>&amp;cat=<xsl:value
            <xsl:value-of select="@next-label"/>
          </a>
        </xsl:if>
      </td>
    </tr>
  </table>
</xsl:template>

<xsl:template match="week">
  <tr>
    <xsl:apply-templates/>
  </tr>
</xsl:template>

<xsl:template match="day">
  <xsl:variable name="dt" select="concat(ancestor::calendar/@year, '-', ancestor::calendar/@month, '-')
  <td>
    <xsl:choose>
      <xsl:when test="boolean(number(@active))">
        <xsl:attribute name="class">calactive</xsl:attribute>
        <a>
          <xsl:attribute name="href">index.vsp?date=<xsl:value-of select="$dt"/><xsl:value-of select=".
          <xsl:apply-templates/>
        </a>
      </xsl:when>
      <xsl:otherwise>
        <xsl:apply-templates/>
      </xsl:otherwise>
    </xsl:choose>
  </td>
</xsl:template>
</xsl:stylesheet>
```

The XML Schema for RSS v2.0 file (rss.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="channel">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="title"/>
        <xs:element ref="link"/>
        <xs:element ref="description"/>
        <xs:element ref="managingEditor"/>
        <xs:element ref="pubDate"/>
        <xs:element ref="generator"/>
        <xs:element ref="webMaster"/>
        <xs:element ref="image"/>
        <xs:element ref="cloud"/>
        <xs:element ref="item" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="cloud">
    <xs:complexType>
      <xs:attribute name="domain" type="xs:string" use="required"/>
      <xs:attribute name="port" type="xs:short" use="required"/>
      <xs:attribute name="path" type="xs:string" use="required"/>
      <xs:attribute name="registerProcedure" type="xs:string" use="required"/>
      <xs:attribute name="protocol" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="comments" type="xs:anyURI"/>
  <xs:element name="description" type="xs:string"/>
  <xs:element name="generator" type="xs:string"/>
  <xs:element name="guid" type="xs:anyURI"/>
  <xs:element name="height" type="xs:int"/>
  <xs:element name="image">
    <xs:complexType>
```

```
        <xs:sequence>
          <xs:element ref="title"/>
          <xs:element ref="url"/>
          <xs:element ref="link"/>
          <xs:element ref="description"/>
          <xs:element ref="width"/>
          <xs:element ref="height"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="item">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="title"/>
          <xs:element ref="guid"/>
          <xs:element ref="comments"/>
          <xs:element ref="pubDate"/>
          <xs:element ref="description"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="link" type="xs:anyURI"/>
    <xs:element name="managingEditor" type="xs:string"/>
    <xs:element name="pubDate" type="xs:string"/>
    <xs:element name="rss">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="channel" minOccurs="1" maxOccurs="1"/>
        </xs:sequence>
        <xs:attribute name="version" type="xs:string" use="required" fixed="2.0"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="title" type="xs:string"/>
    <xs:element name="url" type="xs:anyURI"/>
    <xs:element name="webMaster" type="xs:string"/>
    <xs:element name="width" type="xs:int"/>
  </xs:schema>
```

14.7.2. Blogger Clients Compatibility

The Virtuoso Blog server implementation has been tested against the following blog client applications:

Radio Userland (Manila Blogger Bridge tool and Upstreaming), available for MS Windows and MacOS X

NewzCrawler , available for MS Windows

w.bloggar , available for MS Windows

blogBuddy , available for MS Windows

Zempt , available for MS Windows

BlogMan , available for MS Windows

thinblog , Java based client

The steps required to allow third-party clients to operate with Virtuoso are:

1. Create a WebDAV account using the Server Administration Interface making sure that the home directory is created as /DAV/<username>/
2. Enable the XML-RPC bridge on a SOAP endpoint (see "XML-RPC endpoint configuration" section above)
3. Most client tools accept a username and password, enter the credentials and URI for XML-RPC endpoint.
4. Allow the client tool time to refresh the Blog data.
5. Post a simple message, verify its creating using the Server Administration Interface's Weblog link.

Interoperability Notes. The "Zempt" application will report an error upon startup saying that "mt.supportedTextFilters" are not supported, however it it will continue to work with the Virtuoso's blogging server.

14.7.3. Blogs Management User Interface

The Server Administration Interface provides a way to manage existing blogs. The Blog user interface is accessible from the Web Services >> Weblogs menu. The page will show existing blogs on the server and will allow you to erase, edit existing, or make

new blog sites.

Blog channels and categories are also configurable from this interface.

14.7.4. Community Blog Site

The community blog site feature pools all blog efforts into one page providing a blog community portal. No personal blogs are altered, the community site provides a super-set for all blogs.

The community blog site is activated by enabling blogging capabilities on the DAV administrator account "dav". This can be achieved at two places, on the Users Administration interface or the Web Services >> Weblogs interface in the Server Administration Interface menu.

14.7.5. Blogger API

The Virtuoso server supports the following methods available via XML-RPC for Weblog management and operation:

```
blogger.newPost()

blogger.editPost()

blogger.deletePost()

blogger.getRecentPosts()

blogger.getUsersBlogs()

blogger.getTemplate()

blogger.setTemplate()

blogger.getUserInfo()
```

Note:
First name is the Movable Type "username" up to the first space character, and last name is the "username" after the first space character.

14.7.6. MetaWeblog API

```
metaWeblog.newPost()

metaWeblog.editPost()

metaWeblog.getPost()

metaWeblog.getRecentPosts()
```

14.7.7. Movable Type API

```
mt.getRecentPostTitles()

mt.getCategoryList()

mt.setPostCategories()

mt.getPostCategories()

mt.getTrackbackPings()

mt.publishPost()
```

```
mt.supportedMethods()
```

Note:
the value of "appkey" is ignored by Movable Type in all of the Blogger XML-RPC methods.

14.7.8. Atom API

The Atom API is a protocol for publishing and editing a blog entries using the common HTTP verbs:

GET - used to retrieve a blog post or perform query

PUT - used to update an existing blog post

POST - used to create a new post or perform an update action

DELETE - used to remove a post

Authentication. The Atom API implementation accepts both HTTP basic and digest authentication. All requests for post content modification require an authorization. Furthermore, the account used for authentication is checked to be the blog owner or have granted privileges on that particular blog. The retrieval operations; which no post modification is possible are not required to ask for authentication.

Introspection. When editing the content of a blog, the first thing is to find out the servers capabilities. So the 'introspection' file lists all the functions that a given site supports.

Example 14.54. Intropsection

Request:

```
GET /Atom/Http/intro?b=128 HTTP/1.1
Host: example.com
```

Response:

```
HTTP/1.1 200 OK
Server: Virtuoso
Connection: Keep-Alive
Date: Mon, 10 Nov 2003 10:21:55 GMT
Content-Type: text/xml; charset="ISO-8859-1"
Content-Length: 417

<?xml version="1.0" encoding="ISO-8859-1"?>
<introspection xmlns="http://example.com/newformat#" >
  <create-entry>http://example.com/Atom/Http/entry?b=128</create-entry>
  <user-prefs>http://example.com/Atom/Http/prefs?b=128</user-prefs>
  <search-entries>http://example.com/Atom/Http/search?b=128</search-entries>
  <categories>http://example.com/Atom/Http/categories?b=128</categories>
</introspection>
```

Posting a new entry. To post a new entry, the HTTP POST must be performed with MIME type '`application/x.atom+xml` ' for the request content. If server accepts the request a `Location:` HTTP header will be returned to the client containing the URL to the newly created post.

Example 14.55. Posting an Entry

```
POST /Atom/Http/entry?b=128 HTTP/1.1
Content-Type: application/x.atom+xml
Content-Length: nnn
Authorization: ...

<?xml version="1.0" encoding="iso-8859-1"?>
<entry xmlns="http://example.com/newformat#">
  <title>My Entry</title>
  <author>
    <name>Blog Author</name>
  </author>
  <issued>2003-11-04T17:29:29</issued>
```

```
  <content type="application/xhtml+xml" xml:lang="en-us">
    <p xmlns="">Hello world!</p>
  </content>
</entry>
```

The response from server would be:

```
HTTP/1.1 201 Created
Location: http://host:port/Atom/Http/entry?b=128&id=nn
```

Editing an Existing Entry. An entry can be edited via the HTTP PUT method. The URL used for PUT must be one returned by the `Location:` header of the POST response or those returned from entry-search. The content of PUT request is same as for posting a new entry.

Example 14.56. Editing a weblog entry

```
POST /Atom/Http/entry?b=128&id=nn HTTP/1.1
Content-Type: application/x.atom+xml
Content-Length: nnn
Authorization: ...

<?xml version="1.0" encoding="iso-8859-1"?>
<entry xmlns="http://example.com/newformat#">
  <title>My Entry</title>
  <author>
    <name>Blog Author</name>
  </author>
  <issued>2003-11-04T17:29:29</issued>
  <content type="application/xhtml+xml" xml:lang="en-us">
    <p xmlns="">Hello world! (edit test)</p>
  </content>
</entry>
```

The server response is:

```
HTTP/1.1 205 Reset Content
```

Retrieving an Entry. Retrieval of an existing entry is the simple, usual HTTP GET request with URL to the entry.

Example 14.57. Retrieving an Entry

```
GET /Atom/Http/entry?b=128&id=nn HTTP/1.1
Accept: application/x.atom+xml
```

Server will respond with

```
HTTP/1.1 200 OK
Content-Type: application/x.atom+xml
Content-Length: nnn

<?xml version="1.0" encoding="iso-8859-1"?>
<entry xmlns="http://example.com/newformat#">
  <title>My Entry</title>
  <author>
    <name>Blog Author</name>
  </author>
  <issued>2003-11-04T17:29:29</issued>
  <content type="application/xhtml+xml" xml:lang="en-us">
    <p xmlns="">Hello world! (edit test)</p>
  </content>
</entry>
```

Deleting an Entry. To remove permanently an existing entry an HTTP DELETE method is used with URL to the entry.

Example 14.58. Deleting an Entry

```
DELETE /Atom/Http/entry?b=128&id=nn HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK
```

Searching Entries. To retrieve a subset of messages based some criteria a search should be performed. The current implementation supports two variants: range and last_N.

Example 14.59. Searching

```
GET /Atom/Http/search?b=128?last=7 HTTP/1.1
Accept: application/x.atom+xml
Host: example.com
Authorization: ...
```

or

```
GET /Atom/Http/search?b=128&start-range=2&end-range=27 HTTP/1.1
Accept: application/x.atom+xml
Host: example.com
Authorization: ...
```

Response from server will look like:

```
<search-results xmlns="http://example.com/newformat#">
<match title="Classic ASP Framework. Make your Classic...">http://example.com/Atom/Http?b=128&amp;id=6</m
<match title="DB Navigator - an open source .NET control">http://example.com/Atom/Http?b=128&amp;id=4</ma
<match title="Sort an Array with Non-Comparable Items">http://example.com/Atom/Http?b=128&amp;id=5</match
<match title="SignedXML Licensing Sample">http://example.com/Atom/Http?b=128&amp;id=7</match>
<match title="TicTacToe">http://example.com/Atom/Http?b=128&amp;id=1</match>
<match title="Check your eMails in Hotmail using HTTPMail">http://example.com/Atom/Http?b=128&amp;id=2</m
<match title="Slashdot.org RSS Feed Reader">http://example.com/Atom/Http?b=128&amp;id=3</match>
</search-results>
```

Each sub element of the search result contains the title of the post and its URL. These URLs can be used retrieved using the GET method, or modified via PUT/DELETE methods.

Atom Client API

The Atom Client API is a mirror of the blogger API functionality, but uses the Atom protocol instead of XML-RPC. Before the Atom API can be put to use, the introspection file must be retrieved in order to know the URLs for the requests.

- `atom.new_Post (in uri varchar, in req "blogRequest", )`

  This is to post a new entry to the blog identified by 'uri'. The 'req' must be initialized with authentication and entry data. This function returns the URL to the new entry. This could be used with sequential GET/PUT or DELETE requests for it.
- `atom.edit_Post (in uri varchar, in req "blogRequest", )`

  This is to modify an existing entry identified by 'uri'. The 'req' must be initialized with authentication and entry data. The function returns true on success or will signal an error.
- `atom.delete_Post (in uri varchar, in req "blogRequest", )`

  This is to delete an existing entry identified by 'uri'. The 'req' must be initialized with authentication data. The function returns true on success or will signal an error.
- `atom.get_Post (in uri varchar, in req "blogRequest", )`

This is to retrieve an existing post entry identified by 'uri'. The 'req' must be initialized with authentication data. This function will return on success an MTWeblogPost UDT, filled with post entry data.

- `atom.get_RecentPosts (in uri varchar, in req "blogRequest", in num int, )`

  This is to retrieve last-N post entries identified by blog search URL - 'uri'. The 'req' must be initialized with authentication data. 'num' is number of entries to return.

  This function returns an array of arrays containing the title and URL of the matching blog posts.
- `atom.get_Intro (in uri varchar, )`

  This is used to return Introspection information for a blog identified by introspection file - 'uri'.

Example 14.60. Using the Atom Client API

Getting the introspection info about a blog

```
declare x any;
x := atom.get_Intro ('http://example.com/Atom/Http/intro?b=128');
-- x will contain
-- (("create-entry" "http://example.com/Atom/Http/entry?b=128" )
--  ("user-prefs"
-- "http://example.com/Atom/Http/prefs?b=128" ) ("search-entries"
-- "http://example.com/Atom/Http/search?b=128" ) ("categories"
-- "http://example.com/Atom/Http/categories?b=128" ) )
```

make a new entry

```
declare resp any;
declare req "blogRequest";
declare stru "MTWeblogPost";

req := new "blogRequest" ('', '', '', 'dav', 'dav');
stru := new "MTWeblogPost" ();
stru.title := 'My title'
stru.description := '<p>hello</p>';
req.struct := stru;
resp := atom.new_Post ('http://example.com/Atom/Http/entry?b=128', req);
```

The response, resp, would contain: `http://example.com/Atom/Http/entry?b=128&id=1`

edit an entry:

```
declare resp any;
declare req "blogRequest";
declare stru "MTWeblogPost";

req := new "blogRequest" ('', '', '', 'dav', 'dav');
stru := new "MTWeblogPost" ();
stru.title := 'test'
stru.description := '<p>hello from me</p>';
req.struct := stru;
atom.edit_Post ('http://example.com/Atom/Http/entry?b=128&id=1', req);
```

delete it:

```
atom.delete_Post ('http://example.com/Atom/Http/entry?b=128&id=1', req);
```

See Also:
RFC 5023: The Atom Publishing Protocol

14.7.9. XML-RPC Endpoint Configuration

The Virtuoso blog server can be contacted directly by SOAP XML-RPC. A virtual directory can be created with blogger APIs (Blogger, MetaWeblog, MoveableType, Atom) ability, which requires a mapping to the SOAP endpoint and grants to use the blogger API. This can be done in two ways:

1. **Using the graphical Server Administration Interface.** You can use your web browser to configure a virtual directory based on the "XML-RPC link" template and with bloggerAPI enabled:

    a. Open a web browser on the Administration User Interface and navigate to: Internet Domains; HTTP Virtual Directories; Edit URL mappings.
    b. Create a new virtual directory.
    c. Select the template for "XML-RPC link".
    d. Check the option 'bloggerAPI' is enabled, this will expose all available blogger APIs to the endpoint (Blogger, MetaWeblog, MoveableType, Atom).
    e. Configure the logical path.
    f. Click Add to save the definition.

    See Also:

    Virtual Directory Administration UI

    vhost_define()

    vhost_remove()

    The new logical path now will support XML-RPC and will support blogger API requests.
2. **Programmatical: Script & ISQL.** This method gives you full control, you must perform all steps to allow full support for the blogger API on a virtual directory. Below is a template list of commands that could be in a script.

    Using a script, `blogserver.sql` , with following contents:

```
vhost_define (
  lpath=>'/RPC2',
  ppath=>'/SOAP/',
  soap_user=>'$U{usr}',
  soap_opts=>vector ('XML-RPC', 'yes')
  );

grant execute on "blogger.newPost" to $U{usr};
grant execute on "blogger.editPost" to $U{usr};
grant execute on "blogger.deletePost" to $U{usr};
grant execute on "blogger.getPost" to $U{usr};
grant execute on "blogger.getRecentPosts" to $U{usr};
grant execute on "blogger.getUsersBlogs" to $U{usr};
grant execute on "blogger.getTemplate" to $U{usr};
grant execute on "blogger.setTemplate" to $U{usr};
grant execute on "blogger.getUserInfo" to $U{usr};
grant execute on "metaWeblog.newPost" to $U{usr};
grant execute on "metaWeblog.editPost" to $U{usr};
grant execute on "metaWeblog.getPost" to $U{usr};
grant execute on "metaWeblog.getRecentPosts" to $U{usr};
grant execute on "mt.getRecentPostTitles" to $U{usr};
grant execute on "mt.getCategoryList" to $U{usr};
grant execute on "mt.setPostCategories" to $U{usr};
grant execute on "mt.getPostCategories" to $U{usr};
grant execute on "mt.getTrackbackPings" to $U{usr};
grant execute on "mt.publishPost" to $U{usr};
grant execute on "mt.supportedMethods" to $U{usr};
```

    One can start ISQL using the following parameters:

```
$ isql 1111 dba dba -u usr="<SQL user for execution>" blog_server.sql
```

    Where "<SQL user for execution>" is an existing SQL user account other than dba.

Note:

Note:

If you specify the "dba" as user for SOAP execution in a virtual directory, the grant statements will not be needed and this will open a security hole. So this approach is not recommended. It is always better to have a separate user for SOAP execution with limited rights.

14.7.10. Blog Hooks - Customizing the Blog Server

Virtuoso provides the default system for blogging using one of the supported . Virtuoso provides several PL hooks to customize the blog server. The following hooks are available for definition:

authenticate_<appkey> (in req blogRequest)

newPost_<appkey> (in req blogRequest)

editPost_<appkey> (in req blogRequest)

deletePost_<appkey> (in req blogRequest)

getPost_<appkey> (in req blogRequest)

getRecentPosts_<appkey> (in req blogRequest, in numberOfPosts int)

See Also:

XML-RPC section

blogRequest is a UDT defined as follows:

```
create type "blogPost" as (
                          "content" varchar,            -- message
                          "dateCreated" datetime,       -- timestamp
                          "postid" varchar,             -- message ID
                          "userid" int                  -- creator
                        ) temporary self as ref
;

create type "blogRequest" under "blogPost"
                        as
                        (
                          user_name varchar,            -- user name
                          passwd varchar,               -- credentials
                          appkey varchar,               -- application key
                          blogid varchar,               -- web log ID
                          postId varchar,               -- message ID
                          auth_userid integer,          -- user ID
                          publish smallint,             -- not used, but still in API
                          struct DB.DBA.MWeblogPost     -- used in WebMetaLog
                        ) temporary self as ref

constructor method blogRequest (
                          appkey varchar,
                          blogid varchar,
                          postId varchar,
                          user_name varchar,
                          passwd varchar
                          )
;
```

Example 14.61. Customizing the BLOG Server

```
-- SIMPLE BLOG APPLICATION
-- application key: 0123456789
--
-- the following UDTs are used in API calls
--
-- single message
-- create type "blogPost" as (
--                         "content" varchar,
--                         "dateCreated" datetime,
--                         "postid" varchar,
--                         "userid" int
--                       ) temporary self as ref
--
```

```
--
-- a blog request
-- create type "blogRequest" under "blogPost"
--                       as
--                       (
--                         user_name varchar,
--                         passwd varchar,
--                         appkey varchar,
--                         blogid varchar,
--                         postId varchar,
--                         auth_userid integer,
--                         publish smallint
--                       ) temporary self as ref
--
-- as metaWeblog API not not have appkey, we are assuming 'META_WEBLOG' for it.

drop table BLOG;

create table BLOG (
                        B_APPKEY varchar,
                        BLOG_ID varchar,
                        B_CONTENT long varchar,
                        B_POST_ID varchar,
                        B_TS timestamp,
                        B_USER_ID integer,
                        primary key (B_APPKEY, BLOG_ID, B_POST_ID)
);

-- APPLICATION LEVEL HOOKS
create procedure authenticate_app (inout req blogRequest)
{
    declare pwd varchar;
    declare id int;
    dbg_obj_print ('auth');
    whenever not found goto nf;
    select U_PWD, U_ID into pwd, id from WS.WS.SYS_DAV_USER where U_NAME = req.user_name;
    if (isstring (pwd))
      {
        if ((pwd[0] = 0 and pwd_magic_calc (req.user_name, req.passwd) = pwd)
                            or (pwd[0] <> 0 and pwd = req.passwd))
          {
            req.auth_userid := id;
            return;
          }
      }
  nf:
    signal ('42000', 'Access denied');
}
;

create procedure
authenticate_0123456789 (in req blogRequest)
{
  authenticate_app (req);
}
;

create procedure
newPost_0123456789 (in req blogRequest)
{
   req.postId := cast (sequence_next ('blogger.postid') as varchar);
   insert into BLOG (B_APPKEY, BLOG_ID, B_CONTENT, B_POST_ID, B_USER_ID)
       values (req.appkey, req.blogid, req.content, req.postId, req.auth_userid);
}
;

create procedure
editPost_0123456789 (in req blogRequest)
{
   update BLOG set B_CONTENT = req.content where B_APPKEY = req.appkey and B_POST_ID = req.postId;
}
;
```

```
create procedure
deletePost_0123456789  (in req blogRequest)
{
  delete from BLOG where B_APPKEY = req.appkey and B_POST_ID = req.postId;
}
;

create procedure
getPost_0123456789  (in req blogRequest)
{
      declare content, datecreated, userid any;
      declare post blogPost;

      whenever not found goto nf;
      select sprintf ('%V', blob_to_string (B_CONTENT)), B_TS, B_USER_ID into content, datecreated, useri
            from BLOG where B_APPKEY = req.appkey and B_POST_ID = req.postId;

      post := new blogPost ();
      post."content" := content;
      post."dateCreated" := datecreated;
      post."postid" := req.postId;
      post."userid" := userid;
      dbg_obj_print ('getPost', post);
      return post;
    nf:
      signal ('22023', 'Cannot find a post with Id = ' || req.postId);
}
;

create procedure
getRecentPosts_0123456789  (in req blogRequest, in numberOfPosts int)
{
  declare ret, elm any;
  declare post blogPost;

  ret := vector ();
  for select sprintf ('%V', blob_to_string (B_CONTENT)) as B_CONTENT, B_TS, B_USER_ID, B_POST_ID
        from BLOG where B_APPKEY = req.appkey and BLOG_ID = req.blogId order by B_TS desc do
   {
     post := new blogPost ();
     post."content" := B_CONTENT;
     post."dateCreated" := B_TS;
     post."postid" := B_POST_ID;
     post."userid" := B_USER_ID;
     ret := vector_concat (ret, vector (post));
     numberOfPosts := numberOfPosts - 1;
     if (numberOfPosts <= 0)
       goto endg;
   }
endg:
  return ret;
}
;
```

14.7.11. Blogger Client API

`varchar` **`blogger.new_Post`** (in *uri* varchar, in *req* "blogRequest", in *content* varchar) ;

`boolean` **`blogger.delete_Post`** ((in *uri* varchar, in *req* "blogRequest")) ;

`boolean` **`blogger.edit_Post`** ((in *uri* varchar, in *req* "blogRequest", in *content* varchar)) ;

`blogPost` **`blogger.get_Post`** ((in *uri* varchar, in *req* "blogRequest")) ;

vector **blogger.get_Recent_Posts** ((in *uri* varchar, in *req* "blogRequest", in *lim* int)) ;

Example 14.62. The Blogging Client

Create a new message.

```
SQL> select blogger.new_Post ('http://example.com/RPC2',
    blogRequest ('0123456789', 'home', '', 'dav', 'dav'),'test');
callret
VARCHAR
_______________________________________________________________________________

6

1 Rows. -- 267 msec.
```

Edit a message created with previous step:

```
SQL> select blogger.edit_Post ('http://example.com/RPC2',
    blogRequest ('0123456789', 'home', '6', 'dav', 'dav'), 'test edited');
callret
VARCHAR
_______________________________________________________________________________

1

1 Rows. -- 194 msec.
```

Get the message, result will be in a blogPost UDT:

```
SQL>  dbg_obj_print (blogger.get_Post ('http://example.com/RPC2',
      blogRequest ('0123456789', 'home', '6', 'dav', 'dav')));

Done. -- 120 msec.

---- server console ----
{
        REF:[ref:0xa0deb00 obj:0xa1ed168 DB.DBA.blogPost]
        content=N"test edited"
        dateCreated={ts 2003-04-08 15:34:13.000000}
        postid=N"6"
        userid=2
}
------------------------
```

get list of messages, in our case the result will be a vector with one element of blogPost UDT.

```
SQL> dbg_obj_print (blogger.get_Recent_Posts ('http://example.com/RPC2', blogRequest ('0123456789', 'home

Done. -- 124 msec.

---- server console ----
({
        REF:[ref:0xa2426a8 obj:0xa20af40 DB.DBA.blogPost]
        content=N"test edited"
        dateCreated={ts 2003-04-08 15:34:13.000000}
        postid=N"6"
        userid=2
}
 )
------------------------

<programlisting><![CDATA[
SQL> select blogger.delete_Post ('http://example.com/RPC2',
    blogRequest ('0123456789', 'home', '6', 'dav', 'dav'));
callret
VARCHAR
_______________________________________________________________________________
```

```
1

1 Rows. -- 337 msec.
```

There are more examples on using the API in the tutorials.

14.7.12. xmlStorageSystem API

xmlStorageSystem is an Web storage system for documents that are programmable over XML-RPC and SOAP 1.1. Uploaded files are accessible via HTTP. xmlStorageSystem has methods that allow users to register with the service; upload a set of files; query the server to find out its capabilities; and to register a notification request.

See Also:
XML-RPC section

The API methods are:

```
xmlStorageSystem.registerUser()
xmlStorageSystem.mailPasswordToUser()
xmlStorageSystem.getServerCapabilities()
xmlStorageSystem.deleteMultipleFiles()
xmlStorageSystem.saveMultipleFiles()
xmlStorageSystem.requestNotification()
```

Relation to the WebDAV Repository

The xmlStorageSystem stores uploaded files in the WebDAV repository. Upon user registration a "blog" sub-folder will be created in the user-account home directory, into which these files will be uploaded and stored using the xmlStorageSystem API.

Authentication. The XML Storage System uses WebDAV enabled accounts to authenticate. The passwords sent via the xmlStorageSystem API (except registerUser, see below) must be an MD5 hash of the real user password. In this way clear text password cannot be captured from network sniffers.

Cloud notification services. The aggregators scan for changes in an interval, but in some situations we want to know when a channel has changed immediately. Notification makes it possible to always be in sync with bandwidth conservation and unneeded loops. This is possible to register an aggregator using a xmlStorageSystem.requestNotification function, also to make aggregator to know about such service it's parameters are exposed in <cloud> sub-element of <channel> of the RSS files. The notification works as follows: aggregator gets registered via xmlStorageSystem.requestNotification; Then if a registered link changes, the aggregator will be notified via specified protocol and method. The aggregator needs to register itself every 24 hour, otherwise the registration will expire. For full explanation of the cloud element of RSS file see reference at "http://backend.userland.com/rss#ltcloudgtSubelementOfLtchannelgt".

Enabling XML Storage System in a Virtual Directory

The XML Storage System can be enabled by creating a virtual directory as a SOAP enabled directory, where the SOAP users is set to DBA or any other user granted execute permissions to the xmlStorageSystem methods.

Example 14.63. Enabling an xmlStorageSystem Virtual Directory

via ISQL tool:

```
SQL> vhost_define (lpath=>'/xmlStorageSystem', ppath=>'/SOAP/',
    soap_user=>'dba');
```

The xmlStorageSystem takes into account the following user options:

maxFileSize, integer - maximum file length allowed for upload

maxBytesPerUser, integer - maximum total bytes

These are set upon initial registration to 1Mb/40Mb respectively and can be changed via the Visual Server Administration Interface -> WebDAV -> WebDAV users -> Options link.

14.7.13. User's Blog quota

In reality blog users are going to consume data space via weblog posts. For maximum flexibility to blog host administrators the following quotas are available:

"Post Quotas": A maximum number of posts "Cumulative Post Size": A threshold that is based on a cumulative size of posts. These are settable to a user which may do a blog posts via WebDAV - User accounts UI.

14.7.14. Posting a message in to the Blog

To post a message to the blog usually is used various tools implementing a blogger APIs as clients. These are standalone tools running on a PC or other platform, but not in all cases one have these tolls on hand. Therefore a web interface is needed. The default home page (index.vspx) implements authentication schema and allows blog owners to post to their blogs. To login to that page blogger should go to the site home page and when supply valid credentials will be authenticated and thus possible to make posts/edit existing etc. For authenticated user in that case is also possible to change details, settings channel subscriptions.

14.7.15. Multi-author blogging

In many cases many people needs authoring in common blog. For that case web admin have to create a blog which is owned by "group" (not by web user) and then granting this role to users which are authors will give them access to post on that blog. As group cannot be authenticated, thus it's not possible to login as group and post on behalf of it; the post's author have to be a valid web user account. This feature does not contradict with possibility authors to have their own blogs, they can have own and post to many "group" blogs. Granting access to a "group" blog can be done via WebDAV users UI, or with GRANT statement via ISQL tool.

14.7.16. Posting a comments

The blog is a something that one person or group of persons (authors) exposing to the public. So in some cases this material may need commentary from other people than authors. This is a option to the blog settings, when it's enabled (default) the home page of the blog (via permaLink) will offer a post form to make a comment. When making a comment the author's details can be recorded and re-used based on cookie; comment poster may control that option. Once comment is posted it can be read in same place (permaLink) as in discussion group.

14.7.17. Blog Post Upstreaming (bridging)

The blog upstreaming allows bloggers to keep in sync their blogs on different servers. It is possible for blogger to define a routing target, where his/her posts will be sent after local post is made, updated or deleted. This feature using blogger APIs (Blogger, MetaWeblog, MovableType) to pass messages to the remote. Therefore the target server must support one of these APIs.

The upstreaming works in following manner:

The blogger defines a routing job, which is of type "Upstream". To do this can be used Weblog Bridge UI following the steps: enter the XML-RPC endpoint, username and password valid there; hit "Fetch" to retrieve blogs on that host where the account can post; select desired account ; specify a time interval for update and which categories of posts to exclude; press "add" button.

This is is a equivalent of executing a simple insert statement :

```
SQL> INSERT INTO DB.DBA.SYS_ROUTING
        (
        R_JOB_ID,
        R_TYPE_ID,
        R_PROTOCOL_ID,
        R_DESTINATION,
        R_DESTINATION_ID,
        R_AUTH_USER,
        R_AUTH_PWD,
        R_ITEM_ID,
        R_FREQUENCY,
```

```
        R_EXCEPTION_ID)
        VALUES
        (
        1,      -- unique for job
        1,      -- Upstreaming job
        1,      -- Blogger API (2 for metaWeblog, 3 from mt)
        'http://example.com/RPC2', -- target endpoint
        'home', -- target BlogID
        'visitor', -- user
        'secret',  -- secret
        '128',     -- local Blog ID (which to replicate)
        60,        -- hourly update
        '0;1;'     -- exclude posts within categories with ID 0 and 1
        );
```

This will make so when post is added/edited/deleted to make entry(es) in SYS_BLOGS_ROUTING_LOG table. Then on specified R_FREQUENCY a scheduled job will make blogger XML-RPC requests to the target server using API as specified in R_PROTOCOL_ID field. If target server is down the entries will be kept for next job round.

The upstreaming feature can be extended easily to make another form of routing. The e-mail notification services in blogging is just one of them.

The following shows how this feature can be triggered with simple insert command:

```
INSERT INTO DB.DBA.SYS_ROUTING
        (R_JOB_ID,R_TYPE_ID,R_PROTOCOL_ID,R_DESTINATION,R_ITEM_ID,R_FREQUENCY,R_EXCEPTION_ID)
        VALUES(
        2,  -- unique for job
        2,  -- Email notification
        4,  -- STMP protocol
        'mail.domain.com:25', -- mail server
        '128', -- local Blog ID
        10,     -- scan every 10 minutes
        '0;1;'  -- exclude posts within categories with ID 0 and 1
        );
```

The same can be done via Weblog UI - Notification setup

14.7.18. Weblogs API

This API consists of "weblogUpdates.ping" XML-RPC call and server-side implementation.

The blog owner may enable weblogs pings as option in the blog settings. Then when it's blog gets updated a ping request will be sent on periodic basis to the "http://rpc.weblogs.com/weblogUpdates" server.

The server also implements weblogUpdates.ping method. If this method is exposed on a XML-RPC enabled endpoint, other parties may use it for notification.

The server implementation consist of : weblogUpdates.ping (in weblogname varchar, in weblogurl varchar, out Result struct). The party sends a ping with blog name and url and receive a common for blogger API response as struct (flError boolean, message varchar). Also a public accessible http://host:port/blog/weblogs.xml file is available for aggregators to track the latest additions made via Weblogs ping call. Please note that this file generates during on weblog site setup. hence we will have this file only if community blog pages are generated.

The weblogs.xml file follows the XMLSchema below:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="weblog">
    <xs:complexType>
      <xs:attribute name="name" type="xs:string" use="required"/>
      <xs:attribute name="url" type="xs:string" use="required"/>
      <xs:attribute name="when" type="xs:int" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="weblogUpdates">
    <xs:complexType>
```

```
      <xs:sequence>
        <xs:element ref="weblog" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="version" type="xs:string" use="required" fixed="1"/>
      <xs:attribute name="updated" type="xs:string" use="required"/>
      <xs:attribute name="count" type="xs:int" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

14.7.19. Subscriptions

Every blogger read and link to it's blog a many other sources, which are usually RSS feeds or other Web content. As RSS feeds (doesn't matter in which format RSS/RDF) are common practice to offer XML data from blogs, this is preferable source for most of the bloggers. The utilities for collecting such data are known as aggregators.

The blogging support implements a own channel/blog aggregator, so bloggers may import RSS as specifying a URL to it, or using a directory formats as OCS and OPML. Secondly the channel feeds requested by blogger will be refreshed on a periodic basis (depending of channel parameters) and will feed the data locally. Once the channel is refreshed a blogger may do "BlogThis!" when browsing the local copy of the channel. This function is available via Admin UI - Weblog UI, or via "Channels" on blog home page when owner is authenticated.

14.7.20. Trackback API

The TrackBack provides a method of notification between websites: it is a method of person A saying to person B, "This is something you may be interested in." To do that, person A sends a TrackBack ping to person B. TrackBack ping call: this is a small message sent from one webserver to another via HTTP POST or GET. There are few situations where track back is useful: *As a form of remote comments rather than posting the comment directly on person B's weblog, person A posts it on his own weblog, then sends a TrackBack ping to notify person B. *Person A has written a post that group of people are interested in; then it sends a trackback to a community server and there visitors can show it.

The implementation consist of : * the "trackback" method which can be invoked via REST(HTTP POST) * auto-discovery of the trackback URL in default blog home page

trackback (varchar id, varchar url, varchar title, varchar excerpt, varchar blog_name, varchar __mode) the method can be invoked via REST and returns a XML response.

A typical trackback post request look-like as :

```
POST /mt-tb/Http/trackback?id=1
Content-Type: application/x-www-form-urlencoded
title=Some+Title&url=http://www.some.domain/&excerpt=An+excerpt&blog_name=foo
```

A successful trackback response :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<response>
  <error>0</error>
</response>
```

When "__mode" parameter is specified via GET request, then trackback method returns a RSS v0.91 XML data for given post enclosed in response element.

A sample request and response follows:

```
http://host:port/mt-tb/Http/trackback?id=3&__mode=rss

<response>
<error>0</error>
<rss version="0.91">
<channel>
  <title/>
  <link> http://....  </link>
  <description> Description text ....  </description>
</channel>
```

```
</rss>
</response>
```

14.7.21. Pingback API

The Pingback is another form of trackback, just protocol and parameters are different. In Pingback is used XML-RPC instead of REST model. Also the requester sends his own and target url as part of request.

The implementation consists of "pingback.ping" XML-RPC method and auto-discovery in blog home page.

"pingback.ping" (in sourceURI varchar, in targetURI varchar) return varchar

The sourceURI is URL for page of requester; targetURI is page on the target server. The pingback.ping method will retrieve sourceURI and will record the data as trackback it targetURI exists on same server and it's valid permaLink. On success it returns a string 'Success' otherwise XML-RPC error code.

See Also:
Pingback 1.0

14.7.22. E-mail Notifications

The blog implementation offers a two kind of notifications: notify via e-mail a blog owner when a comment is posted; or one could subscribe to receive blog posts via e-mail. Both notifications are disabled by default and can be turned on using Weblog UI. When this flag is on a scheduled job will send e-mail to the blogger for every new comment posted to the given blog.

The blog owner notification can be turned on as setting BI_COMMENTS_NOTIFY to 1 in table SYS_BLOG_INFO for corresponding blog entry. Other way is to use Weblog settings UI.

If blog owner decide, can define a notification route. To do that he can add a routing job of type SMTP to the routing table. See "Blog Upstreaming" discussed earlier in this chapter how to do that. Also it's possible via Weblog settings UI to add such route via "Notification" link. Note that when define such route , better practice is to designate a category which to be considered as for subscription.

Once such routing job is defined for given blog, users can subscribe and therefore receive e-mails for posts which blogger assign to a category for notification.

14.7.23. Comments tracking options

When a blogger enable E-mail Notification for comments on his/her blog, the following options are available:

HTML - the notification will be sent in HTML format, the message will contain URLs to the post and comment API. Reply to this mail will add new comment, if mail-gateway is enabled by Web administrator.

HTML Form - this option will send an email containing comment and form, which could be used to invoke directly comment API from mail agent (if it supports posting a form)

Text - this option is same as first, but uses plain text instead of HTML encoding.

combined : combines all of the above as attachments.

14.7.24. Subscription Harmonizer API

Considering situation where blogger have many subscribed channels and wish these to be in sync on different places it usually subscribe/unsubscribe in home, work public blog. To be easiest for one to keep all mirrored blog instances (see Upstreaming discussed above) to have same channel subscription can be used Subscription Harmonizer API. This API consist of four methods available via XML-RPC , so developers can use them to put in sync blogger channel subscriptions.

"subsHarmonizer.setup" (in username varchar, in "password" varchar, in array any) This is used to setup on server channels which are subscribed locally. The user authenticates using name and password and sends to server list of RSS URLs in "array" parameter. This action is usually performed once , on initial subscription.

"subsHarmonizer.subscribe" (in username varchar, in "password" varchar, in array any) This is used when in local blog instance blogger makes new subscription(s), the "array" will contains only entries have been added since last call to that or subsHarmonizer.setup call. The target server will resolve URL list and will make records in SYS_BLOG_CHANNEL and SYS_BLOG_CHANNEL_INFO tables.

"subsHarmonizer.unsubscribe" (in username varchar, in "password" varchar, in array any) This function is opposite of subsHarmonizer.setup , it is used when blogger unsubscribe from some channels. The server action is to remove channel subscriptions from blog instance. Note that in this case server keeps channel definition , so it can be re-used when another user get subscribed to it, this is to minimize bandwidth consumption for channel characteristics retrieval.

All of the above returns boolean.

"subsHarmonizer.startup" (in username varchar, in "password" varchar) This function is used from blog application to get from server list of subscriptions which are currently registered. In that case blog application uses the array of URLs returned to set it's local list of channels.

14.7.25. Mobile Blogging (Moblog)

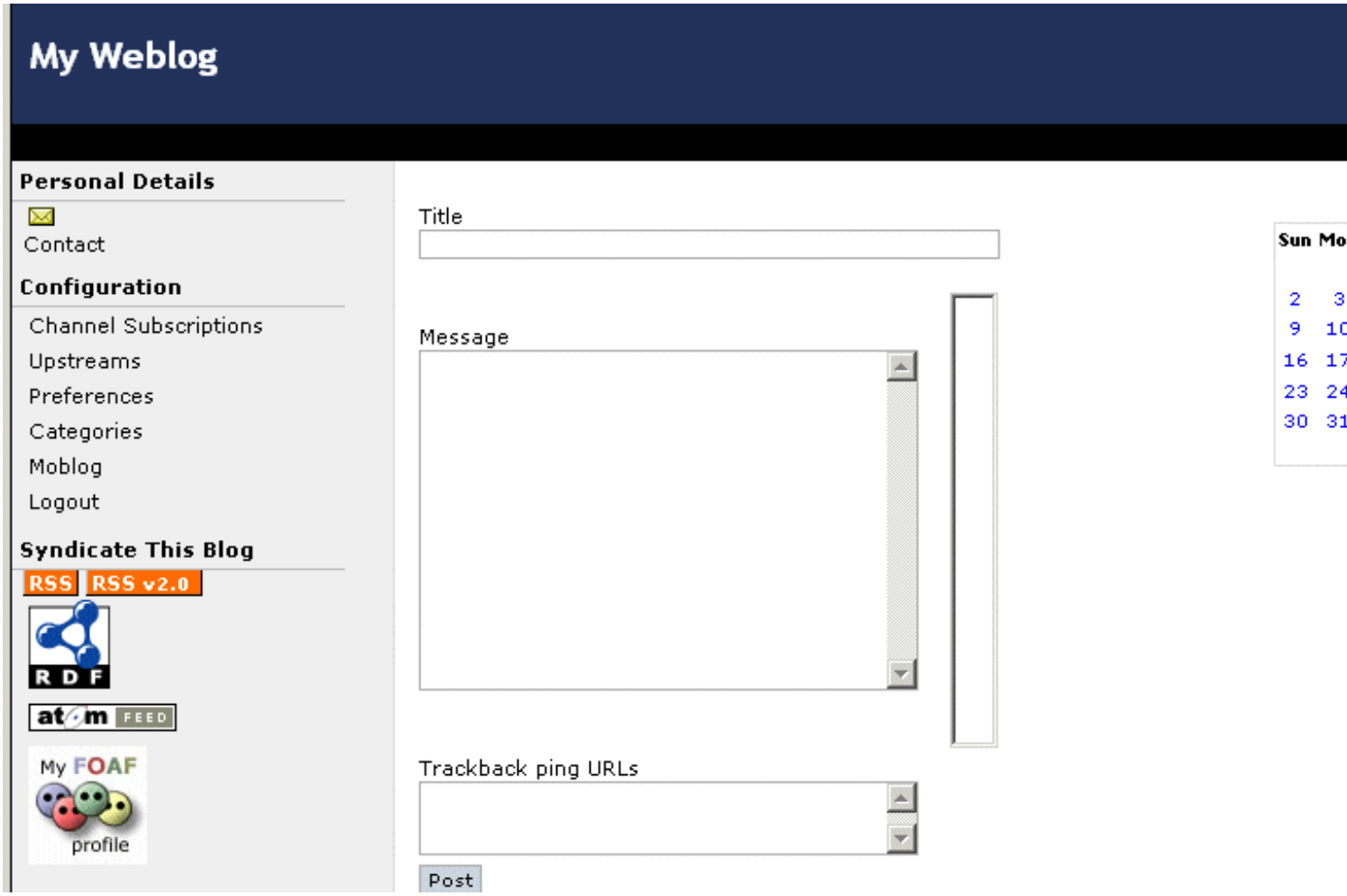
The Mobile blogging (moblog) is synonym of messages containing images made via mobile devices (such as mobile telephones etc.) with ability to record a image and send them via SMTP.

Moblog"ing has become popular as means for users to easily, automatically and immediately post text and images back to their blog site what ever their location.

The Virtuoso server can act as mail server (see Storing Email in Virtuoso chapter, and howto configure mail servers in same chapter). Thus it"s possible to capture mails sent via mobile devices and expose in a Blog. But main problem remains: SMTP have no authentication. So this functionality is achieved via UI which shows all existing mail messages sent via mobile devices and containing a images, allowing to the blog owner to decide which image to go to his/her blog. As an alternative post can be automatic if blogger check this option.

Mobile blogging (Moblog) can be configured via the Virtuoso blog UI by selecting the Moblog link under the Configuration section as indicated in the diagram below:

Figure 14.51. Blog page



The Moblog configuration page enables the details of the POP3 server from which messages are to be retrieved from is located as indicated in the diagram below:

Figure 14.52. Blog page

- *Server Address and Port*

- Hostname and port number of the POP3 Mail Server.

- *Account Username*

  - username of POP3 Mail server account.

- *Account Password*

  - username of POP3 Mail server account.

- *Enable Automatic Moblogging*

  - enabled automatic posting of messages to Blog site.

- *Allowed MIME types*

  - mime type of messages to be retrieved form POP3 server, which can be of type image, vidoe or audio.

- *Set*

  - this button sets the POP3 mail server setting entered above.

- *Fetch*

  - this button retrieves available moblog messages from the POP3 mail server.

Moblog can handle different types of attachments, this depends of list of MIME type patterns (SQL like syntax), which blogger specifies in Moblog UI. Further any new incoming message containing attachment matching any of that list will be exposed in the UI for posting, or it will be posted automatically if "Automatic post" option is checked.

Note: feeding a mail box can also be done via "Mail", "Get Mail with POP3" Admin UI utility. In this case blogger authenticates in this utility, and feeds it"s local mailbox from some POP3 server. The consequential actions to put a message as blog message are same as described above.

As alternative to feeding via own POP3 account, Moblog message can be sent to special mail-gateway account. Such account can be set from Web admin only and can be used from all bloggers to pass Moblog messages to their blogs. Only requirement in that case is blogger to include "@@blogId@@=[blog Id]" string as a single line whenever in the message body. Further mail-gateway processing will detect such rule and will expose the message in the Moblog UI, or automatic post will be performed.

14.7.26. Posting a dynamic content

The Virtuoso blogging system allows to insert into a post a result from SQL, XPATH or XQuery expressions. The way this can be done is to have a SQL access enabled for the Web user making the post, so such blogger can insert into a post body (note that this is not available for title) a XMLTemplate. (see "XML Templates" for reference) Then at the render time the template will be executed on behalf of the blogger who is made the post and result will be shown. There is no limitation how many XML Templates can be included in a single post. Please note that bloggers without SQL access cannot do such posts. Also the SQL granting rules will be applied when XML Template is executed.

The following will render a simple table containing list of Demo.demo.Shippers table from Demo database.

```
<table border="1">
    <sql:query xmlns:sql="urn:schemas-openlink-com:xml-sql" >
        select 1 as tag , null as parent, CompanyName as [tr!1!td!element]
        from Demo.demo.Shippers
        for xml explicit
    </sql:query>
</table>
```

The following post will render a table containing a names of people which is organized in simple OPML file (located in the tutorial system) using XQuery.

```
<div>
    <sql:xquery  xmlns:sql="urn:schemas-openlink-com:xml-sql"
        sql:context="http://example.com/tutorial/apps/blog_query/">
        <![CDATA[
        <table border="1">
            { for $o in document ("opml.xml")//outline
               return <tr><td>{ string ($o/@text) }</td></tr>
            }
```

```
            </table>
            ]]>
          </sql:xquery>
        </div>
```

14.7.27. Notification Services

This UI is used to define a mail routing job. So it is used for notification of blog visitors for post updates. If administrator define a mail target and refresh interval; visitors to the given blog can subscribe to the 'mailing list' for blog updates. Once visitor have subscribed, on every new post an email will be sent to he/she. If the mail is deleted, the mailing list for blog update notification will be canceled.

14.7.28. Rendering the RSS feed in WML format

The user's blog RSS file could be rendered in WML format for accessing with mobile devices. To do that built-in PL post-processing hook for blog virtual directory is introduced (DB.DBA.BLOG_RSS2WML_PP). It working together with a XSL-T style-sheet which converts the RSS in WML format.

To enable this functionality the blogger should enter '*wml-default*' as RSSFilter option on settings page. Or if he/she has an own XSL-T filter for WML can enter its URL instead.

The DB.DBA.BLOG_RSS2WML_PP post-processing hook:

```
create procedure DB.DBA.BLOG_RSS2WML_PP ()
  {
    declare accept, upar, pars any;
    declare lines any;
    lines := http_request_header ();
    accept := http_request_header (lines, 'Accept');
    if (not isstring (accept))
      accept := '';
    upar := http_request_get ('QUERY_STRING');
    if (regexp_match ('text/vnd\.wap\.wml', accept))
      {
        if (http_path () like '%/rss.xml')
          {
            declare opts, filt, bid any;
            whenever not found goto exitp;
            select top 1 BI_BLOG_ID into bid from SYS_BLOG_INFO where
            http_path () like BI_HOME || '%' order by length (BI_HOME) desc;
            select deserialize (blob_to_string (BI_OPTIONS)) into opts from SYS_BLOG_INFO where BI_BLOG_I

            if (not isarray(opts))
              opts := vector ();

            filt := get_keyword ('RSSFilter', opts, '');

            if (filt = '*wml-default*')
              filt := 'http://local.virt/rss2wml.xsl';

            if (not isstring (filt) or not xslt_is_sheet (filt))
              goto exitp;

            if (length (upar) = 0)
              {
                http_xslt (filt);
              }
            else
              {
                declare rss, xt, xsl any;
                rss := http_get_string_output ();
                xt := xml_tree_doc (rss);
                http_rewrite ();
                xsl := xslt (filt, xt, vector ('id', upar));
                http_value (xsl, null);
              }
            http_header ('Content-Type: text/vnd.wap.wml\r\n');
            exitp:;
          }
```

```
        }
      return;
    }
;
```

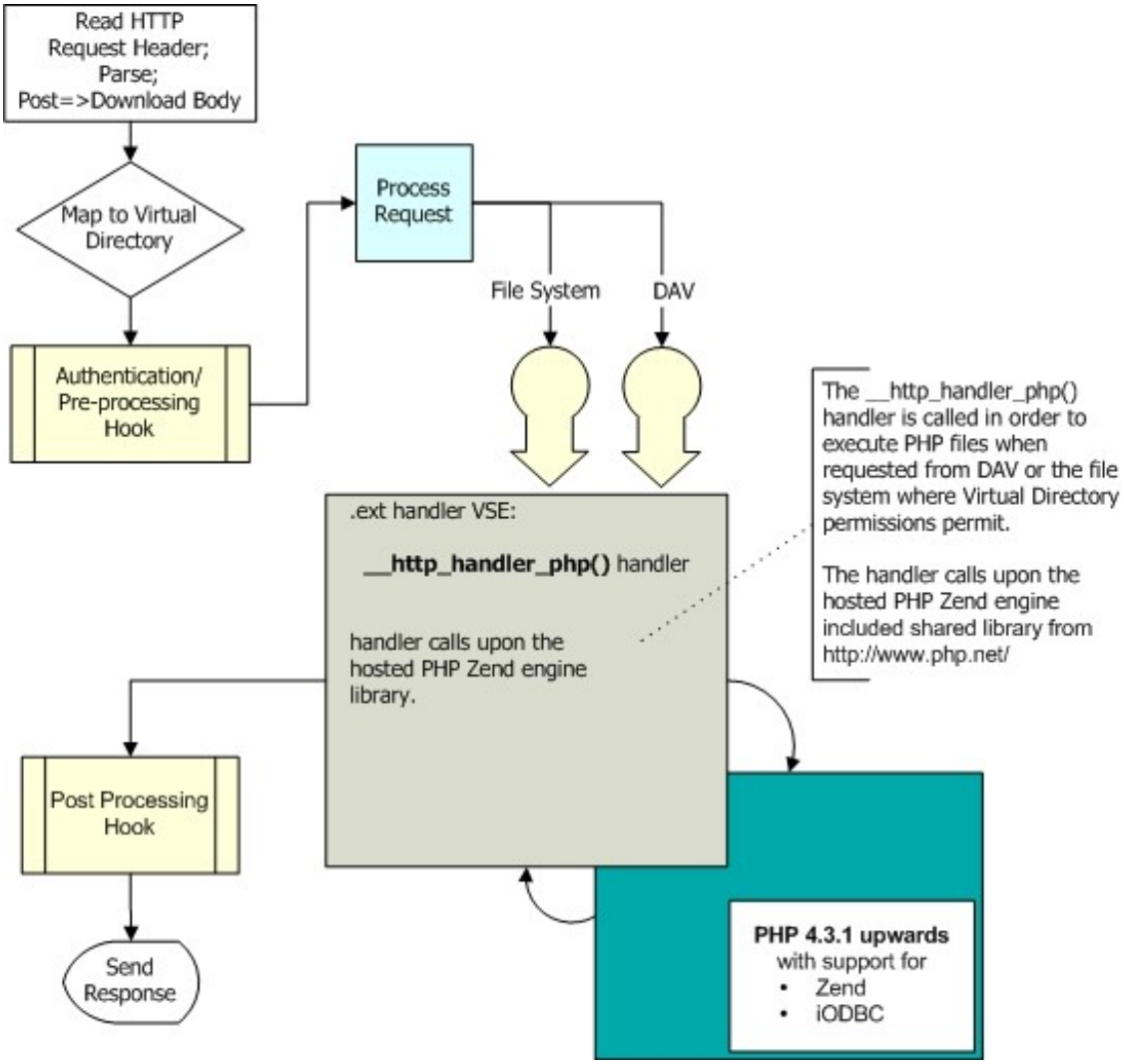
The rss2wml.xsl style sheet:

```
<?xml version='1.0'?>

<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0"
  xmlns:vb="http://example.com/weblog/">
  <xsl:output method="xml" indent="yes" doctype-public="-//WAPFORUM//DTD WML 1.1//EN" doctype-system="htt

  <xsl:param name="id" />

  <xsl:template match="/">
    <wml>
      <card
xml:id="card1">
        <xsl:apply-templates />
      </card>
    </wml>
  </xsl:template>

  <xsl:template match="item">
    <xsl:choose>
      <xsl:when test="boolean($id!='')">
        <xsl:if test="substring-after(link,'?') = $id">
          <p>
            <xsl:value-of select="vb:tidy_xhtml (string (description), '*default*')"
              disable-output-escaping="yes"/>
            <br/>
            <xsl:if test="preceding-sibling::item">
              <a>
                <xsl:attribute name="href">rss.xml?<xsl:value-of select="substring-after(preceding-siblin
            </xsl:if>
            <xsl:if test="following-sibling::item">
              <a>
                <xsl:attribute name="href">rss.xml?<xsl:value-of select="substring-after(following-siblin
            </xsl:if>
          </p>
        </xsl:if>
      </xsl:when>
      <xsl:otherwise>
        <p>
          <a>
            <xsl:attribute name="href">rss.xml?<xsl:value-of select="substring-after(link,'?')"/></xsl:at
            <xsl:value-of select="title"/>
          </a>
        </p>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

  <xsl:template match="text()"/>
</xsl:stylesheet>
```

14.8. Deploying PHP Applications

The PHP server extension allows Virtuoso to execute PHP (v4) pages stored in the file system or in Virtuoso's WebDAV repository. PHP pages run inside the Virtuoso process.

Figure 14.53. The HTTP PHP handler



The VSE __http_handler_php() has been implemented so that the file extension '.php' is recognized by Virtuoso to switch between 'normal' mode and extension processing for PHP mode in the HTTP/WebDAV services.

PHP (recursive acronym for "PHP: Hypertext Preprocessor") is a widely-used Open Source general-purpose scripting language that is suited for Web-based development. Here is an example of a simple PHP page:

Example 14.64. PHP introductory example

```
<html>
    <head>
        <title>Example</title>
    </head>
    <body>

        <?php
        echo "Hi, I'm a PHP script!";
        ?>

    </body>
</html>
```

Notice how this is similar to VSP script, and different from a script written in other languages like Perl or C -- instead of writing a program with lots of commands to output HTML, you write an HTML script with some embedded code to do something. The PHP code is enclosed in special start and end tags that allow you to jump into and out of "PHP mode".

See Also:

See Also:

PHP Online Documentation

In-Process Data Access Client

14.8.1. Building the Virtuoso Server With PHP Extension

1. Firstly you need to have the PHP4 library installed with Zend and ODBC interface enabled. On UNIX-es this can be enabled by doing 'configure --enable-experimental-zts --with-iodbc'. On Windows the library must be downloaded from the php.net site.
2. Make sure that the iODBC library is installed if you are preparing to build on UNIX platform.
3. Make the bif_server_php executable in the bifsdk/examples/ directory. This same directory contains the source file for the bif_server_php executable: bif_server_php.cpp.

The Virtuoso distribution may already contain a binary executable of the PHP extension of the Virtuoso's Web Service. In which case you may skip the build process and start the server with the PHP extensions instead of starting the normal virtuoso server executable.

To start and test the Virtuoso server with the PHP extension do the following:

1. Make a simple file 'info.php' and place it in the HTTP root directory. The content of the info.php file should be the single line:

   ```
   <?phpinfo();?>
   ```

   This function produces an HTML page containing various PHP processor information.
2. Start the server. For Linux platform: bif_server_php -f, for Windows platform: bif_server_php-odbc-t.exe -f
3. Test the installation by entering the http://[host:port]/info.php as the URL of a browser of your choosing.

Note:
Windows users will need to install the php4ts.dll (from the php.net site) before running the PHP enabled Virtuoso server. This DLL must be in the system path when Virtuoso starts and can typically be placed in the %SYSTEMROOT% directory without any path modifications.

Note:
The PHP library used with Virtuoso must be version 4.3.1 or greater. From this version the PHP library includes a serious PHP CGI vulnerability fix, without which the Virtuoso server will fail to start for security reasons.

14.8.2. PHP Extension Functions

The following functions have been added to the Virtuoso server in order to enable PHP processing.

PHP Server Handler.

__http_handler_php (in *file_name* varchar,
in *params* vector (opt1, value [,...]) or string session,
in *lines* vector (opt, value[,...]),
inout *what* any) ;

This function will be detected and called automatically by the Virtuoso HTTP/WebDAV server when a request for a file with extension .php is made.

- The

  file_name

  argument will be the path to the file when the PHP page is stored on the file system or the actual content of the page when a WebDAV resource is processed.
- The content of request entity body will be stored in a string session and passed down to the __http_handler_php as the

  params

argument.

- The HTTP request header will be passed as the

  lines

  argument.
- The

  what

  parameter is used for in two ways: to indicate whether the first argument is a path/file name to a file on the filesystem or the content of the required WebDAV resource. The HTTP/WebDAV server will return an array of two elements to this parameter.

PHP Processor.

__http_php_str (in *string* varchar,
in *params* vector (opt1, val1 [,...]) , varchar) ;

This function takes a string containing PHP code (page), and parameters supplied in the params array and returns the result from the PHP engine as a string. This can be useful for performing PHP transformations in PL or VSP code.

- the

  string

  is the source of the PHP page
- *params*

  is a string or array of strings containing parameter name and parameter value pairs. In the case of a single string it must contain form parameters in the application/x-www-form-urlencoded encoding. In case of an aref of strings it must contain the name and value for parameters. (Like the params argument in the VSPs).

14.8.3. PHP Examples

Example 14.65. Examples of the php_str() usage

Unless the examples are shown as executed in the ISQL tool, this can be made also in the Virtuoso/PL code.

```
SQL> select php_str ('<?php echo "Hello World"?>');
callret
VARCHAR
_______________________________________________________________________________

Hello World

SQL> select php_str ('<?php print abs (-1);?>');
callret
VARCHAR
_______________________________________________________________________________

1

SQL> set MACRO_SUBSTITUTION off;
SQL> select php_str ('<?php echo $a?>', 'a=Hello+World');
callret
VARCHAR
_______________________________________________________________________________

Hello World

SQL> select php_str ('<?php echo $a?>', vector ('a', 'Hello World'));
callret
VARCHAR
```

```
Hello World

SQL> select php_str ('<?php echo "$a $b"?>', 'a=Hello+World&b=Hello+Again+World');
callret
VARCHAR
_______________________________________________________________________________

Hello World Hello Again World

SQL> select php_str ('<?php $a=1; $b=2; $c=3; $d=$a+$b+$c; echo $d?>');
callret
VARCHAR
_______________________________________________________________________________

6

SQL>  select php_str ('<?php $a=8; $b=4; $c=8; echo $a|$b&$c?>');
callret
VARCHAR
_______________________________________________________________________________

8
```

Example 14.66. Examples of the php redirect page

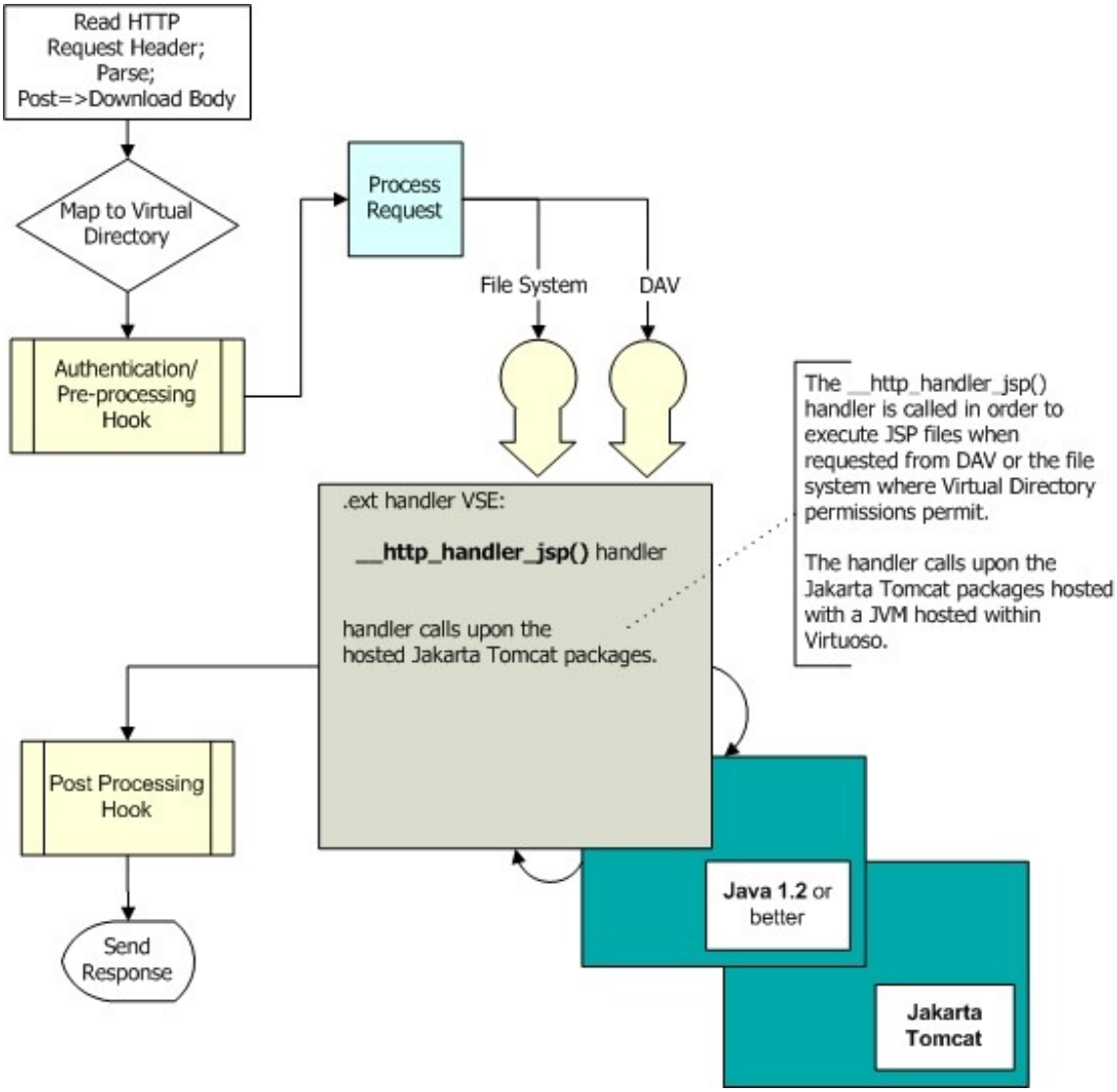
This is test how the PHP processor can instruct the HTTP server to send a custom response and header to the user agent. The following page can stored on the file system or in the WebDAV repository (in which case execution rights must be enabled). Hitting that page will redirect user-agent to the index.html page in HTTP server root.

```
<?php
  header ("HTTP/1.1 302 Found");
  header ("Location: /index.html");
?>
```

14.9. Deploying JSP Applications

The Virtuoso server with Java hosting support can be used to execute JSP pages, however, Virtuoso requires a JSP container engine first. Jakarta provide such an engine that can be hosted inside Virtuoso with Java support. This section describes how JSP's can be executed using Jakarta Tomcat JSP implementation.

Figure 14.54. The HTTP JSP handler using Jakarta Tomcat



14.9.1. Environment Setup & Verification

The following components are required:

Java SDK (or runtime only) needs to be installed. Version 1.2 or later.

Jakarta Tomcat package must be installed and configured. This can be obtained from the Jakarta project site (http://jakarta.apache.org/tomcat/)

The virtuoso.ini file must edited to including the following settings in the `[Parameters]` section:

```
.....
JavaClasspath = <jdk install dir>/lib/tools.jar:<tomcat install dir>/bin/bootstrap.jar
JavaVMOption1 = -Djava.endorsed.dirs=<tomcat install dir>/bin:<tomcat install dir>/common/endorsed
JavaVMOption2 = -Dcatalina.base=<tomcat install dir>
JavaVMOption3 = -Dcatalina.home=<tomcat install dir>
JavaVMOption4 = -Djava.io.tmpdir=<tomcat install dir>/temp
.....
```

Note:

The above has been tested for the jakarta-tomcat-4.1.18 distribution only. Setting may vary for any other versions of Jakarta Tomcat, in which case we advise you consult the Tomcat documentation.

Run the `virtuoso-*-javavm*-t` binary, The Virtuoso server that includes Java hosting.

Start the Jakarta Tomcat server inside virtuoso server process using the following command from ISQL:

```
SQL> java_call_method ('org.apache.catalina.startup.BootstrapService', null, 'main', 'V',
     vector ('[Ljava/lang/String;', vector ('start')));
```

In order to invoke the above command automatically at server startup you might put it in the autoexec.isql init script.

Make a virtual directory for accessing JSP server, here is an example:

```
vhost_define (lpath=>'/tomcat',ppath=>'http://example.com:8080/');
```

Verify that your installations are correct by pointing your web browser of choice to `http://example.com:8890/tomcat/` (assuming you are working on your local machine). Run some of JSP examples provided with Tomcat distribution to verify that they are executed without errors.

Note:
For debugging purposes you may run the virtuoso-*-javavm*-t server with foreground option to receive Java error messages, if any occur.

See Also:
In-Process Data Access Client

14.10. Perl Hosting

Virtuoso functionality can be enhanced through external libraries by loading shared objects or DLLs. The new functions are written in a language of the developer's choice and compiled to produce a shared library appropriate to the operating system. The path to the shared library must be declared in the Virtuoso INI file and the server restarted before it can be used.

The Perl language is hosted within Virtuoso in this way. hosting_perl.so is the library used.

The Virtuoso INI file uses a [Plugins] configuration section for listing shared libraries for the server to load upon startup. An example of this is:

```
[Plugins]
LoadPath = /home/virtuoso/hosting
Load1 = Hosting, hosting_perl.so
..
```

The "Hosting" type defines the entry points for initialization of the runtime hosting environment, destruction of the user environment and execution of a file or string containing commands in the hosted language. It also returns a list of file extensions that it is capable of processing. Virtuoso dynamically defines memory-resident (no disk image) HTTP server handlers for each specified type.

The Perl hosting plugin supports 'pl' extension. Hence, upon initialization of the hosting plugin, Virtuoso defines the `__http_handler_pl(..)` function according to the API for file type handlers in the Virtuoso HTTP server. With this handler in place, each hit on a .pl file (file system or WebDAV) with appropriate execute permissions will cause the HTTP server to execute the code within it and return the result instead of simply the file's contents.

The handler will call the __hosting_http_handler VSE with a special set of parameters to represent the HTTP environment correctly.

Virtuoso will normally call the plugin to memory as required, and expel it when finished. This behavior can be controlled by the INI file parameter:

```
[HTTPServer]
PersistentHostingModules = 1/0 default 0
```

Setting `PersistentHostingModules` to "1" prevents Virtuoso from removing the interpreters from the HTTP threads after each request.

See Also:
VSEI Plugins

Example 14.67. Using the Perl Plugin

Executing Perl code directly:

```
select __hosting_http_handler ('pl', 'print "hello world"; ', '', vector (), 'helloworld.pl');
```

```
returns : hello world
```

Executing a perl script file (perl/test_print.pl in the Virtuoso working directory):

```
/perl/test_print.pl
-------------------
#!/usr/bin/perl
print "hello world file";

select __hosting_http_handler ('pl', 'perl/test_print.pl');

returns : hello world file
```

Note:
The hosting_perl hosting module uses the perl `tie()` function to "tie" up the STDIN, STDOUT, STDERR, exit() and %ENV perl objects. Untying any of these may lead to unpredictable results.

14.11. Python Hosting

Virtuoso functionality can be enhanced through external libraries by loading shared objects or DLLs. The new functions are written in a language of choice and compiled to produce a shared library appropriate to the operating system. The path to the shared library must be declared in the Virtuoso INI file and the server restarted before it can be used.

The Python language is hosted within Virtuoso in this way. `hosting_python.so` (or `hosting_python.dll`) is the library used.

The Python hosting module requires the Python library, version 2.2.2 or above, to build and use the module, which then must be installed in the Plugins-Load-path defined in the Virtuoso ini file. The source code and build script (`build.py`) for building the `hosting_python.so` module are included in the Unix distribution in the `custom/hosting/python` directory. The Unix installer will offer to build it near the end of the installation process. The build.py is a kind of Makefile. On unix it requires `libtool` and `cc` in the path. On windows it requires `cl.exe` in the path.

The Virtuoso INI file uses a [Plugins] configuration section for listing shared libraries for the server to load upon startup. An example of this is:

```
[Plugins]
LoadPath = /home/virtuoso/hosting
Load1 = Hosting, hosting_python.so
..
```

The "Hosting" type defines the entry points for initialization of the runtime hosting environment, destruction of the user environment and execution of a file or string containing commands in the hosted language. It also returns a list of file extensions that it is capable of processing. Virtuoso dynamically defines memory-resident (no disk image) HTTP server handlers for each specified type.

The Python hosting plugin supports the 'py' extension. Hence, upon initialization of the hosting plugin, Virtuoso defines the `__http_handler_py(..)` function according to the API for file type handlers in the Virtuoso HTTP server. With this handler in place, each hit on a .py file (file system or WebDAV) with appropriate execute permissions will cause the HTTP server to execute the code within it and return the result instead of simple the file contents.

The python interpreter has a global lock unrelated to the Virtuoso hosting module, thus no more than one thread can run python code at a time.

The handler will call the __hosting_http_handler VSE with a special set of parameters to represent the HTTP environment correctly. Virtuoso will, by default call the plugin to memory as required, and expel it when finished. This behavior can be controlled by the INI file parameter:

```
[HTTPServer]
PersistentHostingModules = 1/0 default 0
```

Setting `PersistentHostingModules` to "1" prevents Virtuoso from removing the interpreters from the HTTP threads after each request.

See Also:
VSEI Plugins

Example 14.68. Using the Python Plugin

Executing Python code directly:

```
select __hosting_http_handler ('py', 'print "hello world"; ', '', vector (), 'helloworld.py');

returns : hello world
```

Executing a python script file (python/test_print.py in the Virtuoso working directory):

```
python/test_print.py
--------------------
#!/usr/bin/python
print "hello world file";

select __hosting_http_handler ('py', 'python/test_print.py');

returns : hello world file
```

14.12. Ruby Hosting

Virtuoso functionality can be enhanced through external libraries by loading shared objects or DLLs. The new functions are written in a language of choice and compiled to produce a shared library appropriate to the operating system. The path to the shared library must be declared in the Virtuoso INI file and the server restarted before it can be used.

The Ruby language is hosted within Virtuoso in this way. `hosting_ruby.so` (or `hosting_ruby.dll`) is the library used.

The Ruby hosting module requires the Ruby library, version 1.8.3 or above, to build and use the module, which then must be installed in the Plugins-Load-path defined in the Virtuoso ini file. The source code for building the `hosting_ruby.so` module are included in the Unix distribution in the `custom/hosting/ruby` directory. The Unix installer will offer to build it near the end of the installation process.

The Virtuoso INI file uses a [Plugins] configuration section for listing shared libraries for the server to load upon startup. An example of this is:

```
[Plugins]
LoadPath = /home/virtuoso/hosting
Load1 = Hosting, hosting_ruby.so
..
```

The "Hosting" type defines the entry points for initialization of the runtime hosting environment, destruction of the user environment and execution of a file or string containing commands in the hosted language. It also returns a list of file extensions that it is capable of processing. Virtuoso dynamically defines memory-resident (no disk image) HTTP server handlers for each specified type.

The Ruby hosting plugin supports the 'rb' extension. Hence, upon initialization of the hosting plugin, Virtuoso defines the `__http_handler_rb(..)` function according to the API for file type handlers in the Virtuoso HTTP server. With this handler in place, each hit on a .rb file (file system or WebDAV) with appropriate execute permissions will cause the HTTP server to execute the code within it and return the result instead of simple the file contents.

The ruby interpreter has a global lock unrelated to the Virtuoso hosting module, thus no more than one thread can run ruby code at a time.

The handler will call the __hosting_http_handler VSE with a special set of parameters to represent the HTTP environment correctly. Virtuoso will, by default call the plugin to memory as required, and expel it when finished. This behavior can be controlled by the INI file parameter:

```
[HTTPServer]
PersistentHostingModules = 1/0 default 0
```

Setting `PersistentHostingModules` to "1" prevents Virtuoso from removing the interpreters from the HTTP threads after each request.

See Also:
VSEI Plugins

Example 14.69. Using the Ruby Plugin

Executing Ruby code directly:

```
select __hosting_http_handler ('rb', 'print "hello world"; ', '', vector (), 'helloworld.rb');

returns : hello world
```

Executing a ruby script file (ruby/test_print.rb in the Virtuoso working directory):

```
ruby/test_print.rb
-------------------
#!/usr/bin/ruby
print "hello world file";

select __hosting_http_handler ('rb', 'ruby/test_print.rb');

returns : hello world file
```

Chapter 15. XML Support

Abstract

This chapter covers Virtuoso's XML, full text retrieval and related functions.

Virtuoso provides free text indexing capabilities for textual data, including XML data. Virtuoso supports extraction of XML documents from SQL datasets. It also supports XPath , XQuery , XSLT and XML Schema validation.

An SQL long varchar, long xml or xmltype column in a database table can contain XML data as text or in a binary serialized format, respectively. If a column value is a well-formed XML fragment there are special operations that can be performed on the value. There is an SQL data type that represents an XML entity. A string representing a well-formed XML entity can be converted into an entity object representing the root node. XPath expressions can then be applied to the entity in order to retrieve other entities or sets of entities. Returning an XML entity to a client application or printing it out on a dynamic web page will produce the text representation of the entity, complete with start and end tags, attributes, namespaces, and so forth.

An entity object can be stored as a value of a long varchar or varchar column. This will produce and store the text corresponding to the entity. Storing the same into a long xml or xmltype column will provide a more compact representation and will guarantee well-formedness and optionally schema validity, even if the data comes in as text.

A long varchar column can contain huge XML documents as special "persistent XML entity" objects. XML entities of this sort consume only a little amount of memory and load small portions of data from disk to memory on demand; an application can handle XML documents that are order of magnitude larger than the amount of available memory. LONG XML column type is convenient if a column contains only valid XML documents. An application can save XML entities to LONG XML columns and retrieve them back without calling any type conversion functions. A special user-defined type XMLType allows the use of an XML entity as a base for deriving user-defined types from it.

The `xpath_contains` SQL predicate can be used to test whether a column value matches a given XPath expression. If the XPath expression specifies a node set and that set is non-empty for a given column value, it is possible to bind a result variable to each element of the set. In this way a single row of data in a table can produce multiple rows in an SQL result set, one for each entity selected by the XPath predicate.

If there is a free text index on a column it is possible to define that the content be checked for well-formedness. In this case both `contains` and `xpath_contains` predicates can be applied to the same column in the same query. You should create a free text index on your XML data if you expect any content-based searches. The free text index will recognize specific features of XML and allow their use in searches.

The `xcontains` SQL predicate is a combination of XPath and free text, making automatic use of the free text index for evaluating an XPath expression. This predicate also allows you to test text values of entities for complex free text conditions such as proximity.

There is a user interface for using this feature on DAV resources. The use of the `xpath_contains` predicate is not limited to DAV resources though. The use of XPath with XML values is independent of free text indexing and of XML views.

Virtuoso offers functions for XPath processing of well-formed XML strings in SQL. Together with the Virtuoso free text support, these functions offer a powerful free-form and structured content retrieval system. You can search for XPath matches in any XML-populated column. One practical example would be the RES_CONTENT column of the WS.WS.SYS_DAV_RES table, which contains the contents of documents stored in the WebDAV repository. As with the sample database, which contains the XML sources for this documentation, you may store XML documents directly in the WebDAV repository.

An SQL statement can return complex XML documents based on relational data. Virtuoso supports both Microsoft's "FOR XML" syntax and the standard set of SQLX XML composing functions like `XMLELEMENT` and `XMLAGG` . Very complicated processing can be done in a single statement that combines XML composing functions, `xquery_eval` and `xslt` .

Virtuoso/PL routines can modify XML entities in DOM style (functions like `XMLAppendChildren` and `XMLReplace`). Doing local changes this way can be simpler than via XSLT or XQuery; DOM modifications also help speed-critical applications to avoid unnecessary copying of data.

Virtuoso's XML parser can read XML documents of any complexity. It can validate source documents against DTD and XML schema, load compound documents or their fragments, recover from errors in ill-formed HTML documents like popular browsers

can.

15.1. Rendering SQL Queries as XML (FOR XML Clause)

Virtuoso extends SQL-92 with the FOR XML clause that allows any SQL result set to be turned into XML according to some simple rules. The notation and functionality are similar to those offered by Microsoft SQL Server and IIS.

The FOR XML clause has 3 variants:

RAW. Make an XML entity from each row of the result set; do not attempt to construct hierarchies. Each row's data is enclosed in a <ROW/> element and each column is either an attribute or child element.

AUTO. A hierarchy is constructed with one level for each table of the join for which at least one column is selected. The table whose column is first mentioned in the selection will be the topmost element, the next table its child, etc. Each level of the tree will consist of one type of element. A parent element will have multiple children if consecutive rows do not differ in the column values coming from the parent element. When a table's column values differ from the previous row, the element and all children thereof are closed and a new element is started, with children filled out from other columns of the result set.

EXPLICIT. This mode gives more control on the resulting tree's structure while requiring a more elaborate query structure. In this mode, the query will be a UNION ALL of many joins and each row will specify exactly one element. Which type of element this is and where in the tree it will be placed are determined by the values of the first two columns, TAG and PARENT.

In all modes, columns may either be attributes or sub-elements. The *ELEMENT* keyword after the FOR XML clause forces all columns to be rendered as sub-elements; attribute are the default.

In all modes except explicit, the names of elements are the unprefixed table names and the names of attributes are the columns' names in the result set. If tables have correlation names the correlation names are used in the output instead of the table names. Expressions are allowed in the selections but these should be named using AS. In AUTO mode Virtuoso assumes expressions belong to the topmost element.

The FOR XML clause is generally allowed in SELECT statements in place of the FOR UPDATE clause. However it only has an effect when the statement is executed through the `xml_auto()` function.

See Also:

The SQL-XML Statements page described in the Visual Server Administration Interface section provides a fast graphical way of supplying an SQL statement to Virtuoso and saving the view as a resource accessible from the WebDAV store.

15.1.1. FOR XML EXPLICIT Mode

This mode gives the developer the most control over the generated result tree but requires a verbose query formulation. Each row must begin with two integer columns, the first identifying the element represented by the row and the second the parent element type of this element. Consider:

```
select 1 as tag, null as parent,
       "CategoryID" as [category!1!cid],
       "CategoryName" as [category!1!name],
       NULL as [product!2!pid],
       NULL as [product!2!name!element]
from "Demo".."Categories"
union all
select 2, 1, "category" ."CategoryID", NULL, "ProductID", "ProductName"
    from "Demo".."Categories" "category", "Demo".."Products" as "product"
    where "product"."CategoryID" = "category"."CategoryID"
order by [category!1!cid], 5
for xml explicit;
```

This query makes a two level tree where Categories have Product children. The selection in the first UNION term specifies the element types in the result set. The two first columns, TAG and PARENT are required in all EXPLICIT queries. Subsequent columns have an extended AS declaration that specifies which element they belong to, what that element is called in XML and what the column will be called. A row where TAG has a value of 1 will pick the columns which has [xxx!1!yyy] as their alias; rows with a TAG of 2 will pick columns with an alias with [xxx!2!yyy] and so on.

If consecutive rows have a different TAG but the same PARENT, these will be siblings of different types. This possibility does not exist with the other FOR XML modes.

If the PARENT is 0 or NULL, then any previously open elements in the result are closed and the element of the row becomes a top-level element. When PARENT refers to the TAG of a presently open element in the set, all children of that element are closed and the row's element is inserted as the next child of the last element with the TAG equal to the new row's PARENT. All open tags are closed at the end of the result set.

Note

Since each level of the tree is generated by a different term in the UNION ALL, an ORDER BY will invariably be needed to group the children after their parents. If the parent rows have NULLs in place of the child row's key values, the parent gets sorted first because NULL collates first.

15.1.2. Examples of FOR XML

This section gives one example of each mode of FOR XML combined with the `xml_auto()` function to help us display the results simply. First we create a procedure that enables us to supply SQL and return XML using the `xml_auto()` function.

```
create procedure xmla (in q varchar)
{
  declare st any;
  st := string_output ();
  xml_auto (q, vector (), st);
  result_names (q);
  result (string_output_string (st));
}
```

Now we can apply this to a couple of examples:

Example 15.1. XML RAW

```
xmla ('select "category"."CategoryID", "CategoryName",
    "ProductName", "ProductID"
    from "Demo".."Categories" "category", "Demo".."Products" as "product"
    where "product"."CategoryID" = "category"."CategoryID" FOR XML RAW');

<ROW CategoryID="1" CategoryName="Beverages" ProductName="Chai" ProductID="1">
</ROW>
<ROW CategoryID="1" CategoryName="Beverages" ProductName="Chang" ProductID="2">
</ROW>
<ROW CategoryID="1" CategoryName="Beverages" ProductName="Guaraná Fantástica" ProductID="24">
</ROW>
<ROW CategoryID="1" CategoryName="Beverages" ProductName="Sasquatch Ale" ProductID="34">
</ROW>
<ROW CategoryID="1" CategoryName="Beverages" ProductName="Steeleye Stout" ProductID="35">
</ROW>
<ROW CategoryID="1" CategoryName="Beverages" ProductName="Côte de Blaye" ProductID="38">
</ROW>
<ROW CategoryID="1" CategoryName="Beverages" ProductName="Chartreuse verte" ProductID="39">
</ROW>
<ROW CategoryID="1" CategoryName="Beverages" ProductName="Ipoh Coffee" ProductID="43">
</ROW>
<ROW CategoryID="1" CategoryName="Beverages" ProductName="Laughing Lumberjack Lager" ProductID="67">
</ROW>
.....
```

As we can see, RAW mode produces a simple row-by-row account of the data encased within the <ROW.../> tags. This is the simplest mode.

Example 15.2. XML AUTO

```
xmla ('select "category"."CategoryID", "CategoryName",
    "ProductName", "ProductID"
    from "Demo".."Categories" "category", "Demo".."Products" as "product"
    where "product"."CategoryID" = "category"."CategoryID" FOR XML AUTO ELEMENT');

<category>
 <CategoryID>1</CategoryID> <CategoryName>Beverages</CategoryName><product>
 <ProductName>Chai</ProductName> <ProductID>1</ProductID></product>
```

```
<product>
 <ProductName>Chang</ProductName> <ProductID>2</ProductID></product>
<product>
 <ProductName>Guaraná Fantástica</ProductName> <ProductID>24</ProductID></product>
<product>
 <ProductName>Sasquatch Ale</ProductName> <ProductID>34</ProductID></product>
<product>
 <ProductName>Steeleye Stout</ProductName> <ProductID>35</ProductID></product>
<product>
 <ProductName>Côte de Blaye</ProductName> <ProductID>38</ProductID></product>
<product>
 <ProductName>Chartreuse verte</ProductName> <ProductID>39</ProductID></product>
<product>
 <ProductName>Ipoh Coffee</ProductName> <ProductID>43</ProductID></product>
<product>
 <ProductName>Laughing Lumberjack Lager</ProductName> <ProductID>67</ProductID></product>
<product>
.....
```

In contrast to RAW mode, AUTO produces results that are more tree-like. Only one category element is used for each category, and that contains all the children of the category.

Example 15.3. XML EXPLICIT

```
xmla ('
select 1 as tag, null as parent,
       "CategoryID" as [category!1!cid],
       "CategoryName" as [category!1!name],
       NULL as [product!2!pid],
       NULL as [product!2!name!element]
from "Demo".."Categories"
union all
select 2, 1, "category" ."CategoryID", NULL, "ProductID", "ProductName"
    from "Demo".."Categories" "category", "Demo".."Products" as "product"
    where "product"."CategoryID" = "category"."CategoryID"
order by [category!1!cid], 5
FOR XML EXPLICIT');

<CATEGORY CID="1" NAME="Beverages">
<PRODUCT PID="1">
 <NAME>Chai</NAME></PRODUCT>
<PRODUCT PID="2">
 <NAME>Chang</NAME></PRODUCT>
<PRODUCT PID="24">
 <NAME>Guaraná Fantástica</NAME></PRODUCT>
<PRODUCT PID="34">
 <NAME>Sasquatch Ale</NAME></PRODUCT>
<PRODUCT PID="35">
 <NAME>Steeleye Stout</NAME></PRODUCT>
<PRODUCT PID="38">
 <NAME>Côte de Blaye</NAME></PRODUCT>
<PRODUCT PID="39">
 <NAME>Chartreuse verte</NAME></PRODUCT>
<PRODUCT PID="43">
 <NAME>Ipoh Coffee</NAME></PRODUCT>
<PRODUCT PID="67">
 <NAME>Laughing Lumberjack Lager</NAME></PRODUCT>
<PRODUCT PID="70">
 <NAME>Outback Lager</NAME></PRODUCT>
<PRODUCT PID="75">
 <NAME>Rhönbräu Klosterbier</NAME></PRODUCT>
<PRODUCT PID="76">
 <NAME>Lakkalikööri</NAME></PRODUCT>
</CATEGORY>
<CATEGORY CID="2" NAME="Condiments">
<PRODUCT PID="3">
.....
```

In this example, we specify precisely the tree structure we wish, and construct the EXPLICIT query to produce that tree. Many times programmers know what the resulting XML should look like but do not know how to get exactly what they want. FOR XML EXPLICIT can be very useful in these cases.

15.1.3. Functions

`xml_auto()`

15.1.4. FOR XML Syntax

```
for__xml ::= FOR XML <mode> [ ELEMENT ]

<mode> ::= RAW | AUTO | EXPLICIT

<explicit column> ::=  scalar_exp AS '[' <element> '!' <tag no> '!'
                    <column name> [ '!' <option> ] ']'

<tag no> ::= INTNUM

<column name> ::= IDENTIFIER

<element> ::= IDENTIFIER

<option> ::= IDENTIFIER
```

The <explicit column> should be used in the selection list of the first term of the UNION ALL construct in a FOR XML EXPLICIT query. Virtuoso provides this functionality separately from any Web server context, although these are principally expected to be used inside VSP pages.

The text of <option> part of the <explicit column> is ignored but if it is present then the value is placed into a sub-element of the element for the row, not into an attribute.

15.2. XML Composing Functions in SQL Statements (SQLX)

The preferred means of constructing XML data from SQL is to use the standard SQLX SQL extension.

SQLX is a collection of functions added for creating XML entities from standard relational queries. Basically, you write a SQL statement with calls to SQLX functions in the selection and a piece of XML is created.

There are five XML composing functions:

`XMLELEMENT()` creates a single XML element that can contain an arbitrary number of attributes and sub-elements.

`XMLATTRIBUTES()` lists XML attributes to be placed in the XML element created by an enclosing call of `XMLELEMENT()` .

`XMLCONCAT()` returns a forest of XML values by concatenating a list of XML values or forests.

`XMLAGG()` is an aggregate function that creates a forest of XML values by concatenating the XML values that are returned from multiple rows.

`XMLFOREST()` is similar to `XMLATTRIBUTES()` but returns a forest of elements instead of list of attributes.

These functions belong to the SQLX standard, an emerging SQL standard for XML. All the functions take a variable number of arguments.

`XMLELEMENT` is used to construct XML elements from relational data. It takes as parameters the element name, an optional collection of attributes for the element (returned by `XMLATTRIBUTES` call), column names, strings, `XMLELEMENT` , `XMLFOREST` , `XMLCONCAT` , and `XMLAGG` calls, and an entity objects (returned by corresponding functions, e.g. `xtree_doc` , `xpath_eval` , `xquery_eval`) which will make up the content of the element (an exception from this is an attribute entity returned by `xquery_eval` - in this case it is joined to the list of the element's attributes). Column names, strings and attribute entities returned by `xpath_eval` will make up the text content of the element. The others will make up the children of the element.

In the `XMLATTRIBUTES` clause, the value expressions are evaluated to get the values for the attributes.

XMLFOREST produces a forest of XML elements from a given list of arguments. It accepts a list of SQL value expressions as its arguments, and produces an XML element from each value returned.

XMLCONCAT produces a forest of elements by concatenating a list of XML values. XMLCONCAT accepts a list of XML value expressions as its arguments, and produces a forest of elements by concatenating the XML values that are returned from the same row to make one value. If an argument of the XMLCONCAT is an entity object returned by xquery_eval or path_eval , it must not be an attribute.

The following statements create the same result sets:

```
select XMLELEMENT ('Person',
                        XMLELEMENT ('firstname', "FirstName"),
                        XMLELEMENT ('lastname', "LastName"),
                        xquery_eval('//country', xtree_doc('<a><country>USA</country></a>')))
from "Demo"."demo"."Employees";

select
      XMLELEMENT ('Person',
                        XMLFOREST ("FirstName"as "firstname", "LastName" as "lastname"),
                        xquery_eval('//country', xtree_doc('<a><country>USA</country></a>')))
from "Demo"."demo"."Employees";

select
      XMLELEMENT ('Person',
                              XMLCONCAT (
                                        XMLELEMENT ('firstname', "FirstName"),
                                        XMLELEMENT ('lastname', "LastName"),
                                        xquery_eval('//country', xtree_doc('<a><country>USA</country></a>
from "Demo"."demo"."Employees";
```

Note:
The second statement is more effective than the others.

In order to return more than one row of values, you can use XMLAGG . XMLAGG is an aggregate function that produces a forest of XML elements from a collection of XML elements. It concatenates the values returned from one column of multiple rows, unlike XMLCONCAT, which concatenates the values returned from multiple columns in the same row.

The parameters that would be used as element names (in the XMLELEMENT and in the 'AS clause ' of the XMLFOREST and XMLATTRIBUTES) must be valid XML names. If the 'AS clause ' is absent in a list of the parameters of the XMLFOREST or XMLATTRIBUTES , Virtuoso uses the partially escaped form of the column name as the element or attribute name. The partially escaped form means that SQL <identifier> characters that are valid characters in a XML NAME are not changed, SQL <identifier> character that is not valid XML NAME character is replaced with "_xHHHH_", where HHHH is character's upper case hexadecimal code. For example, "first_name" is replaced with "first_x005F_name", "last name" is replaced with "last_x0020_name".

The following example creates an 'FullAddress' element with

- four attributes, three of them ('PostalCode', 'Address', 'City') are produced by XMLATTRIBUTES, and the fourth attribute - 'country' is calculated by xquery_eval
- 'Region' subelement, that is produced by xtree_doc
- text content, that is produced by xpath_eval
- 'emp' subelement with text content from the column "LastName", that is created by nestedXMLELEMENT

```
select XMLELEMENT ('FullAddress',
                        XMLATTRIBUTES ( "PostalCode", "Address", "City"),
                        xtree_doc ('<Region>WA</Region>'),
                        xquery_eval('//@country', xtree_doc('<a country="USA"/>')),
                        xpath_eval('//@Phone', xtree_doc('<a Phone="(206) 555-9857"/>')),
                        XMLELEMENT('emp', "LastName"))
from "Demo"."demo"."Employees"

----------------------------

   <FullAddress PostalCode="98122" Address="507 - 20th Ave. E. Apt. 2A" City="Seattle" country="USA">
        <Region>WA</Region>
        (206) 555-9857
        <emp>Davolio</emp>
```

```
  </FullAddress>
  <FullAddress PostalCode="98401" Address="908 W. Capital Way" City="Tacoma" country="USA">
      <Region>WA</Region>
      (206) 555-9857
      <emp>Fuller</emp>
  </FullAddress>
. . .
```

See Also:

XMLAGG()

XMLATTRIBUTE()

XMLCONCAT()

XMLELEMENT()

XMLFOREST()

XML composing functions deal with arguments of arbitrary type, but the result is always an XML entity that can contain only elements and strings. Hence there is a set of type casting rules. These rules are quite common for any XML DOM model, so they're similar to those listed for DOM function arguments :

If an instance of XMLType is passed then its internal XML entity is used.

If an array representation of an XML tree entity is passed then it is used exactly like XML entity.

If an argument is NULL then it is fully ignored, as if there is no such argument at all.

If an argument is not of a type listed above and not a string then it is cast to a string first.

A root node of some document (or of some generic XML entity) can not appear in the middle of the resulting tree. So if a root node is passed then all child nodes of the root (i.e. every top-level node of the document) will be added.

SQL/XML standards introduce a special name "forest of XML elements" for an ordered list of XML elements, like one returned by `XMLFOREST()` . In Virtuoso, forest can contain XML nodes of any sort, not only XML elements, so it can also contain strings, processing instructions and comments. Virtuoso processes any non-empty "forest" as if it were the root node of a "generic XML entity", and items of the forest were top-level nodes of that entity. Hence, a forest can be passed to any function that accepts an value of type "XML entity". The only potential problem is that this entity is well-formed if and only if the forest is non-empty. If an empty forest is serialized to an XML text then the result is an empty string that is not an acceptable input for an XML parser.

It is important to remember that the XML document can not contain two neighbour text nodes and that the text node can not be an empty string. If two consequent strings appear in the list of values of a forest or in the list of children of an new element then they are replaced with a single node that is a concatenation of these string. Similarly, if an empty string appears in the list of values of a forest or in the list of children of an new element then it is removed from the list.

15.3. Virtuoso XML Services

15.3.1. XPATH Implementation and SQL

Virtuoso offers XPATH as a query language for XML views. The statement is there converted into SQL in the context of the mapping defined by the __view XPATH option, which is mandatory. An XPATH query string is a valid top level SQL statement. This is interpreted as a single select or union of selects with the result columns being specified by various XPATH options.

The basic query string

```
XPATH [__view "cat"]/category
```

will select any top level category elements from the cat XML view, defined with CREATE XML VIEW. This has a single result column with the serialization string of the selected entity as value. This string starts with the category start tag and ends with the corresponding end tag. As many result rows are generated as there are top level category nodes in the view.

This basic behaviour can be modified by XPATH options enclosed in brackets after the XPATH keyword. These options allow specifying the output columns of the generated select statement.

15.3.2. XPATH Query Options

```
<xp option> ::=
    __* | __http | __key   | __view NAME | __tag NAME | __quiet | __base_uri STRING |
    xmlns:NAME '=' STRING | xmlns '=' STRING |
    __lang STRING | __enc STRING | KEYWORD '=' KEYWORD

<option list> ::=  | <option list> '[' <option> [...] ']'
```

The option list may occur between the XPATH keyword and the start of the path, e.g.

```
XPATH [__key __view "cat"] /category
```

The effects of the options are as follows:

__http. Send the serialization of the result entities to the HTTP client. This may only be used inside a VSP page context.

__key. Select the key of the selected entities instead of the serialization text.

__*. Select all columns of the selected entity instead of its serialization text. This is only valid when __view is specified and the result set is homogeneous.

__view STRING. Specify that the query be interpreted in the context of the specified CREATE XML VIEW, directly accessing the tables. This is mandatory since the elements referenced can only be mapped to tables in the context of an XML view.

__quiet. Specify that the query should not signal non-fatal errors. This option is for cases when an incomplete result is anyway better than nothing. Typical example is search in the collection of documents where not all documents are valid.

__base_uri STRING. Specify the base URI that can be used to resolve relative URIs in calls of XPATH functions `processXQuery()` , `processXSLT()` and `processXSLT()` . This is similar to the effect of "declare base-uri" in XQuery.

xmlns:NAME '=' STRING. This declares a pair of namespace prefix and namespace URI for use in the query expression, e.g. xmlns:xs="http://www.w3.org/2001/XMLSchema" or xmlns:ora="http://schemas.oracle.com/xpath/extension"

xmlns '=' STRING. This declares the default namespace URI for use in the query expression, e.g. xmlns="http://www.example.com/my-schema". This namespace will become both default element namespace and default function namespace.

__lang STRING. This declares the language that is used for text search expressions. This is for internationalization purposes only. See subsection "Adding New Languages And Encodings Into Virtuoso" for more details.

__enc STRING. This declares encoding of strings that are used for text search expressions. This is for internationalization purposes only. See subsections "Encoding in XPath Expressions" and "Adding New Languages And Encodings Into Virtuoso" for more details.

KEYWORD '=' KEYWORD. This is useful only if the expression uses XPATH functions like `document()` or `doc()` or if the XPATH expression is an argument of `xpath_contains` or similar special SQL predicate. These are configuration options that XPATH processor provides to the XML parser when the processor should read a document. Section "Configuration Options of the DTD Validator" lists all supported options.

15.3.3. XML Views - Representing SQL Data as Dynamic and Persistent XML

The XML view mechanism allows generating XML content from relational data and to query relational data as if it were XML without first converting it to XML.

CREATE XML VIEW statement

```
<xml view> ::=
     CREATE XML VIEW <name> as [ <namespaces_def> ] <element list> <opt_public>
```

```
<xml view> ::=
     CREATE XML VIEW <name> as [ <namespaces_def> ] <query spec> [ELEMENT] <opt_public>

<namespaces_def> ::= '[' <id_namespace> '='( name | <path string> ) [ ...]  ']'

<element list> ::= <element> [, ...]*

<element> ::=
         <table> <correlation name> AS [ <id_namespace> ':' ] <element> <columns>
         [ ON '('  <search condition> ')']
        [primary key '(' column_commalist ')' ]
        [ELEMENT]
         [ '{' <element list>'}' ]

<opt_public> ::=  PUBLIC <path string> OWNER <DAV owner name> [PERSISTENT] [INTERVAL <minutes>]

<columns> ::= '(' <column> [, ...] ')'

<column> ::=   <column name> | <column name> AS [ <id_namespace> ':' ] <attribute name>

<id_namespace> :: = identifier
```

The XML view declaration establishes a 'virtual document' a context within which XML hierarchy relationships can be translated into arbitrary joins. The virtual document can be then materialized into an actual set of persistent XML elements or used to generate SQL from XPATH.

Each table in the declaration generates an element into the result document. SQL views can be used as tables to accommodate for hidden joins, sub-queries, ordering and aggregates. If a view is used, which by nature has no primary key, the primary key clause should be used to define a uniquely identifying set of view columns.

The only restriction on the XML view declaration is that each branch has a fixed depth.

The structure of joins used to make the text can be specified in two ways: As a SQL query specification, that is a SELECT from a list of tables with a WHERE clause specifying the joins, or as a tree of join elements. The first form is called automatic and the second is called explicit. The automatic form allows generating a tree with as many levels as there are tables in the join, with elements derived from the rows at each level occupying each level of the hierarchy

With both forms the columns of the tables are mapped into either attributes or child elements of the element representing each row. In the explicit mode, the attribute / element choice can be made for each table, in the automatic mode for the entire view. The explicit mode also allows specifying a different element / attribute name whereas the automatic mode takes the name from the column name. Even if the columns are presented as child elements in the output text, they should be referenced as attributes in XPATH queries evaluated in the context of the view.

An XML namespaces can be used in XML view with two restrictions - all namespace names must be different and the first three letters of namespace name must not be 'xml', except for default namespace with name 'xmlns'.

Explicit XML Views

In the explicit form each level of the hierarchy is declared as a list of child elements. Each such element maps one table or view into an entity according to a join condition. The join conditions can reference columns from the associated table and columns from tables in parent elements. The join condition can also have scalar filtering conditions. A top element's join condition may only specify scalar conditions.

Each set of sibling child nodes is delimited by braces {}. The top level of the view typically consists of one element in the outermost braces. This element has itself a child list delimited by braces. Each such list can have more than one different element.

Each element specifies:

- SQL table
- Correlation name for use in subsequent joins for this table
- XML element name to use for delimiting a row of this table
- List of columns, with optional XML element or attribute names
- join condition - will relate rows of this table to rows of the table in the enclosing element. If this element is at the top level, this can only consist of scalar conditions

- Optional PRIMARY KEY clause, needed if the table in this element is a view, does not have a primary key or if a non-primary key unique identity is desired
- Optional ELEMENT flag
- Optional list of child elements, delimited by braces

Note:

A correlation name is mandatory for all the tables.

The column list can mention a single column or a single column renamed into an XML attribute of a different name. If a column of a table is referenced in a subsequent join condition it must appear in the output columns list. Expressions are not directly allowed but a view with expression columns can be used.

The opt_public clause, when present, offers a shorthand for calling xml_view_publish at the same time as making the definition. This makes a DAV resource reflecting the contents of the view. The contents may either be generated on demand or persisted as a DAV accessible XML document. In the latter case the document may be regenerated at a fixed interval. The interval is expressed in minutes.

The path is expressed as an absolute path from the root collection of the DAV server.

Note:

This root collection may be mapped into various places in the web server's URL space.

```
create xml view xx ... public '/xx.xml' owner 'dav' persistent interval 1;
```

is equivalent to:

```
create xml view xx ...;
xml_view_publish (xx, /xx.xml', 'dav', 1, 1);
```

A DAV resource created in this manner can be deleted as any DAV resource. The XML view itself is not affected but a possibly existing refresh job will be automatically deleted.

One XML view can be published several times with different names and owners. There may also exist persistent and non-persistent publications of the same view.

The CREATE XML VIEW statement defines stored procedures for generating an XML text fragment corresponding to each element declared in the view.

The names of the procedures are composed as follows:

```
create procedure http_view_<view name>
        (inout output_mode)

create procedure http_<view name>_<element name>_<correlation name>
        (in pk1 any, ..., in output_mode integer)
```

An http output procedure is created for each <element> in the create xml view declaration. It takes the primary key columns of the table in question in key order plus a mode flag. It then outputs the serialization of the specified element and any child elements. For an output mode of 0 the result goes directly to an HTTP client. For an output_mode of 1 the procedure returns the serialization as a string. Note that for this to work the tables in question must be real tables and the join conditions must only reference the next higher table in the create xml view tree. Further, the primary key columns of each table should be mentioned in the columns list for that table along with any foreign keys referenced in subsequent join conditions.

Example 15.4. Examples

```
create xml view "cat" as
{
  "Demo"."demo"."Categories" "C" as "category"
        ("CategoryID", "Description" as "description")
    {
      "Demo"."demo"."Products" "P" as "product"  ("ProductName")
        on ("P"."CategoryID" = "C"."CategoryID")
    }
}
```

This declares a two level hierarchy with a category node for each category and a product child node for each product in the category.

```
create xml view "cats_e" as
  select "category"."CategoryID", "CategoryName",
    "ProductName", "ProductID"
    from "Demo".."Categories" "category", "Demo".."Products" as "product"
    where "product"."CategoryID" = "category"."CategoryID" element;
```

Here is a similar example, this time using the element option.

The procedures are

```
xmlg_cat
http_cat_category_C (in categoryid any, in _out integer);
http_cat_product_P (in productid any, in _out integer);
```

In the following example the function returns the selected items as an XML fragment. Consecutive elements are separated by new-lines for readability.

Example 15.5. Example

```
SQL> call "http_cat_category_C" (1, 1);

1 sets? Done. -- 5 msec.
RESULT=
<category CategoryID="1" description="Soft drinks, coffees, teas, beers, and ales" >
<product ProductName="Chai" ></product>
<product ProductName="Chang" ></product>
<product ProductName="Guarana Fantastica" ></product>
<product ProductName="Sasquatch Ale" ></product>
<product ProductName="Steeleye Stout" ></product>
<product ProductName="Côte de Blaye" ></product>
<product ProductName="Chartreuse verte" ></product>
<product ProductName="Ipoh Coffee" ></product>
<product ProductName="Laughing Lumberjack Lager" ></product>
<product ProductName="Outback Lager" ></product>
<product ProductName="Rhönbrooou Klosterbier" ></product>
<product ProductName="Lakkalikööri" ></product>
</category>
```

The below example shows how to use a SQL view for hiding a join. The below view generates for each table a set of column children and a set of index children, which in turn have column children.

```
create view KEY_COLS as select KP_KEY_ID, KP_NTH, C.*
        from SYS_KEY_PARTS, SYS_COLS C where COL_ID = KP_COL;

create xml view "schema" as
{
  DB.DBA.SYS_KEYS k as "table" ("KEY_TABLE" as "name",
        KEY_ID as "key_id", KEY_TABLE as "table")
        on (k.KEY_IS_MAIN = 1 and k.KEY_MIGRATE_TO is null)
        { DB.DBA.KEY_COLS  c as "column" (COLUMN as name)
                on (k.KEY_ID = c.KP_KEY_ID)
                primary key (COL_ID),
        DB.DBA.SYS_KEYS i as "index" (KEY_NAME
          as "name", KEY_ID as "key_id", KEY_N_SIGNIFICANT as "n_parts")
          on (i.KEY_TABLE = k.KEY_TABLE and i.KEY_IS_MAIN = 0 and i.KEY_MIGRATE_TO is null)
          {
            DB.DBA.KEY_COLS ic as "column" (COLUMN as "name")
              on (ic.KP_NTH < i.KEY_N_SIGNIFICANT and ic.KP_KEY_ID = i.KEY_ID)
              primary key (COL_ID)
              }
        }
};
```

The following query will return the subtree describing the Customers table in the demo database:

```
XPATH [__view 'schema']
        /table[@name = 'Demo.demo.Customers'];

<table name="0" key_id="1011" table="Demo.demo.Customers" >
<column name="CustomerID" ></column>
<column name="CompanyName" ></column>
<column name="ContactName" ></column>
<column name="ContactTitle" ></column>
<column name="Address" ></column>
<column name="City" ></column>
<column name="Region" ></column>
<column name="PostalCode" ></column>
<column name="Country" ></column>
<column name="Phone" ></column>
<column name="Fax" ></column>
<index name="City" key_id="1012" n_parts="2" >
<column name="City" ></column>
<column name="CustomerID" ></column>
</index>
<index name="CompanyName2" key_id="1013" n_parts="2" >
<column name="CompanyName" ></column>
<column name="CustomerID" ></column>
</index>
<index name="PostalCode2" key_id="1014" n_parts="2" >
<column name="PostalCode" ></column>
<column name="CustomerID" ></column>
</index>
<index name="Region" key_id="1015" n_parts="2" >
<column name="Region" ></column>
<column name="CustomerID" ></column>
</index>
</table>
```

Automatic XML Views - Creating XML Views from SELECT Statements

The automatic form of CREATE XML VIEW will take a select statement and infer a hierarchy from it, based on the order of tables in the from clause. The parent table should be to the left of its children. This is practical if the tables form a hierarchy in application terms, like orders and order lines or departments and employees. This notation allows arbitrary depth but all siblings at the same level will be of the same type. Elements of child rows will be child elements of the element of their parent row, where the join condition identifies the child rows for one parent row.

The columns in the selection will appear as attributes or child elements of the rows selected. The names of the attributes will be the names of the columns. The names of the siblings will be the names of the tables in the from clause, without qualifiers or owners. Expressions should not appear in the selection. If the use of expressions is required then you may create a SQL view first to facilitate this.

The ELEMENT keyword may be present at the end of the select, before the publishing keywords. This will cause all columns to be represented as child elements of the element corresponding to the row. Note that even if the element switch is present, the values will appear like attributes in an XPATH query inside the view.

Note

SQL views or derived tables may not appear directly in the select. The reason for this is that a procedure is generated for each level of the generated XML tree and that this must take unique identifying column values for the element in question. If one desires to use a view, the explicit form should be used, with the primary key option specified where appropriate.

Add to text for explicit XML views

Each set of sibling child nodes is delimited by braces {}. The top level of the view typically consists of one element in the outermost braces. This element has itself a child list delimited by braces. Each such list can have more than one different element.

Each element specifies:

- SQL table
- Correlation name for use in subsequent joins for this table

- XML element name to use for delimiting a row of this table
- List of columns, with optional XML element or attribute names
- join condition - will relate rows of this table to rows of the table in the enclosing element. If this element is at the top level, this can only consist of scalar conditions
- Optional PRIMARY KEY clause, needed if the table in this element is a view, does not have a primary key or if a non-primary key unique identity is desired
- Optional ELEMENT flag
- Optional list of child elements, delimited by braces

Example 15.6. Add to examples:

```
create xml view "cats_e" as
  select "category"."CategoryID", "CategoryName",
    "ProductName", "ProductID"
    from "Demo".."Categories" "category", "Demo".."Products" as "product"
    where "product"."CategoryID" = "category"."CategoryID" element;

Add to text: after 'free text and xml'
```

xml_view_publish

```
DB.DBA.xml_view_publish (in view_name varchar, in dav_path varchar,
        in dav_owner varchar, in is_persistent integer, in refresh_interval integer)
```

This presents an XML view as a DAV resource. The view name is the name in the create xml view statement, note that this is case sensitive and is never converted since it is a string, not an identifier. The path must be absolute and is interpreted as relative from the DAV root collection. The DAV user is the owner of the resource. If is_persistent is non-zero the resource will be materialized from the view's description. The refresh interval is only applicable if the resource is materialized. If so, this is an interval in minutes. A value of 0 means no automatic refresh.

The reverse operation of xml_view_publish is deleting the DAV resource. xml_view_publish may be called several times to alter the owner or refresh interval.

15.3.4. External Entity References in Stored XML

When an XML document is stored as either text, long xml, xmltype or in the persistent XML format it can contain references to external parsed entities with the <!entity ...> declaration and the &xx; syntax. These are stored as references and not expanded at storage time if the entity is external.

Such references are transparently followed by XPATH and XSLT. A run time error occurs if the referenced resource cannot be accessed when needed. The reference is only followed if the actual subtree is selected by XPATH or XSLT. The resource is retrieved at most once for each XPATH or XSLT operation referencing it, regardless of the number of times the link is traversed. This is transparent, so that the document node of the referenced entity appears as if it were in the place of the reference.

External entity references have an associated URI, which is either absolute with protocol identifier and full path or relative. Relative references must be resolved with respect to the base URI of the referencing document. If the document is stored as a column value in a table it does not have a natural base URI, hence the application must supply one if relative references are to be supported. This is done by specifying an extra column of the same table to contain a path, in the form of collections delimited by slashes, just as the path of a DAV resource or a Unix file system path.

This base URI is associated with an XML column with the IDENTIFIED BY declaration:

```
create table XML_TEXT (
        XT_ID integer,
        XT_FILE varchar,
        XT_TEXT long varchar identified by xt_file,
                primary key (XT_ID)
        );

create index XT_FILE on XML_TEXT (XT_FILE);
```

Thus, each time the value of xt_text is retrieved for XML processing by XPATH_CONTAINS or XCONTAINS the base URI is taken from xt_file.

The complete URI for the xt_text of a column of the sample table will be:

```
virt://<qualified table name>.<uri column>.<text column>:<uri column value>
```

An example would be:

```
"virt://DB.DBA.XML_TEXT.XT_FILE.XT_TEXT:sqlreference.xml"
```

The .. and . in relative paths are treated as with file names when combining relative references to base URI's. A relative reference without a path just replaces the last part of the path in the base URI.

See
xml_uri_get and xml_uri_merge for more details.

15.3.5. Using XPATH in SQL Queries and Procedures

An XPATH expression can appear as a SQL query expression, that is, as a derived table or subquery predicate or scalar subquery. This means that the XPATH expression is expanded compile time to the corresponding SQL. The mapping of the XPATH hierarchy to tables and joins is given by the __view XPATH option, which is mandatory.

The XPATH keyword introduces an embedded XPATH expression. The XPATH text is presented as a string literal. Note that the tokenization rules are different for XPATH and SQL, so having XPATH as a string makes it clear which rules apply to parsing which part of the composite query.

Example 15.7. Example

```
select * from (XPATH '[__* __view "cat"]
        //product') P order by "P."ProductName"";
```

will evaluate the //product query in the context of the cat XML view and produce a result set consisting of all the attributes of the product entity as defined in the view.

Note:
The __key and __* XPATH options are central here in defining the result columns of the XPATH. The default result column of an XPATH expression is the serialization of the selected entity or scalar, which is most of the time impractical in a SQL context.

Parameters in XPATH

The '$' sign introduces a parameter in XPATH. The identifier following the dollar sign should reference a SQL column or variable defined in the surrounding context. The name of the parameter can contain a dot for referencing a column with a correlation name.

For instance, to make a VSP page that outputs the category tree which contains a specific product, one may write:

```
<HTML>
<?vsp
      declare N varchar;
      N := {?'name'};
      for (XPATH '[__http __view ''cat'']
            /category[product/@ProductName = $N]' do ; ?>
</HTML>
```

This will iterate over the categories containing a product with ProductName equal to the URL parameter 'name'. Note the __http option that causes the text of the selected entities to go directly to the HTTP client. Note the double '' escape for the XML view name inside the SQL string literal forming the name.

Also note that the N parameter is in upper case to work in all case modes. In some modes SQL identifiers will be converted automatically to upper case but this conversion does not apply inside XPATH.

```
select * from "Demo".."Categories" C
        where exists (XPATH '[__view "ord"]
        //products[@CategoryID = $C.CategoryID]');
```

This example selects the categories of products that have been mentioned in the ord XML view.

Syntax Notes

The main difference of SQL and XPATH is that the '-' is not a breaking character in XPATH and that XPATH is case sensitive without any implicit identifier case conversion.

15.3.6. XQUERY and XML view

Similarly to XPATH, XQUERY may also be used as a query language for XML views. Virtuoso offers a special case of FLWR expression for this purpose. It is possible to use the `xmlview()` function in FOR clause expressions for querying XML views. This function is similar to document () in the sense that it sets the source of the path to be the logical root of the referenced XML view. The XML view must be a constant known at compile time. A SQL query against the appropriate tables of the XML view is internally constructed and evaluated at run time, producing XML fragments from the selected rows. At no point will non-selected parts of the evaluation of the XML view be physically created. The path expression following from `xmlview()` may contain filters involving XQuery variables bound in the scope of the path expression, thus allowing joining SQL data to XQuery variable values.

The XQUERY string

```
for $cat in xmlview("cat")/category return {$cat}
```

in the query

```
select xquery_eval(' {for $cat in xmlview("cat")/category return <q>{$cat}</q>}',
                     xtree_doc('<dummy_tag/>'));
```

is equivalent to the XPATH query string

```
XPATH [__view "cat"]/category
```

described above.

The expression xmlview("viewname")/path is not a valid top level SQL statement, but may be used by xquery_eval() function. The path statement is translated into SQL query in the context of the "viewname" (i.e. the necessary table names are taken from "viewname" XML view), so that only the desired relational data will be queried. Functionality of this kind of SQL queries is similar to functionality of the SQL fetch statement, i.e. such a query provides iteration over the result set of a cursor. The query is executed once for each row in the query expression's result set. Thus using the CREATE XML VIEW statement and a XQUERY FOR clause expression with `xmlview()` function allows you to query the database and to return the results in the form of an XML document and to avoid redundant data access. This kind of queries also allows computing joins between two or more documents and restructuring data.

Note

The <dummy_tag/> tag is not used and it is necessary only as an arbitrary argument for xtree_doc() functions.

Example 15.8. Example 1

```
create xml view "product" as
{
  "Demo"."demo"."Products" p as "product"
      ("ProductID", "ProductName" as "product_name","UnitPrice" as "price",
      "SupplierID","CategoryID")
    {
      "Demo"."demo"."Suppliers" s as "supplier" ("SupplierID","CompanyName")
        on (s."SupplierID" = p."SupplierID")
       ,
      "Demo"."demo"."Categories" c as "category" ("Description")
        on (c."CategoryID" = p."CategoryID")

    }
}
```

This declares a two level hierarchy with a product node for each product and a supplier child node and a category child node of the product.

The following query will return the XML document in which each category node will contain all suppliers supplying products of the given category.

```
select xquery_eval('
  <document>
    {
      for $cat in xmlview("cat")/category
      return (
        <category description={$cat/@description}>
          {
            distinct (
              for $prod in xmlview("product")/product
              where $cat/@CategoryID=$prod/category/@CategoryID
              return $prod/supplier )
          }
        </category> )
    }
  </document>
  ',
  xtree_doc('<dummy_tag/>')
  );
```

returns the XML document in which each category node contains all suppliers supplying products of the given category.

Example 15.9. Example 2

Let a document named suppliers.xml contains supplier elements; each supplier element in turn contains supplier_id and supplier_name subelements. The following query

```
select xquery_eval('
  <supplier_product>
    {
      for $supp in document("suppliers.xml")/supplier
      return (
        <supplier>{$supp/supplier_name }
          <product_name>
            {
              for $prod in xmlview("product")/product
              where string($supp/supplier_id)=$prod/supplier/@SupplierID
              return string($prod/@product_name)
            }
          </product_name>
        </supplier>)
    }
  </supplier_product>',
  xtree_doc('<dummy_tag/>')
  );
```

returns the XML document that contains supplier elements; each supplier element in turn contains supplier_name and product_name elements.

The previous query and the following one show that it is possible to use variables in a XPATH expression following `xmlview()` functions.

Example 15.10. Example 3

The query

```
select xquery_eval('
  <document>
    {
      distinct(
        let $ex:= "Ex%"
        for $prod in xmlview("product")//supplier[@CompanyName like $ex]
        return <supp_id>{$prod/@SupplierID}</supp_id>)
    }
  </document>',
```

```
  xtree_doc('<dummy_tag/>'));
```

is equivalent to

```
select xquery_eval('
  <document>
    {
      distinct(
        for $prod in xmlview("product")//supplier [@CompanyName like "Ex%"]
        return <supp_id>{$prod/@SupplierID}</supp_id>)
    }
  </document>',
  xtree_doc('<dummy_tag/>'));
```

and selects all suppliers having attribute "CompanyName" starting with "Ex".

Optimization in the queries with xmlview() function

At least two methods may be used to accelerate the execution of queries with xmlview() function. The first method assumes that a path statement following xmlview() function should contain maximum conditions to reduce the result set. For example, the query

```
select xquery_eval('<w>
  {
    for $prod in xmlview("product")/product[@ProductID="1"]
    return <q>{$prod}</q>
  }</w>',
  xtree_doc('<dummy_tag/>'));
```

will be executed faster than

```
select xquery_eval('<w>
  {
    for $prod in xmlview("product")/product where $prod[@ProductID="1"]
    return <q>{$prod}</q>
  }</w>',
  xtree_doc('<dummy_tag/>'));
```

due to the SQL query produced from the path expression 'product[@ProductID="1"]' reduces the result set in comparison with the SQL query produced from the path expression 'product' in the second query.

If we execute a join of two (or more) XML views (or XML document and XML view), i.e. the query consists of the nested loops, the second method proposes to carry out a piece of query of the nested loop which is independent of the outer loop outside the outer loop and uses LET clause for it. For example, the query

```
select xquery_eval('<document>
  {
    let $prod_set:=(for $prod in xmlview("product")/product return $prod)
    for $cat in xmlview("cat")/category
    return (<category description={$cat/@description}$gt;
      {distinct(for $prod in $prod_set
                where $cat/@CategoryID=$prod/category/@CategoryID
                return $prod/supplier)}</category>)}</document>',
   xtree_doc('<dummy_tag/>'));
```

is equivalent to the query in the example 1, but it is about 5 times faster than original one. This method is especially useful for full joins. In this case we do not have a full join and this query may be optimized without the use of temporary result sets if the 'where' clause is replaced with proper filter:

```
select xquery_eval('<document>
  {
    for $cat in xmlview("cat")/category
    let $catID := $cat/@CategoryID
    return (<category description={$cat/@description}$gt;
      {distinct(
        for $supp in xmlview("product")/product[category/@CategoryID=$catID]/supplier
        return $supp)
```

```
        }</category>)
    }
    </document<',
    xtree_doc('<dummy_tag/>'));
```

This variant requires no memory for storing $prod_set and it never fetches redundant fields from "Demo"."demo"."Products" table but it heavily needs index for "Demo"."demo"."Products" on "CategoryID" field. If such index is built the last variant is about 10 times faster than the query in example 1. Similarly, the query in example 2 may be optimized as follows:

```
select xquery_eval('
  <supplier_product>
   {
     for $supp in document("suppliers.xml")/supplier
     let $supp_id:=string($supp/@supplier_id)
     return (
      <supplier>{$supp/supplier_name}
        <product_name>
          {
            for $prod in xmlview("product")/product[supplier/@SupplierID=$supp_id]
            return string($prod/@product_name)
          }
        </product_name>
      </supplier>)
   }
  </supplier_product>',
  xtree_doc('<dummy_tag/>')
  );
```

and it speeds up the operation by more than 15 times.

Restrictions in XPATH expressions following the xmlview() function

Virtuoso does not support certain kinds of XPATH expressions applied to the xmlview() function.

1. A path expression must not contain any functions, because it is impossible to translate most of the functions to SQL queries.

2. A path expression must not contain numeric XQUERY variables in the arithmetic expressions. Let a document named products.xml contains product elements; each product element has numeric attribute ProductPrice. A run time error occurs if the following query would be used

```
select xquery_eval('
  <document>
   {
     for $prod_doc in document("products.xml")/products/product/@ProductPrice
       for $prod_view in xmlview("product")/product[@price>$prod_doc+1]
     return <q>{$prod_doc, $prod_view/@product_name}</q>
   }
   </document>',
   xtree_doc('<dummy_tag/>'));
```

because the type of $prod_doc is considered as string. As it is mentioned in the previous restriction the using a function in a path expression (e.g. xmlview("product")/product[@price>number($prod_doc)+1]) is not allowed. The correct query is as follows:

```
select xquery_eval('
  <document>
    {
      for $prod_doc in document("products.xml")/products/product/@ProductPrice
        let $prod_doc2:=number($prod_doc)
        for $prod_view in xmlview("product")/product[@price>$prod_doc2+1]
      return <q>{$prod_doc, $prod_view/@product_name}</q>
    }
  </document>',
  xtree_doc('<dummy_tag/>'));
```

3. A path expression must not contain XQUERY variables with the following paths. The following query will not be executed

```
select xquery_eval('
  <document>
```

```
    {
      for $cat in xmlview("cat")/category,
          $prod in xmlview("product")/product[@CategoryID=$cat/@CategoryID]
      return <q>{$prod/supplier}</q>
    }
  </document>',
  xtree_doc('<dummy_tag/>'));
```

The correct query may be given as

```
select xquery_eval('
  <document>
    {
      for $cat in xmlview("cat")/category
      let $cat_id:=$cat/@CategoryID
      for $prod in xmlview("product")/product[@CategoryID=$cat_id]
      return <q>{$prod/supplier}</q>
    }
  </document>',
  xtree_doc('<dummy_tag/>'));
```

4. Virtuoso does not support a selection of n-th element in a path expression. The following query will not be executed

```
select xquery_eval('
  <document>
    {
      for $cat in xmlview("cat")//product[1] return <q>{$cat}</q>
    }
  </document>',
  xtree_doc('<dummy_tag/>'));
```

5. Virtuoso does not support a dereference (=>) in a path expression.

6. It is not recommended to use the long varchar, long varbinary and long nvarchar data types with the logical and boolean operations in a filter of the path expression. For example, the execution of the following query

```
select xquery_eval('
  <document>
    {
      for $prod in
        xmlview("product")/product[@SupplierID<5]/category[@Description like "Sw%"]
      return <q>{$prod}</q>
    }
  </document>',
  xtree_doc('<dummy_tag/>'));
```

may return an error, because the field "Description" has LONG VARCHAR type in the table "demo"."Categories".

15.3.7. Mapping Schemas as XML Views

Virtuoso supports creating XML views by using annotated XSD schemas referred to as mapping schemas. A file containing a mapping schema may be loaded by calling the `xml_load_mapping_schema_decl()` function. A name (without extension .xsd) of the file containing a mapping schema is considered to be the name of the xml view, defined by the given mapping schema.

A loaded mapping schema may be queried in the same way as one would query XML views defined using the CREATE XML VIEW statement with XPATH:

```
XPATH [__view "xml_view_name"]/xpath_query
```

Example 15.11. Loading and Querying a Mapping Schema

The XML view "Catmp" from the file "Catmp.xsd" may be loaded using the following statement:

```
 select  xml_load_mapping_schema_decl (
    'http://localhost.localdomain/xmlrepository/',
    'Catmp.xsd',
```

```
        'UTF-8',
        'x-any' ) ) );
```

where the contents of "Catmp.xsd" is

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:sql="urn:schemas-microsoft-com:mapping-schema">
<xsd:annotation>
  <xsd:appinfo>
    <sql:relationship name="CategoryProduct"
          parent="Demo.demo.Categories"
          parent-key="CategoryID"
          child="Demo.demo.Products"
          child-key="CategoryID" />
  </xsd:appinfo>
</xsd:annotation>

  <xsd:element name="category" sql:relation="Demo.demo.Categories" type="CategoryType" />
   <xsd:complexType name="CategoryType" >
     <xsd:sequence>
        <xsd:element name="product"
                    sql:relation="Demo.demo.Products"
                    sql:relationship="CategoryProduct" >
           <xsd:complexType>
              <xsd:attribute name="ProductName" type="xsd:string" />
           </xsd:complexType>
        </xsd:element>
     </xsd:sequence>
        <xsd:attribute name="CategoryID"  type="xsd:integer" />
        <xsd:attribute name="description"  sql:field="Description"  type="xsd:string" />
    </xsd:complexType>
</xsd:schema>
```

The XML view "Catmp" loaded from the file "Catmp.xsd" is similar to XML view "cat" defined by CREATE XML VIEW in the section Explicit Xml Views .

The query

```
XPATH [__view 'Catmp'] /category[@* = 1];
```

will now return the following result:

```
<category CategoryID="1" description="Soft drinks, coffees, teas, beers, and ales" >
  <product ProductName="Chai" ></product>
  <product ProductName="Chang" ></product>
  <product ProductName="Guarana Fantastica" ></product>
  <product ProductName="Sasquatch Ale" ></product>
  <product ProductName="Steeleye Stout" ></product>
  <product ProductName="Cote de Blaye" ></product>
  <product ProductName="Chartreuse verte" ></product>
  <product ProductName="Ipoh Coffee" ></product>
  <product ProductName="Laughing Lumberjack Lager" ></product>
  <product ProductName="Outback Lager" ></product>
  <product ProductName="Rhonbrau Klosterbier" ></product>
  <product ProductName="Lakkalikoori" ></product>
</category>
```

Mapping schemas provide more flexibility than XML views defined by the `CREATE XML VIEW` statement. In the following mapping schema a constant element, "CustomerOrders", an element that does not map to any database table or column but may appear in the resulting XML as a container element of other subelements, is specified by the `sql:is-constant` annotation.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:sql="urn:schemas-microsoft-com:mapping-schema">
<xsd:annotation>
  <xsd:appinfo>
    <sql:relationship name="CustOrders"
```

```
        parent="Demo.demo.Customers"
        parent-key="CustomerID"
        child="Demo.demo.Orders"
        child-key="CustomerID" />
  </xsd:appinfo>
</xsd:annotation>

  <xsd:element name="Customer" sql:relation="Demo.demo.Customers" >
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="CustomerOrders" sql:is-constant="1" >
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Order" sql:relation="Demo.demo.Orders"
                           sql:relationship="CustOrders"
                           maxOccurs="unbounded" >
                <xsd:complexType>
                   <xsd:attribute name="OrderID" type="xsd:integer" />
                   <xsd:attribute name="OrderDate" type="xsd:date" />
                   <xsd:attribute name="CustomerID" type="xsd:string" />
                </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
          <xsd:attribute name="CustomerID" type="xsd:string" />
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

After loading the "Cust_constant.xsd" file containing the given mapping schema, the xpath query:

```
XPATH [__view 'Cust_constant'] /category[@* = 1];
```

will return the following result:

```
<Customer CustomerID="ALFKI" >
  <CustomerOrders >
    <Order CustomerID="ALFKI" OrderDate="1995-09-25 00:00:00.000000" OrderID="10643" ></Order>
    <Order CustomerID="ALFKI" OrderDate="1995-11-03 00:00:00.000000" OrderID="10692" ></Order>
    <Order CustomerID="ALFKI" OrderDate="1995-11-13 00:00:00.000000" OrderID="10702" ></Order>
    <Order CustomerID="ALFKI" OrderDate="1996-02-15 00:00:00.000000" OrderID="10835" ></Order>
    <Order CustomerID="ALFKI" OrderDate="1996-04-15 00:00:00.000000" OrderID="10952" ></Order>
    <Order CustomerID="ALFKI" OrderDate="1996-05-09 00:00:00.000000" OrderID="11011" ></Order>
  </CustomerOrders>
</Customer>
    . . .
```

Virtuoso does not support all mapping schema annotations at this time. The following are currently unsupported:

`sql:encode`
`sql:use-cdata`
`sql:overflow-field`
`sql:inverse`
`sql:hide`
`sql:guid`
`sql:max-depth`

Also, there are some restrictions to the structure of mapping schemas:

Attributes can not contain sql:relationship annotation.
Subelement having no sql:is-constant annotation and not mapped to any table can not contain subelements and attributes.
Recursive relationships is not supported.

 **Note:**

Note:

The XML views, defined by mapping schemas may not be queried using XQUERY. They can however be referenced with the xmlview XPATH functions in path expressions inside an XQuery query.

See Also:

SQLXML 3.0 documentation: Creating XML Views by Using Annotated XSD Schemas.

15.3.8. Differences Between SQLX, FOR XML and XML Views

A SQLX or FOR XML query has no effect on the database schema. It is a transient event and does not generate procedures or other schema elements.

These define an ad hoc mapping of a result set to XML. There is no possibility of using XPATH to specify a search without first constructing the whole tree. An XML view on the other hand provides a mapping context in which one can make XQUERY or XPATH queries that are mapped into SQL and the XML is only generated after applying the conditions.

XML views will usually be more efficient in complex cases and the notation there may be simpler than the EXPLICIT notation in FOR XML. For simple cases SQLX or FOR XML is the more convenient of the two. SQLX or FOR XML does not restrict the SQL being used and will allow free use of subqueries, expressions, derived tables, qualified joins etc.

15.4. Querying Stored XML Data

15.4.1. XPATH_CONTAINS SQL Predicate

XPath expressions can be used in SQL statements to decompose and match XML data stored in columns. The *xpath_contains* SQL predicate can be used either to test for an XML value matching a path expression or to extract one or more entities from the XML value. These values can then be used later in the query as contexts for other XPath expressions.

```
xpath_contains (xml_column, xp_expression[, query_variable]);
```

The first argument, *xml_column* is the name of the column on which to perform the XPath search. The second argument, *xp_expression* , takes an XPath expression.

The third argument is an optional query variable that gets bound to each result entity value of the xpath expression. If this variable is omitted the xpath_contains predicate will qualify the query by returning true for matches. In this case the result will only return one row per match. If the variable is present, the result set could contain multiple rows per result set row of the base table, one row for each match.

Consider the example:

```
select xt_file, t from xml_text
  where xpath_contains (xt_text, '//chapter/title[position () = 1]', t);
```

This SQL statement will select the first title child of any chapter entities in the XML documents in the xt_text column of the table *xml_text* . There can be several matching entities per row of xml_text. The result set will contain a row for each matching entity.

In XPath terms the path expression of *xpath_contains* is evaluated with the context node set to the root node of the XML tree represented by the value of the column that is the first argument of xpath_contains. This node is the only element of the context node set.

Note:

The 't' variable in the above example gets bound to XML entities, not to their string values or other representations. One can thus use these values as context nodes for other expressions.

The XPATH expression can have a list of options in the beginning. The list of options is surrounded by square brackets. Options in the list are delimited by spaces. The most popular option is *__quiet* that allows to process a set of rows if not all stored documents are valid XMLs; if an error is signalled by the XML parser when it prepares a content document for the XPATH in question and the XPATH contains *__quiet* then the error is suppressed and the row is silently ignored as if XPATH found nothing. One can configure the DTD validator of the parser by placing its configuration parameters in the list of XPATH options.

The following example is almost identical to the previous one but it works even if not all values of *xt_text* are valid XMLs, and the resulting values of the 't' variable are standalone entities even if source documents in xt_text contain external generic entities.

```
select xt_file, t from xml_text
  where xpath_contains (xt_text, '[__quiet BuildStandalone=ENABLE]//chapter/title[position () = 1]', t);
```

15.4.2. Using xpath_eval()

The `xpath_eval()` function is used to filter out parts of an XML fragment that match a given XPATH expression. It can be used to retrieve multiple-node answers to queries, as it is often the case that more than one node-set matches. Consider the following statements that create a table with XML stored inside.

```
CREATE TABLE t_articles (
        article_id int NOT NULL,
        article_title varchar(255) NOT NULL,
        article_xml long varchar
        );

insert into t_articles (article_id, article_title) values (1, 'a');
insert into t_articles (article_id, article_title) values (2, 'b');

UPDATE t_articles SET article_xml = '
<beatles id = "b1">
<beatle instrument = "guitar" alive = "no">john lennon</beatle>
<beatle instrument = "guitar" alive = "no">george harrison</beatle>
</beatles>'
WHERE article_id = 1;

UPDATE t_articles SET article_xml = '
<beatles id = "b2">
<beatle instrument = "bass" alive = "yes">paul mccartney</beatle>
<beatle instrument = "drums" alive = "yes">ringo starr</beatle>
</beatles>'
WHERE article_id = 2;
```

Now we make a query that will return a vector of results, each vector element corresponding to a node-set of the result.

```
SELECT xpath_eval('//beatle/@instrument', xml_tree_doc (article_xml), 0)
        AS beatle_instrument FROM t_articles WHERE article_id = 2;
```

The repeating nodes are returned as part of a vector, the third argument to `xpath_eval()` is set to 0, which means that it is to return all nodes.

Otherwise, we can select the first node-set by supplying 1 as the third parameter to `xpath_eval()` :

```
SELECT xpath_eval('//beatle/@instrument', xml_tree_doc (article_xml), 1)
        AS beatle_instrument FROM t_articles WHERE article_id = 2;
```

See Also:

`xpath_eval()`

`xquery_eval()`

`xmlupdate()`

15.4.3. External Entity References in Stored XML

When an XML document is stored as either text or in persistent XML format it can contain references to external parsed entities with the <!entity ...> declaration and the &xx; syntax. These are stored as references and not expanded at storage time if the entity is external. Such references are transparently followed by XPath and XSLT. A run-time error occurs if the referenced resource cannot be accessed when needed. The reference is only followed if the actual subtree is selected by XPath or XSLT. The resource is retrieved at most once for each XPath or XSLT operation referencing it, regardless of the number of times the link is traversed. This is transparent, so that the document node of the referenced entity appears as if it were in the place of the reference.

External entity references have an associated URI, which is either absolute, with protocol identifier and full path, or relative. Virtuoso resolves relative references with respect to the base URI of the referencing document. If the document is stored as a column value in a table it does not have a natural base URI; therefore, the application must supply one if relative references are to be supported. This is done by specifying an extra column of the same table to contain a path, in the form of collections delimited by slashes, like the path of a DAV resource or a Unix file system path. This base URI is associated with an XML column with the IDENTIFIED BY declaration:

```
create table XML_TEXT (
        XT_ID integer,
        XT_FILE varchar,
        XT_TEXT long varchar identified by xt_file,
                primary key (XT_ID)
        );

create index XT_FILE on XML_TEXT (XT_FILE);
```

Thus, each time the value of *xt_text* is retrieved for XML processing by `xpath_contains()` or `xcontains()` the base URI is taken from *xt_file* . The complete URI for the *xt_text* of a column of the sample table would be:

```
virt://<qualified table name>.<uri column>.<text column>:<uri column value>
```

An example would be:

```
"virt://DB.DBA.XML_TEXT.XT_FILE.XT_TEXT:sqlreference.xml"
```

The '..' and '.' in relative paths are treated like file names when combining relative references to base URIs. A relative reference without a path just replaces the last part of the path in the base URI.

See Also:

`xml_uri_get()` and `xml_uri_merge()` for more details.

15.4.4. XML Schema & DTD Functions

The following functions can be used to generate XML Schema or DTD information about a given SQL query:

`xml_auto_schema()`

`xml_auto_dtd()`

Example 15.12. Generating XML Schema and DTD Data

This example shows trivial use of the two functions `xml_auto_schema()` and `xml_auto_dtd()` .

```
SQL> select xml_auto_schema('select u_name from sys_users', 'root');
callret
VARCHAR
_______________________________________________________________________________

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <xsd:annotation>
  <xsd:documentation>
   Schema for output of the following SQL statement:

   <![CDATA[select u_name from sys_users]]>

  </xsd:documentation>
 </xsd:annotation>

 <xsd:element name="root" type="root__Type"/>

 <xsd:complexType name="root__Type">
  <xsd:sequence>
   <xsd:element name="SYS_USERS" type="SYS_USERS_Type" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="SYS_USERS_Type">
  <xsd:attribute name="U_NAME" type="xsd:string"/>
```

```
 </xsd:complexType>

</xsd:schema>

1 Rows. -- 1843 msec.
SQL> select xml_auto_dtd('select u_name from sys_users', 'root');
callret
VARCHAR
_______________________________________________________________________________

<!-- dtd for output of the following SQL statement:
select u_name from sys_users
-->

<!ELEMENT root (#PCDATA | SYS_USERS)* >
<!ELEMENT SYS_USERS (#PCDATA)* >
<!ATTLIST SYS_USERS
        U_NAME  CDATA   #IMPLIED        >

1 Rows. -- 411 msec.
```

15.4.5. Using XML and Free Text

Virtuoso integrates classic free text retrieval and XML semi-structured query features to offer a smart, scalable XML repository. When a column is declared as indexed XML with the CREATE TEXT XML INDEX statement the text is checked for well-formedness at time of storage. The specific XML structure of the text is also considered when making the free text index entries. This XML-aware free text index is used for processing XPath queries in the *xcontains* SQL predicate. This predicate is only applicable to columns for which there is an XML free text index.

Arbitrary free text criteria can appear inside the XPath expression of *xcontains* . These are introduced by the XPath extension function `text-contains()` , which may only be used within *xcontains* as it relies on the underlying free text index.

Note

`xpath_contains()` does not require the existence of a free text index and can thus apply to any well-formed XML content.

15.4.6. XCONTAINS predicate

This predicate is used in a SQL statement, it returns "true" if a free text indexed column with XML content matches an XPATH expression. Optionally produces the matching node set as a result set.

Syntax

```
xcontains_pred:
        xcontains (column, expr [, result_var [, opt_or_value ...]])

opt_or_value:
          DESCENDING
          | START_ID ',' scalar_exp
          | END_ID ',' scalar_exp
          | SCORE_LIMIT ',' scalar_exp
          | OFFBAND column

result_var:
          IDENTIFIER
          | NULL
```

The *column* must refer to a column for which there exists a free text index.

The *expr* must be a narrow or wide string expression whose syntax matches the rules in 'XPATH Query Syntax'.

The *result_var* variable is a query variable which, if present, will be successively bound to each element of the node set selected by the XPATH expression. if the value is not a node set and is true, the variable will be once bound to this value. The scope of the variable is the containing select and its value is a scalar or an XML entity. The *result_var* can be not an identifier but a NULL keyword to explicitly indicate that no query variable is required.

The *START_ID* is the first allowed document ID to be selected by the expression in its traversal order, e.g. least or equal for ascending and greatest or equal for descending.

END_ID is the last allowed id in the traversal order. For descending order the START_ID must be >= END_ID for hits to be able to exist. For ascending order the START_ID must be <= END_ID for hits to be able to exist.

DESCENDING specifies that the search will produce the hit with the greatest ID first, as defined by integer or composite collation. This has nothing to do with a possible ORDER BY of the enclosing statement. Even if there is an ORDER BY in the enclosing statement the DESCENDING keyword of xcontains has an effect in the interpretation of the STRT_ID and END_ID xcontains options.

RANGES specifies that the query variable following the RANGES keyword will be bound to the word position ranges of the hits of the expression inside the document. The variable is in scope inside the enclosing SELECT statement.

SCORE_LIMIT specifies a minimum score that hits must have or exceed to be considered matches of the predicate.

OFFBAND specifies that the following column will be retrieved from the free text index instead of the actual table. For this to be possible the column must have been declared as offband with the CLUSTERED WITH option of the CREATE TEXT INDEX statement.

If the select statement containing the xcontains predicate does not specify an exact match of the primary key of the table having the xcontains predicate, then the contains predicate will be the 'driving' condition, meaning that rows come in ascending or descending order of the free text document ID. If there is a full equality match of the primary key of the table, this will be the driving predicate and xcontains will only be used to check if the text expression matches the single row identified by the full match of the primary key.

The xcontains predicate may not appear outside of a select statement and may only reference a column for which a free text index has been declared. The first argument must be a column for which there is such an index. The text expression may be variable and computed, although it must be constant during the evaluation of the select statement containing it.

The xcontains predicate must be a part of the top level AND of the WHERE clause of the containing select. It may not for example be a term of an OR predicate in the select but can be AND'ed with an OR expression.

Example 15.13. Selecting Title Elements Called 'Key'

```
select xt_file from xml_text2 where
 xcontains (xt_text, '//title = "Key"');
```

The query retrieves the *xt_file* for rows whose *xt_text* is an XML document containing 'Key' as the text value of a title element.

If not all values in *xt_text* are valid XMLs then '__quiet' option can be useful to disable error signalling. It is unusual to get an incorrect XML stored in a column that has free text XML index because both on insert and on update the text is parsed by an free text indexing routine, but the error is possible if e.g. a non-standalone document is stored and an important external entity was available at indexing time but disappeared later. Thus a modified example might be better for a column with non-standalone documents;

```
select xt_file from xml_text2 where
 xcontains (xt_text, '[__quiet] //title = "Key"');
```

Selecting Title Element that Contains a Specified Text

```
select n from xml_text2 where
 xcontains(xt_text,
 '//title[. = "AS Declaration - Column Aliasing"]',0,n);
```

The query retrieves each title element from each row of *xml_text2* where the *xt_text* contains title elements with the text value "AS Declaration - Column Aliasing."

Note

The equality test is case- and whitespace-sensitive, as normal in XPath. The free text index is used for the search but the final test is done according to XPath rules.

See Also:
The CONTAINS Predicate.

15.4.7. text-contains XPath Predicate

```
text-contains (node-set, text-expression)
```

This XPath predicate is true if any of the nodes in *node-set* have text values matching the *text-expression*. The *text-expression* should be a constant string whose syntax corresponds to the top production of the free text syntax for patterns in `contains()`. The string also may not consist exclusively of spaces or noise words.

See Also:
"Noise Words" in the Free Text Search chapter .

Example 15.14. Selecting All Titles About Aliasing

```
select n from xml_text2 where
  xcontains (xt_text,
  '//title[text-contains (., "Aliasing")]', 0, n);
```

This selects all title elements that contain the word "Aliasing" using free text match rules: case insensitive and whole word.

Example 15.15. Select All Trees with Elements Containing "sql reference"

```
select n from xml_text2 where
  xcontains (xt_text,
  '//*[text-contains (., '"sql reference"')]',
  0, n);
```

This selects all elements whose text value contains the phrase "sql reference". Free text matching rules apply. This produces all nodes in document order for all documents which contains the phrase, starting with the document node and following downward including all paths to the innermost element(s) whose text contains the phrase.

15.4.8. XML Free Text Indexing Rules

XML documents are inserted into the free text index as follows:

The process works on the parsed XML tree; therefore character and local entity references are expanded.

Whole words of text content, bounded by delimiters used for free text, are each assigned an ordinal number. Noise words defined in the noise.txt file used by free text indexing are not counted.

Attribute names and values are not indexed.

Element start and end tags are indexed using the expanded names - that is, prefixed with the namespace URI + ':'.

An element start tag's ordinal number is one less than the ordinal number of the first whole word in the text value.

A close tag's ordinal number is one greater than that of the last word in the text value.

From these rules follows that:

```
<html>
  <body>
   <title>Title of Document</title>
   <p>Some <b>bold</b> text </p>
  </body>
</html>
```

will be indexed as follows:

```
<html>          0
<body>          0
<title>         0
```

```
Title           1
of              - no number, noise word
Document                2
</title>                3
   <p>          3
Some            4
 <b>            4
bold            5
</b>            6
 text           6
</p>            6
  </body>               6
</html>         6
```

As a result, the phrase "some bold text" is the string value of the <p> tag and will match the free text expression "some bold text" even though there is mark-up in it. Conversely, the phrase "Document some bold" does not match. Words will not considered adjacent if there is a mix of opening and closing tags. They will only be considered adjacent if there are solely one or more either opening or closing tags between them. This can be circumvented by using the *NEAR* connective instead of the phrase construct.

A free text condition will only be true of an element if all the words needed to satisfy the condition are part of the element's string value. This string value includes text children of descendants.

15.4.9. XML Processing & Free Text Encoding Issues

XML document may be written in a variety of encodings, and it may cause errors if an incorrect encoding is used for reading a document. Most common errors can easily be eliminated by writing proper XML prologs in documents, but this is not always possible, e.g. if documents are composed by third-party applications. Virtuoso provides various tools to support different types of encodings and to specify encodings to use if a given document has no XML prolog.

Encodings: The Difference Between Encodings & Character Sets

Not all documents may be converted to Unicode by using simple character sets. Some of them are stored in so-called "multibyte" encodings. It means that every letter (or ideograph) is represented as a sequence of one or more bytes, not by exactly one byte. The conversion from such representation to Unicode and back is usually significantly slower than simple transformation via character sets, so these representations are supported by data import operations only, but not by internal RDBMS routines.

The Virtuoso Server "knows" some number of built-in encodings, such as UTF-8, UTF-16BE and UTF-16LE. It can load additional encoding descriptions from a "UCM" file, and can automatically create a new encoding from a known charset with the same name. See the UCM Encodings section for more details.

An encoding may be used in the following places:

The XML/HTML parser to convert source text to Unicode.

The free-text indexing engine to convert plain-text or XML documents to Unicode during the indexing.

It may be used by the compiler of free-text search expressions to convert string constants of the expression to Unicode.

It may be used to convert string constants of XPath/XQuery expressions.

You can only use character sets, not encodings as an ODBC connection character set, as a character set attribute of a column of a database table, as an output encoding of the built-in XSLT processor (it is for future versions). UTF-8 is an exception, it is supported in many places where other encodings are not.

Security Note:

Two strings converted to Unicode may be identical, but this does not guarantee that their source strings were equal byte-by-byte due to the nature of some encodings. For this reason you should avoid processing authorization data that are neither in Unicode nor in one of the standard character sets (single-byte encodings). Multibyte encodings and user-defined character sets may be unsafe for such purposes.

UCM Encodings

The description of a multibyte encodings is much longer than the description of a character set. It is inconvenient to keep such amounts of data inside the executable. Virtuoso can load descriptions of required encodings from external files in UCM format. Every UCM file describes one encoding.

Virtuoso loads UCM files at system initialization. The list of UCM files is kept in the Virtuoso INI file under a section called [Ucms]. This section should contain a UcmPath parameter and one or more parameters with names Ucm1, Ucm2, Ucm3 and so on (up to Ucm99).

The UcmPath parameter specifies the directory where UCM files are located, and every UcmNN parameter specifies the name of a UCM file to load and a list of names that the encoding can be identified by the <?xml ... encoding="..." ?> XML preamble. A vertical bar character is used to delimit names in the list.

Example 15.16. Sample [Ucms] Section

```
[Ucms]
UcmPath = /usr/local/javalib/ucm
Ucm1 = java-Cp933-1.3-P.ucm,Cp933
Ucm2 = java-Cp949-1.3-P.ucm,Cp949|Korean
```

This section describes two UCM files located in /usr/local/javalib/ucm directory: data from java-Cp933-1.3-P.ucm will be used for documents in the 'Cp933' encoding; data from java-Cp949-1.3-P.ucm will be used for documents in the 'Cp949' encoding and for documents in the 'Korean' encoding (because these two names refers to the same encoding).

Note:

The encoding name specified inside the UCM file itself is not used.

The Virtuoso server will log the results of processing each UCM file specified in the Virtuoso INI file. If a UCM file specified is not found or contains syntax errors, the error is logged, otherwise only the type and name(s) of the encoding are logged.

Note:

If the virtuoso.ini contains a misspelled name of a parameter or section, the parameter (or a whole section) is ignored without being reported as an error. It is always wise to verify that the log contains a record about the encoding(s) you load.

See Also:

UCM files can be found freely from various sites concerning the "International Components for Unicode" project, such as: IBM ICU Homepage or the IBM UCM files directory .

The C Interface chapter contains further information regarding user customizable support for new encodings and languages. For almost all tasks, it is enough to define a new charset or to load an additional UCM file, but some special tasks may require writing additional C code.

The *Encoding* Attribute

If an XML document contains the *encoding* parameter in its

```
<?xml ... ?>
```

prolog declaration, it will be properly decoded and converted into UTF-8, so the application code is free from encoding problems. If the value of this attribute is the name of a pre-set or user-defined character set, that character set will be used. Virtuoso will recognize names such as *UTF-8* and *UTF8* as multi-character or special encodings. Virtuoso recognizes both official names and aliases.

If an encoding is not specified in an XML prolog, or if the document contains no prolog, the default encoding will be used to read the document. If a built-in SQL function invokes the XML parser, it will have an optional argument *parser_mode* to specify whether source text should be parsed as strict XML or as HTML. If the source text is 8-bit, then UTF-8 will be used as the default encoding for "XML mode", and ISO-8859-1 (Latin-1) will be the default for "HTML mode". If the source text is of some wide-character type, Unicode is the default. To make another encoding the default, you may specify its official or alias name as the *content_encoding* argument of a built-in function you call.

Encoding in XPath Expressions

Sometimes applications should perform XPath queries using the encoding specified by a client. For example, a search engine may ask a user to specify a pattern to search and use the browser's current encoding as a hint to parse the pattern properly. In such cases you may wish to use the *__enc* XPath option to specify the encoding used for the rest of XPath string:

Example 15.17. Specifying Search Encodings in XPath

Create a sample table and store an XML with non-Latin-1 characters

```
create table ENC_XML_SAMPLE (
  ID integer,
  XPER long varchar,
  primary key (ID)
);

insert into ENC_XML_SAMPLE (ID, XPER)


values (
  1,
  xml_persistent ('<?xml version="1.0" encoding="WINDOWS-1251" ?>
    <book><cit>Îí äîáàâëë
      êàðòîøêè,
    ïîñîëèë è
    ïîñòàâèë
    àêâàðèóì íà
    îãîíü
    (Ì.Æâàíåöêèé
    )</cit></book>'
  )
);
...
```

Find the IDs of all XML documents whose texts contain a specified phrase. Note that there are pairs of single quotes (not double quotes) around *KOI8-R* . The encoding name should be in single quotes, but because it is inside a string constant the quotes must be duplicated.

```
select ID from ENC_XML_SAMPLE where
  xcontains (XPER, '[__enc ''KOI8-R''] //cit[text-contains(.,
  "''ÐÏÓÔÁ×ÉÌ
    ÁË×ÁÒÉÕÍ
    ÎÁ ÏÇÏÎØ''")]');
```

Encoding in Free Text Search Indexes & Patterns

Like XML applications, free text searching may have encoding problems, and Virtuoso offers a similar solution for them.

Both the CREATE TEXT INDEX statement and vt_create_text_index() Virtuoso/PL procedure have an optional argument to specify the encoding of the indexed data. The specified encoding will be applied to all source text documents (if the TEXT INDEX was created), or to all XML documents that have no encoding attribute of the sort <?xml ... encoding="..." ?> (if the TEXT XML INDEX was created).

The option __*enc* may be specified at the beginning of free text search pattern, even if the pattern is inside an XPath statement:

Example 15.18. Specifying an Encoding for Free Text Searching

Create a sample table and store a sample of text with non-Latin-1 characters (assuming that client encoding is Windows-1251)

```
create table ENC_TEXT_SAMPLE (
  ID integer,
  TEXT long nvarchar,
  primary key (ID)
);

insert into ENC_TEXT_SAMPLE (ID, XPER)
```

```
values (
  1,
  '<?xml version="1.0" encoding="WINDOWS-1251" ?>
Îí äîáàâëë
    êàðòîøêè,
    ïîñìîòðè è
    ïîñòàâëë
    àêâàðèóì
    íà îãîíü
    (È.Æâàíåöêèé')
);
...
```

Find the IDs of all text documents whose texts contain a specified phrase.

```
select ID from ENC_SAMPLE where
  contains (TEXT, '[__enc ''KOI8-R'']
    "ÐÏÓÔÁ×ÉÌ
    ÁË×ÁÒÉÕÍ
    ÎÁ ÏÇÏÎØ"'
  );
```

Encoding may be applied locally to an argument of the text-search predicate. It may be used if the document contains citations in different encodings or if the XML document contains non-ASCII characters in names of tags or attributes, or if the encoding affects character codes of ASCII symbols such as '/' or '['.

```
select ID from ENC_XML_SAMPLE where
  xcontains (XPER, '//cit[text-contains(., "[__enc ''KOI8-R'']
    ''ÐÏÓÔÁ×ÉÌ
    ÁË×ÁÒÉÕÍ ÎÁ
    ÏÇÏÎØ''")]'
  );
```

Note:
You may have free-text a expression written as a literal constant: e.g. if the argument of text-contains XPath function is a literal constant. Be careful to not declare the __enc twice, once in the beginning of the whole XPath expression and then again in the beginning of the free-text expression constant, because words of the text expression will thus be converted twice.

15.5. Using UpdateGrams to Modify Data

Updategrams allow database updates to be defined as XML. This is ultimately achieved by mapping the XML nodes against corresponding database columns. Updategrams can be used to replace existing data access components in a middle tier. A typical application will include a middle tier consisting of business logic and data access code. The data access code will interact with the database using disconnected recordsets and command objects calling stored procedures. Most of the data access section of the middle tier can be replaced with updategrams.

Most data access tiers (both middle tier code and stored procedures) will deal individually with specific database tables or groups of related tables. This can inhibit performance and often several round trips to the database are required to complete a transaction. Updategrams solve this problem by including all the data in an XML document that is then mapped to database tables and columns. The entire database update can then be accomplished at once. This update can include inserting, updating and deleting data.

The `xmlsql_update()` function supports XML-based insert, update, and delete operations performed on an existing table in the database. `xmlsql_update()`

15.5.1. Updategrams Basics

The general format of an updategram is:

```
<sql:sync xmlns:sql="xml-sql">
 <sql:before>
    <TABLENAME [sql:id="value"] col="value" col="value"?../>
 </sql:before>
```

```
 <sql:after>
    <TABLENAME [sql:id="value"] [sql:at-identity="value"]
      col="value" col="value"?../>
 </sql:after>
</sql:sync>
```

or

```
<sql:sync xmlns:sql="xml-sql">
        <sql:before>
                <TABLENAME [sql:id="value"]>
                   <col>"value"</col>
                   <col>"value"</col>
                   ...
                </TABLENAME>
                ...
        </sql:before>
        <sql:after>
                <TABLENAME [sql:id="value"] [sql:at-identity="value"]>
                   <col>"value"</col>
                   <col>"value"</col>
                   ...
                </TABLENAME>
                ...
        </sql:after>
</sql:sync>
```

15.5.2. Elements Description

The *<sync>* tag of the updategram signifies the beginning of an operation(s) The rows specified in the *<before>* refer to existing records in the database. The rows specified in the *<after>* block refer to what the user wants in the database. *<TABLENAME.../>* identifies target table.

The *sql:at-identity* attribute stores the last identity value added by the system (if possible). This identity value can then be used in subsequent operations.

The *sql:id* attribute is used to mark rows. This forces an association between the record specified in the *<before>* and *<after>* block in the update gram. When there are multiple instances specified, it is recommended that *sql:id* attribute be used for all the instances.

Each *<TABLENAME.../>* refers to a single table. Multiple *<TABLENAME.../>* entries are allowed in the same *<before>* or *<after>* tags, or in both *<before>* and *<after>* tags; however, nesting is not allowed. The *<before>* and *<after>* tags are optional. A missing tag is the same as having a tag with no content.

15.5.3. Determining Actions

If only the *<after>* block is specified, the rows specified in the *<after>* block are inserted in the table(s). If both the *<before>* and *<after>* blocks are specified, then rows specified in the *<after>* block for which there are no corresponding rows in the *<before>* block are inserted in the table(s).

In an update operation, the rows specified in the *<before>* block refer to existing rows in the database. The corresponding rows in the *<after>* block reflect what the user wants in the database. A row update operation is performed if there is a row in both the *<before>* and *<after>* sections with the same set of values for the attributes that uniquely identify a row in a table. Rows specified in the *<before>* block must be valid in the database for the updategram to successfully update the rows.

In a delete operation, if only the *<before>* block is specified in the update gram, the rows specified in the *<before>* block are deleted from the table(s). If both the *<before>* and *<after>* blocks are specified, the rows for which there are no corresponding rows in the *<after>* block are deleted from the table(s).

15.5.4. Using Input Parameters

Parameters declarations should be described in the <header> section of the updategram. There should be one *<param>* row for each parameter.

General syntax:

```
<sql:header xmlns:sql="xml-sql">
  <sql:param name="PARAM_NAME" [default="DEFAULT_VALUE"]/>
  ...
</sql:header>
```

Where *PARAM_NAME* is the name of the parameter and *DEFAULT_VALUE* is optional default of parameter Parameters in updategram should have *$PARAM_NAME* instead of a value. On processing, Virtuoso replaces *$PARAM_NAME* with the corresponding value from the *<input_parameters>* given to the function `xmlsql_update()` .

15.5.5. Examples

Given the following tables:

```
CREATE TABLE Orders (
    OrderID int identity,
    CustomerID varchar(10),
    EmpID int,
    PRIMARY KEY (OrderID));

CREATE TABLE OrderDetails (
    OrderID int,
    ProductID int,
    Quantity int);
```

A. Update Gram to Insert a Record

```
xmlsql_update (xml_tree_doc (xml_tree (
'<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql">
<sql:sync>
  <sql:after>
    <Orders CustomerID="TEST" EmpID="99"/>
  </sql:after>
</sql:sync>
</ROOT>')));
```

B. Updategram with an *at-identity* Attribute

```
xmlsql_update (xml_tree_doc (xml_tree (
'<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql">
<sql:sync>
  <sql:after>
    <Orders sql:at-identity="x" CustomerID="VINET" EmpID="10"/>
      <OrderDetails OrderID="x" ProductID="1" Quantity="50"/>
      <OrderDetails OrderID="x" ProductID="2" Quantity="20"/>
    <Orders sql:at-identity="x" CustomerID="HANAR" EmpID="11"/>
      <OrderDetails OrderID="x" ProductID="1" Quantity="30"/>
      <OrderDetails OrderID="x" ProductID="4" Quantity="25"/>
  </sql:after>
</sql:sync>
</ROOT>')));
```

C. Updategram to Delete a Record

```
xmlsql_update (xml_tree_doc (xml_tree (
'<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql">
<sql:sync>
  <sql:before>
    <Orders CustomerID="HANAR" EmpID="11"/>
  </sql:before>
</sql:sync>
</ROOT>')));
```

D. Updategram to Update a Record

```
xmlsql_update (xml_tree_doc (xml_tree (
'<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql">
<sql:sync>
```

```
  <sql:before>
    <Orders sql:id="1" CustomerID="VINET" EmpID="10"/>
  </sql:before>
  <sql:after>
    <Orders sql:id="1" CustomerID="VINET_NEW" EmpID="11"/>
  </sql:after>
</sql:sync>
</ROOT>')));
```

E: Using a different syntax for updategrams - entities in place of attributes - example D can be transformed to:

```
xmlsql_update (xml_tree_doc (xml_tree (
'<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql">
<sql:sync>
  <sql:before>
    <Orders sql:id="1">
      <CustomerID>VINET</CustomerID>
      <EmpID>10</EmpID>
    </Orders>
  </sql:before>
  <sql:after>
    <Orders sql:id="1">
      <CustomerID>VINET_NEW</CustomerID>
      <EmpID>11</EmpID>
    </Orders>
  </sql:after>
</sql:sync>
</ROOT>')));
```

Note that two syntaxes cannot be mixed in one document.

F: Using input parameters

Assume the following table:

```
CREATE TABLE Shippers(
  ShipperID INTEGER,
  CompanyName VARCHAR(40),
  Phone VARCHAR(24),
  PRIMARY KEY (ShipperID));

xmlsql_update (xml_tree_doc (xml_tree (
'<DocumentElement xmlns:sql="urn:schemas-microsoft-com:xml-sql">
    <sql:header>
      <sql:param name="ShipperID" default="2"/>
      <sql:param name="CompanyName" default="United Package New"/>
      <sql:param name="Phone" default="(503) 555-3199 (new)"/>
    </sql:header>
    <sql:sync>
        <sql:before>
        </sql:before>
        <sql:after>
            <Shippers sql:id="1" ShipperID="\$ShipperID"
            CompanyName="\$CompanyName" Phone="\$Phone"/>
        </sql:after>
    </sql:sync>
</DocumentElement>')),
    vector ('ShipperID','10','CompanyName','DHL','Phone','+359 32 144'));
                -- <- this is a array with input parameters
```

This will add one record to the Shippers table with the data in the array. Note that the slash/dollar sign pair '\$' transforms to dollar sign '$' only

15.6. XML Templates

Virtuoso XML templates allow execution of queries over HTTP in order to achieve any combination of the following:

Execution of SQL queries returned in an XML formatted resultset.

Execution of XQuery and XPath queries against XML and SQL-XML documents in the Virtuoso WebDAV repository.

Use an XML notation (Updategrams) to insert/update/delete SQL data.

An XML template is an XML file containing a query, optional parameters with default values for the query, a place to specify an XSL stylesheet, and a section for specifying updatagram based synchronization metrics. They are meant to be executed as an XML described short-cut to an query result, an XML document. The XML document returned from calling an XML template can be served raw, or transformed using XSLT.

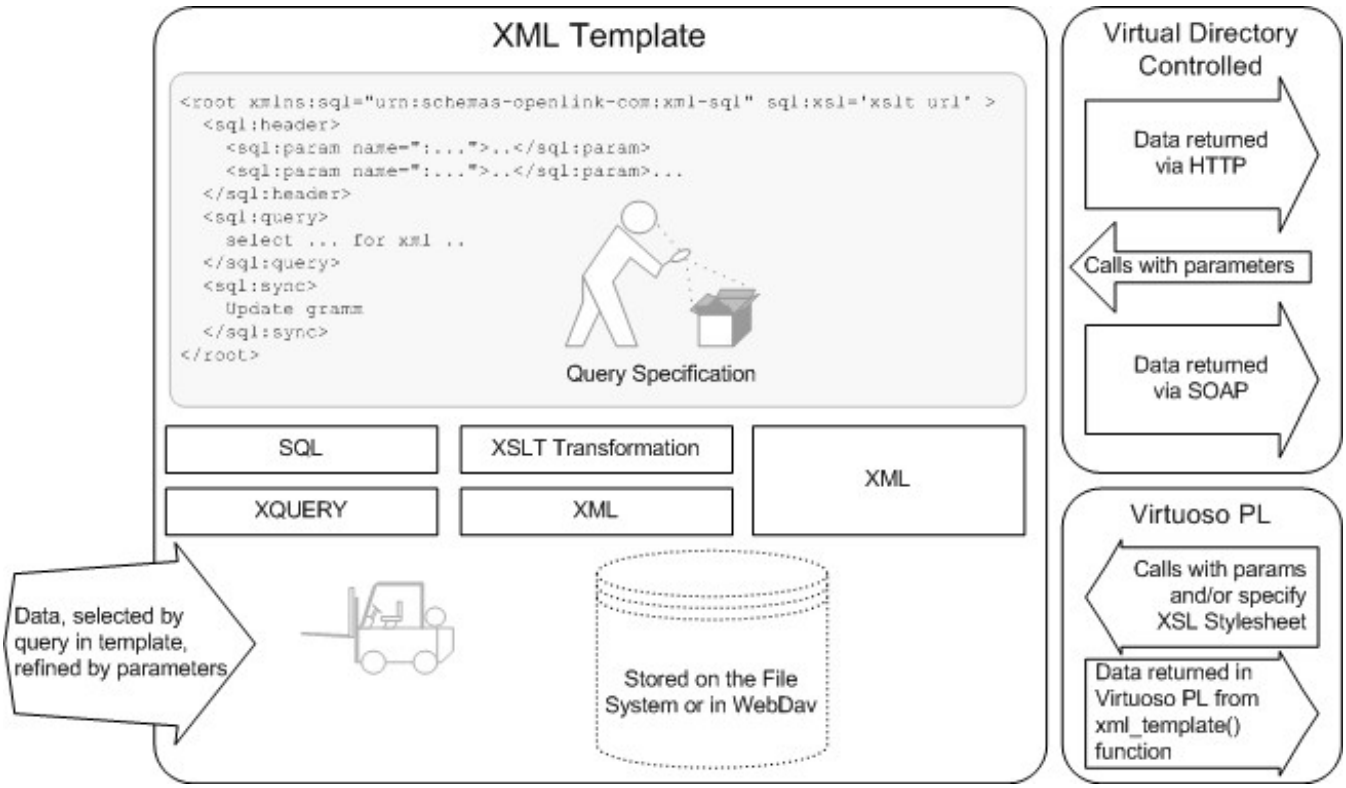
XML templates can be executed from within Virtuoso procedure language using the `xml_template()` function, or published as SOAP compliant XML Web Services, which makes them accessible to any SOAP and WSDL aware environment.

XML templates provide quick easy access to results from a SQL query as usual, but now this can be saved to a file. The results are not saved, just the query definition. You can use this feature to rapidly produce dynamic reports that can potentially be rendered in different ways by providing an alternate stylesheet. The report can be refined on the fly by providing parameters for the query. The output is reachable via HTTP directly by providing the URL to the template.

When XML templates are executed via HTTP, the XSLT transformation will be performed automatically if the "sql:xsl" attribute is specified in the templates root element. This means that the default XSLT transformation cannot be avoided if the template is executed over HTTP. However, you have the option whether to apply the transformation or not when using the `xml_template()` function. XML templates containing <sql:query> elements return either results or messages regarding the queries. An XSLT transformation can be made on the result from `xml_template()` using the returned XSLT sheet URL. Hence it is conditional in PL programming. The application developers can choose to either use the style sheet specified in the template, use another style sheet, or skip XSLT transformation entirely. This feature provides full flexibility for the procedure developer.

XML templates are intended for execution over HTTP. The template file can be stored either on the file system, WebDAV repository or stored on another HTTP server being referenced by a URL parameter. Specifying a template as a URL parameter naturally introduces a new potential security risk associated with the template owners web server.

Figure 15.1. Conceptual View of XML Templates



Queries and updategrams can be parameterized. The parameters must be defined in the header element, which consists of zero or more param elements. The default value of the parameter is enclosed within the param element whose values are typically replaced during the execution phases. When XML templates are executed from the `xml_template()` function the parameters are specified in a vector as an input parameter. When XML templates are executed via HTTP the parameters are contained in the URL.

Important

All <sql:param> parameter definitions in the template for SQL queries need to be named parameters, and the names must be preceded by a colon i.e. :ParamName (note colon at the front)
. The colon is not required for updategrams.

If an error occurs as a result of executing a SQL query or updategram, the comment will be included in the result detailing the error. The subsequent queries and updategrams will still be executed.

See Also:

The Exposing Persistent Stored Modules as Web Services section as this also describes how XML Templates can be published to web services as a SOAP messages using a PL Wrapper. This is achieved using the Administrative Interface.

`xml_template()`

15.6.1. Syntax

The format of an XML template is shown below, illustrating how SQL queries and updategrams are specified:

```
<root xmlns:sql="urn:schemas-openlink-com:xml-sql" sql:xsl='xslt url' >
  <sql:header>
    <sql:param name=":...">..</sql:param>
    <sql:param name=":...">..</sql:param>...
  </sql:header>
  <sql:query>
    select ... for xml ..
  </sql:query>
  <sql:sync>
    Update gramm
  </sql:sync>
</root>
```

The root element can contain two attributes. The first is the required namespace declaration of xmlns:sql="urn:schemas-openlink-com:xml-sql", required to identify the XML as an XML template. The second attribute allows the specification of an optional XSLT stylesheet to be applied to the resulting XML document, if applicable. The XSLT stylesheet file can be specified as either a relative or absolute URL.

The following elements can be defined in the "sql" namespace for an XML Template:

- The

  "root"

  tag can be any name, its purpose is to provide the top-level element required for any well-formed XML, and provides the root element for the resulting XML document, if any.
- <*sql:header*>

  This tag is used to hold the parameters for templates execution such as <sql:param> elements. The <sql:header> element allow us to define multiple parameters.
- <*sql:param*>

  This element is used to define a named parameter that is passed to the query and/or updategram inside the template. Each <xsl:param> element defines one parameter. Multiple <xsl:param> elements can be specified within the <sql:header> element.
- <*sql:query*>

  This element is used to specify SQL/XML queries. It is possible to specify multiple <sql:query> elements within a single template.
- <*sql:sync*>

  This element is used to contain updategrams.

<sql:query> and <sql:sync> entries are executed sequentially in the order they appear as separate transactions. Therefore, if you have multiple <sql:query> or <sql:sync> elements in the template, if one fails, the others will proceed.

XML templates can be executed directly from Virtuoso PL using the `xml_template()` function.

To allow XML template execution from a Virtuoso virtual directory and it's descendants the special option: 'xml_templates' must be used in the virtual directory definition. This option can be set from the Conductor / Web Application Server / Virtual Domains & Directories or from SQL (or ISQL) using the `vhost_define(.... , opts=>vector('xml_templates', 'yes'));` function. In the usual way, the SQL user specified for VSP execution within the virtual directory definition will be used for executing the templates within such web directories. If your XML Templates are to run from a DAV directory then you must allow suitable execute permissions for the file and directory.

When a virtual directory is configured to allow execution of XML templates be aware that this also means that descendant directories inherit this property. Also be aware that this also allows execution of VSP files in the normal way. WebDAV also has another layer of security attributes that will need to be set to enable files to be executable. By default files in DAV do not have execute privileges.

Explicitly specifying "contenttype=text/html" as a URL parameter will instruct the Virtuoso HTTP server to return the HTML output.

See Also

Functions: `xml_template()` , `xmlsql_update()` , `xml_auto()` .

UpdateGrams .

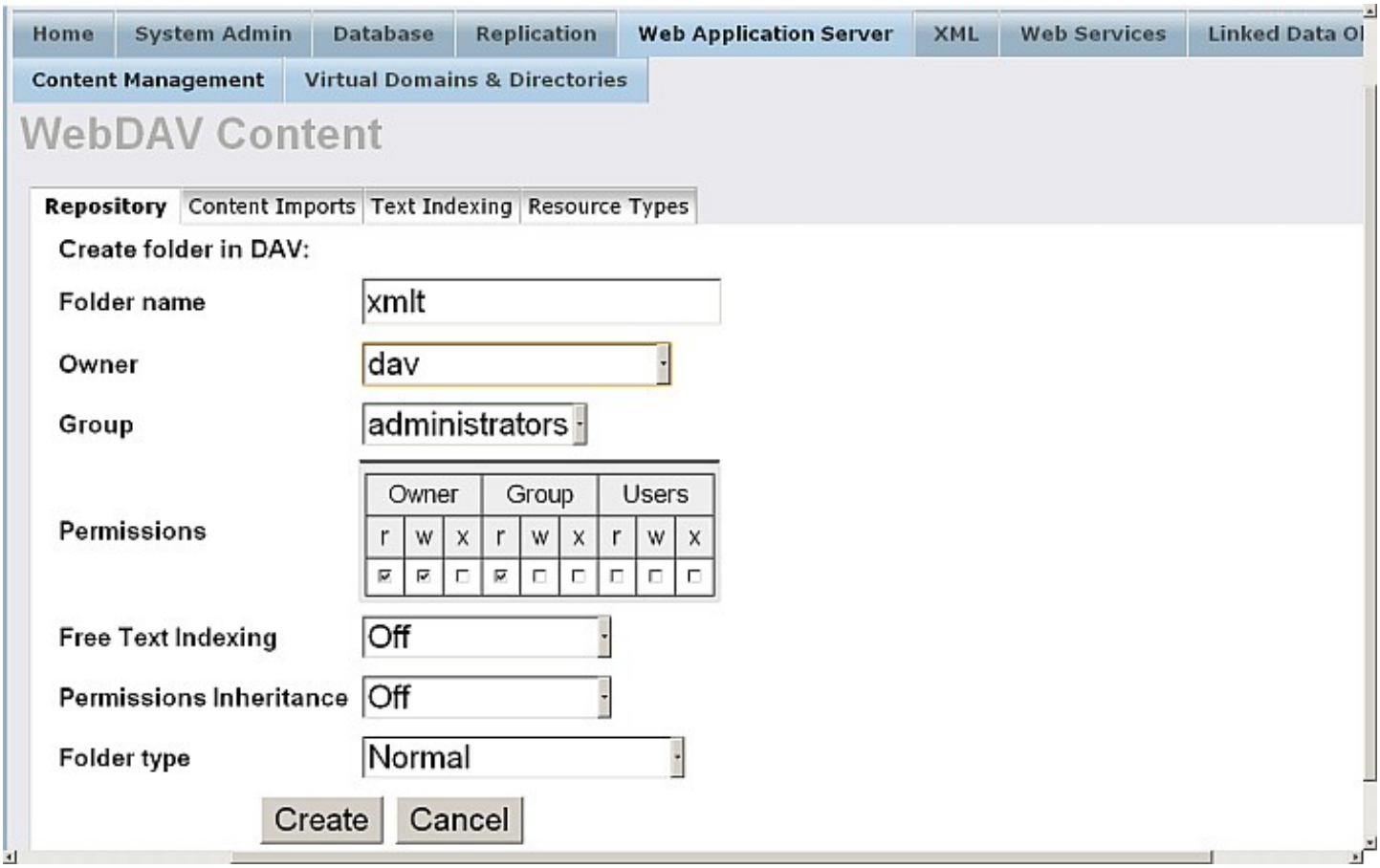
The FOR XML clause .

15.6.2. Saving SQL Queries to XML Template

Using the Conductor Administration Interface, we are going to make and save a query to an XML template in WebDav, and then demonstration its use from a browser.

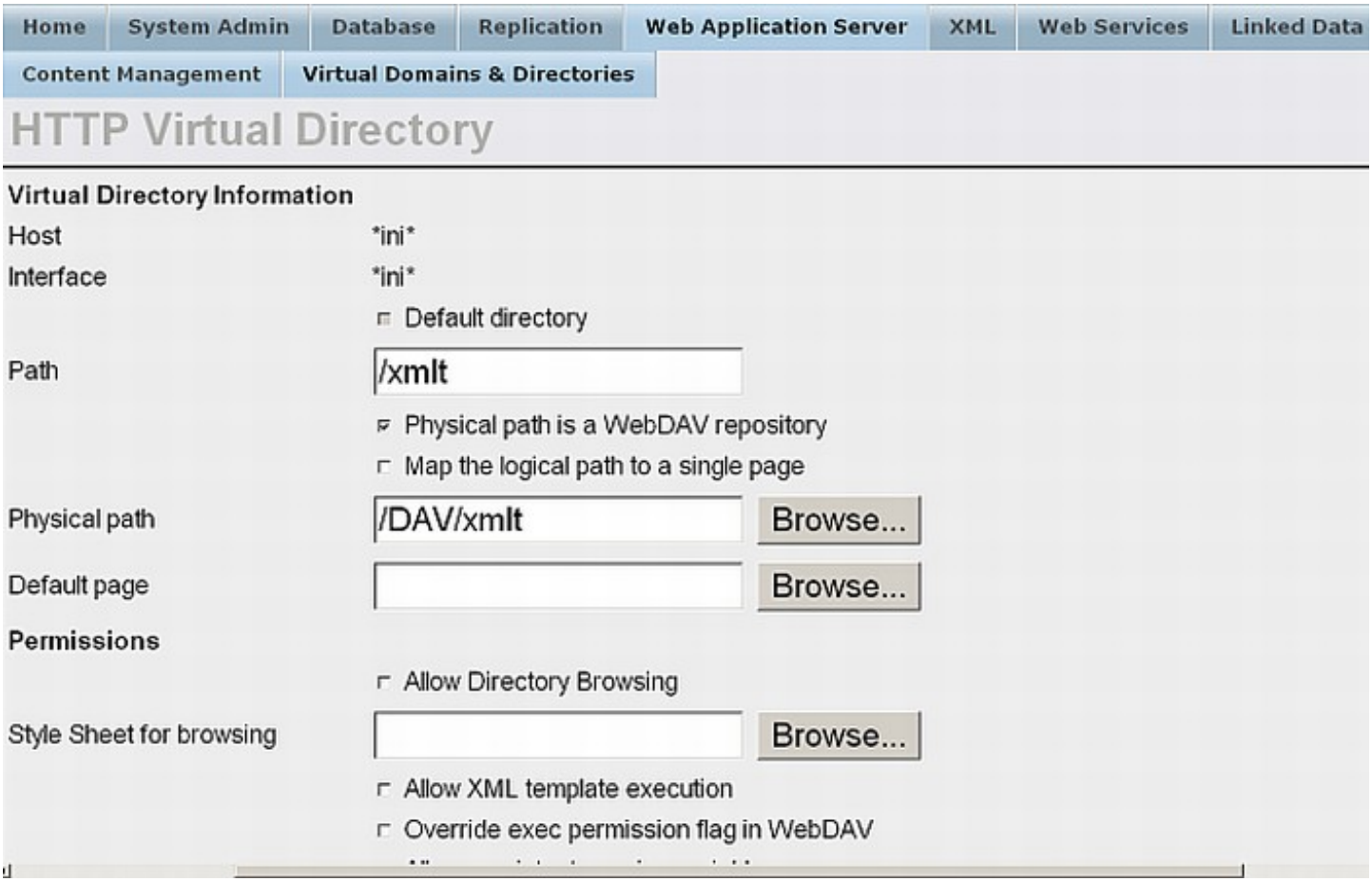
First we will make a new WebDAV directory. From Web Application Server -> Content Management -> Repository click the "New folder" icon and in the shown form enter for "Folder Name": xmlt. Choose for "Owner" dav and click the "Create" button.

Figure 15.2. Creating a new directory (collection) in WebDAV



Next we need to configure a virtual directory mapping to this so that XML Templates can be executed later. Go to Web Application Server / Virtual Domains & Directories and for your {Default Web Site} click the link "New Directory". In the shown form click "Next".

Figure 15.3. Configuring a Virtual Directory to respond to XML Template requests from our Dav



The programmatic equivalent for creating a virtual directory is the command in iSQL:

```
SQL>vhost_define(lpath=>'/DAV/xmlt',
                 ppath=>'/DAV/xmlt/',
                 is_dav=>1,
                 vsp_user=>'demo',
                 opts=>vector('xml_templates', 'yes'));
```

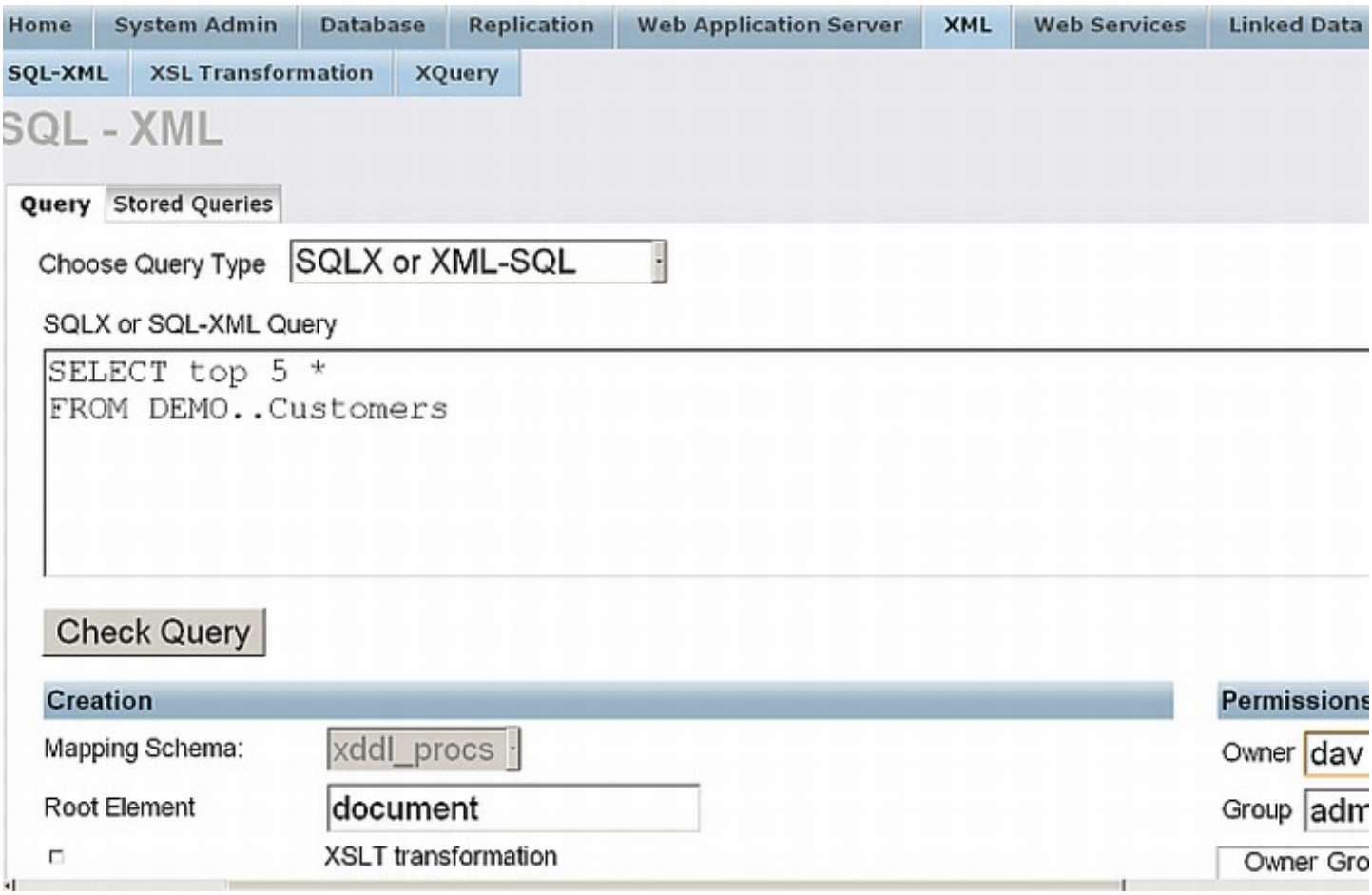
Now we go to Database/Interactive SQL and will use a simple query. Here we are assuming that the Demo database is being used, so we will query the Customers table:

Figure 15.4. A SQL Query

Figure 15.5. Results

Now go to XML -> SQL-XML and enter in the query area the sql statement from above. Here we can supply some "FOR XML" clause, "FOR XML AUTO" being the simplest. Enter in the field "WebDAV resource path for the result" the directory we created earlier, i.e. it should be /DAV/xmlt and add the name of the xmltemplate to which the result will be saved, for example: sql-template.xml So finally the value should be `/DAV/xmlt/sql-template.xml` .

Figure 15.6. The Save XML Template page and settings



Once the template has been saved we can test it. Point your browser at you newly created file, a URL similar to: `http://host:port/DAV/xmlt/sql-template.xml` :

Figure 15.7. The test results: http://host:port/DAV/xmlt/sql-template.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <document>
  <Customers CustomerID="ALFKI" CompanyName="Alfreds Futterkiste" ContactName="Maria Anders" ContactTitle="Sales
    Representative" Address="Obere Str. 57" City="Berlin" Region="" PostalCode="12209" Country="Germany" CountryCode="gm"
    Phone="030-0074321" Fax="030-0076545" ROWGUID="CD83B720-1FEC-11DC-8A76-D44C76F22C3E" />
  <Customers CustomerID="ANATR" CompanyName="Ana Trujillo Emparedados y helados" ContactName="Ana Trujillo"
    ContactTitle="Owner" Address="Avda. de la Constitución 2222" City="México D.F." Region="" PostalCode="05021" Country="Mexico"
    CountryCode="mx" Phone="(5) 555-4729" Fax="(5) 555-3745" ROWGUID="CD853E50-1FEC-11DC-8A76-D44C76F22C3E" />
  <Customers CustomerID="ANTON" CompanyName="Antonio Moreno Taquería" ContactName="Antonio Moreno" ContactTitle="Owner"
    Address="Mataderos 2312" City="México D.F." Region="" PostalCode="05023" Country="Mexico" CountryCode="mx" Phone="(5) 555-
    3932" Fax="" ROWGUID="CD853E51-1FEC-11DC-8A76-D44C76F22C3E" />
  <Customers CustomerID="AROUT" CompanyName="Around the Horn" ContactName="Thomas Hardy" ContactTitle="Sales
    Representative" Address="120 Hanover Sq." City="London" Region="" PostalCode="WA1 1DP" Country="United Kingdom"
    CountryCode="uk" Phone="(171) 555-7788" Fax="(171) 555-6750" ROWGUID="CD853E52-1FEC-11DC-8A76-D44C76F22C3E" />
  <Customers CustomerID="BERGS" CompanyName="Berglunds snabbköp" ContactName="Christina Berglund" ContactTitle="Order
    Administrator" Address="Berguvsvägen 8" City="Luleå" Region="" PostalCode="S-958 22" Country="Sweden" CountryCode="sw"
    Phone="0921-12 34 65" Fax="0921-12 34 67" ROWGUID="CD853E53-1FEC-11DC-8A76-D44C76F22C3E" />
</document>
```

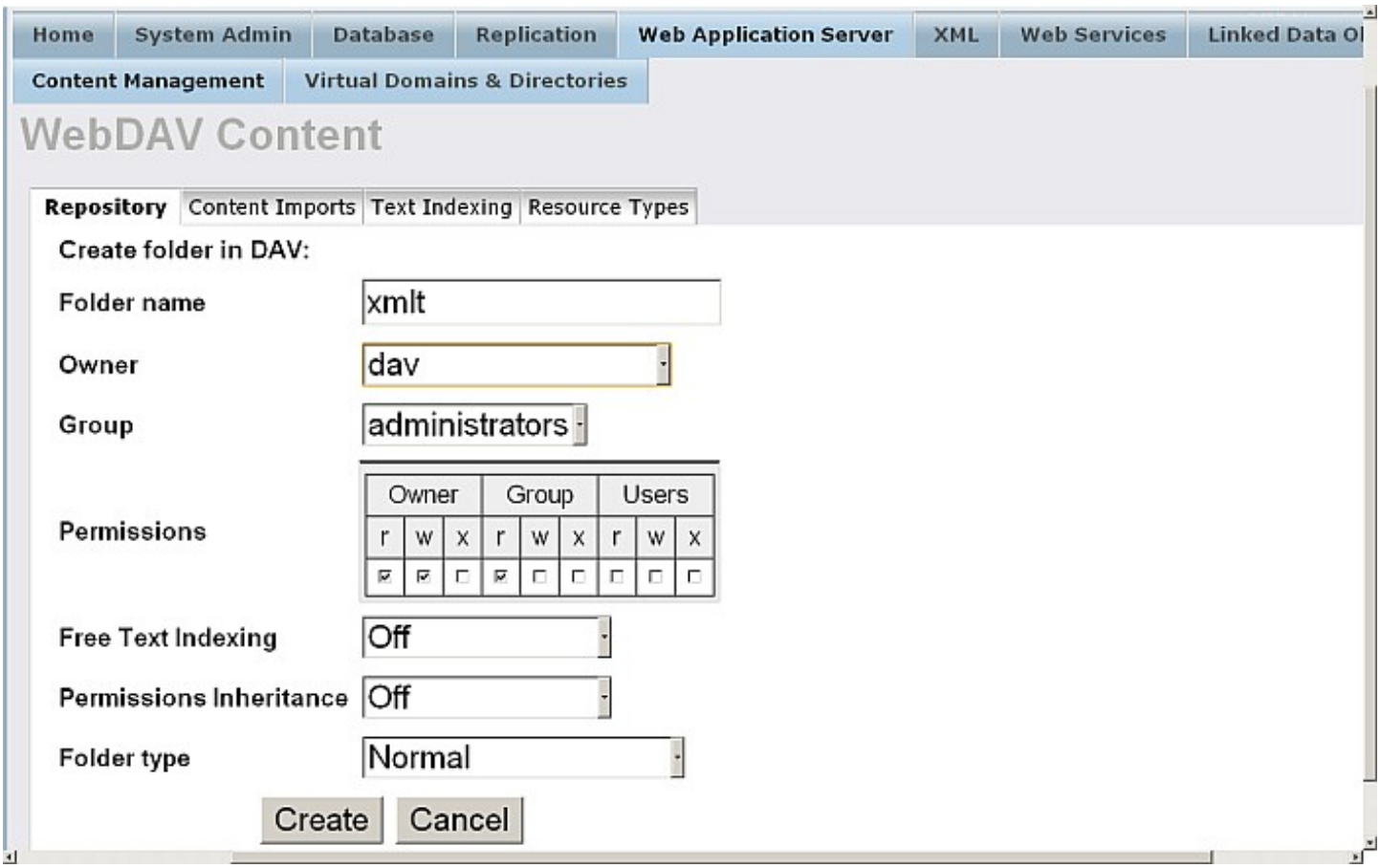
See Also:
Exposing XML Templates as Web Services .

15.6.3. Saving XQUERY Queries to XML Template

Using the Conductor Administration Interface, we are going to make and save a query to an XML template in WebDav, and then demonstration its use from a browser.

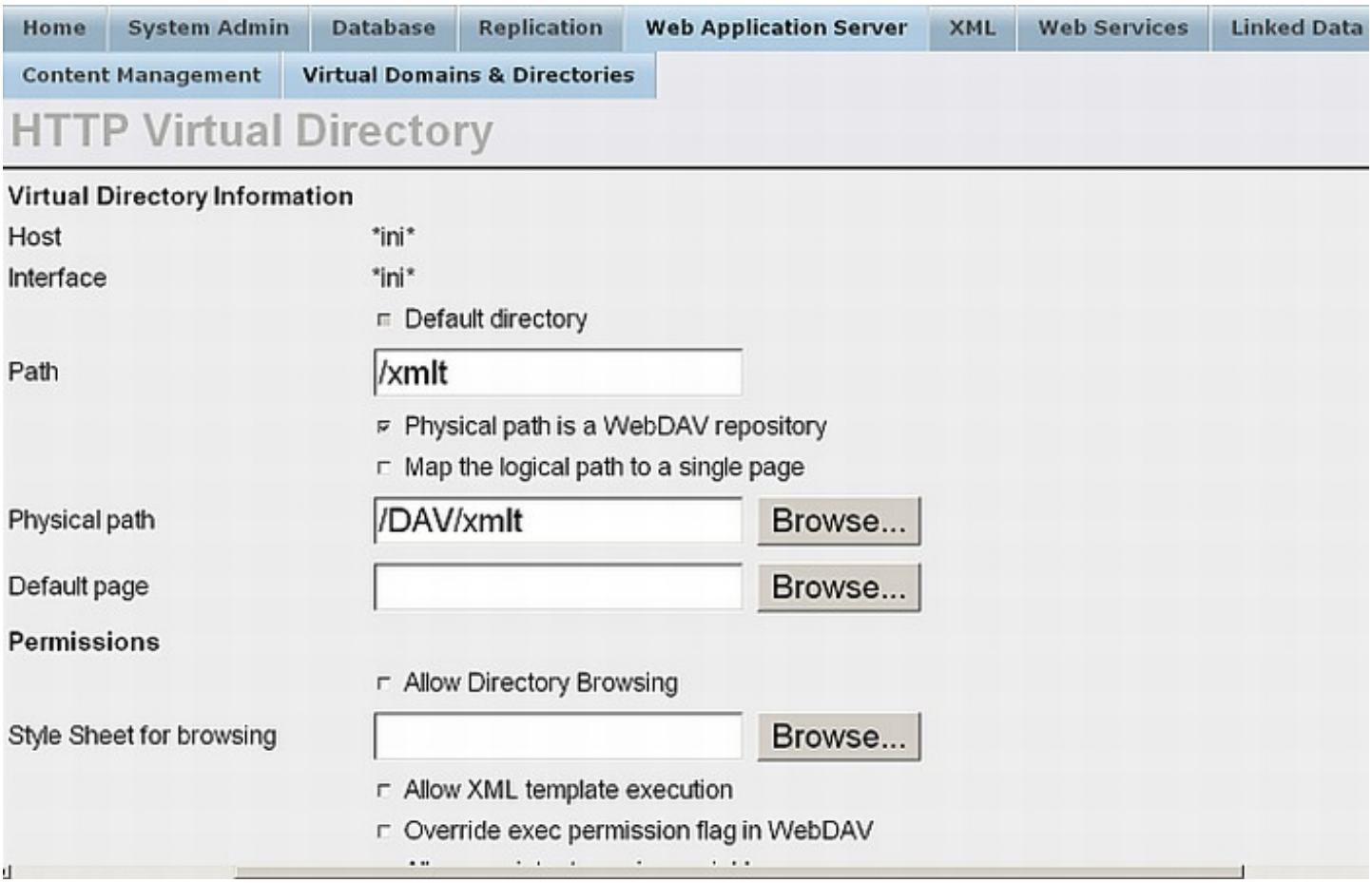
First we will make a new WebDAV directory. From Web Application Server -> Content Management -> Repository click the "New folder" icon and in the shown form enter for "Folder Name": xmlt. Choose for "Owner" dav and click the "Create" button.

Figure 15.8. Creating a new directory (collection) in WebDAV



Next we need to configure a virtual directory mapping to this so that XML Templates can be executed later. Go to Web Application Server / Virtual Domains & Directories and for your {Default Web Site} click the link "New Directory". In the shown form click "Next".

Figure 15.9. Configuring a Virtual Directory to respond to XML Template requests from our Dav



The programmatic equivalent for creating a virtual directory is the command in iSQL:

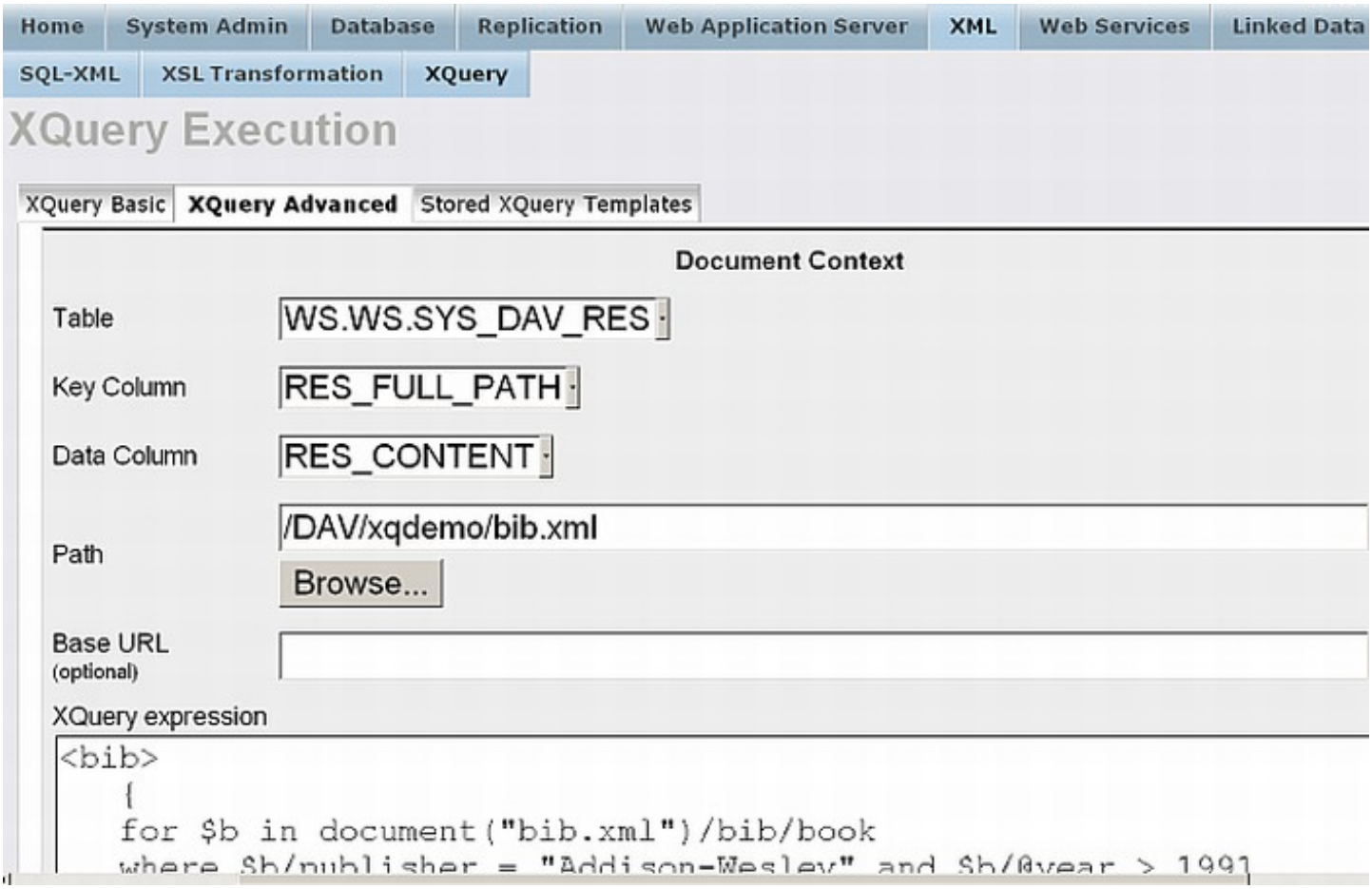
```
SQL>vhost_define(lpath=>'/DAV/xmlt',
                 ppath=>'/DAV/xmlt/',
                 is_dav=>1,
                 vsp_user=>'demo',
                 opts=>vector('xml_templates', 'yes'));
```

Now we go to XML -> XQuery -> XQuery Advanced.

We will start by testing the following query against the Demo database

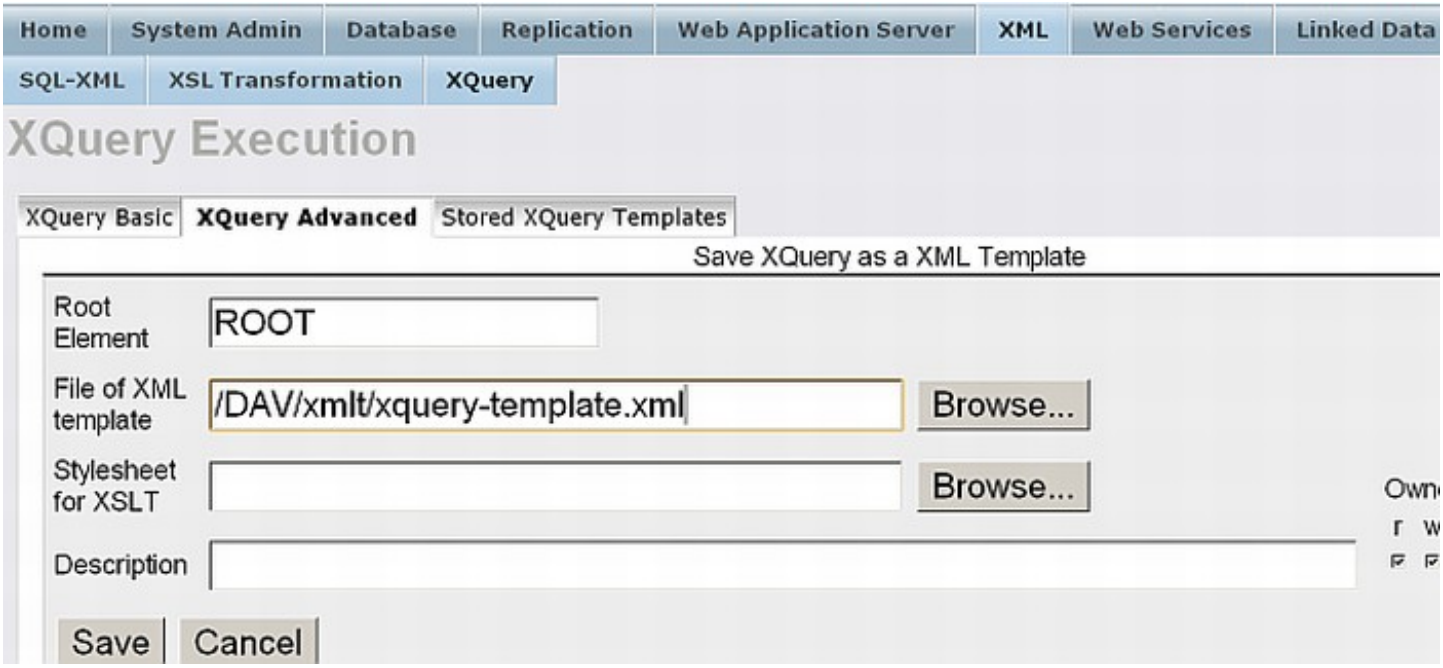
```
<bib>
   {
   for $b in document("bib.xml")/bib/book
   where $b/publisher = "Addison-Wesley" and $b/@year > 1991
   return
      <book year = {$b/@year}>
         {$b/title}
      </book>
   }
</bib>
```

Figure 15.10. XQUERY query against the Demo database



Now we will save this as an XML template by pressing the Save button. The query we used will be copied to the save XML template page. We will ensure that the name of the file is `/DAV/xmlt/xquery-template.xml` :

Figure 15.11. Saving the XML template for the Xquery query



See Also:

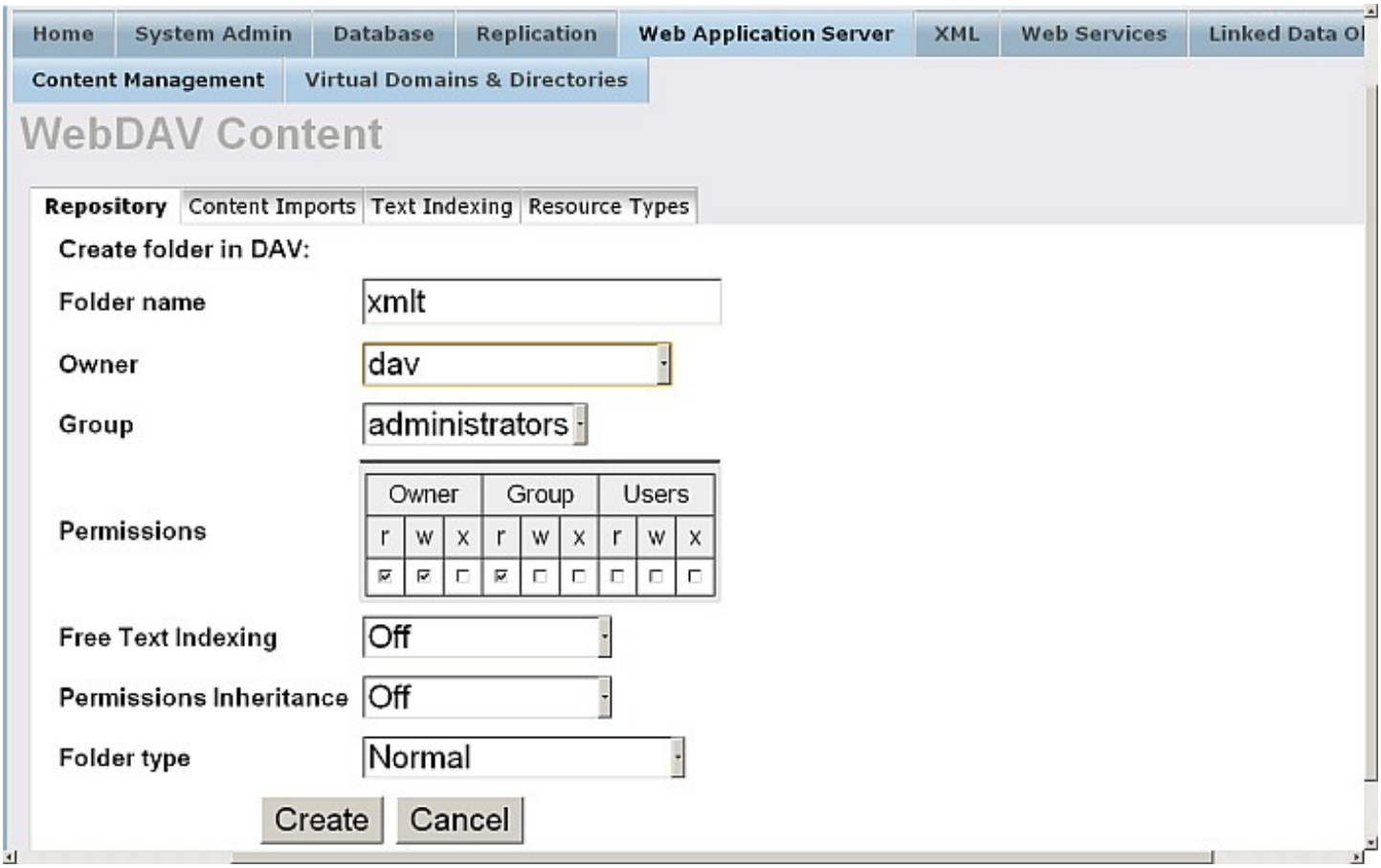
Exposing XML Templates as Web Services .

15.6.4. Saving XPATH Queries to XML Template

Using the Conductor Administration Interface, we are going to make and save a query to an XML template in WebDav, and then demonstration its use from a browser.

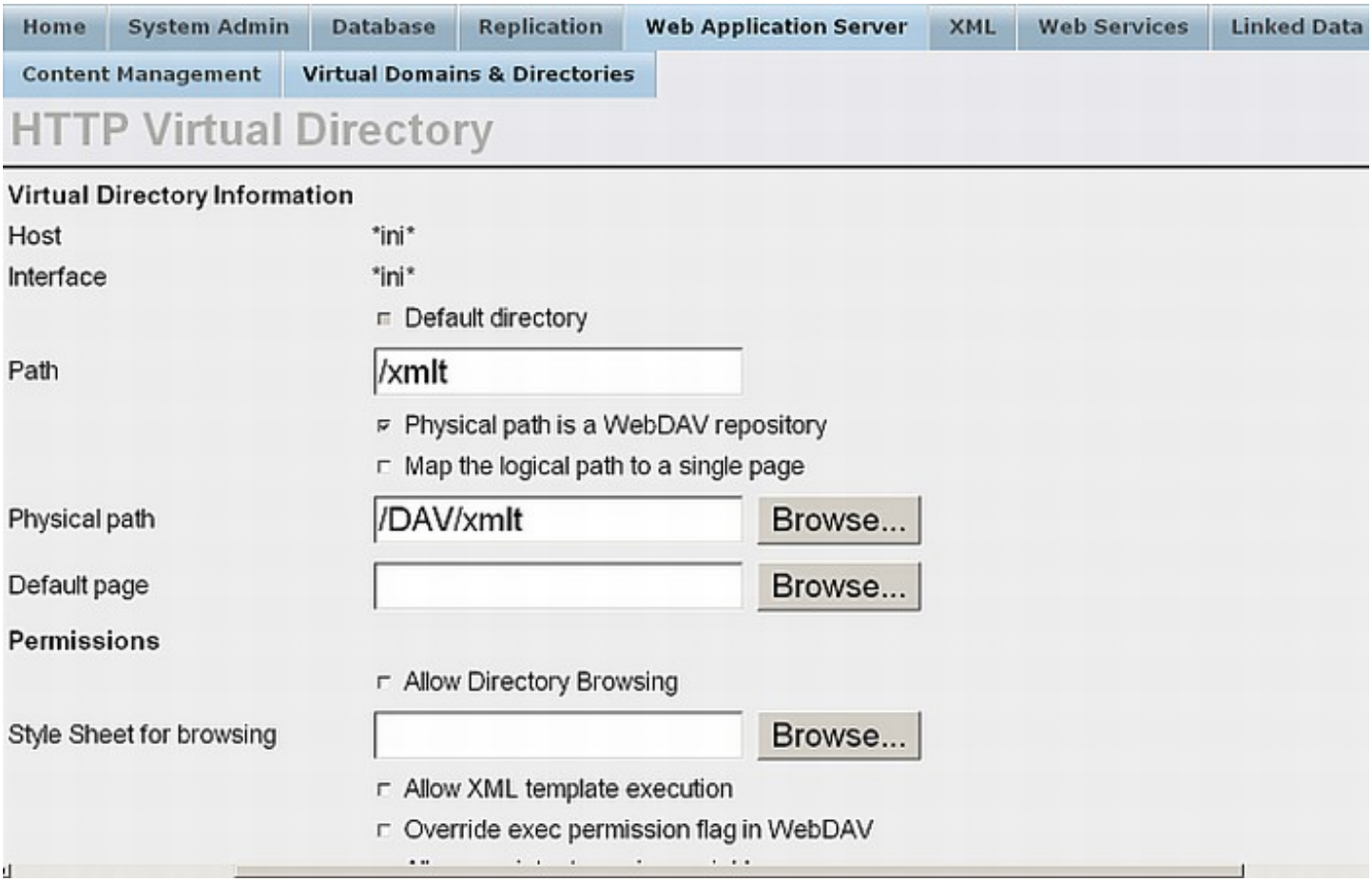
First we will make a new WebDAV directory. From Web Application Server -> Content Management -> Repository click the "New folder" icon and in the shown form enter for "Folder Name": xmlt. Choose for "Owner" dav and click the "Create" button.

Figure 15.12. Creating a new directory (collection) in WebDAV



Next we need to configure a virtual directory mapping to this so that XML Templates can be executed later. Go to Web Application Server -> Virtual Domains & Directories and for your {Default Web Site} click the link "New Directory". In the shown form click "Next":

Figure 15.13. Configuring a Virtual Directory to respond to XML Template requests from our Dav

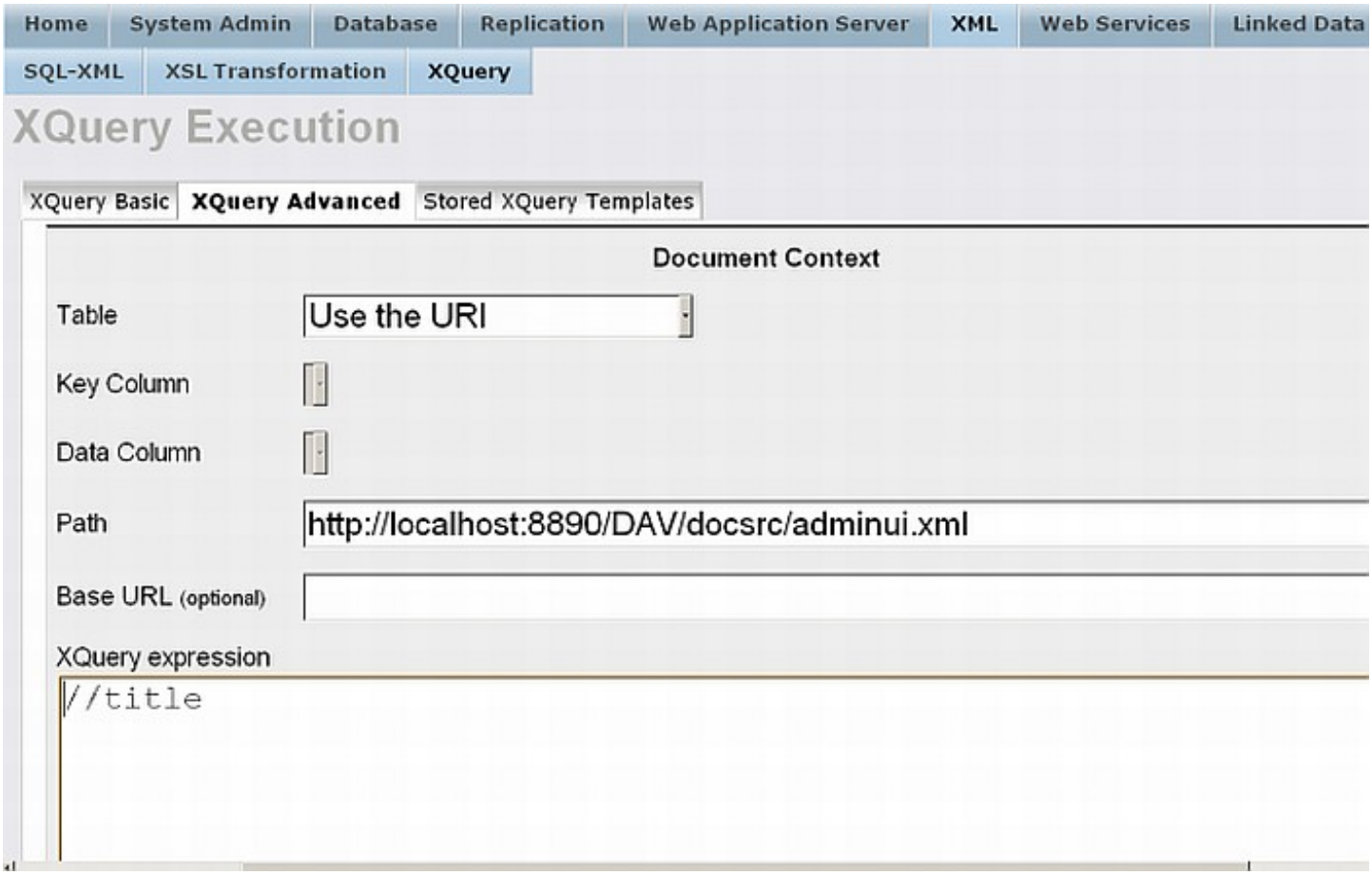


The programmatic equivalent for creating a virtual directory is the command in iSQL:

```
SQL>vhost_define(lpath=>'/DAV/xmlt',
                 ppath=>'/DAV/xmlt/',
                 is_dav=>1,
                 vsp_user=>'demo',
                 opts=>vector('xml_templates', 'yes'));
```

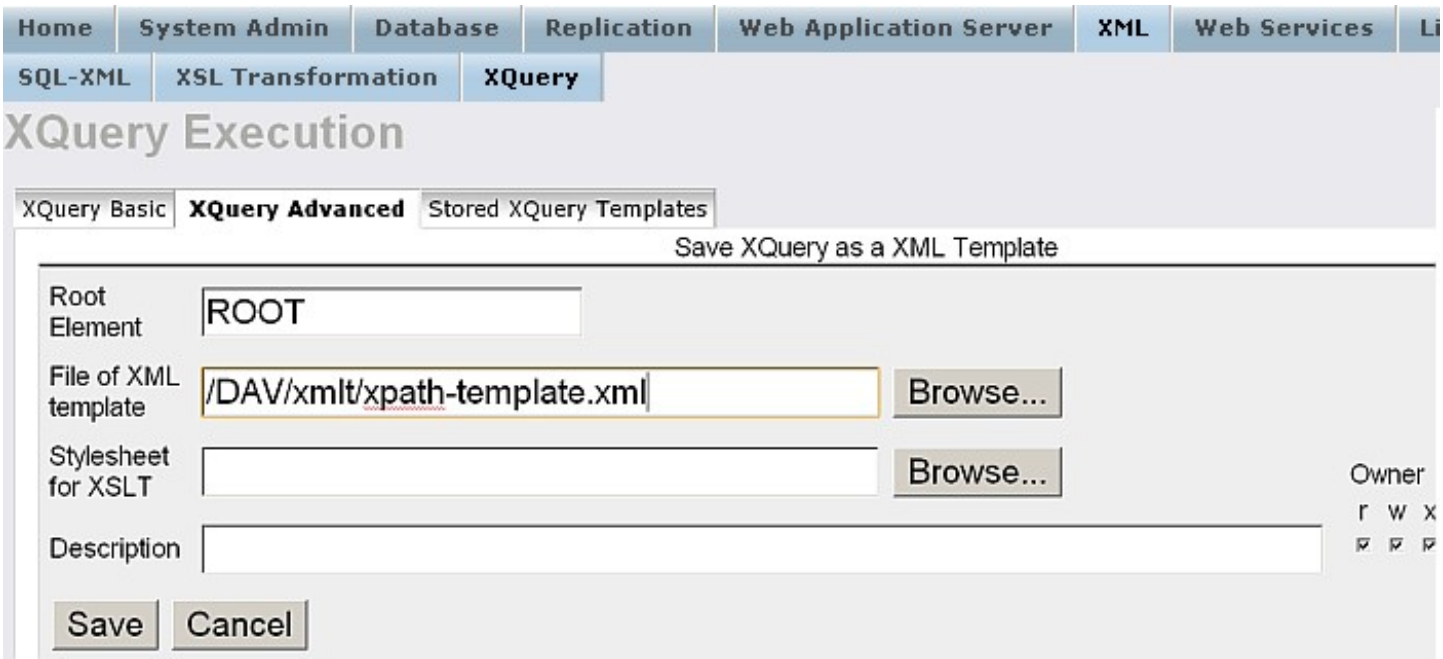
Now we set our sights on the XML files in WebDAV. We will perform a very simple XPATH query on an XML source file from this documentation. We will search for the existence of `<title>` tags in `http://host:port/DAV/docsrc/adminui.xml`:

Figure 15.14. XPATH query on a file in WebDAV



Make a quick mental note of the trivial result. Now we will save our query as an XML template to be executed over HTTP. Pressing the Save button takes us to the save XML Template page. We ensure that the file to save is set to `/DAV/xmlt/xpath-template.xml`:

Figure 15.15.



Once our query has been saved in an XML template we can test it by pointing a web browser at the URL for the file as we configured our virtual directory for, which will be similar to: `http://example.com/DAV/xmlt/xpath-template.xml`:

Figure 15.16.

See Also:

Exposing XML Templates as Web Services .

15.6.5. Programmatic Examples

Example 15.19. Executing an XML template from the file system

First step is to setup the /xmlt directory on file system. The function `vhost_define()` will be used to enable execution of XML templates. DBA privileges are required to use this function:

```
SQL> vhost_define (lpath=>'/xmlt', ppath=>'/xmlt/',
  vsp_user=>'demo', opts=>vector('xml_templates', 'yes'));
```

Now the following files must be stored as: <www-root>/xmlt/file1.xml

```
<?xml version ='1.0' encoding='UTF-8'?>
 <root xmlns:sql='urn:schemas-openlink-com:xml-sql' sql:xsl='shippers.xsl'>
   <!-- XML template example -->

   <!-- parameters declaration -->

    <sql:header>
      <sql:param name=":ShipperID">2</sql:param>
      <sql:param name=":CompanyName">United Package (you should see me)</sql:param>
      <sql:param name=":Phone">(503) 555-3199</sql:param>
    </sql:header>
   <!-- XML updategram , this will update the second record -->

    <sql:sync>
        <sql:before>
            <Shippers sql:id="1" ShipperID=":ShipperID"/>
        </sql:before>
        <sql:after>
            <Shippers sql:id="1" ShipperID=":ShipperID" CompanyName=":CompanyName" Phone=":Phone"/>
        </sql:after>
    </sql:sync>
   <!-- make a parametrized query -->

   <sql:query>
      SELECT ShipperID, CompanyName,Phone FROM Shippers where ShipperID = :ShipperID FOR XML AUTO
   </sql:query>
   <!-- make an error to see what happens -->

   <sql:query>
      select * from NotExist for xml auto
   </sql:query>
</root>
```

Now we want an XSL stylesheet stored as: <www-root>/xmlt/shippers.xslt containing the following:

```
<?xml version="1.0"?>

<xsl:stylesheet
    version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html" indent="yes" />

  <xsl:template match="/">
        <html>
            <head>
            <title>Shippers list</title>
            </head>
            <body>
            <xsl:apply-templates/>
            </body>
        </html>
    </xsl:template>

  <xsl:template match="root">
        <table>
            <tr><td>ID</td><td>Name</td><td>Phone</td></tr>
            <xsl:apply-templates/>
        </table>
    </xsl:template>

  <xsl:template match="Shippers">
        <tr>
            <td><xsl:value-of select="@ShipperID"/></td>
            <td><xsl:value-of select="@CompanyName"/></td>
            <td><xsl:value-of select="@Phone"/></td>
        </tr>
    </xsl:template>
</xsl:stylesheet>
```

Point your web browser to:

```
http://[host:port]/xmlt/file1.xml
```

and then at:

```
http://[host:port]/xmlt/file1.xml?:ShipperID=3
```

for the results.

Example 15.20. Executing an XML template from WebDAV

The first step is to create a new DAV collection and configure it to allow XML template execution. DBA privileges are required for these operations. To create a DAV collection the function `dav_col_create()` can be used, followed by the `vhost_define()` function to allow XML template execution:

```
SQL> dav_col_create ('/DAV/xmlt/', '110100100', 'dav', 'dav', 'dav', 'dav');

SQL> vhost_define (lpath=>'/DAV/xmlt', ppath=>'/DAV/xmlt/',
  is_dav=>1, vsp_user=>'demo', opts=>vector('xml_templates', 'yes'));
```

Now, the file can be uploaded. The same file as in the previous example will be used, copied from the file system location to the DAV location: <www-root>/DAV/xmlt/file1.xml

```
SQL> dav_res_upload ('/DAV/xmlt/file1.xml',
  file_to_string (concat (http_root(), '/xmlt/file1.xml')), 'text/xml',
  '111101101N', 'dav', 'dav', 'dav', 'dav');
```

Likewise the XSLT stylesheet from the previous example will also be used in the same way, placing into the DAV location: <www-root>/DAV/xmlt/shippers.xslt

```
SQL> dav_res_upload ('/DAV/xmlt/shippers.xsl',
  file_to_string (concat (http_root(), '/xmlt/shippers.xsl')),
  'text/xsl', '110100100N', 'dav', 'dav', 'dav', 'dav');
```

This example can now be demonstrated by trying the following URLs in your web browser:

```
http://[host:port]/DAV/xmlt/file1.xml
```

and

```
http://[host:port]/DAV/xmlt/file1.xml?:ShipperID=3
```

15.7. XML DTD and XML Schemas

15.7.1. XML Document Type Definition (DTD)

It is always useful to store a description of an XML document inside the document itself. XML DTD is a set of directives to describe the structure of the document and references to other XML documents.

The XML parser of the Virtuoso Server can recognize and use most DTD directives.

If DTD describes the structure of the document (i.e. the allowed content of elements with particular names) then the XML parser can validate the source document in order to check if it matches to the rules written in the DTD.

If a DTD describes references to external XML entities then the parser can build either "standalone" or "non-standalone" internal representation. To build a "standalone" representation, the XML parser will retrieve all external resources and then it will replace every occurrence of every reference with a copy of the whole content of the resource. To build a "non-standalone" representation, XML parser remembers only referencing information but it tends to not retrieve external documents; they can be retrieved on demand later.

A DTD may describe some attributes as "unique identifiers" and as "references to unique identifiers". If requested, the XML parser can check that identifiers are really unique or that there are no "dangling" references to missing identifiers.

In an ideal world, source documents match the XML standard perfectly and all declared URIs are valid and can be accessed. The real application, however, should read inaccurate data and ignore some minor errors. To let the XML parser signal only errors that really can affect the application, it is possible to precisely configure the reaction of the parser on every sort of problem. In addition, the application can get a detailed human-readable log of diagnostic messages by calling `xml_validate_dtd()` to let user fix the document in question.

If an XML document contains a DTD then DTD is placed before any data nodes (if parts of DTD can be placed in separate documents then they will be retrieved before parsing data nodes). Thus XML parser has enough data to perform "DTD validation" of the source document right during the reading. It lets the parser provide detailed diagnostics with precise location of detected errors.

15.7.2. Configuration Options of the DTD Validator

If some built-in Virtuoso/PL or XPATH function can invoke XML parser then it is probably have a special argument to pass configuration options to the XML parser. The most typical value of such an argument is "configuration string" that is a sequence of pairs parameter=value, delimited by spaces. Instead of single string, it can be passed as a vector of strings of even length of sort

```
vector (parameter1, value1, parameter2, value2 ...)
```

. It can be NULL indicating that the parser can use default values. No errors are reported if a parameter is specified twice, in which case the last specified value will be used. The only exception is the 'Validation' parameter which sets typical values for all parameters; if it is specified then it should be the first parameter in the list. Both parameter names and values are case-insensitive.

Many parameters are used to specify the importance of a particular error. For a particular application some validity constraints may be much more important than others. Because less than perfectly valid XML is common in practice it is important to configure the validator to report only those errors which are relevant to the application. Using configuration parameters, one may specify "importance levels" for every group of problems. There are 5 "importance levels":

Table 15.1. Error Catch Levels

Error Level	Description
FATAL	violations must be reported and further processing of XML source must be terminated right after the term where the violation was detected, without checking for errors in the rest of the document
ERROR	violations must be reported and validation should continue to find further errors, but no XML document will be built if validation is invoked from such function as `xtree_doc()` or `xper_doc()`
WARNING	violations must be reported but XML processing will continue
IGNORE	validator must try to locate violations of given sort but only severe (possibly fatal) violations must be reported
DISABLE	the specified check must be turned off fully, saving time and maybe memory

Some parameters are just switches, with only two values available: 'ENABLE' and 'DISABLE'.

AttrCompletion (ENABLE/DISABLE, default is DISABLE). This is useful when DTD validator is invoked from XML parser. When enabled, the XML document built will contain default values of 'IMPLIED' attributes as if they present in source text. It may be useful if application should perform free-text search on all attribute values including defaults or if XML should be converted in form suitable for external non-validating XML processor or if given XML data should be stored later as part of composite document and composite document will have another DTD with other default values.

AttrMisFormat (FATAL/ERROR/WARNING/IGNORE/DISABLE, default is DISABLE). This describes how to report errors in syntax of values of attributes.

AttrUnknown (FATAL/ERROR/WARNING/IGNORE/DISABLE). This describes how to report attributes whose names are not listed in the DTD.

BadRecursion (FATAL/ERROR/WARNING/IGNORE/DISABLE). This describes how to report circular references, when replacement text of an entity contains reference to this entity again, either directly (e.g. '<!ENTITY bad "some &bad; replacement">) or through other entities (e.g. '<!ENTITY a "&b;"> '<!ENTITY b "&a;">).

BuildStandalone (ENABLE/DISABLE, default is DISABLE). When set to ENABLE, replacement texts of external entities will be inserted instead of references to these entities, thus all data from a composite document will be gathered together into one large XML. This is useful for checking the element content model of the whole document without breaks on references or if parsed XML will be passed to external application as a standalone document.

Fsa (FATAL/ERROR/WARNING/IGNORE/DISABLE). This describes how to report violations of specified element-content model. Virtuoso's DTD validator contains a finite state automaton which can detect the first error in the content of some element, but remaining errors in the same element become "obscured" by the first one and will not be reported. Moreover, if element-content model is not SGML-compatible, some errors may remain undiscovered: it is possible to write a complex rule, so ambiguous that full check of all its interpretations will take prohibitively much time and memory. The validator will simplify such rules to make check faster, thus some errors will not be reported.

FsaBadWs (FATAL/ERROR/WARNING/IGNORE/DISABLE). This describes how to report the most frequent violation of element-content model specified by DTD: the use of whitespace characters in positions where only elements are allowed, not PCDATA. It usually happens when XML is indented for readability. You may wish to specify 'FsaBadWs=IGNORE' to eliminate redundant messages about this violation. Note that if you will specify 'FsaBadWs=DISABLE' then you will disable the check of illegal PCDATA tokens for this particular case, so common rule for 'Fsa' violations will be applied and you will see messages.

FsaSgml (FATAL/ERROR/WARNING/IGNORE/DISABLE). This describes how to report violations of SGML compatibility in element-content model. Some complex DTD rules for elements are not supported by SGML processors and the validator may report the use of such rules.

GeRedef (FATAL/ERROR/WARNING/IGNORE/DISABLE). This describes how to report redundant definitions of generic entities. There redefinitions are errors in SGML but they may be ignored in XML processing. The first definition will be used and others will be ignored.

IdDuplicates (FATAL/ERROR/WARNING/IGNORE/DISABLE). This describes how to report non-unique values of ID attributes. It is a data integrity error, because IDs are usually parts of some primary keys, and are expected to be unique.

IdrefIntegrity (FATAL/ERROR/WARNING/IGNORE/DISABLE). This describes how to report "dangling references". Any value of IDREF attribute and any name from value of IDREFS attribute should appear in the same XML document as value of some ID attribute. You can think that an ID attribute specifies a hyperlink anchor and IDREF is a hyperlink, so it's a data integrity error if a hyperlink points to unknown location.

Include (FATAL/ERROR/WARNING/IGNORE/DISABLE). This configures reading of external sub-documents into "main" document you validate (and maybe load in database). If 'DISABLE', no additional documents will be read, otherwise external parameter-entities, external generic-entities and external DTD will be located, using their SYSTEM names. External documents may reside in file system, in database or in the Web. Absolute SYSTEM names (of form 'protocol://server/resource') will be used without any modifications, relative SYSTEM names should be "resolved", i.e. converted to absolute by adding a prefix from the *base_uri* argument of SQL function.

MaxErrors (a string that represents an integer from 1 to 10000, default is '25'). This specifies how many errors may be logged before the "Too many error messages" fatal error will be reported.

MaxWarnings (a string that represents an integer from 0 to 10000, default is '100'). This specifies how many warnings may be logged before the "Too many warning messages" event will stop their logging.

NamesUnknown (FATAL/ERROR/WARNING/IGNORE/DISABLE). This describes how to report if the document contains element names which are not the declared in DTD. They may be typos in element names or signal that DTD is incomplete or obsolete. In addition, unknown names may be reported as element-content model violations.

NamesUnordered (FATAL/ERROR/WARNING/IGNORE/DISABLE). This describes how to report element names not declared before use in DTD. Proper order ("declare element name before use it") is important solely for compatibility with SGML standard.

NamesUnresolved (FATAL/ERROR/WARNING/IGNORE/DISABLE). This describes how to report if an element name used in the DTD is not declared at all. This may occur if DTD is incomplete or if some declaration in it are ignored conditional sections. Unresolved names cause no data integrity errors while remain unused in data section of the XML document, NamesUnknown parameter defines what happens if they're actually used.

PeRedef (FATAL/ERROR/WARNING/IGNORE/DISABLE). This describes how to report redundant definitions of parameter entities. Similarly to redefinitions of generic entities, there redefinitions are errors in SGML but they may be ignored in XML processing. The first definition will be used and others will be ignored.

Sgml (FATAL/ERROR/WARNING/IGNORE/DISABLE). This describes how to report violations of SGML compatibility. In fact, not all such violations are detected by the Virtuoso Server, because known SGML readers are insensitive to some sorts of violations.

TooManyWarns (FATAL/ERROR/WARNING/IGNORE/DISABLE). This describes how to report "Too many warning messages" event. While "Too many errors" is fatal error and terminates XML processing, "Too many warning messages" may have arbitrary "importance levels".

TraceLoading (ENABLE/DISABLE, default is DISABLE). If set to 'ENABLE', the validator will log every reading of any resource, for easier tracking of URI resolving problems. It's possible that some readings of sub-documents will not be reported: there's a limit for number of records in the log returned by the validator. In addition, sub-documents may be cached inside validator, so only first references to some sub-document will require reading procedure.

Validation (SGML/RIGOROUS/QUICK/DISABLE, default is DISABLE). This loads one of four "preset configurations". It must be the first parameter in configuration string, if used. DISABLE means "do not check for any type of error", QUICK is to check only for violation of "local" validity constraints, with disabled FsaBadWs, IdDuplicates and IdrefIntegrity, RIGOROUS enables these three groups, too, SGML enables all checks including all checks for SGML compatibility.

VcData (ENABLE/DISABLE, default is DISABLE). This describes how to report violations of generic validity constraints in data section of XML document. If constraint is not configured by other parameters listed here, it will be configured by this parameter (or by VcDtd if relates to the text of DTD section).

VcDtd (ENABLE/DISABLE, default is DISABLE). This describes how to report violations of generic validity constraints in DTD section of XML document. If constraint is not configured by other parameters listed here, it will be configured by this parameter (or by VcData if relates to the text of data section).

15.7.3. XML Schema Definition Language

The W3C XML Schema Definition Language is a way of describing and constraining the content of XML documents.

The XML Schema specification consists of three parts. One part defines a set of simple datatypes, which can be associated with XML element types and attributes; this allows XML software to do a better job of managing dates, numbers, and other special forms of information. The second part of the specification proposes methods for describing the structure and constraining the contents of XML documents, and defines the rules governing schema validation of documents. The third part is a primer that explains what schemas are, how they differ from DTDs, and how one builds a schema.

XML Schema introduces new levels of flexibility that may accelerate the adoption of XML for significant industrial use. For example, a schema author can build a schema that borrows from a previous schema, but overrides it where new unique features are needed. XML Schema allows the author to determine which parts of a document may be validated, or identify parts of a document where a schema may apply. XML Schema also provides a way for users of e-commerce systems to choose which XML Schema they use to validate elements in a given namespace, thus providing better assurance in e-commerce transactions and greater security against unauthorized changes to validation rules. Further, as XML Schema are XML documents themselves, they may be managed by XML authoring tools, or through XSLT. The implementation of XML Schema in Virtuoso is based on the W3C XML Schema Specification .

See Also:
XML Schema Part-0: Primer

XML Schema Part-1: Structures

XML Schema Part-2: Datatypes

For parsing the schema definitions the DTD definition of XML Schema is used. This definition was taken from XML Schema Part-1: Structures .

15.7.4. XML Schema Functions

The Virtuoso interface to XML Schema is represented primarily by two functions:

`xml_validate_schema()`

`xml_load_schema_decl()`

The signature of the function `xml_validate_schema()` is the same as the function `xml_validate_dtd()` . It parses and validates an XML document. The root element of the document must contain the *`"schemaLocation"`* attribute with the value of the document's URI.

As described above, the XML Schema Processor implemented within Virtuoso relies on the XML Schema DTD, which is composed of two files: "XMLSchema.dtd" and "datatypes.dtd." These files must be placed in the system directory (see `xml_add_system_path()`).

The following XML Schema items are not fully implemented:

facets support is primitive;

you may only derive by restriction from the "anyType" type;

enumerations are not supported;

the "all" particle is not supported;

elements may not be defined within an element model group declaration;

unions are not supported;

"appinfo," "documentation," "list," and "notation" tags are ignored.

Virtuoso does not cache XML Schema documents; they are completely reprocessed every time the document is loaded.

15.7.5. XML Schema & SOAP

The XML Schema defines primitive types, such as "integer","string" and "float". Users may extend or restrict the primitive types to build their own type system. For example, a type "monthCode" could be an enumerated list of three character strings

corresponding to JAN, FEB, etc... SOAP types can use any of these types: native, primitive XML Schema types or complex types which should be fully described using XML Schema.

SOAP messages that use XML Schema types or complex types contain a fragment of XML Schema or a reference to a schema file so that the data types can be determined and messages validated.

The Virtuoso tutorials contain many examples from the SOAP Interop test suite. Each tutorial displays both the request message and response message from the server when the tests are executed. Below is a sample message exchange of the echoStruct tutorial (SO-S-20) which uses a trivial complex struct composed of an integer, float and a string:

Example 15.21. Sample Run of Tutorial SO-S-20

Struct parameter inputs:

```
varString = hello
varInt = 42
varFloat = 99.004997253418
```

Return value:

```
varString = hello
varInt = 42
varFloat = 99.004997253418
```

Request message:

```
<?xml version='1.0' ?>
<SOAP:Envelope
    SOAP:encodingType='http://schemas.xmlsoap.org/soap/encoding/'
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xmlns:xsd='http://www.w3.org/2001/XMLSchema'
    xmlns:SOAP='http://schemas.xmlsoap.org/soap/envelope/'
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"  xmlns:dt='urn:schemas-microsoft-com:datat
  <SOAP:Body>
    <cli:echoStruct SOAP:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
        xmlns:cli='http://soapinterop/' >
      <inputStruct>
        <varInt xsi:type="xsd:int" dt:dt="int">42</varInt>
        <varFloat xsi:type="xsd:float" dt:dt="float">99.004997</varFloat>
        <varString xsi:type="xsd:string" dt:dt="string">hello</varString>
      </inputStruct>
    </cli:echoStruct>
  </SOAP:Body>
</SOAP:Envelope>
```

Response message:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope
    xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAPSDK4:echoStructResponse
        xmlns:SOAPSDK4="http://soapinterop/">
      <Result href="#id1"/>
    </SOAPSDK4:echoStructResponse>
    <SOAPSDK5:SOAPStruct
        xmlns:SOAPSDK5="http://soapinterop.org/xsd"

xml:id="id1" SOAPSDK3:root="0"
        SOAPSDK2:type="SOAPSDK5:SOAPStruct">
      <varString>hello</varString>
      <varInt>42</varInt>
      <varFloat>99.004997253418</varFloat>
    </SOAPSDK5:SOAPStruct>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

See Also
Extending SOAP Types

SOAP Literal Encodings

15.8. XQuery 1.0 Support

The Virtuoso Server provides support for the XQuery 1.0 XML Query Language specification. This specification is currently in the working draft stage at the W3C XML Query Working Group working in collaboration with the W3C XSL Working Group. Both the syntax and semantics of XQuery will probably vary from version to version.

In addition to the XQuery 1.0 standard, which describes the language, the XQuery 1.0 and XPath 2.0 Functions and Operators Version 1.0 specification describes a set of built-in functions. As with all W3C in-progress efforts, there is a list of open issues detailing problems and unresolved areas; where these affect Virtuoso's implementation, they are noted below.

This chapter is not an XQuery textbook and does not replace XQuery-related specifications of W3C. Only Virtuoso-specific extensions and differences are described here.

The most important deviation from the standard is that Virtuoso does not provide full type information about data values. As a consequence, "typeswitch" and automatic type conversions are not implemented.

15.8.1. Types of XQuery Expressions

The current draft of XQuery lists 10 groups of XQuery expressions:

- Primary expressions, including literals, variable references and function calls.
- Path expressions, including all XPATH 1.0 expressions and a "pointer operator".
- Sequence expressions.
- Arithmetic, comparison and logical operators.
- Element constructors, which allow you to create new element nodes with specified names, lists of attributes and lists of children.
- FLWR (FOR-LET-WHERE-RETURN) expressions, which allow you to create variables for intermediate results and to transform sequences of items on a per-item basis.
- Ordered and unordered expressions to force some sorting of intermediate results or to prevent the XQuery processor from making redundant sorting.
- Control expressions, such as IF.
- Quantified expressions: SOME and EVERY.
- Expressions that test or modify data types.

Not all groups of expressions are implemented. In some groups, not all kinds of clauses are implemented.

In addition to the standard, Virtuoso supports special cases of FLWR expressions to deal with XML views:

- FOR Clause expressions with xmlview() function.

Primary Expressions

XQuery processor uses 32-bit integers on 32-bit platforms and 64-bit integers on 64-bit platforms. Similarly, the scale and precision of floating-point operations may vary from platform to platform.

Note that string literals are handled differently in XPath 1.0 and XQuery. "Ben "amp; Jerry"apos;s" denotes the string "Ben " Jerry's" in XQuery and the string "Ben "amp; Jerry"apos;s" in XPath.

Path Expressions

Any XPath 1.0 expression is a valid XQuery 1.0 path expression, which the Virtuoso XQuery processor supports. When invoked from the XQuery context, the XPath Processor works in accordance with XSLT rules. There are two major differences between standalone and XQuery/XSLT path expressions. First, the meaning of non-qualified name used as NameTest criterion, as described below. Second, the data type used for attributes varies. In XPath or XQuery mode, if a value is calculated by an

attribute:: axis, it is of type *attribute entity*; in standalone XPath, the string value of the attribute is used instead.

As specified in the XQuery 1.0 standard, a node-set returned by an XPath expression may be used as a sequence of items, where every node of that node-set becomes an item of the sequence. The opposite is not true, however. Not every sequence may be converted into a node-set, even if it is a sequence of nodes. If XPath starts from a function which returns a sequence, an error message "Context node is not an entity" is returned. Fortunately, a variable of type

```
sequence
```

may be used as a node-set if all items of the sequence are nodes.

Obsolete drafts of W3C specification contains description of "pointer operator". Virtuoso continues to support this operator to provide backward compatibility. XQuery processor needs DTD data associated with the XML document in question to distinguish ID attributes from other sorts of attributes and to bookmark elements that have ID locations. For more details, see the description of `id()` XPATH function. This function uses same DTD data for same purposes, so for any given document, either both `id()` and "pointer operator" are applicable or both does not work.

Sometimes the "Context node is not an entity" error is signalled if the beginning of the XPath expression is surrounded by parenthesis, even if the expression works fine without these parenthesis. This happens because "(...)" is an "append" operator in XQuery, not just a way to group subexpressions. "append" converts a node-set into a sequence even when it is called with a single argument - that is, without commas inside "(...)". This sequence cannot be used as input for the rest of the XPath expression.

As an syntax extension, special notation of QNames is added and can be used, e.g., in NameTest. An expanded name can be surrounded by delimiters (! and !), like (!http://www.example.com:MyTag!) and this syntax allows names that contain otherwise prohibited characters. This syntax is also useful when the text of the query is generated by software.

Note that the NameTest that consists of an unqualified name has different meanings in Virtuoso XPath and in XQuery. In XPath, NameTest "sample-tag" means "any element whose local-name is equal to sample-tag". In XQuery, the same test means "any element without namespace-uri whose local-name is equal to sample-tag".

Sequence Expressions

XQuery sequences are supported not only in XQuery but can also be handled in XPath and XSLT. When the XQuery processor is invoked from SQL and a sequence is returned to the caller, the sequence is automatically converted into a vector of its elements.

Virtuoso supports all sequence operations listed in current W3C paper plus deprecated operations BEFORE and AFTER.

The *sequence concatenation* operator is available in XPath and XSLT as the `append()` function. In addition, the `tuple()` function is available to get the first items of every given argument sequence and return the sequence of these items.

XQuery operators UNION, INTERSECT, EXCEPT are available in XPath and XSLT as functions `union()`, `intersect()` and `except()`.

Arithmetic, Comparison and Logical operations

Virtuoso shares the implementation of basic arithmetic and comparison operations between XPath, XQuery, XSLT, SQL and Virtuoso/PL processors, so type casting, scale and precision of calculated values are identical across the system. All operators are available in XQuery, in addition, << and >> operators are available in XPath and XSLT as `is_before()` and `is_after()` built-in functions.

Element Constructors

Virtuoso XQuery supports all XQuery 1.0 direct constructors. Previous versions of W3C draft contained the syntax for placing calculated content into the opening tag of direct element constructor, such as

```
<{concat("calculated-", "element-name")} {concat("calculated-", "attribute-name")}={concat("calculated-",
```

. Thus name of element or attribute, or a value of an attribute can be calculated dynamically. This syntax is still supported. The `create-element()` XPath function is implemented to make this functionality available in XPath. Additionally, a special function `create-attribute()` may be used to create a new dynamic attribute entity with value and name calculated, this works similarly to xsl:attribute XSLT instruction.

Similarly, `create-comment()`, `create-element()` and `create-pi()` mimics other XQuery direct constructors in XPath and XSLT.

The XQuery specification states that when sequence of atomic values is converted into content of an element constructor, whitespace character is inserted between adjacent values.

Unlike previous versions of Virtuoso, current XQuery syntax allows you to use "pure XML notation" inside element constructors. Thus there is no strict need to write 'constant' expression

```
<emp empid="12345"><name>John Smith</name><job>Bubble sorter</job></emp>
```

as it is dynamically calculated text, like

```
<emp empid="12345"><name>{'John Smith'}</name><job>{'Bubble sorter'}</job></emp>
```

It is still may be useful to write 'constant' expression in the old way. This artificial restriction simplifies finding syntax errors, because there are syntactically wrong expressions that are still correct "pure XML notation." Alternatively, CDATA sections also may be used to make it obvious that the string is a constant, not an expression with forgotten braces around it:

```
<emp empid="12345"><name><![CDATA[John Smith]]></name><job><![CDATA[Bubble sorter]]></job></emp>
```

The current version of Virtuoso does not support the new XQuery syntax for dynamic constructors.

FLWR Expressions

FLWR expressions are fully supported by Virtuoso XQuery. Moreover, `for()` and `let()` XPath functions are implemented to make this functionality available in XPath and XSLT. In addition, `assign()` and `progn()` functions are available to deal with extension functions, especially when extension functions are called for their side effects.

A special xmlview() function allows very efficient access to SQL data from XML views.

Previous XQuery specifications used "sort by" instead of "order by". The difference is that "sort by" was applicable to the final results of the FLWR statement made by RETURN clause whereas "order by" reorders input data for RETURN. Thus, "order by" can sort outputs using data that do not appear in the final result. E.g., an expression can collect items, "order" them by category and title but output only title and price. This was much harder in previous versions of XQuery because it was necessary to prepare an intermediate result that contained title and price and category, then do "sort" by category and title then use one more FLWR expression to form a result that is free from redundant data about category.

Nevertheless, Virtuoso supports both "sort by" and "order by", to keep backward compatibility. Moreover, "sort by" operator can be freely used with no relation to any FLWR subexpression. Typical use of such a simplified notation is

```
<hit-list>{//track[@rating] sort by (@rating descending)}</hit-list>
```

instead of portable

```
<hit-list>{for $t in //track[@rating] order by $t/@rating descending return $t}</hit-list>
```

Ordered and Unordered Expressions

The current version of Virtuoso does not use ordered/unordered hints. Everything is calculated ordered. This will change in the future but it is not advisable to place "unordered" hints for future use because there's no way to validate these hints. It is better to place appropriate comments but not hints.

Control Expressions

The `if()` special function mimics the XQuery operator for use in XPath and XSLT. Functions `and()` and `or()` are also control expressions because they calculate arguments in strict left-to-right order and may omit the calculation of some results.

Quantified Expressions

Both the *SOME* and *EVERY* operators are implemented. The `some()` and `every()` XPath functions are implemented to make this functionality available in XPath and XSLT.

Expressions That Test or Modify Data types

The operators *IS* , *CASTABLE* , *CAST* , *TREAT* , *TYPESWITCH* and *VALIDATE* are not implemented.

FOR Clause Expressions With xmlview() Function

XML views can be queried using FOR Clause from FLWR expressions. The `xmlview()` function allows XML views to be accessed as if they were XML documents. XPath expressions beginning with the `xmlview()` function will be translated into SQL statements to avoid redundant data access and to avoid creating a whole XML tree.

15.8.2. Details of XQuery Syntax

Virtuoso XQuery uses some syntax extensions. Most visible is an additional notation for qualified names as described above (name is surrounded by "(!...!)" delimiters. An earlier implementation allowed single-line comments started with "#" or "--" continuing to the end of line, this syntax is now obsolete.

The "default namespace declaration" clause is not currently supported, to make the text of XQuery unambiguous. If used, default namespaces must extend element names but not attribute names. Extension function names must be extended as they have non-default namespace prefixes but the names of basic functions should not be extended by the default namespace. Finally, Virtuoso will not preserve any information about used namespace prefixes, so default namespaces will be converted into non-default when the resulting XML entity is printed.

15.8.3. Pre-compilation of XPath and XQuery Expressions

Virtuoso compiles XPath and XQuery expressions as early as it is possible. E.g. if the first argument of `xquery_eval()` is a string constant then the SQL compiler will invoke the XQuery compiler to avoid on-demand compilation(s) of this text.

This feature significantly enhances performance of XQuery expressions embedded in SQL. For a simple search on XML document of average size the compilation time can be three times greater than execution time. In addition, the use of `sql:column()` special XQuery function is possible only when pre-compilation can be done by SQL compiler.

Pre-compilation is impossible if the text of the expression is not a constant. The typical case is passing an XQuery expression as parameter to a function. In this case the expression is compiled during the call of xquery_eval() and stored for future use. If the same string is passed again to the same invocation of xquery_eval() then a stored compiled expression is used.

Only partial pre-compilation is possible if XQuery expression refers to not-yet defined extension functions or to external resources. Partial pre-compilation gives little gain in speed, but it allows the use of `sql:column()`

The most important fact about pre-compilation is that passing parameters into XQuery statement is much more efficient than printing then into the text of the query. This is similar to SQL queries.

Example 15.22. Good and Poor Coding Practices

GOOD The expression is compiled once when SQL query is compiled:

```
select xquery_eval('count(//abstract)', SOURCE_XML) from LIB..ARTICLES;
```

GOOD The expression is compiled once when SQL query is compiled:

```
select xquery_eval('count(//article[@id=$main_id]/abstract)', SOURCE_XML, 1, vector('main_id', MAIN_ID))
  from LIB..ARTICLES;
```

POOR The expression is compiled once per data row. In addition, a hard-to-find error will occur if a value of MAIN_ID may contain double quote or a backslash character.

```
select xquery_eval(sprintf('count(//article[@id="%s"]/abstract)', MAIN_ID), SOURCE_XML)
  from LIB..ARTICLES;
```

GOOD The XQuery expression is compiled once per execution of the SQL query. The SQL compiler pays special attention to queries that import external resources, because the content and availability of these resources may differ from call to call. In addition, importing an external resource is usually not possible during SQL compilation due to deadlock danger, so the

compilation is postponed until run time, but this is not too bad anyway. Even in this sophisticated case, XQuery can contain calls of `sql:column()`.

```
select xquery_eval('
    namespace tools="http://www.example.com/lib/tools/"
    import define "http://www.example.com/lib/tools/common.xqr"
    tools:extract-keywords(//abstract)',
  SOURCE_XML)
from LIB..ARTICLES;
```

GOOD Two XQuery expressions are compiled during SQL compilation.

```
select
  case
    when SOURCE_IS_DOCBOOK then xquery_eval ('//formalpara[title="See Also"]/para', SOURCE_XML)
    else xquery_eval ('//p[@style="seealso"]', SOURCE_XML)
  end
from LIB..ARTICLES;
```

POOR Virtuoso can not pre-compile XQquery expressions. Moreover, only one precompiled expression is cached per occurrence of xquery_eval() in the SQL statement so it is possible that an XQuery compiler will start once per data row.

```
select
  xquery_eval (
    case
      when SOURCE_IS_DOCBOOK then '//formalpara[title="See Also"]/para'
      else '//p[@style="seealso"]'
    end,
  SOURCE_XML)
from LIB..ARTICLES;
```

15.9. XSLT Transformation

Virtuoso implements XSLT 1.0 transformations as SQL callable functions. In general, Virtuoso's XSLT processor follows the XSLT 1.0 standard as far as possible without loss of database-related functionality. The most important difference is in error handling. While XSLT rules assume that almost any error must be recovered, Virtuoso will stop the transformation in case of data access error or in case of serious arithmetic error, such as divide by zero. This is due to the greater complexity of the Virtuoso XSLT processor's environment. Standalone processors generally accept a set of files as input parameters, the only output is the resulting transformation as a file, and all configuration parameters are listed on the command line. Virtuoso's XSLT processor may retrieve both data and stylesheets from local filesystems, local tables, remote datasources, or the Web. Parameters of the stylesheet transformation may be prepared by complicated Virtuoso PL procedures and so on. Plain XSLT processors may continue producing "dirty" output; Virtuoso's processor will stop upon encountering serious in order to produce helpful error diagnostics.

Another important difference is that Virtuoso has one processor for XSLT, XPath and XQuery. The user will find this convenient because an application may use XSLT-specific functions in XPath expressions, XQuery functions in XSLT, and so on. If extension functions are defined for the XSLT processor, they may also be used in XPath and XQuery expressions without any additional operations. One caveat is that this may invalidate some rules related to the document order of nodes in node-sets. Both the XPath 1.0 and XSLT 1.0 standards say that some functions must process the node that is first in document order from a given node-set. In such cases Virtuoso's processor will process the first node found in the node-set, regardless of whether it is actually the first in document order or not. This is done because the old XPath rule is nonsensical if nodes of the node-set are retrieved from more than one document or are created in the query. The processor follows the old rule for 99% of real expressions. Specifically, it fully matches the standards' rules for all XPath expressions that are free of reverse-order axes and non-standard functions.

15.9.1. Namespaces

In the following the

```
xsl:
```

namespace prefix is used to mean "http://www.w3.org/XSL/Transform/1.0". In fact all namespace URI's beginning with "http://www.w3.org/XSL/Transform" are considered as the XSLT namespace. The namespaces

"http://www.w3.org/1999/XSL/Transform" and "http://www.w3.org/TR/WD-xsl" are recognized as synonymous. No other namespace URI's have special significance.

15.9.2. The *<xsl:output>* Tag

The Virtuoso XSLT processor recognizes and verifies the *<xsl:output>* tag. The only *<xsl:output>* attributes that are used are the *method*, *encoding*, *omit-xml-declaration*, *media-type*, *doctype-public* *doctype-system* and *indent*. The processor output is created in the character set that is in effect for the client or the one specified by encoding.

The SQL function `xml_tree_doc_media_type()` returns the media type in effect for the result of the `xslt()` function based on the xsl:output *media-type* and *method* attributes of the stylesheet applied. This method is also used for producing FOR XML AUTO WebDAV resources for HTTP reply generation.

15.9.3. External Parameters in XSLT Stylesheets

The XSLT 1.0 standard specifies that parameters may be passed to the XSLT processor from its environment in order to control data transformation, but the details are implementation-specific. Virtuoso's XSLT Processor will accept default values for global parameters from the optional third argument of the xslt() Virtuoso PL function. This argument, if specified, must be a vector of parameter names and values of the form

```
vector(name1, value1,... nameN,
valueN)
```

, where *name1 ... nameN* must be of type *varchar*, and *value1 ... valueN* may be of any Virtuoso datatype, but may not be null. If a parameter has a null value specified, it is removed from the vector without any further processing. After removal of null values, duplicate names are eliminated. If a name occurs more than once in the vector, only the last one will be retained.

When the XSLT Processor begins the transformation of the document, it creates two stacks of variables, one for global variables and one for local variables. Initially the stacks are empty. Then it prepares temporary indexes for all *<xsl:key>* elements; all future calls of `key()` function will return without any searches through the source document. Only after the processing of all keys of the stylesheet, the XSLT Processor pushes all parameters from the third argument of the `xslt()` function into the stack of local variables. Then, as described in the XSLT specification, it initializes top-level variables and parameters. For every *<xsl:variable>* or *<xsl:param>* element found at the top-level of the main stylesheet or at the top level of some imported stylesheet, the processor calculates the value and pushes the created variable into the stack of local variables. The *<xsl:param>* element is ignored if the specified name is already declared in the vector of parameters or in some stylesheet imported before. When all top-level variables and parameters are initialized, the content of the stack of local variables is moved into the stack of global variables, and the stack of local variables is made empty. During the rest of the XSLT transformation, these variables will be used as global variables. They may be temporarily shadowed by inner declarations of local variables - though not by declarations of local parameters - but they cannot be changed. Note that expressions for values of variables and parameters may contain calls of `key()` function, because temporary indexes are ready to use before the first such expression is calculated. Expressions of *<xsl:key>* elements may not refer to any parameters or variables, due to the same reason.

15.9.4. Functions

The `xslt()` Virtuoso/PL function applies a given stylesheet to a given source XML document and returns the transformed document. There is no restriction to what a VSP page can output, this is usually HTML but can be XML. The function `http_xslt()` allows a stylesheet to be applied to a whole output of an VSP page before it is sent to the user agent. Functions `xslt_sheet()` and `xslt_stale()` allow you to create and destroy a stylesheet dynamically. Function `xmlupdate()` is convenient for very simple "search-and replace" transformations.

15.9.5. XSLT Examples

This section covers some examples of applying XSLT to various stored data.

Example 15.23. XSLT example

```
create procedure xml_view_string (in _view varchar)
{
  declare _body any;
  declare _pf varchar;
  _body := string_output ();
```

```
  http ('<document>', _body);
  _pf := concat ('DB.DBA.http_view_', _view);
  call (_pf) (_body);
  http ('</document>', _body);

  return (string_output_string (_body));
}

create procedure xslt_view (in v varchar, in xst varchar)
{

  declare str, r varchar;
  xslt_sheet (xst, xtree_doc (file_to_string (xst)));
  str := xml_view_string (v);
  r := xslt (xst, xtree_doc (str));
  declare str_out any;
  str_out := string_output ();
  http_value (r, 0, str_out);
  string_to_file ('xslt.out', string_output_string (str_out), 0);
}
```

These functions will take the serialized text of an XML view created with CREATE XML VIEW, transform it with a stylesheet and store the result into a file.

The first function returns a string containing the text generated as the value of the XML view specified as argument. It calls the serialization function, which is

```
DB.DBA.http_view_ +
view name
```

. This function writes the text into the string output stream passed to it. Note that the function wraps the text inside a *<document>* element in order to make it well-formed, since the view serialization function will emit multiple top-level elements, one for each selected row in the root table in the XML view.

The `xslt_view()` function first defines the style sheet, which it takes from a file in this case. The `xslt_sheet()` function is called with the name and root element of the parsed file.

The function next gets the string to process, parses it as XML, and converts the parse tree into an entity object. This is then passed to the `xslt()` function. The result is another entity object. This is finally serialized as XML text and written into the file *xslt.out*.

These examples show how to parse and serialize XML using varchars, string output streams and the entity data type.

The central points are:

1. The XML entity object is a reference to a particular element of a parsed XML document, The underlying document is only visible through this reference. A string is converted into such a document and a reference to the document's root is returned by the

   ```
   xtree_doc (string)
   ```

   function. In previous versions of Virtuoso, the combination

   ```
   xml_tree_doc (xml_tree (string))
   ```

   was used for this purpose; this combination still works to maintain backwards compatibility, but using xtree_doc() is preferable.
2. A string output is used to capture the serialization of data by the `http_value()` function. If the string output were not specified the data would be sent to the HTTP client directly, if we were running in an HTTP request context.

15.9.6. XPath Function Extensions for XSLT

Virtuoso provides a way to extend the abilities of the XSLT processor by creating user defined XPath functions. The functions `xpf_extension()` and `xpf_extension_remove()` allow addition and removal of XPath extension functions.

15.9.7. Status Of XSLT And XPath Implementation

Items marked with (*) are currently not implemented.

XPath Axes

ancestor::
ancestor-or-self::
attribute::
child::
descendant::
descendant-or-self::
following::
following-sibling::
namespace:: (*)
parent::
preceding::
preceding-sibling::
self::

Node Functions

number position ()
number count (node-set)
string local-name (node-set?)
string namespace-uri (node-set?)
string name (node-set?)
boolean lang (string)

Node Set Functions

node-set document (string, string?, integer?, name?, name?, string?)
node-set document-literal (string, string?, integer?, name?, name?, string?)
node-set id (object)
node-set append (object, object, ..., object)

String Functions

string string (object?)
string serialize (object?)
string format-number (number, string, string?)
string concat (object, object, ..., object)
number string-length (string?)
boolean contains (string, string)
boolean text-contains (string, string)
boolean starts-with (string, string)
boolean ends-with (string, string)
string substring (string, number, number?)
string substring-before (string, string)
string substring-after (string, string)

string normalize-space (string?)
string translate (string, string, string)
string replace (string, string, string)

Boolean Functions

boolean boolean (object)
boolean true ()
boolean false ()
boolean not (boolean)
boolean and (boolean, boolean, ..., boolean)
boolean or (boolean, boolean, ..., boolean)
boolean some (name, boolean, node-set)
boolean every (name, boolean, node-set)
boolean function_available (name)

Number Functions

number number (object?)
number floor (number)
number ceiling (number)
number round (number)
number sum (number)
number avg (number)
number max (number)
number min (number)

XSLT-specific Functions

current ()
function-available (string)
unparsed-entity-uri (string)
system-property (string)

XSLT 1.0 Elements and Attributes

xsl:apply-imports
xsl:apply-templates
xsl:attribute
xsl:attribute-set (*)
xsl:call-template
xsl:choose
xsl:comment
xsl:copy
xsl:copy-of
xsl:decimal-format
xsl:element
xsl:fallback (*)
xsl:for-each
xsl:if
xsl:import
xsl:include
xsl:key
xsl:message

xsl:namespace-alias

xsl:number NOTE: formatting and level="any" is not implemented

xsl:otherwise

xsl:output

xsl:param

xsl:preserve-space (*)

xsl:processing-instruction

xsl:sort

xsl:strip-space

xsl:stylesheet

xsl:template

xsl:text

xsl:transform (*) NOTE: use xsl:stylesheet instead

xsl:value-of

xsl:variable

xsl:when

xsl:with-param

15.10. XMLType

XMLType is a predefined type for representing XML entities as UDT objects. This is compatible with Oracle9i and later. You can get better performance and flexibility by using use plain built-in functions that directly operate with XML entities. All predefined member functions of XMLType for extracting fragments from an XMLType are actually wrappers for `xpath_eval()` built-in function. You can declare a column of XMLType but the actual type of the created column will be "LONG XML". Thus any XML-related feature that can be used for "LONG XML" column will work with XMLType. E.g. WITH SCHEMA constraint allows you to force stored values to match a particular schema; XML free text index can accelerate search for documents by content etc.

XMLType behaves like any user-defined type, with the only difference in type conversion rules. If an XML entity is passed as an argument instead of an instance of XMLType, a new instance of XMLType is created by a constructor that takes the entity as an argument. Similarly, functions that accept XML entities as arguments can also accept an instance of XMLType as an actual value of argument.

The following example creates a table with an XMLType column, put two records there and performs a simple search:

```
CREATE TABLE Xml_tab ( xmlval XMLType);

INSERT INTO Xml_tab VALUES (
   xmltype('<?xml version="1.0"?>
               <EMP>
                  <EMPNO>221</EMPNO>
                  <ENAME>John</ENAME>
               </EMP>'));

INSERT INTO Xml_tab VALUES (
   xmltype('<?xml version="1.0"?>
               <PO>
                  <PONO>331</PONO>
                  <PONAME>PO_1</PONAME>
               </PO>'));

-- now extract the numerical values for the employee numbers

SELECT e.xmlval.extract('//EMPNO/text()').getNumVal() as empno
   FROM Xml_tab
   WHERE e.xmlval.existsnode('/EMP/EMPNO')  = 1;
```

To create a new instance of XMLType, the constructor `XMLType()` or a function `createXML()` is used.

Virtuoso can perform XPATH search in XMLType instances: `extract()` and `existsNode()` member functions are convenient for simple searches and any built-in XPATH functions like `xpath_eval()` or `xquery_eval()` can handle

XMLType parameters instead of XML entity parameters. An application can use `getStringVal()` , `getNumVal()` and `getColbVal()` member functions to convert found node to strings, numbers or an XML source text of the node.

Instances of the type can store the URL of the XML schema to which they should conform. This URL can be specified when an instance is constructed; Once an instance is created, its schema URL cannot be changed but a modified copy can be created by `createSchemaBasedXML()` and `createNonSchemaBasedXML()` member functions; `isSchemaBased()` and `getSchemaURL()` member functions check whether the given instance is schema based or not and what particular schema is used.

An schema based XMLType instance can be validated against its schema; If it has been validated against its schema once with no errors detected then a special "VALIDATED" flag is set in the instance indicating that there is no need to validate it again. `schemaValidate()` member function performs the validation, `isSchemaValidated()` queries the "VALIDATED" flag and `setSchemaValidated()` changes the "VALIDATED" flag if application needs optimization tricks.

In addition, any instance can be validated against an arbitrary schema via member function `isSchemaValid()` . Built-in function `xml_validate_schema()` may accept instance of XMLType as its first argument, providing even more validation functionality.

15.11. Changing XML entities in DOM style

An application may need to perform small irregular changes in an XML document. It can be time-consuming to create a modified copy of a whole document for making each small change. It can be also hard to express condition for making the change in terms of XPath in order to use XSLT or XQuery.

Virtuoso offers a way to modify an existing XML entity by local operations like "insert a subtree before current node" or "replace one subtree with a given one". These operations are called DOM operations because they are similar to JavaScript operations with the Document Object Model.

DOM operations are the only operations that change an existing XML document instead of creating a new modified document. It is very important to understand that one document can be shared between many XML entities so changes made via one entity may affect data accessible via other entities that share the same document. Every XML document has special flags to "lock" its content and thus prevent the content from being changed. E.g. XSLT and XQuery processors optimize access to the source document in hope that every access to any entity will produce the same result during the whole run of the processor. If an extension function will modify the source document in the middle then the result of the processing is probably corrupted and this error may be very hard to detect and debug. To prevent this sort of problems, the XSLT processor locks the source at start and unlock at end so any attempt to modify the source during the run will signal an error. The document can be locked in the following cases:

- The document is the source of the XSLT processing that is in progress.
- The document contains the context node that is passed to the XQuery processor at start.
- The document is used as a external generic entity in some other document.
- The document is created by the call of XPath function doc, document or document-literal during the processing of some other document and that document is still in use.

One additional issue of using DOM operations is that any DOM operation invalidates DTD and schema validation information that was associated with the document before the operation.

- **Schema validation status ("is validated" flag).** E.g. if an schema based instance of XMLType is validated against its schema and thus it has set its "is validated" flag to 1 then the flag is to be reset to zero if the XML entity of this instance is modified by DOM operation, because there is no guarantee that the modified document is still valid against the declared schema; if the application will repeat the validation and it is successful then the flag will be set to 1 again.
- **DTD-based processing of ID attributes.** A validating XML parser can build special dictionaries of ID attributes when it reads a document. These dictionaries are used by `id` XPATH function and "pointer operator" of XQuery to find a node labeled by a given ID value without searching the whole document. Any change in document tree invalidates data in these dictionaries so any access to them will signal an error.

15.11.1. Composing Document Fragments From DOM Function Arguments

When a DOM modification function adds some nodes to the tree, the content of these nodes comes from values of arguments of the function. In the most common case, some generic entity is created from one or more arguments and the function adds the content of the entity into the appropriate place of the document. This process is regulated by a small set of very intuitive rules.

All data to be inserted are calculated before any modification of the document is started. For example it is safe to insert into the document a fragment of this document, even if it contains the point of insertion; such operation will never cause infinite loops.

If an instance of XMLType is passed then its internal XML entity is used.

If an argument is neither an XML entity nor a string then it is cast to the string first.

If an argument is NULL then it is fully ignored, as if there is no such argument at all.

A root node of a document can not be added into the tree of other document. Instead, every child node of the root (i.e. every top-level node of the document) is added to the target document, the order of child nodes will be preserved. An XML document usually contains only one top-level element node, but it also can contain comment and processing instruction nodes at top-level. Depending on the application, such comments and instructions may be inappropriate in the middle of the modified document. Thus it is worth to check what the application should insert: the root node with everything inside or only a top-level element node.

If some function accepts a list of arguments as list of values to be inserted, these values will be concatenated into a single generic entity. It is important to remember that the XML document can not contain two neighbour text nodes. If two consecutive arguments are strings then a single node that is a concatenation of these string is added. If the value of one argument is a string and the value of the next argument is a tree fragment that starts with a string then this pair of strings will also be replaced with a concatenation of these strings. When a function inserts a text after some existing text node of the XML document, no new node is created and the existing text node is replaced with the text node that is the concatenation of old and new texts.

In general, you should not expect that adding N values into the document will add N new nodes to the document. It can add a lesser number of nodes due to ignoring NULLs and concatenating some texts. It can also add a greater number of nodes if some values are root nodes with many children.

Chapter 16. RDF Data Access and Data Management

Abstract

Starting with version 4.5, Virtuoso provides built-in support for SPARQL, the standard query language for RDF and the semantic web. Adoption of SPARQL with Virtuoso is effortless, as any existing SQL client applications and stored procedures can take advantage of SPARQL simply by using it in the place of or inside SQL queries. Additionally, Virtuoso offers the standard SPARQL protocol to HTTP clients. From version 5.0.7, Virtuoso can be used as the RDF store/query processor of the Jena and Sesame RDF frameworks.

This chapter discusses Virtuoso's RDF triple storage and query capabilities. This discusses storing RDF data as well as mapping existing relational data into RDF for SPARQL access. Numerous SPARQL language extensions and standard compliance are covered.

In this chapter SPARQL and SPASQL are used as siblings.

See Also:

- Virtuoso ODBC RDF extensions for SPASQL
- Geometry Data Types and Spatial Index Support

16.1. Data Representation

This section covers how Virtuoso stores RDF triples. The IRI_ID built-in data type is introduced, along with the default table structures used for triple persistency. These details are mostly hidden from users of RDF, thus this section is not necessary reading for typical use of Virtuoso with RDF.

16.1.1. IRI_ID Type

The central notion of RDF is the IRI, or URI, which serves as the globally unique label of named nodes. The subject and predicate of a triple are always IRI's and the object may be an IRI or any other XML Schema scalar data type. In any case, an IRI is always distinct from any instance of any other data type.

Virtuoso supports a native IRI_ID data type, internally an unsigned 32 bit or unsigned 64 bit integer value. Small databases can use 32 bit values but if database becomes big then the administrator should execute `DB.DBA.RDF_64BIT_UPGRADE ()` procedure that will switch to 64-bit values. This procedure takes time so if it is known in advance that the database will grow to billions of nodes then it could be convenient to upgrade it while it is empty. An IRI_ID is never equal to any instance of any other type.

Thus, the object column of a table storing triples can be declared as ANY and IRI values will be distinguishable without recourse to any extra flag and IRI's will naturally occupy their own contiguous segment in the ANY type collation sequence. Indices can be defined over such columns. An IRI_ID is never automatically cast into any other type nor any other type into IRI_ID.

The functions iri_id_num (in i IRI_ID) and iri_id_from_num (in n INT) convert between signed integers and IRI_ID's. The function isiri_id (in i any) returns nonzero if the argument is of type IRI_ID, zero otherwise.

The syntax for an IRI_ID literal is *#i<NNN>* or *#ib<NNN>* , where *<NNN>* is up to 20 decimal digits. *#i12345* is equal to *iri_id_from_num (12345)* and *#ib12345* is equal to *iri_id_from_num (12345) + min_64bit_bnode_iri_id ()* .

When received by a SQL client application, the ODBC driver or interactive SQL will bind an IRI_ID to a character buffer, producing the *#i<NNN>* syntax. When passing IRI_ID's from a client, one can pass an integer and use the iri_id_from_num () function in the statement to convert server side. A SQL client will normally not be exposed to IRI_ID's since the SPARQL implementation returns IRI's in their text form, not as internal id's. These will however be seen if reading the internal tables directly.

Note

Nobody, even DBA, should write directly to internal RDF tables, because some data from that tables are cached in a special way and cache is not automatically updated when content of tables has changed.

Example

The following example demonstrates IRI type usage as Virtuoso PL function parameter:

```
SQL>create procedure vs_property_label (in _uri varchar)
  {
    declare res varchar;
    result_names (res);
    for (sparql define input:storage "" select distinct ?graph_rvr_fixed where { graph `iri(?:_uri)` { ?q
    do {
```

```
      result ("graph_rvr_fixed");
    }
  }
;
Done. -- 0 msec.

SQL>select vs_property_label('http://www.openlinksw.com/schemas/virtrdf#');
res
VARCHAR
_______________________________________________________________________________

http://demo.openlinksw.com/Northwind
http://demo.openlinksw.com/tpch
http://demo.openlinksw.com/tpcd
http://demo.openlinksw.com/bsbm
http://demo.openlinksw.com/tutorial/Northwind
http://demo.openlinksw.com/thalia
http://demo.openlinksw.com/tutorial_view
http://demo.openlinksw.com/ecrm
http://demo.openlinksw.com/sys
http://demo.openlinksw.com/Doc
http://demo.openlinksw.com/informix/stores_demo
http://demo.openlinksw.com/oraclehr
http://demo.openlinksw.com/db2sample
http://demo.openlinksw.com/ingrestut
http://demo.openlinksw.com/sybasepubs2
http://demo.openlinksw.com/MSPetShop#
http://demo.openlinksw.com/oracle#
http://demo.openlinksw.com/progress/isports
http://demo.openlinksw.com/wpl_v
http://demo.openlinksw.com/mw_v
http://demo.openlinksw.com/drupal_v
0

22 Rows. -- 241 msec.
```

16.1.2. RDF_BOX Type

While strings, numbers, dates and XML entities are "native" SQL datatypes, RDF literal with non-default type or language have no exact matches among standard SQL types. Virtuoso introduces a special data type called "RDF_BOX" in order to handle that cases. Instance of RDF_BOX consists of data type, language, the content (or beginning characters of a long content) and a possible reference to DB.DBA.RDF_OBJ table if the object is too long to be held in-line in some table or should be outlined for free-text indexing.

Usually applications do not need to access internals of an RDF boxes. This datatype is used in system tables but almost all SPARQL and RDF operations use standard SQL datatypes for arguments and return values.

16.1.3. RDF_QUAD and other tables

The main tables of the default RDF storage system are:

```
create table DB.DBA.RDF_QUAD (
  G IRI_ID,
  S IRI_ID,
  P IRI_ID,
  O any,
  primary key (G,S,P,O) );
create bitmap index RDF_QUAD_OGPS on DB.DBA.RDF_QUAD (O, G, P, S);
```

Each triple (more correctly, each quad) is represented by one row in RDF_QUAD. The columns represent the graph, subject, predicate and object. The IRI_ID type columns reference RDF_IRI, which translates the internal id to the external name of the IRI. The O column is of type ANY. If the O value is a non-string SQL scalar, such as a number or date or IRI, it is stored in its native binary representation. If it is a "very short" string (20 characters or less), it is also stored "as is". Long strings and RDF literal with non-default type or language are stored as RDF_BOX values. Instance of rdf_box consists of data type, language, the content (or beginning characters of a long content) and a possible reference to RDF_OBJ if the object is too long to be held in-line in this table or should be outlined for free-text indexing.

```
create table DB.DBA.RDF_PREFIX (
```

```
  RP_NAME varchar primary key,
  RP_ID int not null unique );
create table DB.DBA.RDF_IRI (
  RI_NAME varchar primary key,
  RI_ID IRI_ID not null unique );
```

These two tables store a mapping between internal IRI id's and their external string form. A memory-resident cache contains recently used IRIs to reduce access to this table. Function id_to_iri (in id IRI_ID) returns the IRI by its ID. Function iri_to_id (in iri varchar, in may_create_new_id) returns an IRI_ID for given string; if the string is not used before as an IRI then either NULL is returned or a new ID is allocated, depending on the second argument.

```
create table DB.DBA.RDF_OBJ (
  RO_ID integer primary key,
  RO_VAL varchar,
  RO_LONG long varchar,
  RO_DIGEST any
)
create index RO_VAL on DB.DBA.RDF_OBJ (RO_VAL)
create index RO_DIGEST on DB.DBA.RDF_OBJ (RO_DIGEST)
;
```

When an O value of RDF_QUAD is longer than a certain limit or should be free-text indexed, the value is stored in this table. Depending on the length of the value, it goes into the varchar or the long varchar column. The RO_ID is contained in rdf_box object that is stored in the O column. Still, the truncated value of O can be used for determining equality and range matching, even if < and > of closely matching values need to look at the real string in RDF_OBJ. When RO_LONG is used to store very long value, RO_VAL contains a simple checksum of the value, to accelerate search for identical values when the table is populated by new values.

```
create table DB.DBA.RDF_DATATYPE (
    RDT_IID IRI_ID not null primary key,
    RDT_TWOBYTE integer not null unique,
    RDT_QNAME varchar );
```

The XML Schema data type of a typed string O represented as 2 bytes in the O varchar value. This table maps this into the broader IRI space where the type URI is given an IRI number.

```
create table DB.DBA.RDF_LANGUAGE (
    RL_ID varchar not null primary key,
    RL_TWOBYTE integer not null unique );
```

The varchar representation of a O which is a string with language has a two byte field for language. This table maps the short integer language id to the real language name such as 'en', 'en-US' or 'x-any'.

Note that unlike datatype names, language names are not URIs.

A short integer value can be used in both RDF_DATATYPE and RDF_LANGUAGE tables for two different purposes. E.g. an integer 257 is for 'unspecified datatype' as well as for 'unspecified language'.

16.1.4. Short, Long and SQL Values

When processing an O, the SPARQL implementation may have it in one of three internal formats, called "valmodes". The below cases apply for strings:

The short format is the format where an O is stored in RDF_QUAD.

The long value is similar to short one but an rdf_box object, that consists of six fields:

- short integer id of type referencing RDT_TWOBYTE, 257 if the type is not specified,
- the string as inlined in O or as stored in RO_VAL or RO_LONG,
- the RO_ID if the string is from RDF_OBJ (otherwise zero),
- the short integer id of language referencing RL_TWOBYTE, 257 if the language is not specified,
- flag whether the stored string value is complete or it is only the beginning that is inlined in O.

The SQL value is the string as a narrow string representing the UTF8 encoding of the value, stripped of data type and language tag.

The SQL form of an IRI is the string. The long and short forms are the IRI_ID referencing RU_IRI_ID of RDF_URL.

For all non-string, non-IRI types, the short, long and SQL values are the same SQL scalar of the appropriate native SQL type. A SQL host variable meant to receive an O should be of the ANY type.

The SPARQL implementation will usually translate results to the SQL format before returning them. Internally, it uses the shortest possible form suited to the operation. For equalities and joining, the short form is always good. For range comparisons, the long form is needed etc. For arithmetic, all three forms will do since the arguments are expected to be numbers which are stored as their binary selves in O, thus the O column unaltered and uncast will do as an argument of arithmetic or numeric comparison with, say, SQL literal constants.

16.1.5. Programatically resolving DB.DBA.RDF_QUAD.O to SQL

This section describes how to resolve programatically the internal representation of DB.DBA.RDF_QUAD.O to its SQL value.

When operating over RDF_QUAD table directly, in order to transform all values obtained from column O to the explicit SQL type in a programmatic way, should be used the following hints depending on the case:

- The SQL value can be extracted as

  __ro2sq(O)

  .
- The datatype can be extracted as

  DB.DBA.RDF_DATATYPE_OF_OBJ (O)

  if IRI_ID of the type is enough or

  __ro2sq (DB.DBA.RDF_DATATYPE_OF_OBJ(O))
- The language can be extracted as

  DB.DBA.RDF_LANGUAGE_OF_OBJ (O)

  .

It could be helpful to be created an Linked Data View for a custom table with formats rdfdf:default or rdfdf:default-nullable for columns similar to O, and let SPARQL perform the rest.

To track SPARQL, use the following functions:

```
select sparql_to_sql_text ('query text here without a leading SPARQL keyword and trailing semicolon')
```

or

```
string_to_file ('filename.sql', sparql_to_sql_text ('query text'), -2);
```

So for example to track the following SPARQL query:

```
SPARQL define input:storage ""
select distinct ?graph_rvr_fixed
from <http://www.openlinksw.com/schemas/virtrdf#>
where { ?qmv virtrdf:qmGraphRange-rvrFixedValue ?graph_rvr_fixed }
```

execute

```
SQL>select sparql_to_sql_text('define input:storage "" select distinct ?graph_rvr_fixed from <http://www.
callret
VARCHAR
_______________________________________________________________________________
```

```
 SELECT  __ro2sq ("s-1-0_rbc"."graph_rvr_fixed") AS "graph_rvr_fixed" FROM (SELECT DISTINCT __rdf_sqlval_
rvr_fixed */ /*]retval*/ ) AS /*tmpl*/ "graph_rvr_fixed"
    FROM DB.DBA.RDF_QUAD AS "s-1-1-t0"
    WHERE /* field equal to URI ref */
        "s-1-1-t0"."G" = __i2idn ( /* UNAME as sqlval */ __box_flags_tweak ( 'http://www.openlinksw.com/s
        AND  /* field equal to URI ref */
        "s-1-1-t0"."P" = __i2idn ( /* UNAME as sqlval */ __box_flags_tweak ( 'http://www.openlinksw.com/s
1))
OPTION (QUIETCAST)) AS "s-1-0_rbc"

1 Rows. -- 321 msec.
```

or

```
SQL>string_to_file ('mytest.sql', sparql_to_sql_text ('define input:storage "" select distinct ?graph_rvr
```

As result will be created file with the given name, i.e. mytest.sql and its content should be:

```
 SELECT  __ro2sq ("s-1-0_rbc"."graph_rvr_fixed") AS "graph_rvr_fixed" FROM (SELECT DISTINCT __rdf_sqlval_
    FROM DB.DBA.RDF_QUAD AS "s-1-1-t0"
    WHERE /* field equal to URI ref */
        "s-1-1-t0"."G" = __i2idn ( /* UNAME as sqlval */ __box_flags_tweak ( 'http://www.openlinksw.com/s
        AND  /* field equal to URI ref */
        "s-1-1-t0"."P" = __i2idn ( /* UNAME as sqlval */ __box_flags_tweak ( 'http://www.openlinksw.com/s
OPTION (QUIETCAST)) AS "s-1-0_rbc"
```

16.1.6. Special Cases and XML Schema Compatibility

We note that since we store numbers as the equivalent SQL binary type, we do not preserve the distinction of byte, boolean etc. These all become integer. If preserving such detail is for some reason important, then storage as a typed string is possible but is not done at present for reasons of compactness and performance.

16.1.7. SQL Compiler Support - QUIETCAST option

The type cast behaviors of SQL and SPARQL are different. SQL will generally signal an error when an automatic cast fails. For example, a string can be compared to a date column if the string can be parsed as a date but otherwise the comparison should signal an error. In SPARQL, such situations are supposed to silently fail. Generally, SPARQL is much more relaxed with respect to data types.

These differences will be specially noticed if actual SQL data is processed with SPARQL via some sort of schema mapping translating references to triples into native tables and columns.

Also, even when dealing with the triple-oriented RDF_QUAD table, there are cases of joining between S and O such that the O can be a heterogeneous set of IRI's and other data whereas the S is always an IRI. The non-IRI to IRI comparison should not give cast errors but should silently fail. Also, in order to keep queries simple and easily optimizable, it should not be necessary to introduce extra predicates for testing if the O is n IRI before comparing with the S.

Due to these considerations, Virtuoso introduces a SQL statement option called QUIETCAST. When given in the OPTION clause of a SELECT, it switches to silent fail mode for automatic type casting.

The syntax is as follows:

```
SELECT ...
FROM .... OPTION (QUIETCAST)
```

This option is automatically added by the SPARQL to SQL translator. The scope is the enclosing procedure body.

16.1.8. Dynamic Renaming of Local IRI's

There are cases where it is desirable to have IRI's in RDF storage that will change to reflect a change of the host name of the containing store. This is specifically true of DAV resource metadata for local DAV resources. Such IRI's must be stored prefixed with `local:`.

If a user application makes statements with such a URI, then these statements will be returned with local: substituted with a prefix taken from the context as described below.

When returning IRI's from id's, this prefix is replaced by the Host header of the HTTP request and if not running with HTTP, with the DefaultHost from URIQA. This behavior is always in effect.

When converting strings to IRI id's, the `local:` prefix may or may not be introduced depending on ini file and other context factors. If DynamicLocal defined in the [URIQA] section of the Virtuoso INI file is on and the host part of the IRI matches the Host header of the HTTP request in context or the DefaultHost if outside of HTTP context, then this is replaced with local: before looking up the IRI ID. Even if DynamicLocal is not on and the `local:` prefix occurs in the IRI string being translated to id, the translating the IRI_ID back to the IRI name will depend on the context as described above.

The effects of DynamicLocal = 1 can be very confusing since many names can refer to the exact same thing. For example, if the DefaultHost is dbpedia.org, `iri_to_id ('http://dbpedia.org/resource/Paris') = iri_to_id ('local:///resource/Paris)` is true and so is `'http://dbpedia.org/resource/Paris' = id_to_iri (iri_to_id ('local://resource/Paris'))` These hold in a SQL client context, i.e. also when connected through RDF frameworks like Jena or Sesame. When running a SPARQL protocol request, the Host: header influences the behavior, likewise when using web interactive SQL in Conductor. Also be careful when loading RDF files that may have URI's corresponding to the local host name.

16.2. SPARQL

16.2.1. SPARQL Implementation Details

Virtuoso's RDF support includes in-built support for the SPARQL query language. It also includes a number of powerful extensions that cover path traversal and business intelligence features. In addition, there is in-built security based on Virtuoso's support for row level policy-based security, custom authentication, and named graphs.

The current implementation does not support some SPARQL features:

- Unicode characters in names are not supported.
- Comments inside SPARQL queries are not supported when the query is inlined in SQL code.

On the other hand, Virtuoso implements some extensions to SPARQL:

- SPARUL statements, such as

  insert , *modify* , *load* etc, are supported.
- The SPARQL compiler can be configured using

  define ...

  clauses, e.g.

  define output:valmode "LONG"

  .
- Expressions are allowed in triple patterns, both in a

  where

  clause and in constructor patterns. Such expressions are delimited by backquotes.
- Expressions are allowed in select statement result lists.
- Parameters can be passed to the query from outside, using

  ?:variablename

  syntax.
- Aggregate functions are supported.
- Subqueries may appear where group patterns are allowed.

- A set of operators has been added to configure the mapping of relational data to RDF (aka Linked Data Views).

The following listing shows the SPARQL grammar expressed in BNF, including all Virtuoso extensions but excluding rules for the syntax of each lexical element. Rule numbers in square brackets are from W3C normative SPARQL grammar. An asterisk indicates that the rule differs from the W3C grammar due to Virtuoso extensions - *[Virt]* means that the rule is Virtuoso-specific, *[DML]* indicates a data manipulation language extension from SPARUL.

```
[1]*    Query             ::=  Prolog ( QueryBody | SparulAction* | ( QmStmt ('.' QmStmt)* '.'? ) )
[1]     QueryBody         ::=  SelectQuery | ConstructQuery | DescribeQuery | AskQuery
[2]*    Prolog            ::=  Define* BaseDecl? PrefixDecl*
[Virt]  Define            ::=  'DEFINE' QNAME (QNAME | Q_IRI_REF | String )
[3]     BaseDecl          ::=  'BASE' Q_IRI_REF
[4]     PrefixDecl        ::=  'PREFIX' QNAME_NS Q_IRI_REF
[5]*    SelectQuery       ::=  'SELECT' 'DISTINCT'? ( ( Retcol ( ','? Retcol )* ) | '*' )
                          DatasetClause* WhereClause SolutionModifier
[6]     ConstructQuery    ::=  'CONSTRUCT' ConstructTemplate DatasetClause* WhereClause SolutionModifier
                          DatasetClause* WhereClause? SolutionModifier
[8]     AskQuery          ::=  'ASK' DatasetClause* WhereClause
[9]     DatasetClause     ::=  'FROM' ( DefaultGraphClause | NamedGraphClause )
[10]*   DefaultGraphClause       ::=  SourceSelector SpongeOptionList?
[11]*   NamedGraphClause         ::=  'NAMED' SourceSelector SpongeOptionList?
[Virt]  SpongeOptionList          ::=  'OPTION' '(' ( SpongeOption ( ',' SpongeOption )* )? ')'
[Virt]  SpongeOption      ::=  QNAME PrecodeExpn
[Virt]  PrecodeExpn       ::=  Expn       (* Only global variables can occur in Expn, local cannot *)
[13]    WhereClause       ::=  'WHERE'? GroupGraphPattern
[14]    SolutionModifier          ::=  OrderClause?
                          ((LimitClause OffsetClause?) | (OffsetClause LimitClause?))?
[15]    OrderClause       ::=  'ORDER' 'BY' OrderCondition+
[16]*   OrderCondition    ::=  ( 'ASC' | 'DESC' )?
                          ( FunctionCall | Var | ( '(' Expn ')' ) | ( '[' Expn ']' ) )
[17]    LimitClause       ::=  'LIMIT' INTEGER
[17]    LimitClause       ::=  'LIMIT' INTEGER
[18]    OffsetClause      ::=  'OFFSET' INTEGER
[18]    OffsetClause      ::=  'OFFSET' INTEGER
[19]*   GroupGraphPattern         ::=  '{' ( GraphPattern | SelectQuery ) '}'
[20]    GraphPattern      ::=  Triples? ( GraphPatternNotTriples '.'? GraphPattern )?
[21]*   GraphPatternNotTriples   ::=
                          QuadMapGraphPattern
                          | OptionalGraphPattern
                          | GroupOrUnionGraphPattern
                          | GraphGraphPattern
                          | Constraint
[22]    OptionalGraphPattern      ::=  'OPTIONAL' GroupGraphPattern
[Virt]  QuadMapGraphPattern       ::=  'QUAD' 'MAP' ( IRIref | '*' ) GroupGraphPattern
[23]    GraphGraphPattern         ::=  'GRAPH' VarOrBlankNodeOrIRIref GroupGraphPattern
[24]    GroupOrUnionGraphPattern          ::=  GroupGraphPattern ( 'UNION' GroupGraphPattern )*
[25]*   Constraint        ::=  'FILTER' ( ( '(' Expn ')' ) | BuiltInCall | FunctionCall )
[26]*   ConstructTemplate         ::=  '{' ConstructTriples '}'
[27]    ConstructTriples          ::=  ( Triples1 ( '.' ConstructTriples )? )?
[28]    Triples           ::=  Triples1 ( '.' Triples? )?
[29]    Triples1          ::=  VarOrTerm PropertyListNotEmpty | TriplesNode PropertyList
[30]    PropertyList      ::=  PropertyListNotEmpty?
[31]    PropertyListNotEmpty      ::=  Verb ObjectList ( ';' PropertyList )?
[32]*   ObjectList       ::=  ObjGraphNode ( ',' ObjectList )?
[Virt]  ObjGraphNode      ::=  GraphNode TripleOptions?
[Virt]  TripleOptions     ::=  'OPTION' '(' TripleOption ( ',' TripleOption )? ')'
[Virt]  TripleOption      ::=  'INFERENCE' ( QNAME | Q_IRI_REF | SPARQL_STRING )
[33]    Verb              ::=  VarOrBlankNodeOrIRIref | 'a'
[34]    TriplesNode       ::=  Collection | BlankNodePropertyList
[35]    BlankNodePropertyList     ::=  '[' PropertyListNotEmpty ']'
[36]    Collection        ::=  '(' GraphNode* ')'
[37]    GraphNode         ::=  VarOrTerm | TriplesNode
[38]    VarOrTerm         ::=  Var | GraphTerm
[39]*   VarOrIRIrefOrBackquoted  ::=  Var | IRIref | Backquoted
[40]*   VarOrBlankNodeOrIRIrefOrBackquoted        ::=  Var | BlankNode | IRIref | Backquoted
[Virt]  Retcol    ::=  ( Var | ( '(' Expn ')' ) | RetAggCall ) ( 'AS' ( VAR1 | VAR2 ) )?
[Virt]  RetAggCall        ::=  AggName '(', ( '*' | ( 'DISTINCT'? Var ) ) ')'
[Virt]  AggName   ::=  'COUNT' | 'AVG' | 'MIN' | 'MAX' | 'SUM'
[41]*   Var       ::=  VAR1 | VAR2 | GlobalVar | ( Var ( '+>' | '*>' ) IRIref )
[Virt]  GlobalVar         ::=  QUEST_COLON_PARAMNAME | DOLLAR_COLON_PARAMNAME
                          | QUEST_COLON_PARAMNUM | DOLLAR_COLON_PARAMNUM
[42]*   GraphTerm         ::=  IRIref | RDFLiteral | ( '-' | '+' )? NumericLiteral
```

```
                        | BooleanLiteral | BlankNode | NIL | Backquoted
[Virt]  Backquoted       ::=  '`' Expn '`'
[43]    Expn             ::=  ConditionalOrExpn
[44]    ConditionalOrExpn        ::=  ConditionalAndExpn ( '||' ConditionalAndExpn )*
[45]    ConditionalAndExpn       ::=  ValueLogical ( '&&' ValueLogical )*
[46]    ValueLogical     ::=  RelationalExpn
[47]*   RelationalExpn   ::=  NumericExpn
                        ( ( ('='|'!='|'<'|'>'|'<='|'>='|'LIKE') NumericExpn )
                        | ( 'IN' '(' Expns ')' ) )?
[49]    AdditiveExpn     ::=  MultiplicativeExpn ( ('+'|'-') MultiplicativeExpn )*
[50]    MultiplicativeExpn       ::=  UnaryExpn ( ('*'|'/') UnaryExpn )*
[51]    UnaryExpn        ::=   ('!'|'+'|'-')? PrimaryExpn
[58]    PrimaryExpn      ::=
                        BracketedExpn | BuiltInCall | IRIrefOrFunction
                        | RDFLiteral | NumericLiteral | BooleanLiteral | BlankNode | Var
[55]    IRIrefOrFunction         ::=  IRIref ArgList?
[52]*   BuiltInCall      ::=
                        ( 'STR' '(' Expn ')' )
                        | ( 'IRI' '(' Expn ')' )
                        | ( 'LANG' '(' Expn ')' )
                        | ( 'LANGMATCHES' '(' Expn ',' Expn ')' )
                        | ( 'DATATYPE' '(' Expn ')' )
                        | ( 'BOUND' '(' Var ')' )
                        | ( 'sameTERM' '(' Expn ',' Expn ')' )
                        | ( 'isIRI' '(' Expn ')' )
                        | ( 'isURI' '(' Expn ')' )
                        | ( 'isBLANK' '(' Expn ')' )
                        | ( 'isLITERAL' '(' Expn ')' )
                        | RegexExpn
[53]    RegexExpn        ::=  'REGEX' '(' Expn ',' Expn ( ',' Expn )? ')'
[54]    FunctionCall     ::=  IRIref ArgList
[56]*   ArgList  ::=  ( NIL | '(' Expns ')' )
[Virt]  Expns    ::=  Expn ( ',' Expn )*
[59]    NumericLiteral   ::=  INTEGER | DECIMAL | DOUBLE
[60]    RDFLiteral       ::=  String ( LANGTAG | ( '^^' IRIref ) )?
[61]    BooleanLiteral   ::=  'true' | 'false'
[63]    IRIref           ::=  Q_IRI_REF | QName
[64]    QName            ::=  QNAME | QNAME_NS
[65]*   BlankNode        ::=  BLANK_NODE_LABEL | ( '[' ']' )
[DML]   SparulAction     ::=
                        CreateAction | DropAction | LoadAction
                        | InsertAction | InsertDataAction | DeleteAction | DeleteDataAction
                        | ModifyAction | ClearAction
[DML]*  InsertAction     ::=
                        'INSERT' ( ( 'IN' | 'INTO ) 'GRAPH' ( 'IDENTIFIED' 'BY' )? )? PrecodeExpn
                        ConstructTemplate ( DatasetClause* WhereClause SolutionModifier )?
[DML]*  InsertDataAction         ::=
                        'INSERT' 'DATA' ( ( 'IN' | 'INTO ) 'GRAPH' ( 'IDENTIFIED' 'BY' )? )?
                        PrecodeExpn ConstructTemplate
[DML]*  DeleteAction     ::=
                        'DELETE' ( 'FROM' 'GRAPH' ( 'IDENTIFIED' 'BY' )? )? PrecodeExpn
                        ConstructTemplate ( DatasetClause* WhereClause SolutionModifier )?
[DML]*  DeleteDataAction         ::=
                        'DELETE' 'DATA' ( 'FROM' 'GRAPH' ( 'IDENTIFIED' 'BY' )? )?
                        PrecodeExpn ConstructTemplate
[DML]*  ModifyAction     ::=
                        'MODIFY' ( 'GRAPH' ( 'IDENTIFIED' 'BY' )? PrecodeExpn?
                        'DELETE' ConstructTemplate 'INSERT' ConstructTemplate
                        ( DatasetClause* WhereClause SolutionModifier )?
[DML]*  ClearAction      ::=  'CLEAR' ( 'GRAPH' ( 'IDENTIFIED' 'BY' )? PrecodeExpn )?
[DML]*  LoadAction       ::=  'LOAD' PrecodeExpn
                        ( ( 'IN' | 'INTO' ) 'GRAPH' ( 'IDENTIFIED' 'BY' )? PrecodeExpn )?
[DML]*  CreateAction     ::=  'CREATE' 'SILENT'? 'GRAPH' ( 'IDENTIFIED' 'BY' )? PrecodeExpn
[DML]*  DropAction       ::=  'DROP' 'SILENT'? 'GRAPH' ( 'IDENTIFIED' 'BY' )? PrecodeExpn
[Virt]  QmStmt           ::=  QmSimpleStmt | QmCreateStorage | QmAlterStorage
[Virt]  QmSimpleStmt     ::=
                        QmCreateIRIClass | QmCreateLiteralClass | QmDropIRIClass | QmDropLiteralClass
                        | QmCreateIRISubclass | QmDropQuadStorage | QmDropQuadMap
[Virt]  QmCreateIRIClass         ::=  'CREATE' 'IRI' 'CLASS' QmIRIrefConst
                        ( ( String QmSqlfuncArglist )
                        | ( 'USING' QmSqlfuncHeader ',' QmSqlfuncHeader ) )
[Virt]  QmCreateLiteralClass     ::=  'CREATE' 'LITERAL' 'CLASS' QmIRIrefConst
                        'USING' QmSqlfuncHeader ',' QmSqlfuncHeader QmLiteralClassOptions?
```

```
[Virt]  QmDropIRIClass   ::=  'DROP' 'IRI' 'CLASS' QmIRIrefConst
[Virt]  QmDropLiteralClass       ::=  'DROP' 'LITERAL' 'CLASS' QmIRIrefConst
[Virt]  QmCreateIRISubclass      ::=  'IRI' 'CLASS' QmIRIrefConst 'SUBCLASS' 'OF' QmIRIrefConst
[Virt]  QmIRIClassOptions        ::=  'OPTION' '(' QmIRIClassOption (',' QmIRIClassOption)* ')'
[Virt]  QmIRIClassOption         ::=
                        'BIJECTION'
                        | 'DEREF'
                        | 'RETURNS' STRING ('UNION' STRING)*
[Virt]  QmLiteralClassOptions    ::=  'OPTION' '(' QmLiteralClassOption (',' QmLiteralClassOption)* ')'
[Virt]  QmLiteralClassOption     ::=
                        ( 'DATATYPE' QmIRIrefConst )
                        | ( 'LANG' STRING )
                        | ( 'LANG' STRING )
                        | 'BIJECTION'
                        | 'DEREF'
                        | 'RETURNS' STRING ('UNION' STRING)*
[Virt]  QmCreateStorage  ::=  'CREATE' 'QUAD' 'STORAGE' QmIRIrefConst QmSourceDecl* QmMapTopGroup
[Virt]  QmAlterStorage   ::=  'ALTER' 'QUAD' 'STORAGE' QmIRIrefConst QmSourceDecl* QmMapTopGroup
[Virt]  QmDropStorage    ::=  'DROP' 'QUAD' 'STORAGE' QmIRIrefConst
[Virt]  QmDropQuadMap    ::=  'DROP' 'QUAD' 'MAP' 'GRAPH'? QmIRIrefConst
[Virt]  QmDrop   ::=  'DROP' 'GRAPH'? QmIRIrefConst
[Virt]  QmSourceDecl     ::=
                        ( 'FROM' QTABLE 'AS' PLAIN_ID QmTextLiteral* )
                        | ( 'FROM' PLAIN_ID 'AS' PLAIN_ID QmTextLiteral* )
                        | QmCondition
[Virt]  QmTextLiteral    ::=  'TEXT' 'XML'? 'LITERAL' QmSqlCol ( 'OF' QmSqlCol )? QmTextLiteralOptions?
[Virt]  QmTextLiteralOptions     ::=  'OPTION' '(' QmTextLiteralOption ( ',' QmTextLiteralOption )* ')'
[Virt]  QmMapTopGroup    ::=  '{' QmMapTopOp ( '.' QmMapTopOp )* '.'? '}'
[Virt]  QmMapTopOp       ::=  QmMapOp | QmDropQuadMap | QmDrop
[Virt]  QmMapGroup       ::=  '{' QmMapOp ( '.' QmMapOp )* '.'? '}'
[Virt]  QmMapOp          ::=
                        ( 'CREATE' QmIRIrefConst 'AS' QmMapIdDef )
                        | ( 'CREATE' 'GRAPH'? QmIRIrefConst 'USING' 'STORAGE' QmIRIrefConst QmOptions? )
                        | ( QmNamedField+ QmOptions? QmMapGroup )
                        | QmTriples1
[Virt]  QmMapIdDef       ::=  QmMapTriple | ( QmNamedField+ QmOptions? QmMapGroup )
[Virt]  QmMapTriple      ::=  QmFieldOrBlank QmVerb QmObjField
[Virt]  QmTriples1       ::=  QmFieldOrBlank QmProps
[Virt]  QmNamedField     ::=  ('GRAPH'|'SUBJECT'|'PREDICATE'|'OBJECT') QmField
[Virt]  QmProps          ::=  QmProp ( ';' QmProp )?
[Virt]  QmProp           ::=  QmVerb QmObjField ( ',' QmObjField )*
[Virt]  QmObjField       ::=  QmFieldOrBlank QmCondition* QmOptions?
[Virt]  QmIdSuffix       ::=  'AS' QmIRIrefConst
[Virt]  QmVerb           ::=  QmField | ( '[' ']' ) | 'a'
[Virt]  QmFieldOrBlank   ::=  QmField | ( '[' ']' )
[Virt]  QmField          ::=
                        NumericLiteral
                        | RdfLiteral
                        | ( QmIRIrefConst ( '(' ( QmSqlCol ( ',' QmSqlCol )* )? ')' )? )
                        | QmSqlCol
[Virt]  QmCondition      ::=  'WHERE' ( ( '(' SQLTEXT ')' ) | String )
[Virt]  QmOptions        ::=  'OPTION' '(' QmOption ( ',' QmOption )* ')'
[Virt]  QmOption         ::=  ( 'SOFT'? 'EXCLUSIVE' ) | ( 'ORDER' INTEGER ) | ( 'USING' PLAIN_ID )
[Virt]  QmSqlfuncHeader  ::=  'FUNCTION' SQL_QTABLECOLNAME QmSqlfuncArglist 'RETURNS' QmSqltype
[Virt]  QmSqlfuncArglist         ::=  '(' ( QmSqlfuncArg ( ',' QmSqlfuncArg )* )? ')'
[Virt]  QmSqlfuncArg     ::=  ('IN' | QmSqlId) QmSqlId QmSqltype
[Virt]  QmSqltype        ::=  QmSqlId ( 'NOT' 'NULL' )?
[Virt]  QmSqlCol         ::=  QmSqlId | spar_qm_sql_id
[Virt]  QmSqlId          ::=  PLAIN_ID | 'TEXT' | 'XML'
[Virt]  QmIRIrefConst    ::=  IRIref | ( 'IRI' '(' String ')' )
```

Example: Using OFFSET and LIMIT

Virtuoso uses a zero-based index for OFFSET. Thus, in the example below, the query returns 1000 rows starting from, and including, record 9001 of the result set. Note that the default value of the MaxSortedTopRows parameter in the [Parameters] section of the virtuoso.ini configuration file defaults to 10000, so in this example its value will need to have been increased beforehand.

```
SQL>SELECT ?name
ORDER BY ?name
OFFSET 9000
LIMIT 1000
```

LIMIT applies to the solution resulting from the graph patterns specified in the WHERE CLAUSE. This implies that SELECT and CONSTRUCT/DESCRIBE queries will behave a little differently. In the case of a SELECT, there is a straight translation i.e. LIMIT 4 implies 4 records in the result set. In the case of CONSTRUCTs where the solution is a graph (implying that the existence of duplicates and/or unbound variables is common) LIMIT is basically a maximum triples threshold of: [Solution Triples] x [LIMIT].

Example query:

```
SQL>SPARQL
prefix dct:<http://purl.org/dc/terms/>
prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#>

CONSTRUCT { ?resource dct:title ?title ;
                      a ?type }

FROM <http://msone.computas.no/graphs/inferred/classification>
FROM <http://msone.computas.no/graphs>
FROM <http://msone.computas.no/graphs/instance/nfi>
FROM <http://msone.computas.no/graphs/instance/mo>
FROM <http://msone.computas.no/graphs/ontology/mediasone>
FROM <http://msone.computas.no/graphs/vocab/mediasone>
FROM <http://msone.computas.no/graphs/inferred/nfi/realisation1>
FROM <http://msone.computas.no/graphs/inferred/mo/realisation1>
FROM <http://msone.computas.no/graphs/inferred/nfi/realisation2>
FROM <http://msone.computas.no/graphs/inferred/mo/realisation2>
FROM <http://msone.computas.no/graphs/inferred/agent-classification>
FROM <http://msone.computas.no/graphs/ontology/mediasone/agent>

WHERE {
  {
?resource a ?type .
  FILTER (?type = <http://www.w3.org/2002/07/owl#Class> ) .
  ?resource rdfs:label ?title .
  } UNION {
?resource a ?type .
  FILTER (?type in (
          <http://musicbrainz.org/mm/mm-2.1#Track> ,
          <http://www.csd.abdn.ac.uk/~ggrimnes/dev/imdb/IMDB#Movie> ,
          <http://xmlns.com/foaf/0.1/Image> ,
          <http://www.computas.com/mediasone#Text> ) ) .
  ?resource dct:title ?title .
  }
  FILTER regex(?title, "turi", "i")
}
ORDER BY ?title LIMIT 4 OFFSET 0
```

Example: Prevent Limits of Sorted LIMIT/OFFSET query

The DBpedia SPARQL endpoint is configured with the following INI setting:

```
MaxSortedTopRows = 40000
```

The setting above sets a threshold for sorted rows. Thus, when using basic SPARQL queries that include OFFSET and LIMIT the following query will still exist the hard limit set in the INI:

```
DEFINE sql:big-data-const 0
SELECT DISTINCT  ?p ?s
FROM <http://dbpedia.org>
WHERE
  {
    ?s ?p <http://dbpedia.org/resource/Germany>
  }
ORDER BY ASC(?p)
OFFSET  40000
LIMIT   1000
```

returns the following error on execution:

```
HttpException: 500 SPARQL Request Failed
```

```
Virtuoso 22023 Error SR353: Sorted TOP clause specifies more then 41000 rows to sort.
Only 40000 are allowed.
Either decrease the offset and/or row count or use a scrollable cursor
```

To prevent the problem outlined above you can leverage the use of subqueries which make better use of temporary storage associated with this kind of quest. An example would take the form:

```
SELECT ?p ?s
WHERE
  {
    {
      SELECT DISTINCT ?p ?s
      FROM <http://dbpedia.org>
      WHERE
        {
          ?s ?p <http://dbpedia.org/resource/Germany>
        } ORDER BY ASC(?p)
    }
  }
OFFSET 50000
LIMIT 1000
```

SPARQL and XQuery Core Function Library

In the current implementation, the XQuery Core Function Library is not available from SPARQL.

As a temporary workaround, string parsing functions are made available, because they are widely used in W3C DAWG examples and the like. They are:

```
xsd:boolean (in strg any) returns integer
xsd:dateTime (in strg any) returns datetime
xsd:double (in strg varchar) returns double precision
xsd:float (in strg varchar) returns float
xsd:integer (in strg varchar) returns integer
```

(assuming that the query contains the declaration: 'PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>')

16.2.2. Query Constructs

Starting from Version 5.0, Virtuoso supports filtering RDF objects triples by a given predicate.

Examples

The boolean functions bif:contains, bif:xcontains, bif:xpath_contains and bif:xquery_contains can be used for objects that come from Linked Data Views as well as for regular "physical" triples. Each of these functions takes two arguments and returns a boolean value. The first argument is a local variable which should also be used as an object field in the group pattern where the filter condition is placed.

In order to execute the examples below please run these commands:

```
SQL>SPARQL CLEAR GRAPH <http://MyTest.com>;
DB.DBA.RDF_QUAD_URI_L ('http://MyTest.com', 'sxml1', 'p_all1', xtree_doc ('<Hello>world</Hello>'));
DB.DBA.RDF_QUAD_URI_L ('http://MyTest.com', 'sxml2', 'p_all2', xtree_doc ('<Hello2>world</Hello2>'));
DB.DBA.RDF_QUAD_URI_L ('http://MyTest.com', 'nonxml1', 'p_all3', 'Hello world');
VT_INC_INDEX_DB_DBA_RDF_OBJ();
DB.DBA.RDF_OBJ_FT_RULE_ADD ('http://MyTest.com', null, 'My test RDF Data');
```

bif:contains

```
SQL>SPARQL
SELECT *
FROM <http://MyTest.com>
WHERE { ?s ?p ?o . ?o bif:contains "world" };

s             p         o
VARCHAR       VARCHAR   VARCHAR
```

```
sxml1          p_all1    <Hello>world</Hello>
nonxml1        p_all3    Hello world
sxml2          p_all2    <Hello2>world</Hello2>

3 Rows. -- 20 msec.
```

bif:xcontains

```
SQL>SPARQL
SELECT *
FROM <http://MyTest.com>
WHERE { ?s ?p ?o . ?o bif:xcontains "//Hello[text-contains (., 'world')]" };
s                 p           o
VARCHAR           VARCHAR     VARCHAR
_______________________________________________________________________________

sxml1             p_all       <Hello>world</Hello>

1 Rows. -- 10 msec.
```

bif:xpath_contains

```
SQL>SPARQL
SELECT *
FROM <http://MyTest.com>
WHERE { ?s ?p ?o . ?o bif:xpath_contains "//*" };

s             p          o
VARCHAR       VARCHAR    VARCHAR
_______________________________________________________________________________

sxml1         p_all1     <Hello>world</Hello>
sxml2         p_all2     <Hello2>world</Hello2>

2 Rows. -- 20 msec.
```

bif:xquery_contains

```
SQL>SPARQL
SELECT *
FROM <http://MyTest.com>
WHERE { ?s ?p ?o . ?o bif:xquery_contains "//Hello2 , world" };

s             p          o
VARCHAR       VARCHAR    VARCHAR
_______________________________________________________________________________

sxml2         p_all2     <Hello2>world</Hello2>

1 Rows. -- 20 msec.
```

16.2.3. SPARQL Web Services & APIs

Introduction

The Virtuoso SPARQL query service implements the SPARQL Protocol for RDF (W3C Working Draft 25 January 2006) providing SPARQL query processing for RDF data available on the open internet.

The query processor extends the standard protocol to provide support for multiple output formats. At present this uses additional query parameters.

Supported features include:

- Support for GET and POST requests
- Support for a variety of transfer MIME-types, including RDF /XML and TURTLE
- Support for: *default-graph-uri*, *named-graph-uri*, *using-graph-uri*, *using-named-graph-uri*, *format* parameters

- Support for: *debug*, *timeout*, *maxrows*, *qtxt*, *query* parameters
- Support for a variety of query types including CONSTRUCT, ASK, DESCRIBE

Virtuoso also supports /sparql-graph-crud/ web service endpoint that implements the current draft of W3C SPARQL Graph Update protocol . Both /sparql /sparql-graph-crud/ endpoints use the same SPARQL user account, so this user should be member of SPARQL_UPDATE group in order to modify data via Graph Update protocol. Note that /sparql/ endpoint has /sparql-auth/ variant that uses web authentication. Similarly, /sparql-graph-crud/ has /sparql-graph-crud-auth/ variant. As soon as user is member of SPARQL_UPDATE group, she/he can modify the stored data via /sparql-graph-crud-auth/ as well as via /sparql-auth/ . The /sparql-graph-crud/ endpoint is primarily for serving requests from applications, not for manual interactions via browser. See more information in our SPARQL Authentication section.

Service Endpoint

Virtuoso uses the pre-assigned endpoints "/sparql" and "/SPARQL" as the defaults for exposing its REST based SPARQL Web Services.

The port number associated with the SPARQL services is determined by the 'ServerPort' key value in the '[HTTPServer]' section of the virtuoso.ini file. Thus, if the Virtuoso instance is configured to listen at a none default port e.g. 8890, the SPARQL endpoints would be accessible at http://example.com:8890/sparql/.

The SPARQL endpoint supports both GET and POST requests. The client chooses between GET and POST automatically, using the length of query text as the criterion. If the SPARQL endpoint is accessed without any URL and requisite SPARQL protocol parameters, an interactive HTML page for capturing SPARQL input will be presented.

Customizing SPARQL Endpoint Page

The SPARQL Endpoint Page can now be customized using a xsl stylesheet.

This works by adding the following line with isql:

```
SQL> registry_set ('sparql_endpoint_xsl', 'http://host:port/path/isparql.xsl');
```

where obviously host, port, path and the name isparql.xsl can be set to anything.

SPARQL Protocol Extensions

Request Parameters

Table 16.1. Request Parameters List

Parameter	Notes	Required?
service	Service URI such as 'http://example.com/sparql/'	Yes
query	Text of the query	Yes
dflt_graph	Default graph URI (string or NULL)	No
named_graphs	Vector of named graphs (or NULL to prevent overriding named graphs specified in the query)	Yes
req_hdr	Additional HTTP headers that should be passed to the service, e.g. 'Host: ...'	No
maxrows	Limit on the numbers of rows that should be returned (the actual size of the result set may differ)	No
xslt-uri	Absolute URL of any XSLT stylesheet file to be applied to the SPARQL query results	No
timeout	Timeout for "anytime" query execution, in milliseconds, values less than 1000 are ignored; see Anytime Queries for more details	No
debug	If set to on, SPARQL Compiler will check if all variables are declared, and if there is variable that is not declared, an error will be raised	No

Response Codes

If the query is a CONSTRUCT or a DESCRIBE then the result set consists of a single row and a single column. The value inside is a dictionary of triples in 'long valmode'. Note that the dictionary object cannot be sent to a SQL client, say, via ODBC. The client may lose the database connection trying to fetch a result set row that contains a dictionary object. This disconnection does not disrupt the server, so the client may readily reconnect to the server, but the disconnected transaction will have been rolled

back.

See Also:

- Virtuoso ODBC RDF extensions for SPASQL

Response Format

All the SPARQL protocol standard MIME types are supported by a SPARQL web service client. Moreover, SPARQL web service endpont supports additional MIME types and in some cases additional query types for standard MIME types.

Server Response Formats

Table 16.2. Server Response Formats

Content-Type	SPARQL query type	Description
'application/sparql-results+xml'	SELECT, ASK	Canonical XML presentation of SPARQL result set
'text/rdf+n3'	SELECT, ASK, CONSTRUCT, DESCRIBE	Turtle
'text/rdf+ttl'	SELECT, ASK, CONSTRUCT, DESCRIBE	Turtle
'text/rdf+turtle'	SELECT, ASK, CONSTRUCT, DESCRIBE	Turtle
'text/turtle'	SELECT, ASK, CONSTRUCT, DESCRIBE	Turtle
'text/n3'	SELECT, ASK, CONSTRUCT, DESCRIBE	Turtle
'application/turtle'	SELECT, ASK, CONSTRUCT, DESCRIBE	Turtle
'application/x-turtle'	SELECT, ASK, CONSTRUCT, DESCRIBE	Turtle
'application/x-nice-turtle'	SELECT, ASK, CONSTRUCT, DESCRIBE	Turtle, like above, but the server will try to use "list" and "[...]" notations to make the document easier to read. This formatting is a slow procedure so long results will be formatted as plain Turtle.
'text/rdf+nt'	SELECT	Format for NT (each triple is printed separately without abbreviations)
'text/plain'	SELECT	Format for NT (each triple is printed separately without abbreviations)
'text/ntriples'	SELECT, CONSTRUCT, DESCRIBE	Format for NT (each triple is printed separately without abbreviations)
'application/x-trig'	SELECT, CONSTRUCT, DESCRIBE	TriG syntax for result sets, triples and quads (or sets of graphs with triples). While it is not used for quads, the output is same as for Turtle.
'application/rdf+xml'	SELECT, CONSTRUCT, DESCRIBE	Canonical RDF/XML presentation
'application/soap+xml'	SELECT	SOAP XML
'application/soap+xml;11'	SELECT	SOAP XML
'text/html'	SELECT	HTML document for plain browsing; it's a TABLE for result set and HTML with micro-data for triples
'text/md+html'	SELECT, CONSTRUCT, DESCRIBE	HTML with microdata; for triples only
'text/microdata+html'	SELECT, CONSTRUCT, DESCRIBE	HTML with microdata; for triples only
'text/x-html+ul'	SELECT, CONSTRUCT, DESCRIBE	HTML with triples grouped into hierarchical list

Content-Type	SPARQL query type	Description
'text/x-html+tr'	SELECT, CONSTRUCT, DESCRIBE	HTML with triples in form of a table
'application/vnd.ms-excel'	SELECT	HTML table for loading data into stylesheets
'text/csv'	SELECT, CONSTRUCT, DESCRIBE	Comma-separated values
'text/tab-separated-values'	SELECT	Tab-separated values
'application/javascript'	SELECT	JavaScript data fragment
'application/json'	SELECT	JSON
'application/sparql-results+json'	SELECT, ASK	JSON result set
'application/odata+json'	SELECT, ASK, CONSTRUCT, DESCRIBE	JSON in ODATA style
'application/microdata+json'	SELECT, CONSTRUCT, DESCRIBE	Microdata as JSON; for triples only
'application/rdf+json'	CONSTRUCT, DESCRIBE	JSON in TALIS style; for triples only
'application/x-rdf+json'	CONSTRUCT, DESCRIBE	JSON in TALIS style; for triples only
'application/x-json+ld'	CONSTRUCT, DESCRIBE	JSON in Linked Data style; for triples only
'application/ld+json'	CONSTRUCT, DESCRIBE	JSON in Linked Data style; for triples only
'text/cxml'	SELECT, CONSTRUCT, DESCRIBE	CXML output for rendering in Pivot Viewer of MS SilverLight?. Result sets and triples are handled in different ways.
'text/cxml+qrcode'	SELECT,CONSTRUCT, DESCRIBE	CXML output with QRcode imprinted into each picture; for result sets and triples
'application/atom+xml'	SELECT, CONSTRUCT, DESCRIBE	Atom-style XML
'application/xhtml+xml'	SELECT	RDFa placed into XHTML; for triples only

Client Response Formats

Table 16.3. Client Response Formats

Content-Type	SPARQL query type	Description
'application/sparql-results+xml'	SELECT, ASK	Canonical XML presentation of SPARQL result set
'text/rdf+n3'	CONSTRUCT, DESCRIBE	Turtle
'text/rdf+ttl'	CONSTRUCT, DESCRIBE	Turtle
'text/rdf+turtle'	CONSTRUCT, DESCRIBE	Turtle
'text/turtle'	CONSTRUCT, DESCRIBE	Turtle
'text/n3'	CONSTRUCT, DESCRIBE	Turtle
'application/turtle'	CONSTRUCT, DESCRIBE	Turtle
'application/x-turtle'	CONSTRUCT, DESCRIBE	Turtle
'application/rdf+xml'	CONSTRUCT, DESCRIBE	Canonical RDF/XML presentation

The current implementation does not support returning the results of SELECT as RDF/XML or 'sparql-results-2'.

If the HTTP header returned by the remote server does not contain a 'Content-Type' line, the client may guess MIME type from the text of the returned body.

Error messages returned from the service are returned as XML documents, using the MIME type application/xml. The documents consist of a single element containing an error message.

Additional Response Formats -- SELECT

Use the format parameter to select one of the following alternate output formats:

Table 16.4. Additional Response formats list -- SELECT

Format Value	Description	Mimetype
HTML	The result is a HTML document containing query summary and tabular results. The format is human-readable but not intended for using by applications because it makes strings undistinguishable from IRIs and loses other details such as exact datatypes of returned values.	text/html
json	Two separate MIME types exist for JSON: JSON serialization of results is 'application/sparql-results+json' and confirms to the draft specification "Serializing SPARQL Query Results in JSON". JSON serialization of triples is 'application/rdf+json' and interoperable with Talis. Sometimes a client needs a JSON but it does not know the type of query it sends to Virtuoso web service endpoint. In this case the client can specify either one MIME-type 'application/json' or both 'application/sparql-results+json' and 'application/rdf+json' in the "Accept" header line and Virtuoso will chose the appropriate one automatically. Similar trick works for other sorts of result types: Virtuoso inspects the whole "Accept" header line to find out the most appropriate return type for the given query.	application/sparql-results+json
json		application/rdf+json
js	Javascript serialization of results generates an HTML table with the CSS class sparql. The table contains a column indicating row number and additional columns for each query variable. Each query solution contributes one row of the table. Unbound variables are indicated with a non-breaking space in the appropriate table cells.	application/javascript
table		text/html
XML		text/html
TURTLE		text/html

Additional Response Formats -- CONSTRUCT & DESCRIBE

Example output of DESCRIBE in rdf+json serialization format

1. Go to the sparql endpoint at http://host:port/sparql, for ex. at http://dbpedia.org/sparql
2. Enter query in the "Query text" area, for ex.:

```
DESCRIBE <http://dbpedia.org/resource/%22S%22_Bridge_II>
```

3. Select for "Display Results As": JSON
4. Click "Run Query" button.
5. As result should be produced the following output:

```
{
  { 'http://dbpedia.org/resource/%22S%22_Bridge_II' : { 'http://www.w3.org/1999/02/22-rdf-syntax-n
      { 'type' : 'uri', 'value' : 'http://dbpedia.org/ontology/Resource' } ,
      { 'type' : 'uri', 'value' : 'http://dbpedia.org/ontology/HistoricPlace' } } ,
    { 'http://dbpedia.org/ontology/added' : { 'type' : 'literal', 'value' : '1973-04-23' , 'dataty
    { 'http://www.w3.org/2003/01/geo/wgs84_pos#lat' : { 'type' : 'literal', 'value' : 39.993057250
    { 'http://www.w3.org/2003/01/geo/wgs84_pos#long' : { 'type' : 'literal', 'value' : -81.7466659
    { 'http://dbpedia.org/property/wikiPageUsesTemplate' : { 'type' : 'uri', 'value' : 'http://dbp
    { 'http://dbpedia.org/property/name' : { 'type' : 'literal', 'value' : '"S" Bridge II' , 'lang
    { 'http://dbpedia.org/property/nearestCity' : { 'type' : 'uri', 'value' : 'http://dbpedia.org/
      { 'type' : 'uri', 'value' : 'http://dbpedia.org/resource/Ohio' } } ,
    { 'http://dbpedia.org/property/latDirection' : { 'type' : 'literal', 'value' : 'N' , 'lang' :
    { 'http://dbpedia.org/property/governingBody' : { 'type' : 'literal', 'value' : 'State' , 'lan
    { 'http://www.georss.org/georss/point' : { 'type' : 'literal', 'value' : '39.99305556 -81.7466
      { 'type' : 'literal', 'value' : '39.9930555556 -81.7466666667' } } ,
    { 'http://xmlns.com/foaf/0.1/name' : { 'type' : 'literal', 'value' : '"S" Bridge II' } } ,
    { 'http://dbpedia.org/property/latDegrees' : { 'type' : 'literal', 'value' : 39 , 'datatype' :
    { 'http://dbpedia.org/property/latMinutes' : { 'type' : 'literal', 'value' : 59 , 'datatype' :
    { 'http://dbpedia.org/property/latSeconds' : { 'type' : 'literal', 'value' : 35 , 'datatype' :
    { 'http://dbpedia.org/property/longDirection' : { 'type' : 'literal', 'value' : 'W' , 'lang' :
```

```
    { 'http://dbpedia.org/property/architect' : { 'type' : 'uri', 'value' : 'http://dbpedia.org/re
    { 'http://dbpedia.org/property/added' : { 'type' : 'literal', 'value' : '1973-04-23' , 'dataty
    { 'http://www.w3.org/2000/01/rdf-schema#label' : { 'type' : 'literal', 'value' : '"S" Bridge I
      { 'type' : 'literal', 'value' : '"S" Bridge II' , 'lang' : 'en' } } ,
    { 'http://dbpedia.org/ontology/architect' : { 'type' : 'uri', 'value' : 'http://dbpedia.org/re
    { 'http://xmlns.com/foaf/0.1/img' : { 'type' : 'uri', 'value' : 'http://upload.wikimedia.org/w
    { 'http://dbpedia.org/property/locmapin' : { 'type' : 'literal', 'value' : 'Ohio' , 'lang' : '
    { 'http://dbpedia.org/property/refnum' : { 'type' : 'literal', 'value' : 73001513 , 'datatype'
    { 'http://dbpedia.org/property/abstract' : { 'type' : 'literal', 'value' : '"S" Bridge II is a
      { 'type' : 'literal', 'value' : '"S" Bridge II bij New Concord, Ohio, is een deel van de Nat
    { 'http://www.w3.org/2004/02/skos/core#subject' : { 'type' : 'uri', 'value' : 'http://dbpedia.
      { 'type' : 'uri', 'value' : 'http://dbpedia.org/resource/Category:Bridges_on_the_National_Re
    { 'http://dbpedia.org/ontology/nearestCity' : { 'type' : 'uri', 'value' : 'http://dbpedia.org/
      { 'type' : 'uri', 'value' : 'http://dbpedia.org/resource/New_Concord%2C_Ohio' } } ,
    { 'http://xmlns.com/foaf/0.1/depiction' : { 'type' : 'uri', 'value' : 'http://upload.wikimedia
    { 'http://dbpedia.org/property/caption' : { 'type' : 'literal', 'value' : 'The bridge in the f
    { 'http://dbpedia.org/property/longDegrees' : { 'type' : 'literal', 'value' : 81 , 'datatype'
    { 'http://dbpedia.org/property/longMinutes' : { 'type' : 'literal', 'value' : 44 , 'datatype'
    { 'http://dbpedia.org/property/longSeconds' : { 'type' : 'literal', 'value' : 48 , 'datatype'
    { 'http://www.w3.org/2000/01/rdf-schema#comment' : { 'type' : 'literal', 'value' : '"S" Bridge
      { 'type' : 'literal', 'value' : '"S" Bridge II bij New Concord, Ohio, is een deel van de Nat
    { 'http://xmlns.com/foaf/0.1/page' : { 'type' : 'uri', 'value' : 'http://en.wikipedia.org/wiki
  { 'http://dbpedia.org/resource/%22S%22_Bridge_II_%28Muskingum_County%2C_Ohio%29' : { 'http://dbp
}
```

Example output of CONSTRUCT in rdf+json serialization format

1. Go to the sparql endpoint at http://host:port/sparql, for ex. at http://dbpedia.org/sparql
2. Enter query in the "Query text" area, for ex.:

```
CONSTRUCT
{
 ?s a ?Concept .
}
WHERE
{
 ?s a ?Concept .
}
LIMIT 10
```

3. Select for "Display Results As": JSON
4. Click "Run Query" button.
5. As result should be produced the following output:

```
{
  { 'http://dbpedia.org/ontology/Place' : { 'http://www.w3.org/1999/02/22-rdf-syntax-ns#type' : {
  { 'http://dbpedia.org/ontology/Area' : { 'http://www.w3.org/1999/02/22-rdf-syntax-ns#type' : { '
  { 'http://dbpedia.org/ontology/City' : { 'http://www.w3.org/1999/02/22-rdf-syntax-ns#type' : { '
  { 'http://dbpedia.org/ontology/River' : { 'http://www.w3.org/1999/02/22-rdf-syntax-ns#type' : {
  { 'http://dbpedia.org/ontology/Road' : { 'http://www.w3.org/1999/02/22-rdf-syntax-ns#type' : { '
  { 'http://dbpedia.org/ontology/Lake' : { 'http://www.w3.org/1999/02/22-rdf-syntax-ns#type' : { '
  { 'http://dbpedia.org/ontology/LunarCrater' : { 'http://www.w3.org/1999/02/22-rdf-syntax-ns#type
  { 'http://dbpedia.org/ontology/ShoppingMall' : { 'http://www.w3.org/1999/02/22-rdf-syntax-ns#typ
  { 'http://dbpedia.org/ontology/Park' : { 'http://www.w3.org/1999/02/22-rdf-syntax-ns#type' : { '
  { 'http://dbpedia.org/ontology/SiteOfSpecialScientificInterest' : { 'http://www.w3.org/1999/02/2
}
```

For interoperability with clients that were developed before current versions of SPARQL protocol and format specs are issued, Virtuoso supports some obsolete variants of standard MIME types. 'text/rdf+n3', 'text/rdf+ttl', 'application/turtle' and 'application/x-turtle' are understood for TURTLE output, 'application/x-rdf+json' and 'application/rdf+json' are for "Serializing SPARQL Query Results in JSON". When a client specifies obsolete MIME type but not its standard variant, an obsolete variant is returned for interoperability.

Strict checking of void variables

To set SPARQL Endpoint to check if all variables in a given query are declared correctly, one should hatch the the Options "Strict checking of void variables" check-box. In that case, if a variable declaration is missing, i.e. not included in the SPARQL Query WHERE clause, then an error will be raised. For example, on attempt to execute the following query:

```
--      Options "Strict checking of void variables" check-box is checked:
select ?y
where
 {
  ?s ?p ?o
 }
```

since the variable `?y` is not declared, the following error will be raised:

```
Virtuoso 37000 Error SP031: SPARQL compiler:
Variable 'y' is used in the query result set but not assigned

SPARQL query:

#output-format:text/html
define sql:signal-void-variables 1 define sql:gs-app-callback "ODS" select ?y
where
{
?s ?p ?o
}
```

View Results Page of SPARQL Query Execution

To view SPARQL Endpoint Results page of SPARQL query execution should be used the parameter *query* i.e the SPARQL Protocol URL should look like:

```
http://cname/sparql?default-graph-uri=&query=...
```

Example

Suppose the following simple query:

```
SELECT *
WHERE
  {
    ?s ?p ?o
  }
LIMIT 10
```

See this example link against Virtuoso Demo Server SPARQL Endpoint with SPARQL Protocol URL.

View Editor Page of SPARQL Query

To view the SPARQL Endpoint editor page of SPARQL query execution should be used the parameter *qtxt* i.e the SPARQL Protocol URL should look like:

```
http://cname/sparql?default-graph-uri=&qtxt=...
```

Example

Suppose the following simple query:

```
SELECT *
WHERE
  {
    ?s ?p ?o
  }
LIMIT 10
```

Suppose also this results page link against Virtuoso Demo Server SPARQL Endpoint with SPARQL Protocol URL.

Replace the parameter name *query* with *qtxt* .

Access the new link , which should present the SPARQL Endpoint Editor page with "Query Text" area filled in with the SPARQL Query from above.

Virtuoso/PL APIs

Virtuoso also provides SPARQL protocol client APIs in Virtuoso PL, so you can communicate with SPARQL Query Services from Virtuoso stored procedures. The APIs are as follows:

Table 16.5. Virtuoso/PL APIs

API	Notes
DB.DBA.SPARQL_REXEC	Behaves like DBA.SPARQL_EVAL, but executes the query on the specified server. The procedure does not return anything. Instead, it creates a result set.
DB.DBA.SPARQL_REXEC_TO_ARRAY	Behaves like DBA.SPARQL_EXEC_TO_ARRAY(), but executes the query on the specified server. The function returns a vector of rows, where every row is represented by a vector of field values.
DB.DBA.SPARQL_REXEC_WITH_META	Has no local SPARQL_EVAL analog. It produces not only an array of result rows together with an array of result set metadata in a format used by the exec() function.

SPARQL Anytime Queries

Starting with version 6, Virtuoso offers a partial query evaluation feature that guarantees answers to arbitrary queries within a fixed time. This is intended for use in publicly available SPARQL or SQL end points on large databases. This enforces a finite duration to all queries and will strive to return meaningful partial results. Thus this provides the same security as a transaction timeout but will be more user friendly since results will generally be returned, also for aggregate queries. Outside of a public query service, this may also be handy when exploring a large data set with unknown properties.

The feature is activated with the statement

```
set result_timeout == <expression>;
```

Find more detailed information in the Anytime Queries section.

Example Dump arbitrary query result as N-Triples

Assume the following arbitrary query:

```
SPARQL define output:format "NT"
CONSTRUCT { ?s a ?t }
FROM virtrdf:
WHERE { ?s a ?t };
```

For iteration over result-set of an arbitrary query, use `exec_next()` in a loop that begins with `exec()` with cursor output variable as an argument and ends with `exec_close()` after it is out of data.

Service Endpoint Security

Earlier releases of Virtuoso secured the SPARQL endpoint via privileges assigned to the service- specific SQL user account "SPARQL". This account was optionally granted "SPARQL_SELECT" or "SPARQL_UPDATE" roles. By default only the "SPARQL_SELECT" role was assigned, enabling all users to at least perform SELECT queries. The "SPARQL_UPDATE" role must be granted to allow updates to the Quad Store - a pre-requisite for the Virtuoso Sponger services to be functional i.e. to allow the Sponger to populate and update the Quad Store. In Virtuoso release 5.0.7, there is a new "SPARQL_SPONGE" role which can be assigned specifically to allow Sponger services to update the Quad Store but not SPARQL users via the SPARQL endpoint.

Restricting a user's access to specific graphs can be done using Virtuoso Graph security functionality, via one of the Virtuoso Data Access APIs: ODBC, JDBC, ADO.Net or PL code.

See Also:

- Virtuoso ODBC RDF extensions for SPASQL

For example, users of OpenLink Data Space (ODS) applications are restricted in the RDF graphs accessible to them as follows:

```
DB.DBA.TABLE_DROP_POLICY ('DB.DBA.RDF_QUAD', 'S');

create procedure DB.DBA.RDF_POLICY (in tb varchar, in op varchar)
{
 declare chost, ret varchar;
 chost := DB.DBA.WA_CNAME ();
 ret := sprintf ('(ID_TO_IRI (G) NOT LIKE \'http://%s/dataspace/%%/private#\' ' ||
 'OR G = IRI_TO_ID (sprintf (\'http://%s/dataspace/%%U/private#\', USER)))', chost, chost);
 return ret;
}
;

grant execute on DB.DBA.RDF_POLICY to public;

DB.DBA.TABLE_SET_POLICY ('DB.DBA.RDF_QUAD', 'DB.DBA.RDF_POLICY', 'S');
```

where DB.DBA.WA_CNAME () is an ODS function returning the default host name.

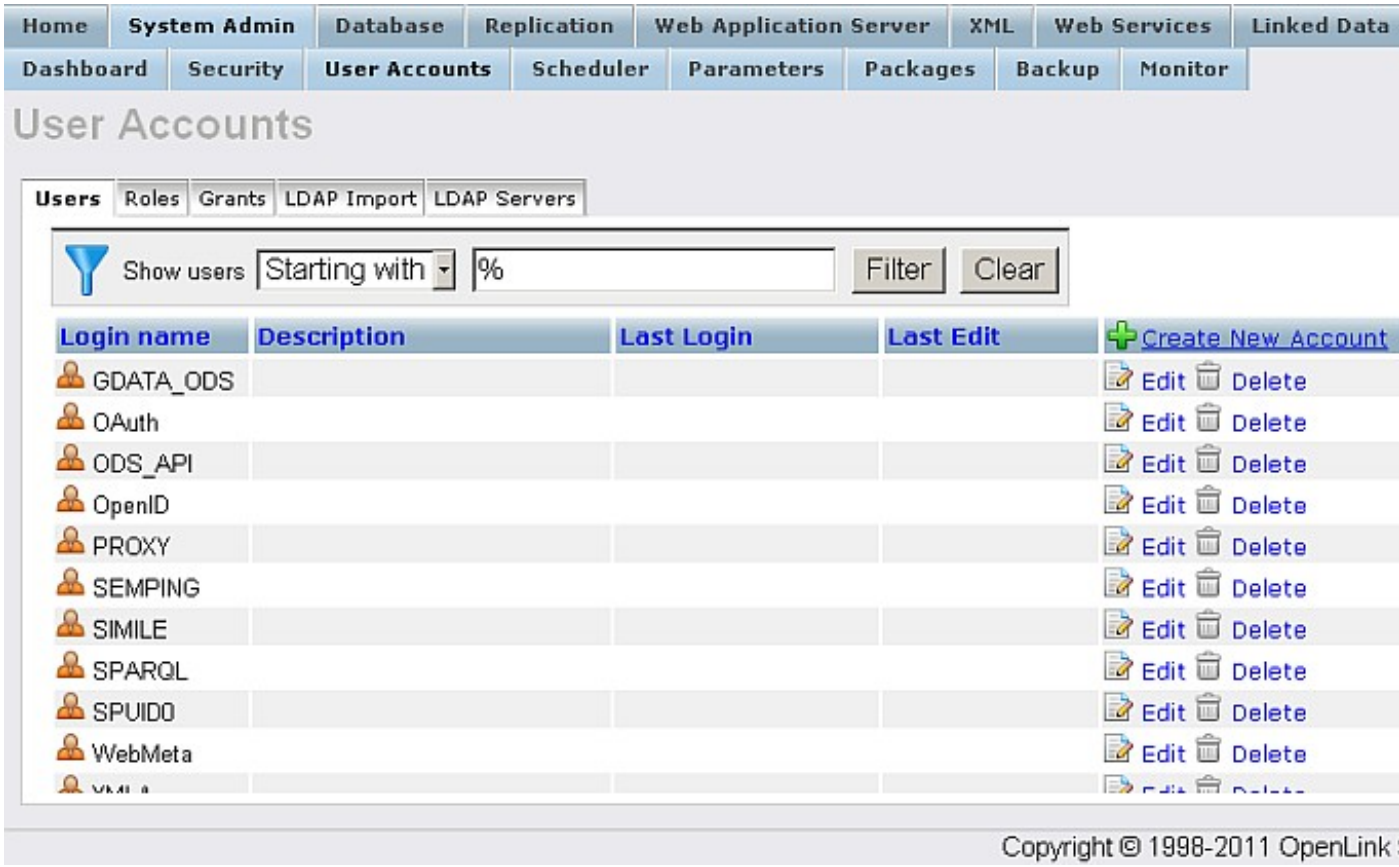
The effect of this policy is to restrict user 'user' to the graph http://cname/dataspace/user/private#

SPARQL Auth Endpoint Usage Example

Virtuoso reserves the path '/sparql-auth/' for a SPARQL service supporting authenticated SPARUL. This endpoint allows specific SQL accounts to perform SPARUL over the SPARQL protocol. To be allowed to login via SQL or ODBC and update physical triples, a user must be granted "SPARQL_UPDATE" privileges. To grant this role:

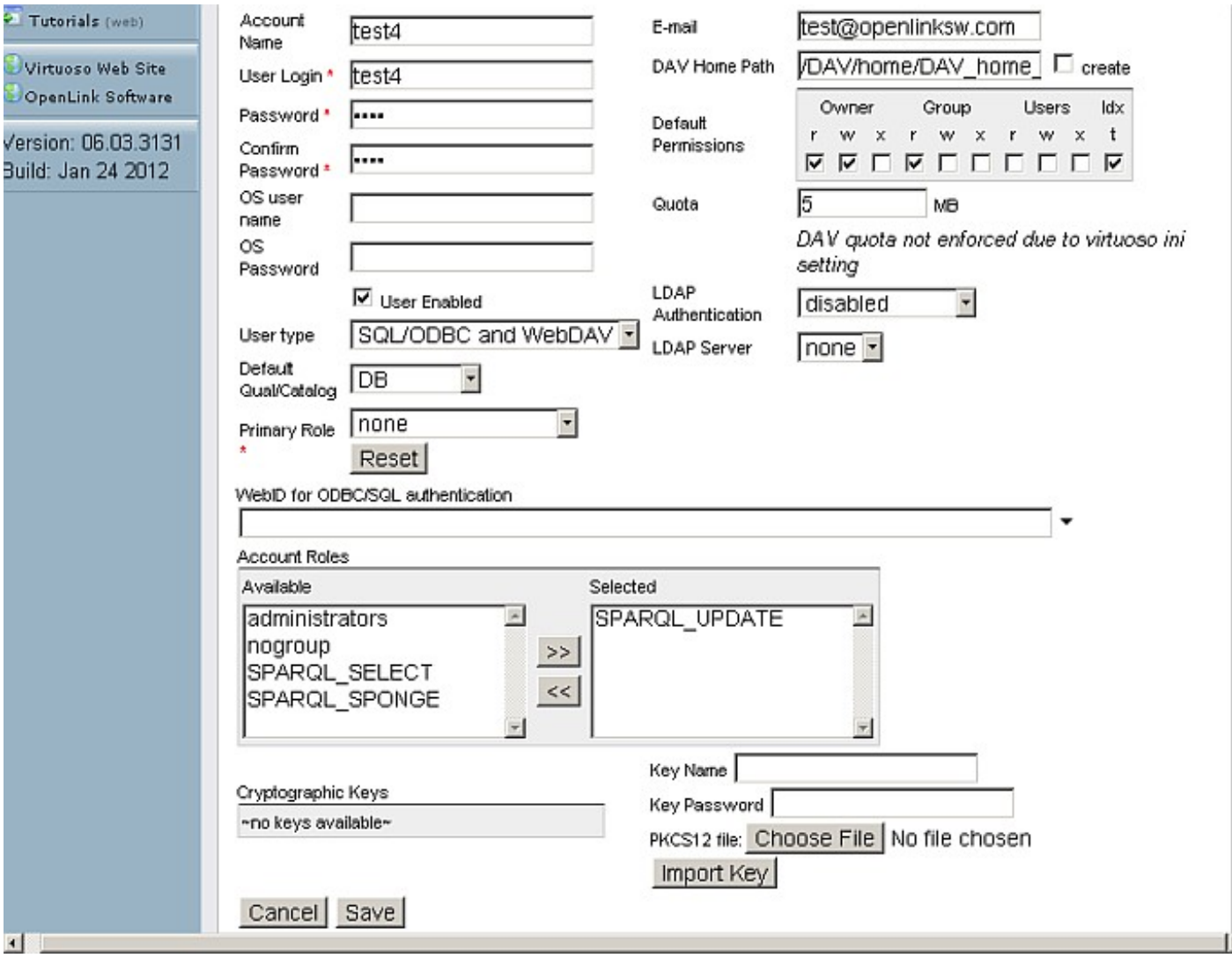
1. Go to the Virtuoso administration UI i.e. http://host:port/conductor
2. Login as user dba
3. Go to System Admin->User Accounts->Users

Figure 16.1. Conductor UI



4. Click the link "Edit"
5. Set "User type" to "SQL/ODBC Logins and WebDAV".
6. Select from the list of available Account Roles "SPARQL_UPDATE" role and click the ">>" button so to add it to the right-hand list.

7. **Figure 16.2. Conductor UI**



8. Click the "Save" button.

Note that if a table is used in an Linked Data View, and this table is not granted to SPARQL_SELECT permission (or SPARQL_UPDATE, which implicitly confers SPARQL_SELECT), then all SELECTs on a graph defined by an Linked Data View will return an access violation error as the user account has no permissions to read the table. The user must have appropriate privileges on all tables included in an Linked Data View in order to be able to select on *all* graphs.

Managing a SPARQL Web Service Endpoint

Virtuoso web service endpoints may provide different default configurations for different host names mentioned in an HTTP request. Host name configuration for SPARQL web service endpoints can be managed via the table *DB.DBA.SYS_SPARQL_HOST* .

```
create table DB.DBA.SYS_SPARQL_HOST (
  SH_HOST       varchar not null primary key, -- host mask
  SH_GRAPH_URI varchar,                 -- default graph uri
  SH_USER_URI   varchar,                  -- reserved for any use in applications
  SH_BASE_URI varchar,                  -- for future use (not used currently) to set BASE in sparql quer
  SH_DEFINES long varchar,              -- additional defines for requests
  PRIMARY KEY (SH_HOST)
)
```

You can find detailed descriptions of the table columns here . Also, please read these notes on managing public web service endpoints.

Authentication

Virtuoso 5.0.7 introduced a new "SPARQL_SPONGE" role which can be assigned specifically for controlling Sponger middleware services which perform writes and graph creation in the RDF Quad Store. This role only allows updates through the Sponger. Quad Store updates via any other route require granting the SPARQL_UPDATE role.

Virtuoso 5.0.11 onwards added three new methods for securing SPARQL endpoints that include:

- SQL authentication
- OAuth
- WebID Protocol based authentication

Each of these authentication methods is associated with a purpose specific default SPARQL endpoint along the following lines:

- http://<cname>/sparql-auth (SQL authentication)
- http://<cname>/sparql-oauth (OAuth)
- http://<cname>/sparql-graph-crud-auth (OAuth CRUD)
- https://<cname>/sparql and https://<cname>/sparql-webid (WebID Protocol)

Note: sparql-ssl is alias of sparql-webid.

The Virtuoso Authentication Server offers a UI with options for managing:

- Application keys and protected SPARQL endpoints: OAuth provides a secure data transmission level mechanism for your SPARQL endpoint. It enables you to interact securely with your RDF database from a variety of locations. It also allows you to provide controlled access to private data to selected user profiles.
- WebID Protocol ACLs: WebID Protocol is an implementation of a conceptual authentication and authorization protocol that links a Web ID to a public key to create a global, decentralized/distributed, and open yet secure authentication system that functions with existing browsers.

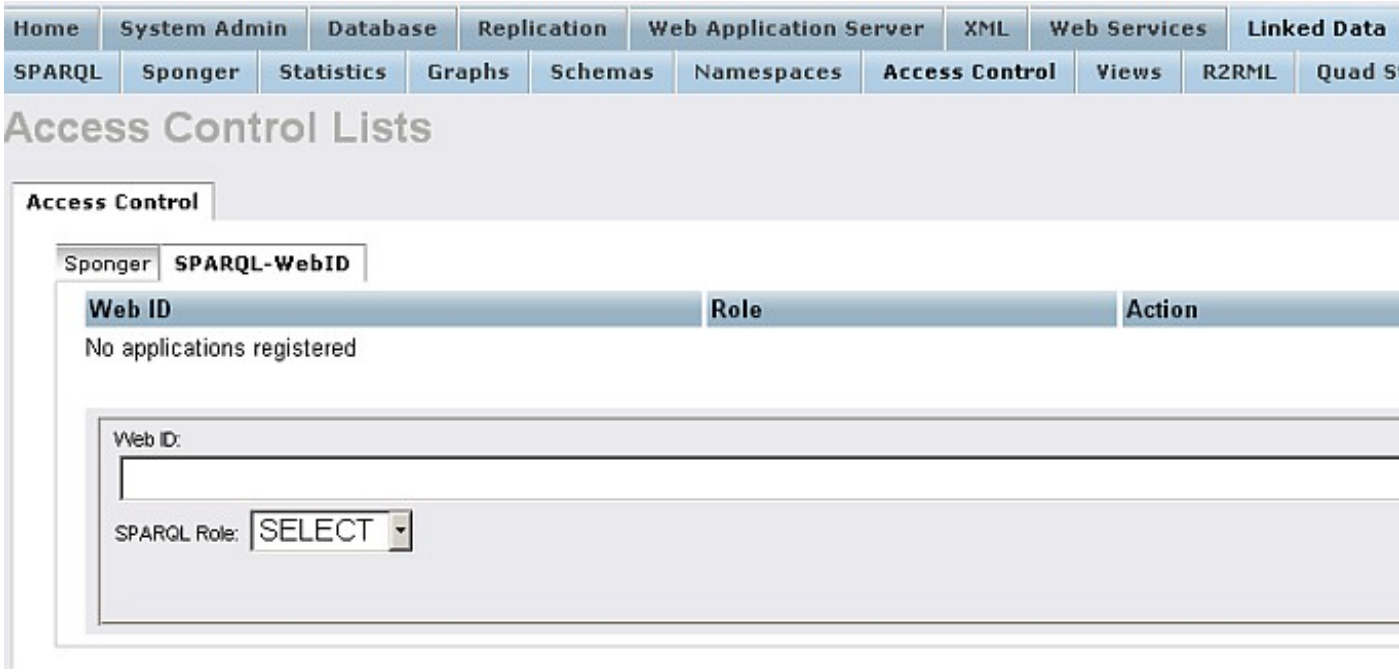
Virtuoso Authentication Server can be installed by downloading and installing the conductor_dav.vad package.

The Authentication UI is accessible from the Conductor UI -> Linked Data -> Access Control -> SPARQL-WebID. Here is sample scenario:

SPARQL-WebID Authentication Example

1. Download and install the conductor_dav.vad package.
2. Generate an X.509 Certificate hosted WebID .
3. Go to http://<cname>:<port>/conductor, where <cname>:<port> are replaced by your local server values.
4. Log in as user "dba" or another user with DBA privileges.
5. Go to Linked Data -> Access Controls -> SPARQL-WebID:

Figure 16.3. SPARQL-WebID

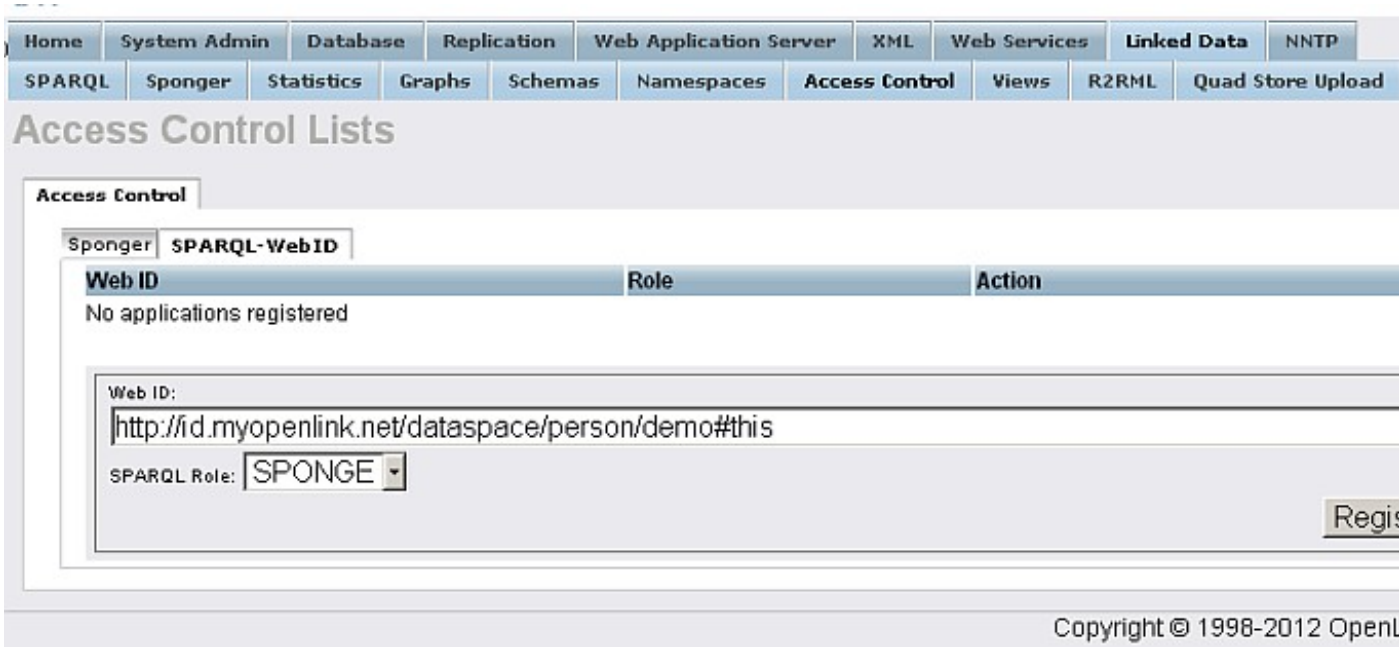


6. Enter in the presented form Web ID for ex.:

   ```
   http://id.myopenlink.net/dataspace/person/demo#this
   ```

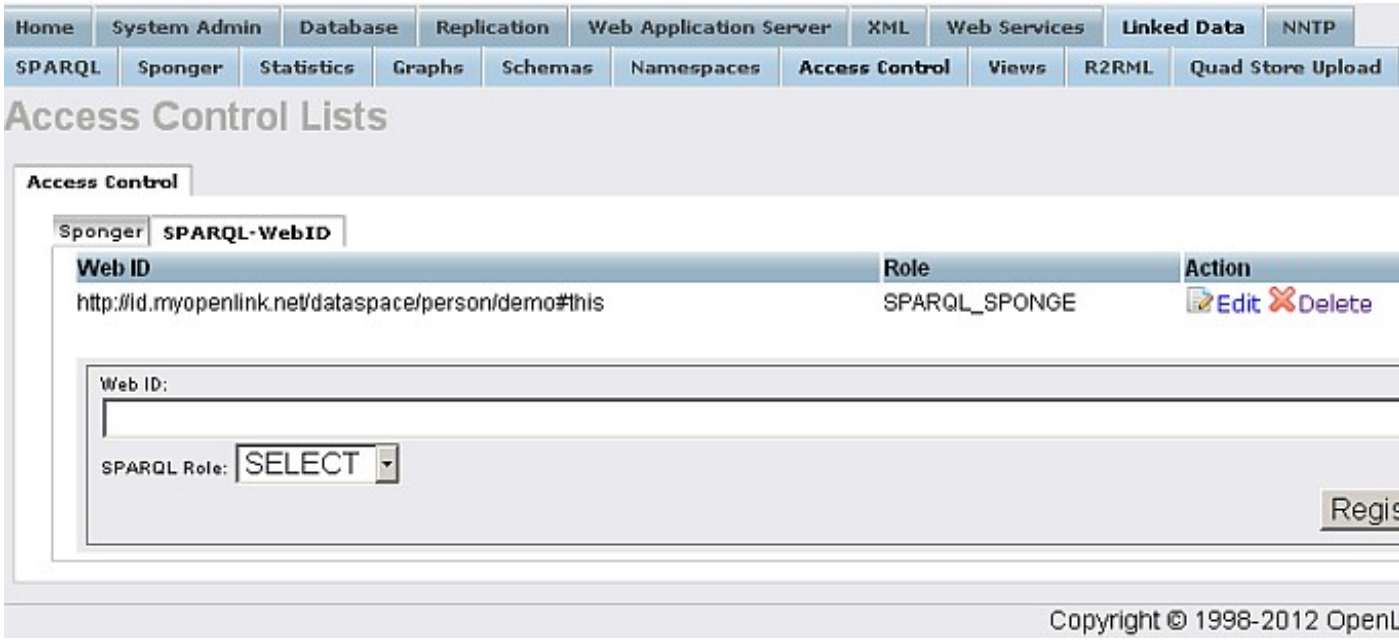
   and select "SPARQL Role" for ex. "Sponge".

Figure 16.4. SPARQL-WebID



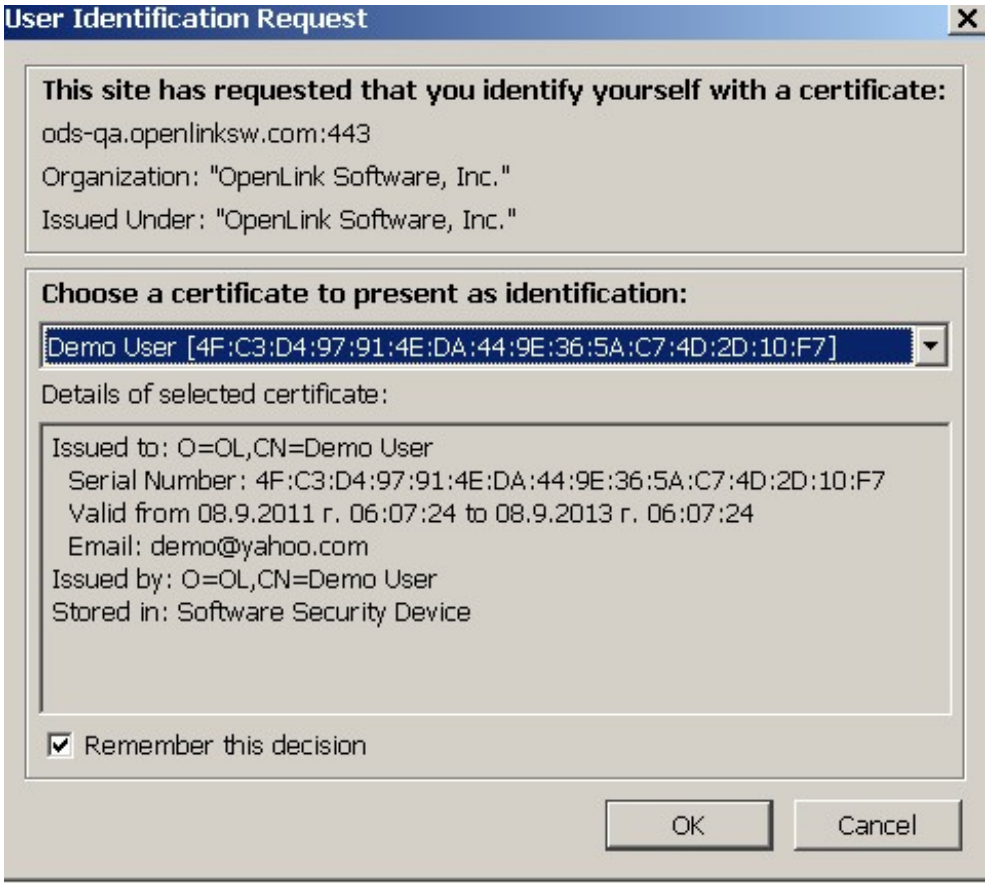
7. Click the "Register" button.
8. As result the WebID Protocol ACLs will be created:

Figure 16.5. SPARQL-WebID



9. Go to the SPARQL-WebID endpoint https://<cname>:<port>/sparql-webid
10. Select the user's certificate from above:

Figure 16.6. SPARQL-WebID



11. As result the SPARQL Query UI will be presented:

Figure 16.7. SPARQL-WebID

12. Execute sample query and view the results:

Figure 16.8. SPARQL-WebID

s	p	o
http://www.openlinksw.com/virtrdf-data-formats#default-iid	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat
http://www.openlinksw.com/virtrdf-data-formats#default-iid-nullable	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat
http://www.openlinksw.com/virtrdf-data-formats#default-iid-nonblank	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat
http://www.openlinksw.com/virtrdf-data-formats#default-iid-nonblank-nullable	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat
http://www.openlinksw.com/virtrdf-data-formats#default	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat
http://www.openlinksw.com/virtrdf-data-formats#default-nullable	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat
http://www.openlinksw.com/virtrdf-data-formats#sql-varchar	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat
http://www.openlinksw.com/virtrdf-data-formats#sql-varchar-nullable	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat
http://www.openlinksw.com/virtrdf-data-formats#sql-varchar-dt	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat
http://www.openlinksw.com/virtrdf-data-formats#sql-varchar-dt-nullable	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat
http://www.openlinksw.com/virtrdf-data-formats#sql-varchar-lang	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat
http://www.openlinksw.com/virtrdf-data-formats#sql-varchar-lang-nullable	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat

SPARQL OAuth Endpoint

OAuth provides a secure data transmission level mechanism for your SPARQL endpoint. It enables you to interact securely with your RDF database from a variety of locations. It also allows you to provide controlled access to private data to selected users.

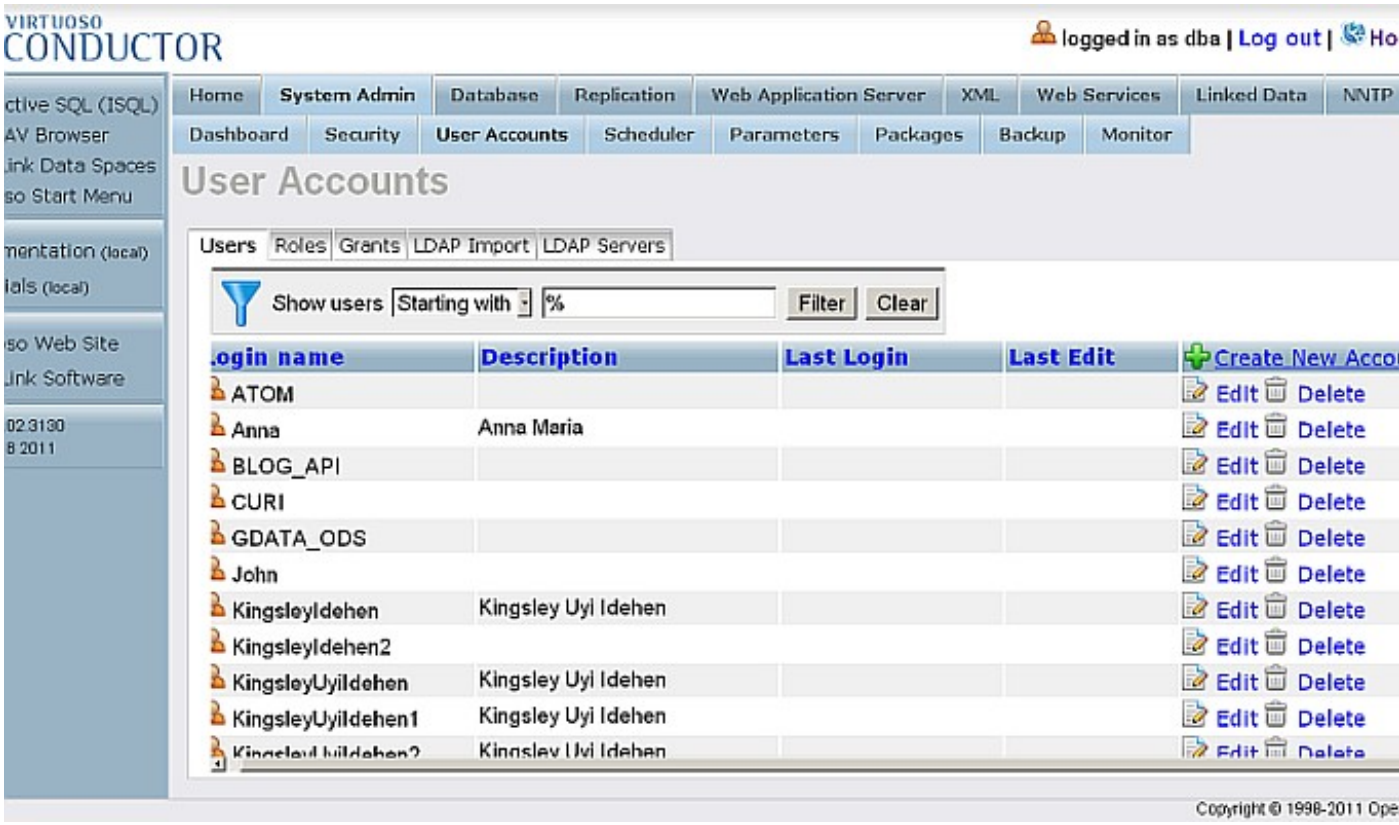
Virtuoso OAuth Server can be installed by downloading and installing the ods_framework_dav.vad package. The OAuth UI is accessible from the URL http://cname:port/oauth

A user must have SQL privileges in order to run secured SPARQL statements.

Here is a sample scenario:

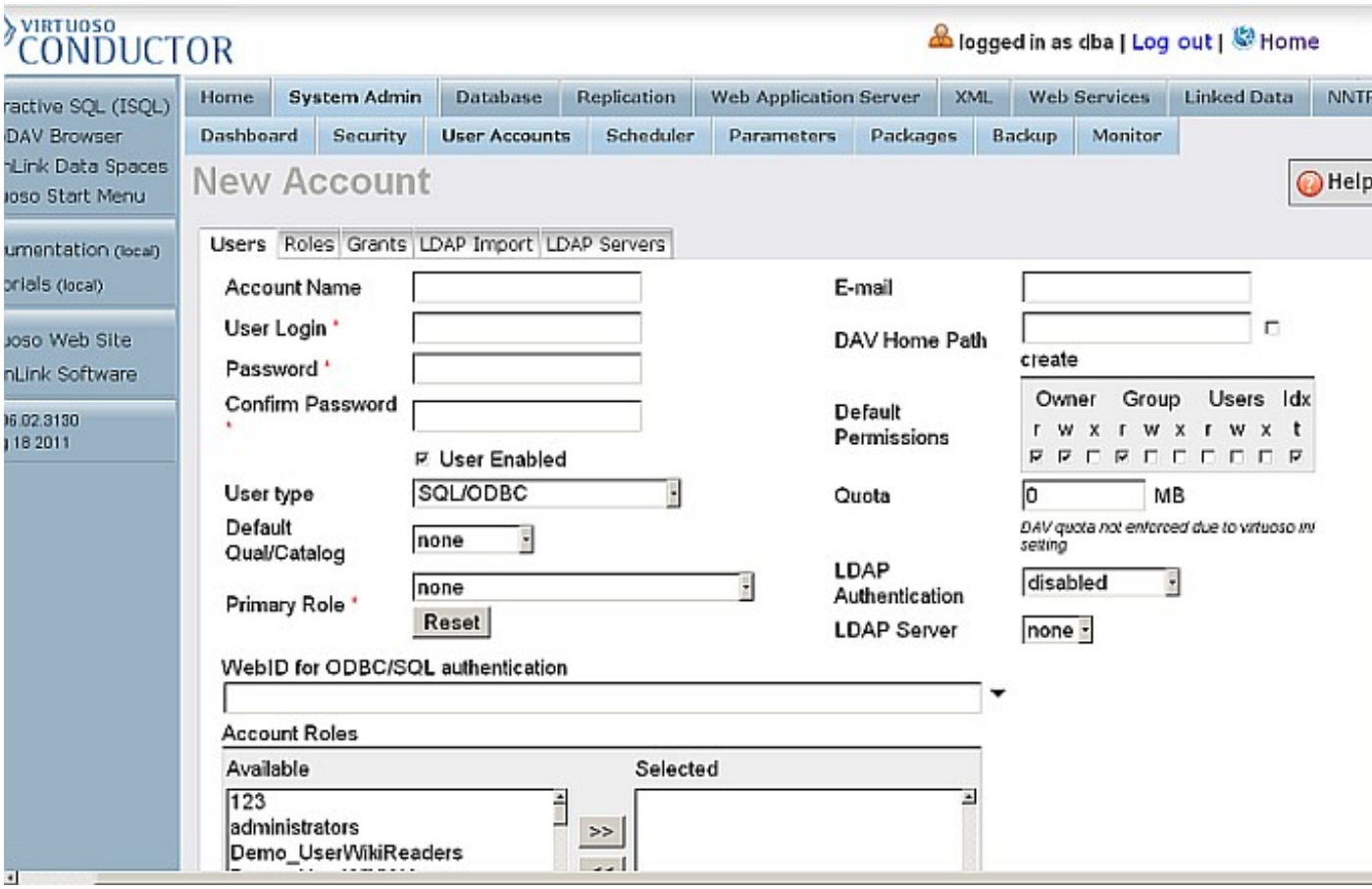
1. Download and install the conductor_dav.vad and ods_framework_dav.vad packages.
2. Generate an X.509 Certificate hosted WebID .
3. Go to http://<cname>:<port>/conductor, where <cname>:<port> are replaced by your local server values.
4. Log in as user "dba" or another user with DBA privileges.
5. Go to System Admin->User Accounts:

Figure 16.9. SPARQL OAuth Endpoint



6. Click "Create New Account":

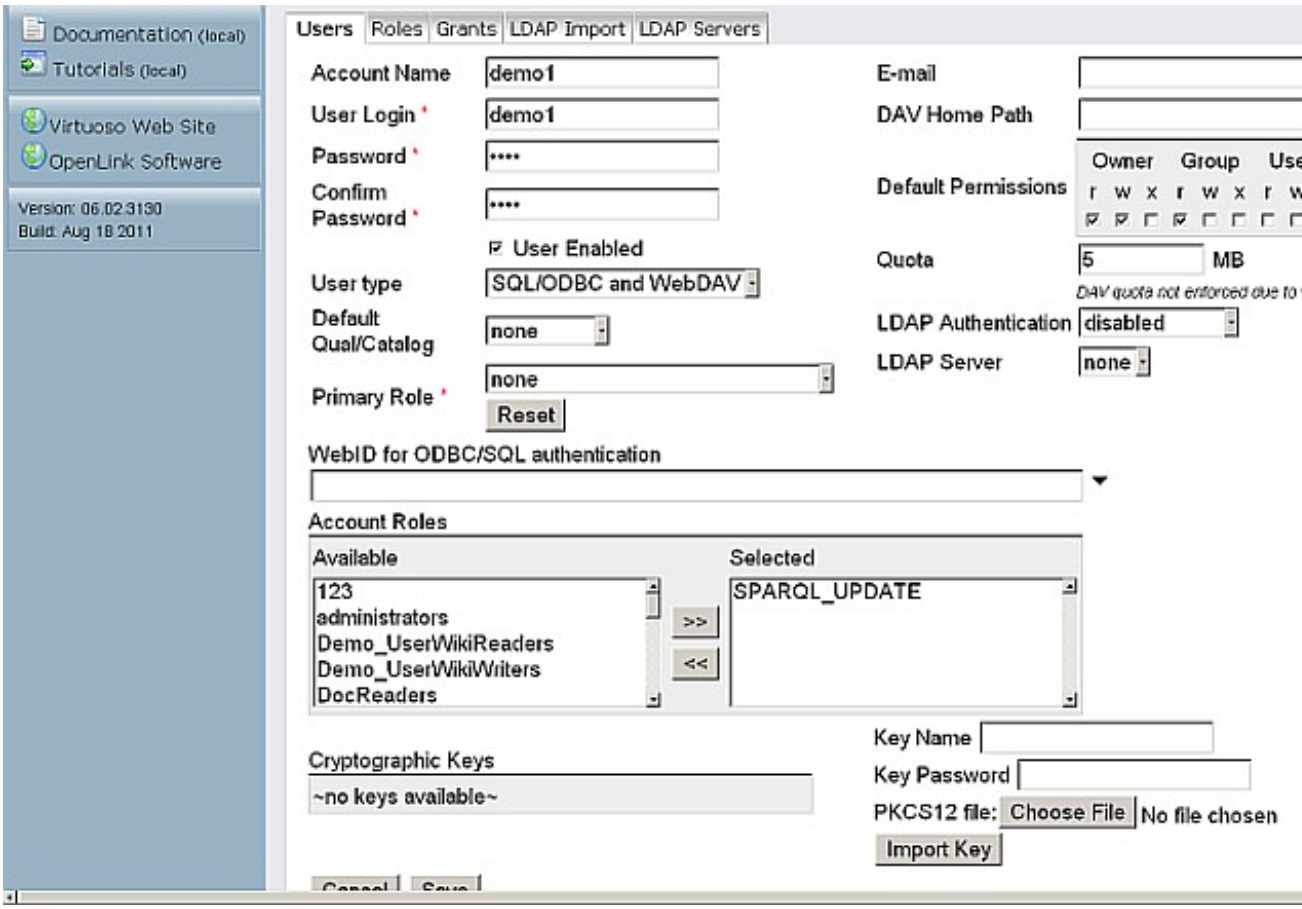
Figure 16.10. SPARQL OAuth Endpoint



7. In the presented form enter respectively:

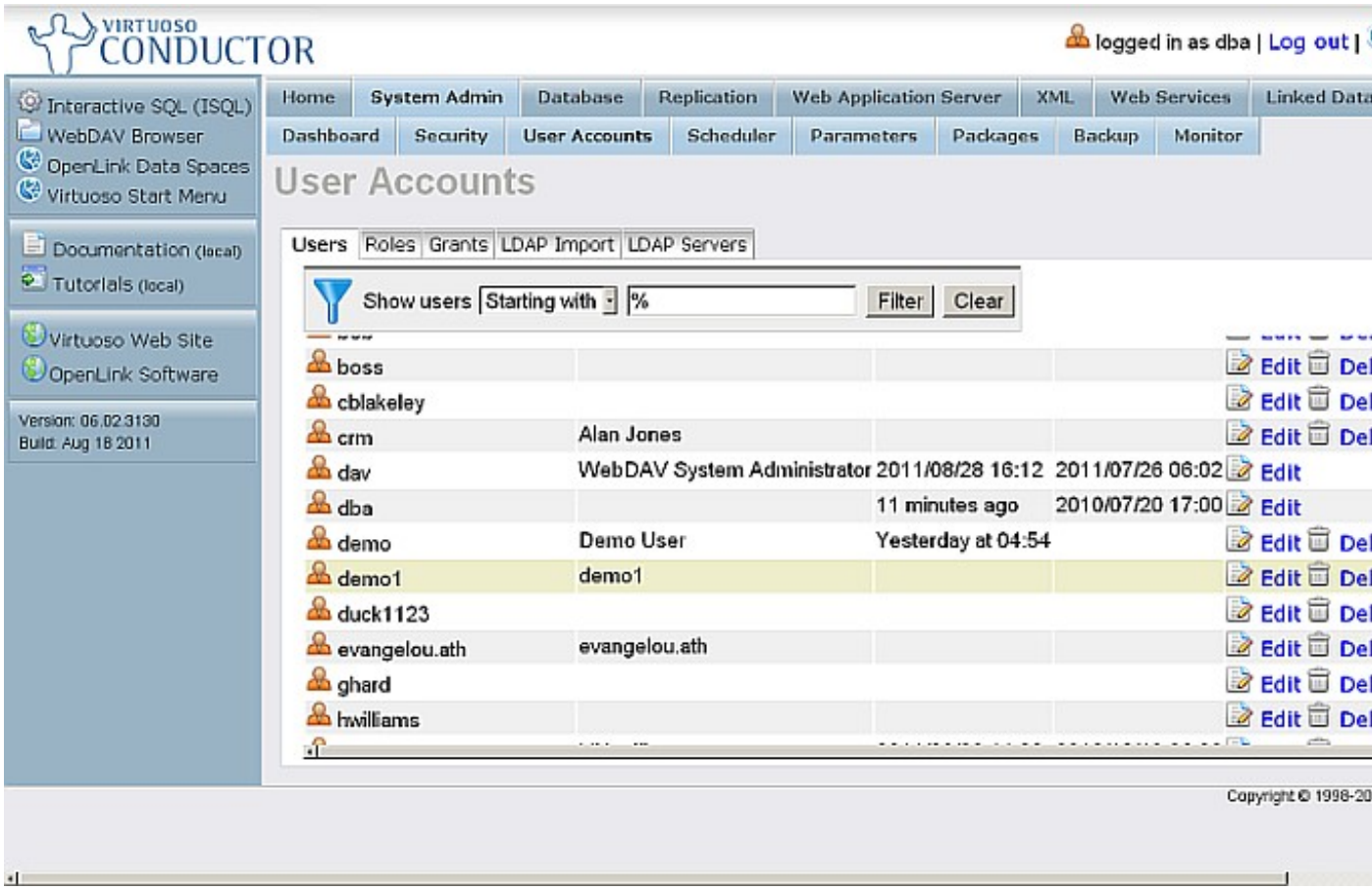
a. Account name, for ex:demo1; a password and then confirm the password;
b. User type: SQL/ODBC and WebDAV;
c. Account role: SPARQL_UPDATE

Figure 16.11. SPARQL OAuth Endpoint



8. Click the "Save" button.
9. The created user should be shown in the list of registered users:

Figure 16.12. SPARQL OAuth Endpoint

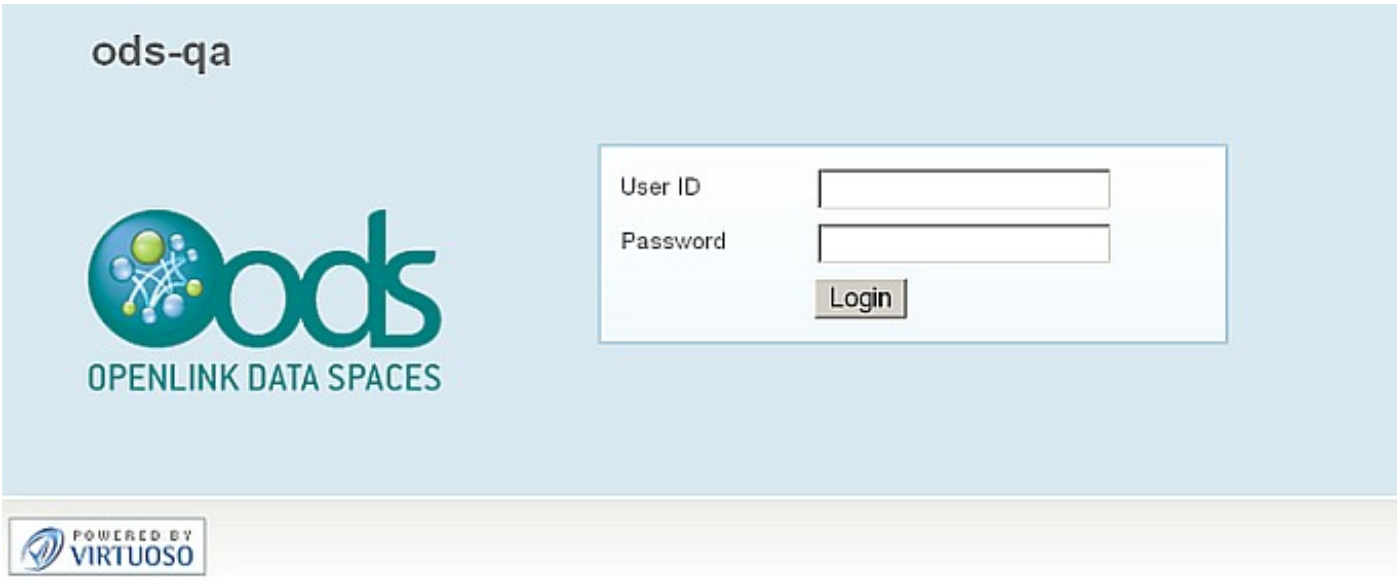


10. Go to http://<cname>:<port>/oauth/, where <cname>:<port> are replaced by your local server values.

Figure 16.13. SPARQL OAuth Endpoint

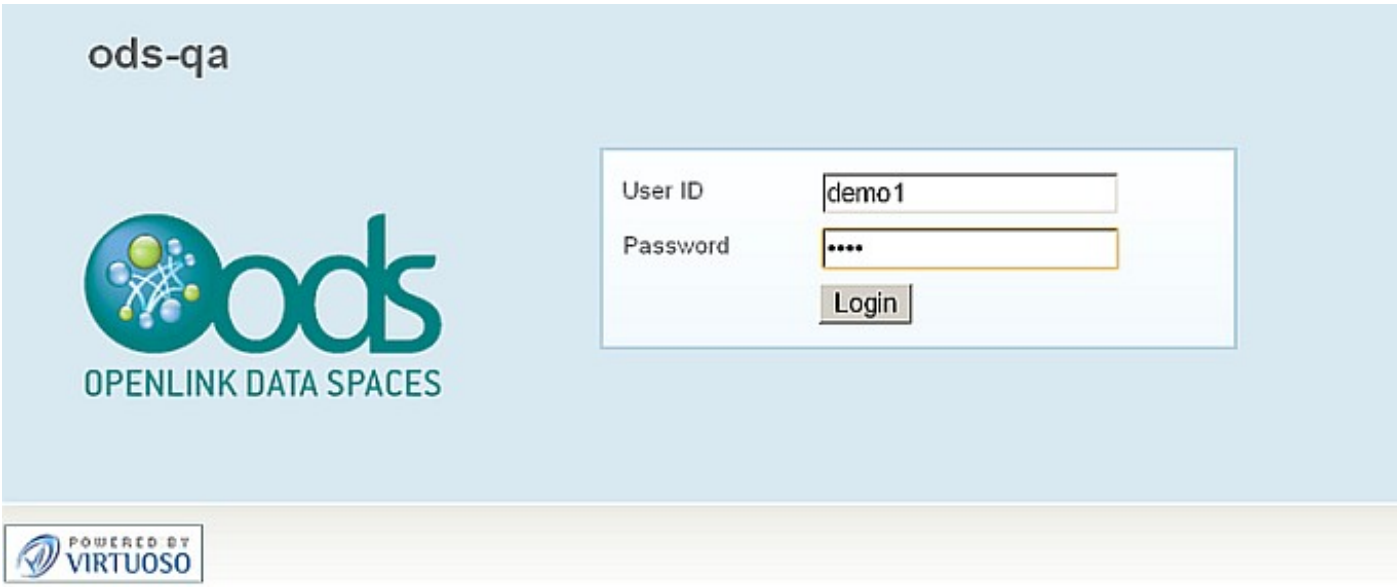
11. Click the "OAuth keys" link:

Figure 16.14. SPARQL OAuth Endpoint



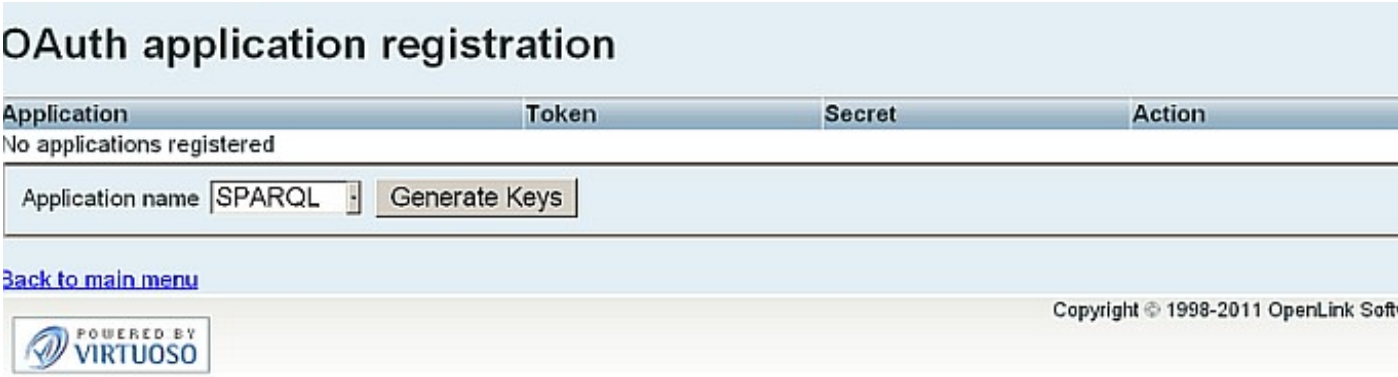
12. Log in as user demo1:

Figure 16.15. SPARQL OAuth Endpoint



13. The OAuth application registration form will be shown.

Figure 16.16. SPARQL OAuth Endpoint



14. Select SPARQL from the "Application name" list, and click the "Generate Keys" button.
15. A Consumer Key for SPARQL will be generated:

```
90baa79108b1d972525bacc76c0279c02d6421e8
```

Figure 16.17. SPARQL OAuth Endpoint

16. Click the "Back to main menu" link.
17. Click the "Protected SPARQL Endpoint" link.
18. The OpenLink Virtuoso SPARQL Query form will be displayed.

Figure 16.18. SPARQL OAuth Endpoint

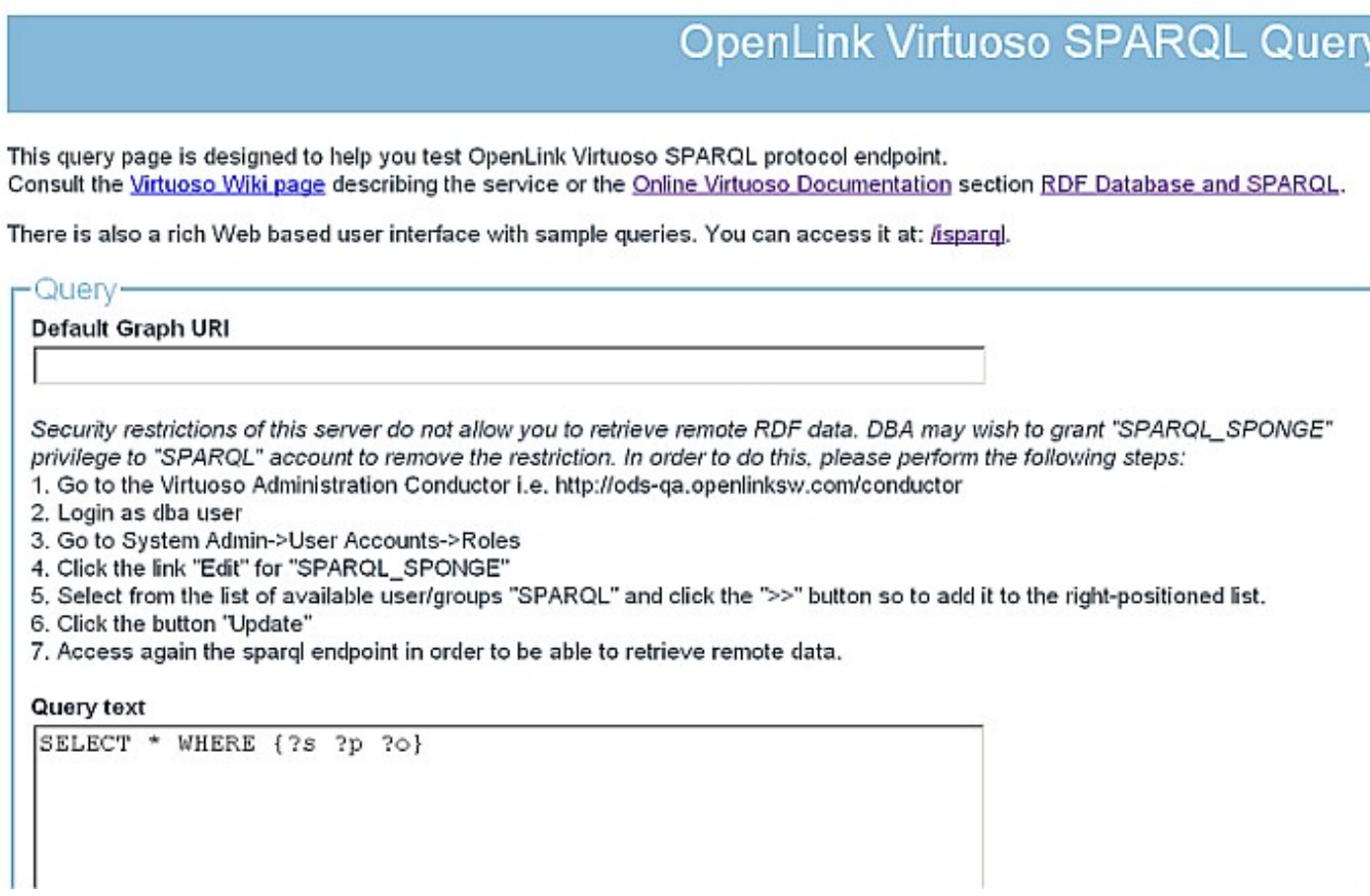


Figure 16.19. SPARQL OAuth Endpoint

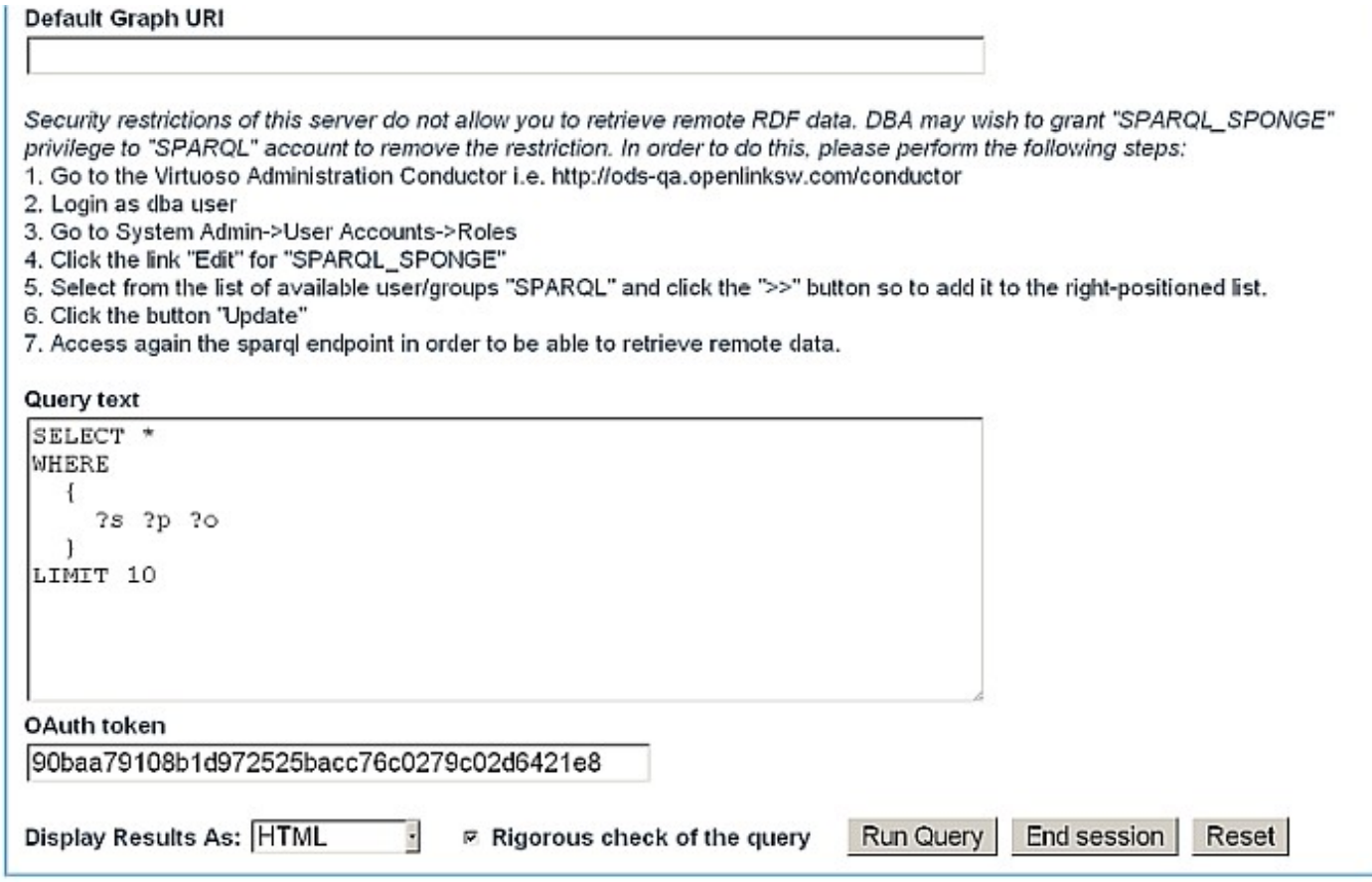
19. Enter a simple query, for ex:

```
SELECT *
WHERE
  {
    ?s ?p ?o
  }
LIMIT 10
```

20. Enter the value from below for the "OAuth token":

```
90baa79108b1d972525bacc76c0279c02d6421e8
```

Figure 16.20. SPARQL OAuth Endpoint

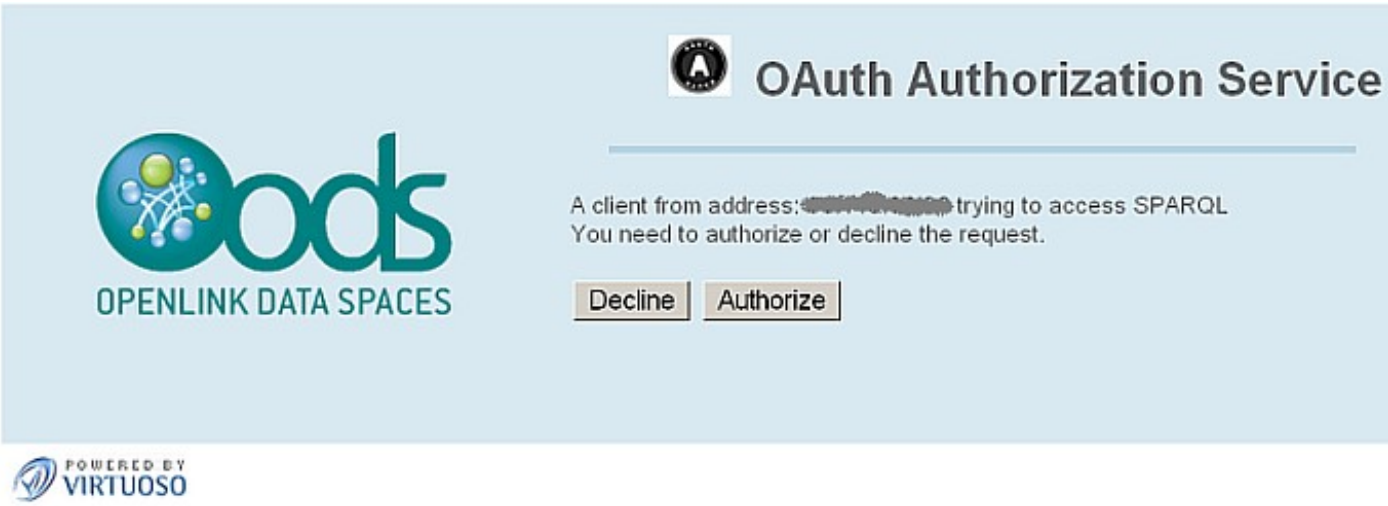


21. Click the "Run Query" button.
22. In the OAuth Authorization Service form enter the password for user demo1 and click the "Login" button.

Figure 16.21. SPARQL OAuth Endpoint

23. Next you should authorize the request:

Figure 16.22. SPARQL OAuth Endpoint



24. On successful authentication and authorization, the query results should be shown:

Figure 16.23. SPARQL OAuth Endpoint

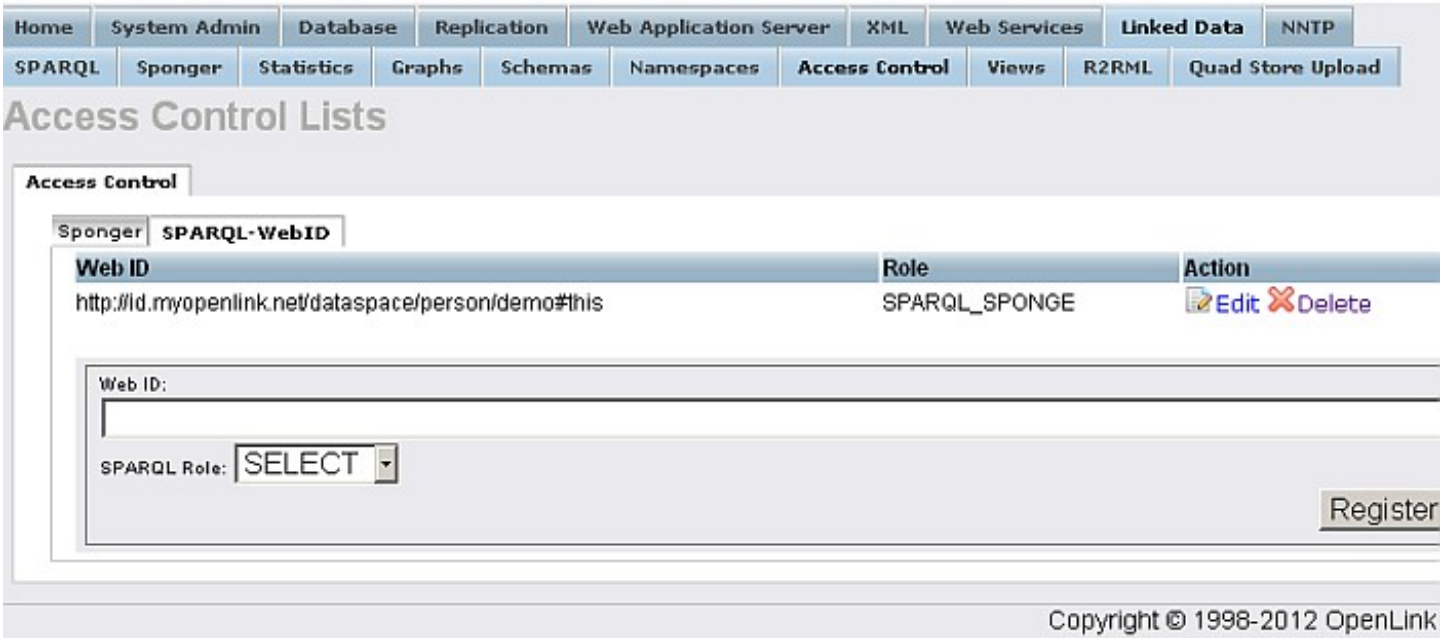
s	p	o
http://www.openlinksw.com/virtrdf-data-formats#default-iid	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapForm
http://www.openlinksw.com/virtrdf-data-formats#default-iid-nullable	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapForm
http://www.openlinksw.com/virtrdf-data-formats#default-iid-nonblank	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapForm
http://www.openlinksw.com/virtrdf-data-formats#default-iid-nonblank-nullable	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapForm
http://www.openlinksw.com/virtrdf-data-formats#default	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapForm
http://www.openlinksw.com/virtrdf-data-formats#default-nullable	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapForm
http://www.openlinksw.com/virtrdf-data-formats#sql-varchar	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapForm
http://www.openlinksw.com/virtrdf-data-formats#sql-varchar-nullable	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapForm
http://www.openlinksw.com/virtrdf-data-formats#sql-varchar-dt	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapForm
http://www.openlinksw.com/virtrdf-data-formats#sql-varchar-dt-nullable	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.openlinksw.com/schemas/virtrdf#QuadMapForm

WebID Protocol ACLs

WebID Protocol is an implementation of a conceptual authentication and authorization protocol that links a Web ID to a public key, to create a global decentralized/distributed, and open yet secure authentication system that functions with existing browsers.

To use WebID Protocol, download and install the conductor_dav.vad VAD package. Once installed, to access the WebID Protocol ACLs UI, go to URL http://cname:port/conductor -> Linked Data -> Access Controls -> SPARQL-WebID .

Figure 16.24. WebID



Configuring WebID Protocol ACLs is with a WebID Protocol certificate and a Web ID allows secure SPARQL queries to be performed against a Virtuoso SPARQL-WebID endpoint and viewing of the query results. The SPARQL-WebID endpoint URL is of the form https://cname:port/sparql-webid

Note: SPARQL-SSL is alias of SPARQL-WebID.

See sample example how to configure a sample WebID Protocol ACL are outlined below:

See Also:
WebID Protocol ODBC Login

Creating and Using a SPARQL-WebID based Endpoint

The following section describes the basic steps for setting up an SSL protected and WebID based SPARQL Endpoint (SPARQL-WebID). The guide also covers the use of Virtuoso PL functions and the Virtuoso Conductor for SPARQL endpoint creation and configuration. It also covers the use of cURL for exercising the newly generated SPARQL-SSL endpoint. Note: SPARQL-SSL is alias of SPARQL-WebID.

1. Setup the CA issuer and https listener
2. To create the /sparql-webid endpoint, install the policy_manager.vad manage or manually define the /sparql-webid endpoint on an HTTPS based listener (HTTPS service endpoint), for example using Virtuoso PL:

```
DB.DBA.VHOST_DEFINE (
         lhost=>'127.0.0.1:443',
         vhost=>'localhost',
         lpath=>'/sparql-webid',
         ppath=>'/!sparql/',
         is_dav=>1,
         auth_fn=>'DB.DBA.FOAF_SSL_AUTH',
         vsp_user=>'dba',
         ses_vars=>0,
         auth_opts=>vector ( 'https_cert',
                             'db:https_key_localhost',
                             'https_key',
                             'db:https_key_localhost',
                             'https_verify',
                             3,
                             'https_cv_depth',
                             10 ),
         opts=>vector ('noinherit', 1),
         is_default_host=>0
);
```

3. Setup the SPARQL-WebID endpoint and define ACLs using the Virtuoso Conductor
4. Export your private key and its associated WebID based X.509 certificate from your Firefox browser or System's Key Manager into PEM (PKCS12) file

    a. If using Firefox use the menu path: Advanced -> View Certificates, then click Backup for your certificate with name "mykey".
    b. The file "mykey.p12" will be created. To disable password protection so that you can use this file in non-interactive mode (e.g. with cURL and other HTTP clients) execute:

```
openssl pkcs12 -in mykey.p12 -out mykey.pem -nodes
```

5. Test the SPARQL-WebID endpoint with cURL: (listening on default HTTPS 443 port):

    - Note: In this example we use the "-k / --insecure" option with cURL since we are going to be using self-signed X.509 certificates signed by self-signed root CA.

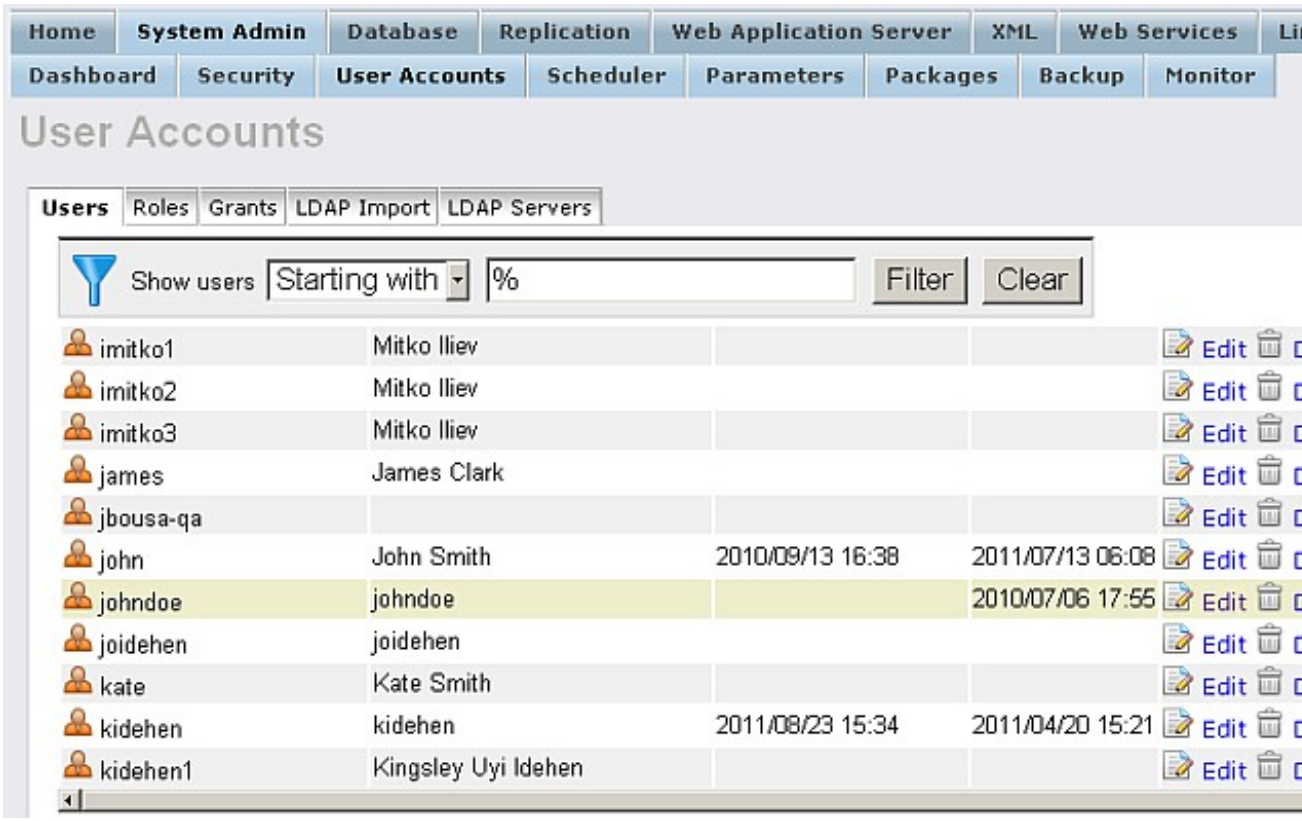
```
curl -k -E mykey.pem "https://example.com/sparql-webid?query=select+*+where+\{+%3Fx+%3Fy+%

@prefix res: <http://www.w3.org/2005/sparql-results#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
_:_ a res:ResultSet .
_:_ res:resultVariable "x" , "y" , "z" .
@prefix ns0:    <https://example.com/tutorial/> .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
_:_ res:solution [
      res:binding [ res:variable "x" ; res:value ns0:hosting ] ;
      res:binding [ res:variable "y" ; res:value rdf:type ] ;
```

```
      res:binding [ res:variable "z" ; res:value "Tutorial" ] ] .
_:_ res:solution [
      res:binding [ res:variable "x" ; res:value ns0:xml ] ;
      res:binding [ res:variable "y" ; res:value rdf:type ] ;
      res:binding [ res:variable "z" ; res:value "Tutorial" ] ] .
_:_ res:solution [
      res:binding [ res:variable "x" ; res:value ns0:repl ] ;
      res:binding [ res:variable "y" ; res:value rdf:type ] ;
      res:binding [ res:variable "z" ; res:value "Tutorial" ] ] .
_:_ res:solution [
      res:binding [ res:variable "x" ; res:value ns0:rdfview ] ;
      res:binding [ res:variable "y" ; res:value rdf:type ] ;
      res:binding [ res:variable "z" ; res:value "Tutorial" ] ] .
_:_ res:solution [
      res:binding [ res:variable "x" ; res:value ns0:services ] ;
      res:binding [ res:variable "y" ; res:value rdf:type ] ;
      res:binding [ res:variable "z" ; res:value "Tutorial" ] ] .
_:_ res:solution [
      res:binding [ res:variable "x" ; res:value ns0:wap ] ;
      res:binding [ res:variable "y" ; res:value rdf:type ] ;
      res:binding [ res:variable "z" ; res:value "Tutorial" ] ] .
_:_ res:solution [
      res:binding [ res:variable "x" ; res:value ns0:bpeldemo ] ;
      res:binding [ res:variable "y" ; res:value rdf:type ] ;
      res:binding [ res:variable "z" ; res:value "Tutorial" ] ] .
_:_ res:solution [
      res:binding [ res:variable "x" ; res:value ns0:web ] ;
      res:binding [ res:variable "y" ; res:value rdf:type ] ;
      res:binding [ res:variable "z" ; res:value "Tutorial" ] ] .
_:_ res:solution [
      res:binding [ res:variable "x" ; res:value ns0:web2 ] ;
      res:binding [ res:variable "y" ; res:value rdf:type ] ;
      res:binding [ res:variable "z" ; res:value "Tutorial" ] ] .
_:_ res:solution [
      res:binding [ res:variable "x" ; res:value ns0:xmlxslt ] ;
      res:binding [ res:variable "y" ; res:value rdf:type ] ;
      res:binding [ res:variable "z" ; res:value "Tutorial" ] ] .
```

6. Import your key it via Conductor UI:

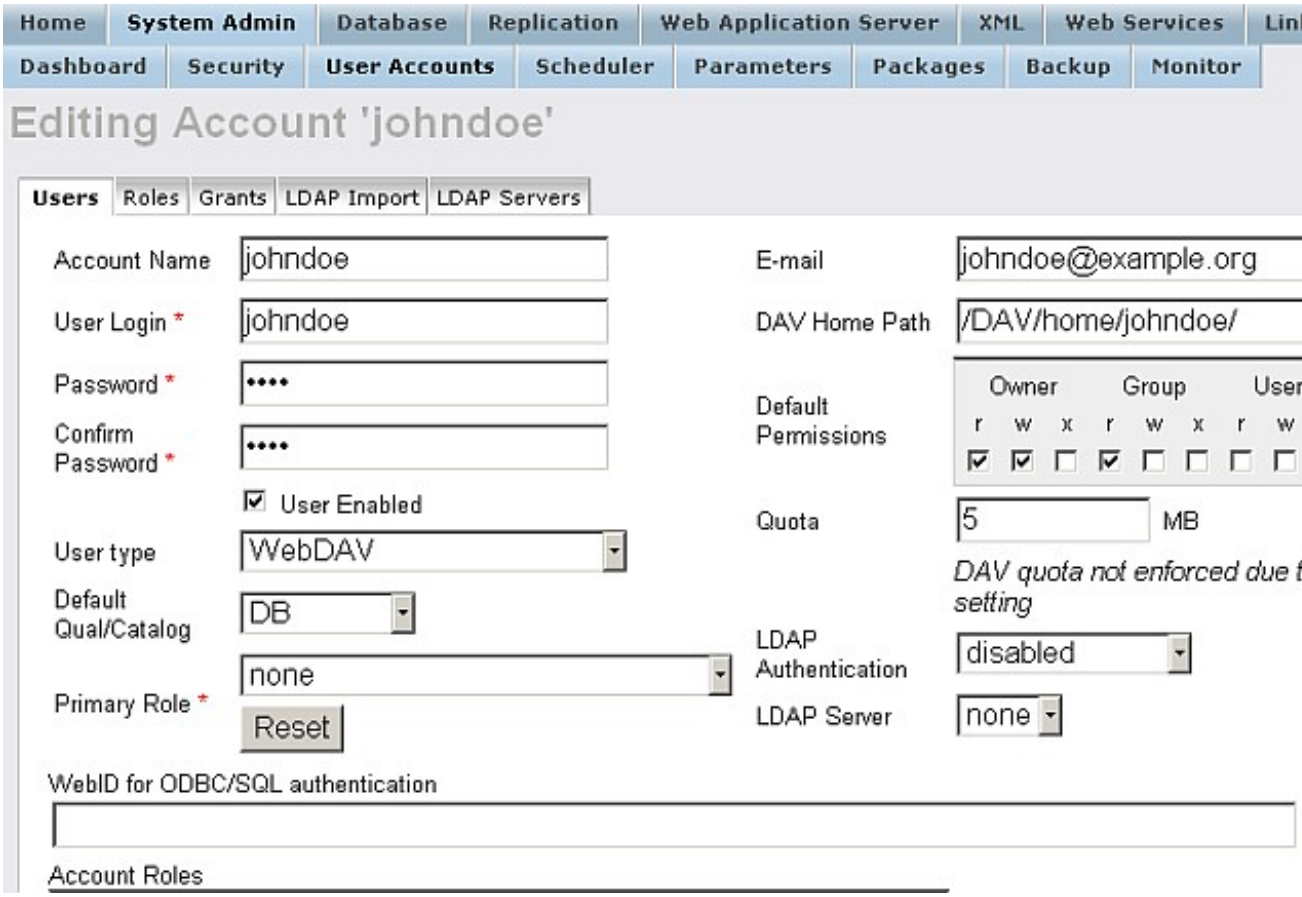
    a. Go to Conductor -> System Admin->User Accounts

Figure 16.25. Import key it via Conductor UI



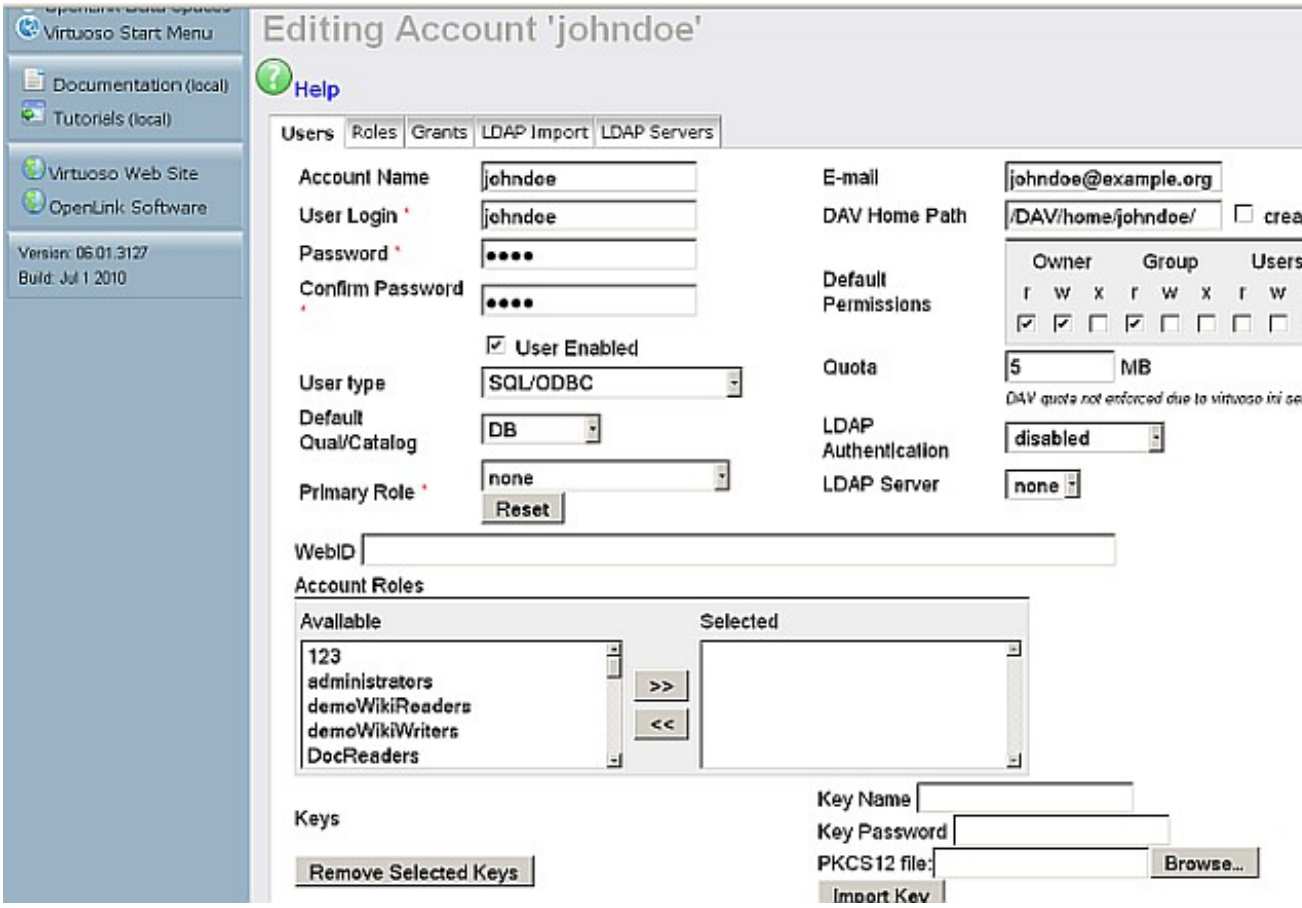
b. Click "Edit" for your user

Figure 16.26. Import key it via Conductor UI



c. Change "User type" to: SQL/ODBC and WebDAV

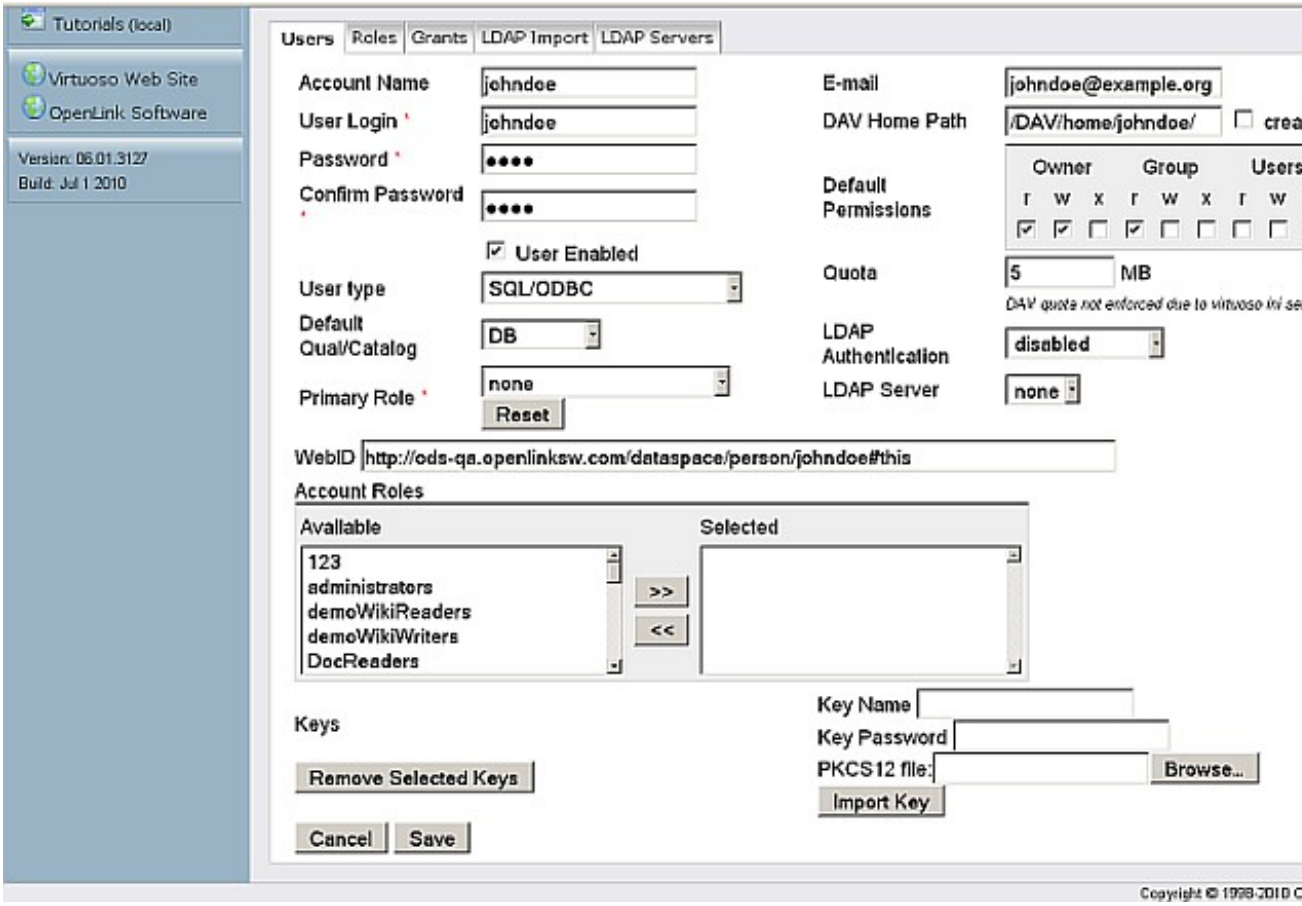
Figure 16.27. Import key it via Conductor UI



d. Enter your ODS user WebID:

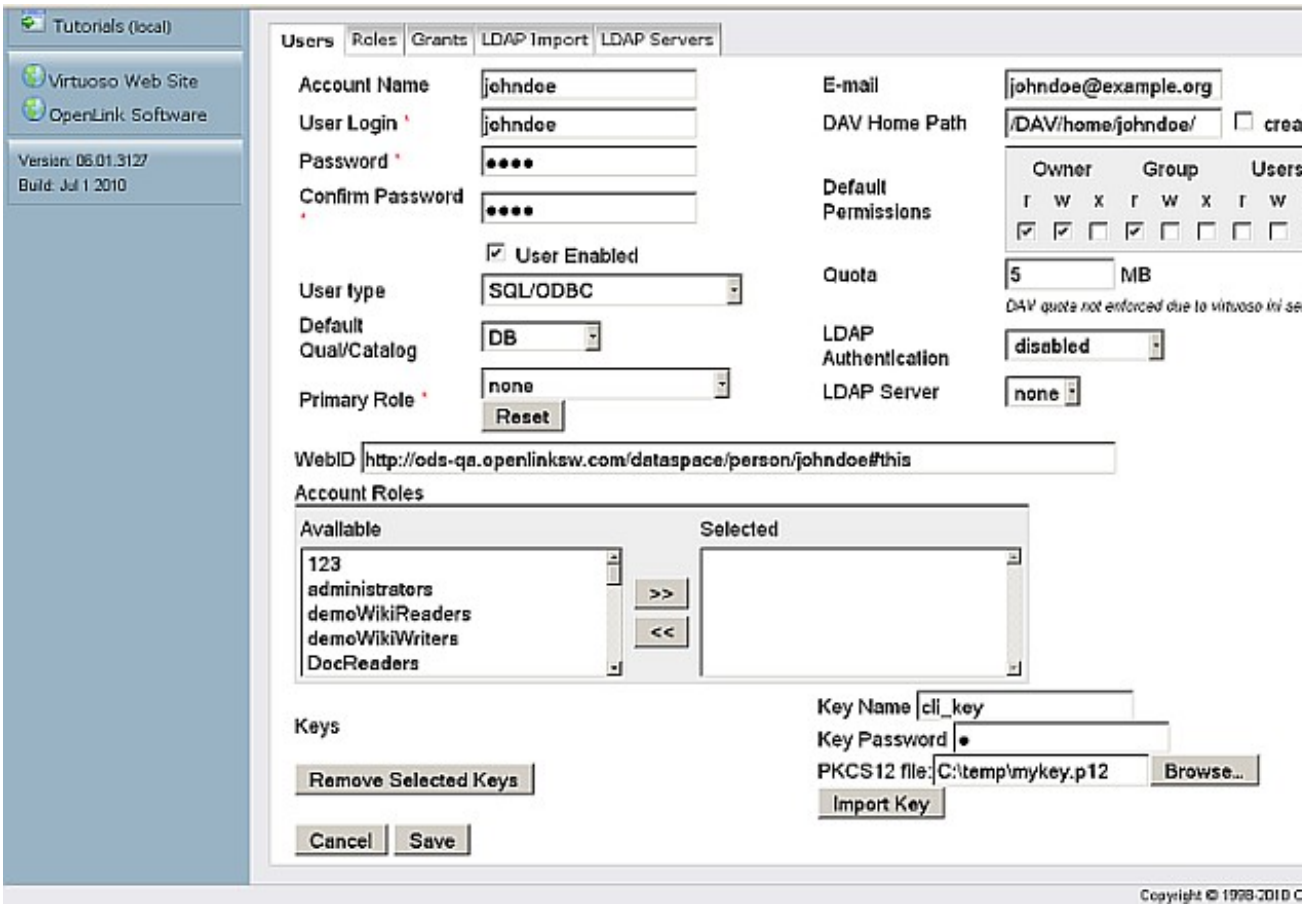
```
http://cname:port/dataspace/person/username#this
```

Figure 16.28. Import key it via Conductor UI



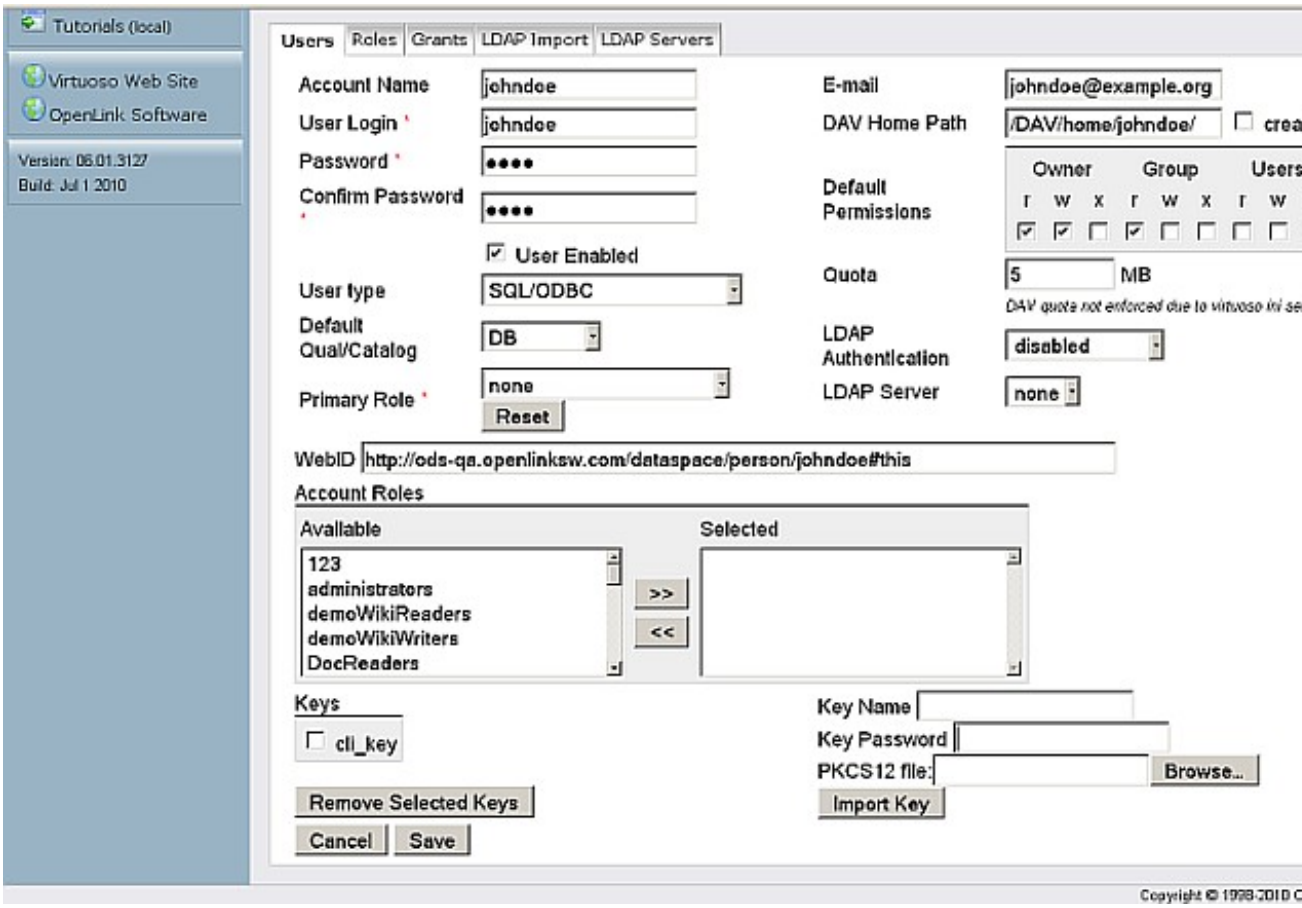
e. Click "Save"
f. Click again "Edit" for your user
g. In "PKCS12 file:" click the Browse" button and select your key.
h. Enter a local Key Name, for e.g., "cli_key"
i. Enter key password

Figure 16.29. Import key it via Conductor UI



j. Click "Import Key"
k. As result the key will be stored with name for ex. cli_key

Figure 16.30. Import key it via Conductor UI



l. Click "Save"

7. Test the SPARQL-WebID endpoint with http_client (listening on default HTTPS 443 port):

a. Log in at Virtuos ISQL with your user credentials:

```
C:\>isql localhost:1111 johndoe****
Connected to OpenLink Virtuoso
Driver: 06.01.3127 OpenLink Virtuoso ODBC Driver
OpenLink Interactive SQL (Virtuoso), version 0.9849b.
Type HELP; for help and EXIT; to exit.
SQL>
```

b. Execute:

```
SQL>select http_client ('https://example.com/sparql-webid?query=select+*+where+{+%3Fx+%3Fy+
b:cli_key', insecure=>1);
callret
VARCHAR
_______________________________________________________________________________

@prefix res: <http://www.w3.org/2005/sparql-results#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
_:_ a res:ResultSet .
_:_ res:resultVariable "x" , "y" , "z" .
@prefix ns0:    <https://example.com/tutorial/> .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
_:_ res:solution [
      res:binding [ res:variable "x" ; res:value ns0:hosting ] ;
      res:binding [ res:variable "y" ; res:value rdf:type ] ;
      res:binding [ res:variable "z" ; res:value "Tutorial" ] ] .
_:_ res:solution [
      res:binding [ res:variable "x" ; res:value ns0:xml ] ;
      res:binding [ res:variable "y" ; res:value rdf:type ] ;
      res:binding [ res:variable "z" ; res:value "Tutorial" ] ] .
_:_ res:solution [
      res:binding [ res:variable "x" ; res:value ns0:repl ] ;
      res:binding [ res:variable "y" ; res:value rdf:type ] ;
      res:binding [ res:variable "z" ; res:value "Tutorial" ] ] .
```

```
_:_ res:solution [
      res:binding [ res:variable "x" ; res:value ns0:rdfview ] ;
      res:binding [ res:variable "y" ; res:value rdf:type ] ;
      res:binding [ res:variable "z" ; res:value "Tutorial" ] ] .
_:_ res:solution [
      res:binding [ res:variable "x" ; res:value ns0:services ] ;
      res:binding [ res:variable "y" ; res:value rdf:type ] ;
      res:binding [ res:variable "z" ; res:value "Tutorial" ] ] .
_:_ res:solution [
      res:binding [ res:variable "x" ; res:value ns0:wap ] ;
      res:binding [ res:variable "y" ; res:value rdf:type ] ;
      res:binding [ res:variable "z" ; res:value "Tutorial" ] ] .
_:_ res:solution [
      res:binding [ res:variable "x" ; res:value ns0:bpeldemo ] ;
      res:binding [ res:variable "y" ; res:value rdf:type ] ;
      res:binding [ res:variable "z" ; res:value "Tutorial" ] ] .
_:_ res:solution [
      res:binding [ res:variable "x" ; res:value ns0:web ] ;
      res:binding [ res:variable "y" ; res:value rdf:type ] ;
      res:binding [ res:variable "z" ; res:value "Tutorial" ] ] .
_:_ res:solution [

1 Rows. -- 281 msec.
```

See Also:

Demo Example Using HTTP client to perform WebID Protocol connection.

Disable Default SPARQL Endpoint

Using iSQL:

1. To disable /sparql, execute:

```
DB.DBA.VHOST_REMOVE (lpath=>'/sparql');
```

2. To add the endpoint again via PL, execute:

```
DB.DBA.VHOST_DEFINE (lpath=>'/sparql/', ppath => '/!sparql/', is_dav => 1, vsp_user => 'dba', opts
```

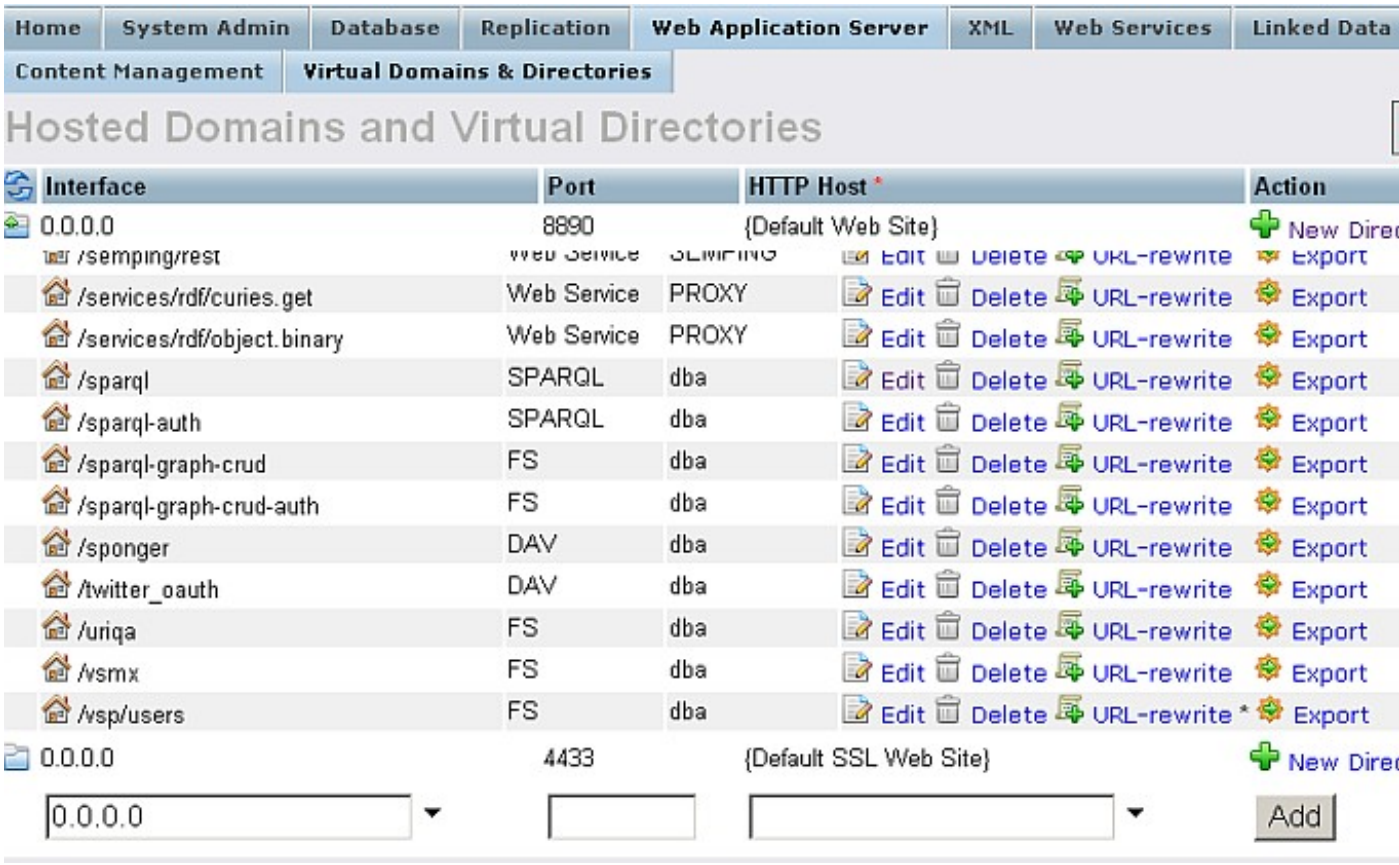
Using Conductor UI:

1. Go to http://cname:port/conductor .
2. Enter user dba credentials.
3. Go to "Web Application Server" -> "Virtual Domains & Directories".

Figure 16.31. Disable SPARQL Endpoint

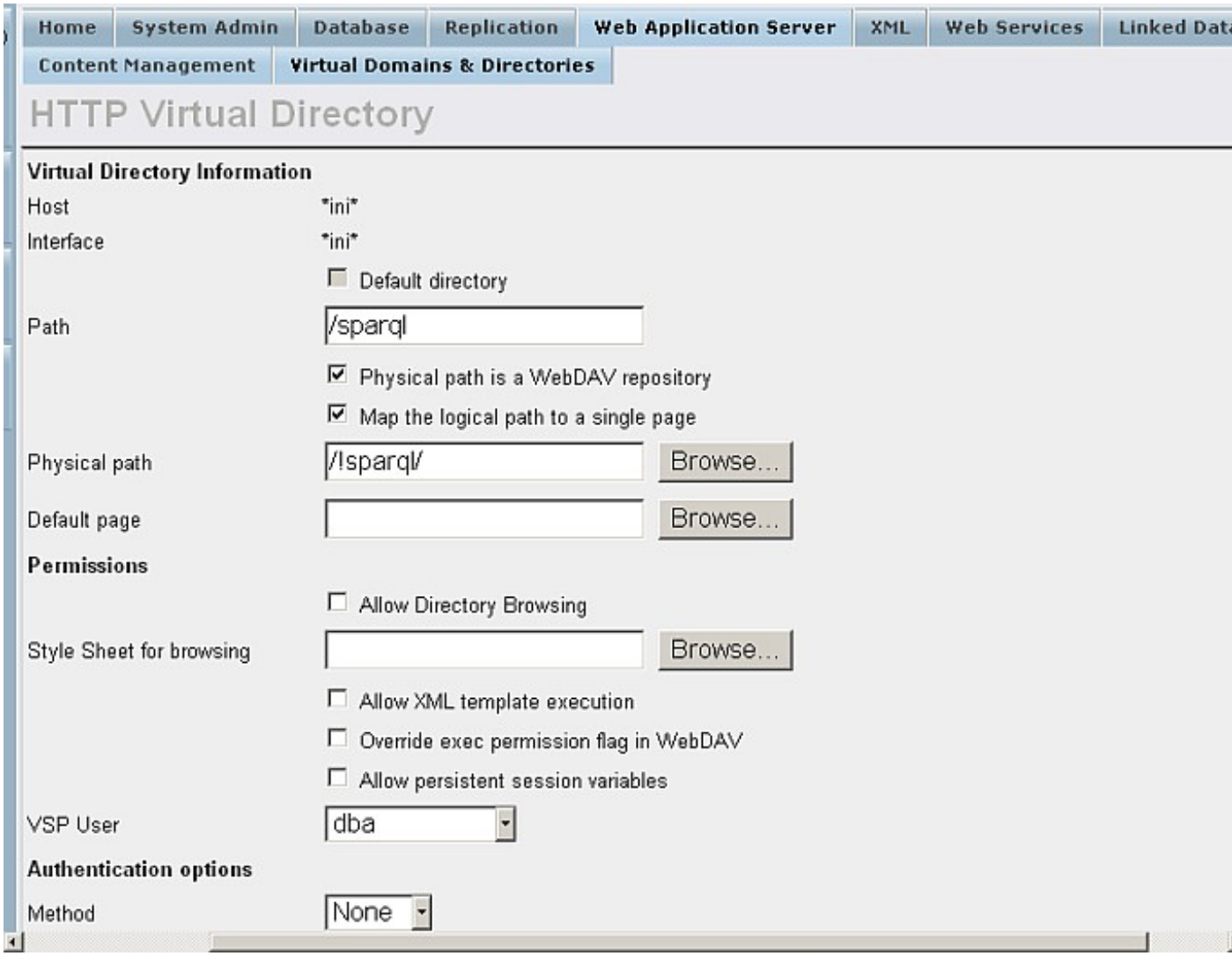
4. Find the logical path "/sparql".

Figure 16.32. Disable SPARQL Endpoint



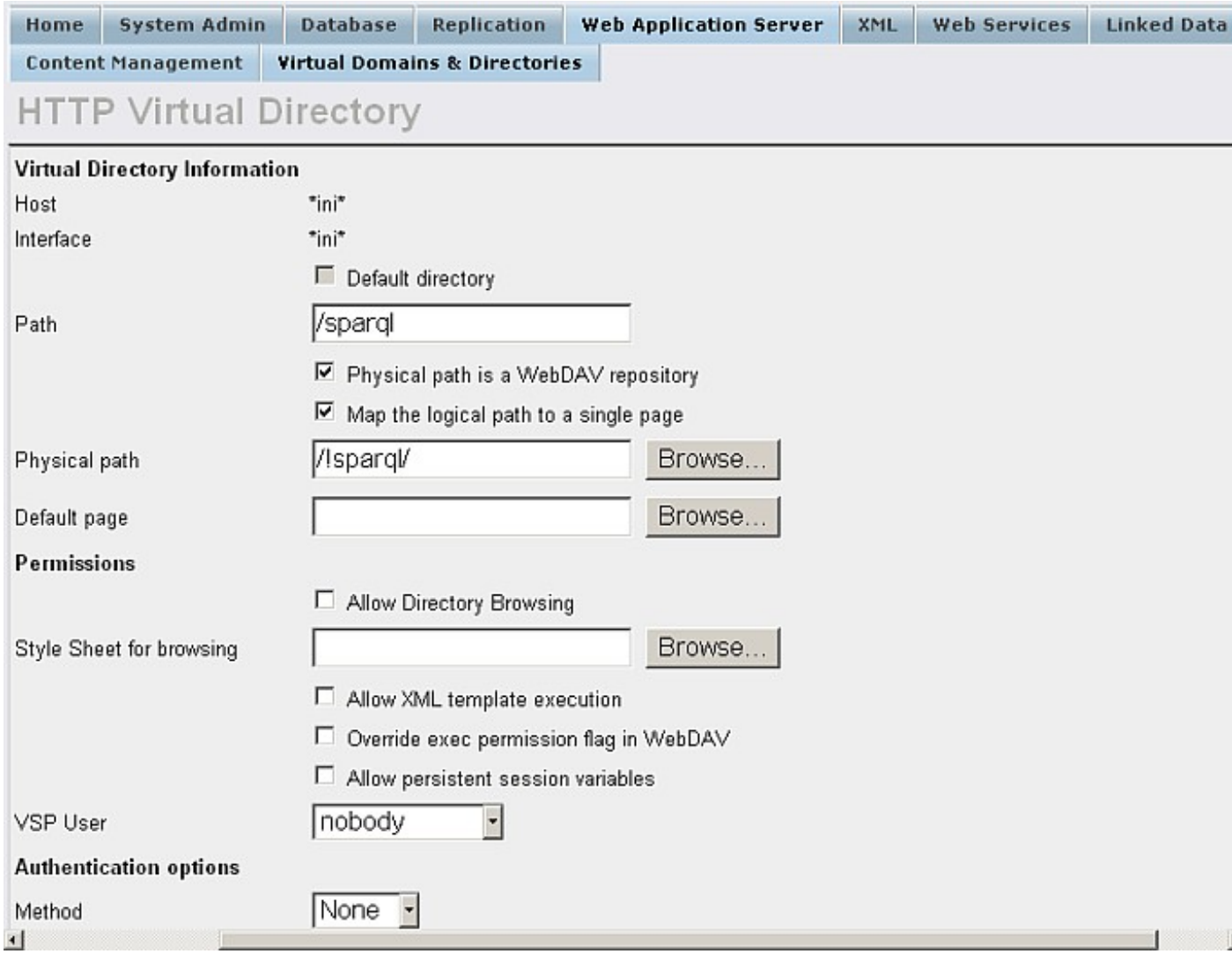
5. Click "Edit" from the "Action" column.

Figure 16.33. Disable SPARQL Endpoint



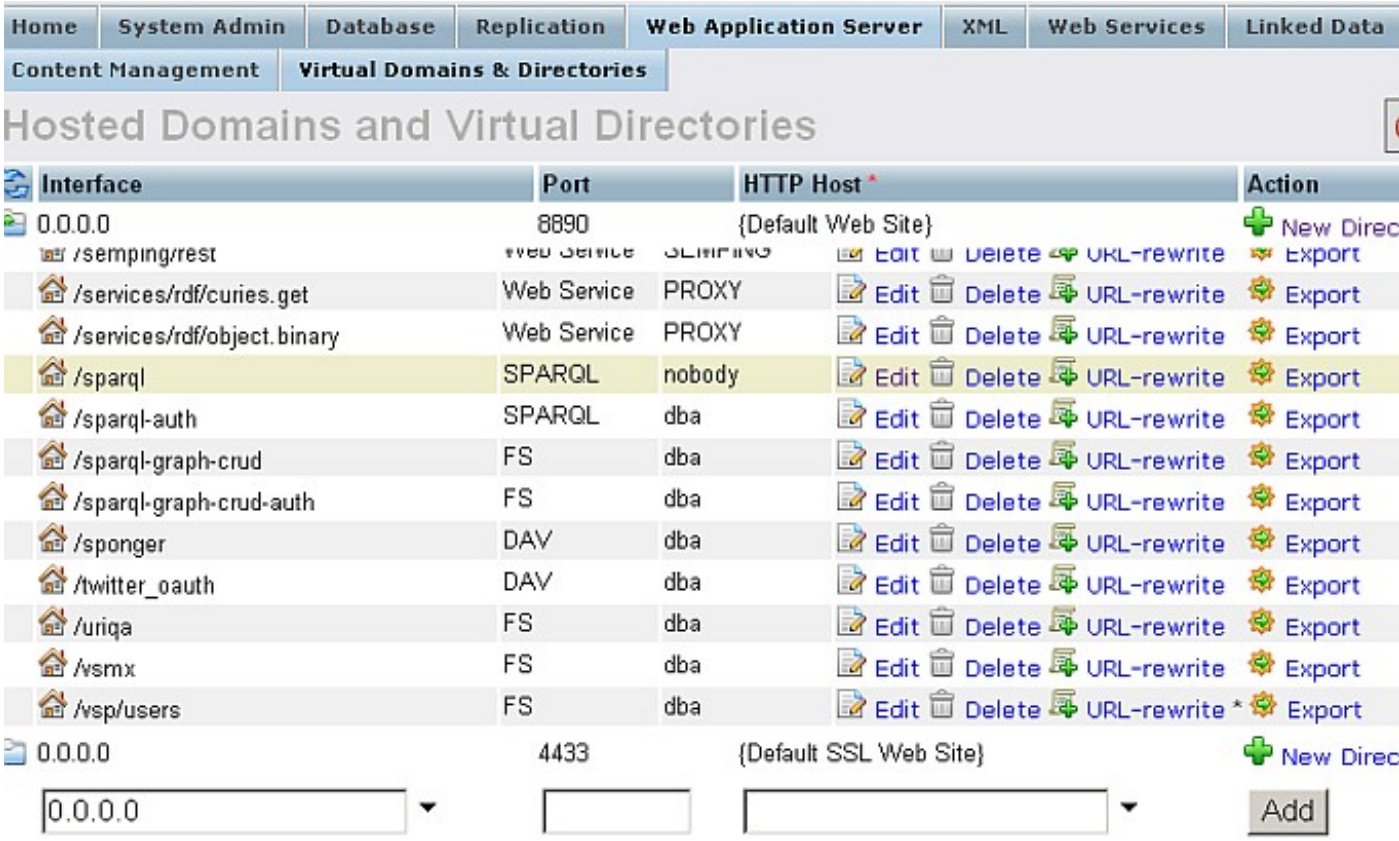
6. Change "VSP User" to "nobody".

Figure 16.34. Disable SPARQL Endpoint



7. Click "Save Changes".
8. As result the SPARQL Endpoint should be shown as disabled:

Figure 16.35. Disable SPARQL Endpoint



Request Methods

Table 16.6. Methods List

Method	Supported?	Notes
GET	Yes	Short queries are sent in GET mode
POST	Yes	Queries longer than 1900 bytes are POST-ed.
DELETE	No	
PUT	No	

Functions

The SPARQL client can be invoked by three similar functions:

Table 16.7. Functions List

Function	Notes
DB.DBA.SPARQL_REXEC	Behaves like DBA.SPARQL_EVAL, but executes the query on the specified server. The procedure does not return anything. Instead, it creates a result set.
DB.DBA.SPARQL_REXEC_TO_ARRAY	Behaves like DBA.SPARQL_EXEC_TO_ARRAY (), but executes the query on the specified server. The function return a vector of rows, where every row is represented by a vector of field values.
DB.DBA.SPARQL_REXEC_WITH_META	Has no local 'SPARQL_EVAL' analog. It produces an array of result rows together with an array of result set metadata in the same format as produced by the exec () function. This function can be used when the result should be passed later to exec_result_names () and exec_result () built-in functions. To process a local query in similar style, an application can use the SQL built-in function exec () - a SPARQL query (with the 'SPARQL' keyword in front) can be passed to exec () instead of a plain SQL SELECT statement.

```
create procedure DB.DBA.SPARQL_REXEC (
    in service varchar, in query varchar, in dflt_graph varchar, in named_graphs any,
    in req_hdr any, in maxrows integer, in bnode_dict any );

create function DB.DBA.SPARQL_REXEC_TO_ARRAY (
    in service varchar, in query varchar, in dflt_graph varchar, in named_graphs any,
    in req_hdr any, in maxrows integer, in bnode_dict any )
    returns any;

create procedure DB.DBA.SPARQL_REXEC_WITH_META (
    in service varchar, in query varchar, in dflt_graph varchar, in named_graphs any,
    in req_hdr any, in maxrows integer, in bnode_dict any,
    out metadata any,  -- metadata like exec () returns.
    out resultset any) -- results as 'long valmode' values.
```

Examples

Virtuoso's SPARQL demo offers a live demonstration of Virtuoso's implementation of the DAWG's SPARQL test-suite , a collection of SPARQL query language use cases that enable interactive and simplified testing of a triple store implementation. If you have installed the SPARQL Demo VAD locally, it can be found at a URL similar to 'http://example.com:8080/sparql_demo/', the exact form will depend on your local configuration. Alternatively, a live version of the documentation is available at Virtuoso Demo Server .

Example SPARQL query issued via curl

```
curl -F "query=SELECT DISTINCT ?p FROM <http://demo.openlinksw.com/DAV/home/demo/rdf_sink/> WHERE {?s ?p
```

The result should be:

```
<?xml version="1.0" ?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instan
 <head>
  <variable name="p"/>
 </head>
 <results distinct="false" ordered="true">
  <result>
   <binding name="p"><uri>http://www.w3.org/1999/02/22-rdf-syntax-ns#type</uri></binding>
  </result>
  <result>
   <binding name="p"><uri>http://xmlns.com/foaf/0.1/nick</uri></binding>
  </result>
  <result>
   <binding name="p"><uri>http://xmlns.com/foaf/0.1/name</uri></binding>
  </result>
  <result>
   <binding name="p"><uri>http://xmlns.com/foaf/0.1/homepage</uri></binding>
  </result>
  <result>
   <binding name="p"><uri>http://xmlns.com/foaf/0.1/knows</uri></binding>
  </result>
  <result>
   <binding name="p"><uri>http://xmlns.com/foaf/0.1/workplaceHomepage</uri></binding>
  </result>
  <result>
   <binding name="p"><uri>http://xmlns.com/foaf/0.1/mbox</uri></binding>
  </result>
 </results>
</sparql>
```

Other Examples of SPARQL query issued via curl

Further example SPARQL queries:

```
curl -F "query=SELECT DISTINCT ?Concept FROM <http://dbpedia.org> WHERE {?s a ?Concept} LIMIT 10" http://

curl -F "query=SELECT DISTINCT ?Concept FROM <http://example.com/dataspace/person/kidehen> WHERE {?s a ?C

curl -F "query=SELECT DISTINCT ?Concept FROM <http://data.openlinksw.com/oplweb/product_family/virtuoso>
```

```
curl -F "query=SELECT DISTINCT ?Concept FROM <http://openlinksw.com/dataspace/organization/openlink> WHER
```

Example with curl and SPARQL-WebID endpoint

```
$ curl -H "Accept: text/rdf+n3"  --cert test.pem -k https://demo.openlinksw.com/dataspace/person/demo
Enter PEM pass phrase: *****
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ns1:    <https://demo.openlinksw.com/dataspace/demo/socialnetwork/demo%27s%20AddressBook/1046#> .
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .
ns1:this        rdf:type        foaf:Person .
@prefix ns3:    <http://www.pipian.com/rdf/tami/juliette.n3#> .
ns3:juliette    rdf:type        foaf:Document .
@prefix ns4:    <https://demo.openlinksw.com/dataspace/person/> .
ns4:demo        rdf:type        foaf:PersonalProfileDocument .
@prefix ns5:    <https://demo.openlinksw.com/dataspace/person/demo#> .
@prefix geo:    <http://www.w3.org/2003/01/geo/wgs84_pos#> .
ns5:based_near  rdf:type        geo:Point .
@prefix ns7:    <https://demo.openlinksw.com/dataspace/demo/socialnetwork/demo%27s%20AddressBook/1042#> .
ns7:this        rdf:type        foaf:Person .
ns5:this        rdf:type        foaf:Person .
@prefix ns8:    <https://demo.openlinksw.com/dataspace/person/demo/online_account/> .
@prefix sioc:   <http://rdfs.org/sioc/ns#> .
ns8:demo        rdf:type        sioc:User .
@prefix ns10:   <https://demo.openlinksw.com/dataspace/demo/socialnetwork/myAddressBook/1001#> .
ns10:this       rdf:type        foaf:Person .
@prefix ns11:   <https://demo.openlinksw.com/dataspace/demo/socialnetwork/demo%27s%20AddressBook/1045#> .
ns11:this       rdf:type        foaf:Person .
@prefix ns12:   <https://demo.openlinksw.com/dataspace/demo#> .
ns12:this       rdf:type        sioc:User .
ns5:org rdf:type        foaf:Organization .
@prefix ns13:   <https://demo.openlinksw.com/dataspace/demo/socialnetwork/demo%27s%20AddressBook/1048#> .
ns13:this       rdf:type        foaf:Person .
@prefix ns14:   <https://demo.openlinksw.com/dataspace/demo/socialnetwork/myAddressBook/1001#this#> .
ns14:org        rdf:type        foaf:Organization .
@prefix ns15:   <https://demo.openlinksw.com/dataspace/person/imitko#> .
ns15:this       rdf:type        foaf:Person .
@prefix ns16:   <https://demo.openlinksw.com/dataspace/demo/socialnetwork/myAddressBook/1049#> .
ns16:this       rdf:type        foaf:Person .
@prefix ns17:   <https://demo.openlinksw.com/dataspace/demo/socialnetwork/myAddressBook/1000#> .
ns17:this       rdf:type        foaf:Person .
ns8:MySpace     rdf:type        sioc:User .
@prefix ns18:   <https://demo.openlinksw.com/dataspace/demo/socialnetwork/demo%27s%20AddressBook/1044#> .
ns18:this       rdf:type        foaf:Person .
@prefix dc:     <http://purl.org/dc/elements/1.1/> .
ns4:demo        dc:title        "demo demo's FOAF file" .
ns14:org        dc:title        "OpenLink" .
ns5:org dc:title        "OpenLink" .
ns18:this       foaf:name       "Kingsley Idehen" .
ns13:this       foaf:name       "Juliette" .
ns17:this       foaf:name       "Kingsley Idehen" .
ns5:this        foaf:name       "demo demo" .
ns15:this       foaf:name       "Mitko Iliev" .
ns10:this       foaf:name       "test test12" .
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
ns5:this        rdfs:seeAlso    ns4:demo .
ns15:this       rdfs:seeAlso    ns4:imitko .
ns4:demo        foaf:maker      ns5:this .
ns15:this       foaf:nick       "imitko" .
ns7:this        foaf:nick       "Orri Erling" .
ns13:this       foaf:nick       "Juliette" .
ns10:this       foaf:nick       "test1" .
ns5:this        foaf:nick       "demo" .
ns18:this       foaf:nick       "Kingsley" .
ns17:this       foaf:nick       "Kingsley" .
ns16:this       foaf:nick       "test2" .
ns1:this        foaf:nick       "TEST" .
ns11:this       foaf:nick       "TEST" .
ns5:this        foaf:holdsAccount       ns8:demo ,
                ns8:MySpace ,
                ns12:this .
@prefix ns21:   <http://example.com/dataspace/person/imitko#> .
ns5:this        foaf:knows      ns21:this ,
                ns17:this ,
```

```
                ns16:this ,
                ns3:juliette ,
                ns10:this ,
                ns7:this .
@prefix ns22:   <http://example.com/dataspace/person/kidehen#> .
ns5:this        foaf:knows      ns22:this ,
                ns18:this ,
                ns11:this ,
                ns1:this .
@prefix ns23:   <http://bblfish.net/people/henry/card#me\u0020> .
ns5:this        foaf:knows      ns23: ,
                ns13:this ,
                ns15:this ;
        foaf:firstName  "demo" ;
        foaf:family_name        "demo" ;
        foaf:gender     "male" ;
        foaf:icqChatID  "125968" ;
        foaf:msnChatID  "45demo78" ;
        foaf:aimChatID  "demo1234" ;
        foaf:yahooChatID        "demo678" ;
        foaf:based_near ns5:based_near .
@prefix ns24:   <http://www.openlinksw.com> .
ns5:this        foaf:workplaceHomepage  ns24: .
ns5:org foaf:homepage   ns24: .
ns5:this        foaf:homepage   ns24: .
ns14:org        foaf:homepage   ns24: .
ns4:demo        foaf:primaryTopic       ns5:this .
ns5:based_near  geo:lat "47.333332" ;
        geo:long        "13.333333" .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ns1:    <https://demo.openlinksw.com/dataspace/demo#> .
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .
ns1:this        rdf:type        foaf:OnlineAccount .
@prefix ns3:    <https://demo.openlinksw.com/dataspace/person/demo/online_account/> .
ns3:MySpace     rdf:type        foaf:OnlineAccount .
ns3:demo        rdf:type        foaf:OnlineAccount .
@prefix ns4:    <https://demo.openlinksw.com/dataspace/person/demo#> .
ns4:this        foaf:holdsAccount       ns3:MySpace ,
                ns1:this ,
                ns3:demo .
@prefix vcard:  <http://www.w3.org/2001/vcard-rdf/3.0#> .
ns4:this        vcard:ADR       ns4:addr .
ns4:addr        vcard:Country   "United States" ;
        vcard:Locality  "New York" ;
        vcard:Region    "Nebraska" .
@prefix ns6:    <http://myspace.com> .
ns3:MySpace     foaf:accountServiceHomepage     ns6: .
@prefix ns7:    <skype:demo?> .
ns3:demo        foaf:accountServiceHomepage     ns7:chat ;
        foaf:accountName        "demo" .
ns3:MySpace     foaf:accountName        "MySpace" .
@prefix ns8:    <http://vocab.org/bio/0.1/> .
ns4:this        ns8:olb "this is short resume of user Demo." .
@prefix ns9:    <https://demo.openlinksw.com/dataspace/> .
ns4:this        foaf:openid     ns9:demo ;
        ns8:keywords    "demo, openlinksw, virtuoso, weblog, rdf" .
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .
@prefix ns1:    <https://demo.openlinksw.com/dataspace/demo/subscriptions/> .
@prefix ns2:    <https://demo.openlinksw.com/dataspace/person/demo#> .
ns1:DemoFeeds   foaf:maker      ns2:this .
@prefix ns3:    <https://demo.openlinksw.com/dataspace/demo/community/> .
ns3:demoCommunity       foaf:maker      ns2:this .
@prefix ns4:    <https://demo.openlinksw.com/dataspace/demo/eCRM/demo%27s%20eCRM> .
ns4:    foaf:maker      ns2:this .
@prefix ns5:    <https://demo.openlinksw.com/dataspace/demo/calendar/> .
ns5:mycalendar  foaf:maker      ns2:this .
@prefix ns6:    <https://demo.openlinksw.com/dataspace/demo/photos/> .
ns6:MyGallery   foaf:maker      ns2:this .
@prefix ns7:    <https://demo.openlinksw.com/dataspace/demo/briefcase/> .
ns7:mybriefcase foaf:maker      ns2:this .
@prefix ns8:    <https://demo.openlinksw.com/dataspace/demo/wiki/> .
ns8:ESBWiki     foaf:maker      ns2:this .
@prefix ns9:    <https://demo.openlinksw.com/dataspace/demo/bookmark/> .
```

```
ns9:mybookmarks foaf:maker      ns2:this .
@prefix ns10:   <https://demo.openlinksw.com/dataspace/demo/weblog/> .
ns10:myblog     foaf:maker      ns2:this .
@prefix ns11:   <https://demo.openlinksw.com/dataspace/demo/socialnetwork/demo%27s%20AddressBook> .
ns11:   foaf:maker      ns2:this .
@prefix ns12:   <https://demo.openlinksw.com/dataspace/demo/community/demo%27s%20Community> .
ns12:   foaf:maker      ns2:this .
ns8:mywiki      foaf:maker      ns2:this .
@prefix ns13:   <https://demo.openlinksw.com/dataspace/demo/eCRM/demo%20demo%27s%20eCRM> .
ns13:   foaf:maker      ns2:this .
@prefix ns14:   <https://demo.openlinksw.com/dataspace/demo/polls/> .
ns14:mypolls    foaf:maker      ns2:this .
@prefix ns15:   <https://demo.openlinksw.com/dataspace/demo/socialnetwork/> .
ns15:myAddressBook      foaf:maker      ns2:this .
ns3:SP2 foaf:maker      ns2:this .
ns2:this        foaf:made       ns11: ,
                ns4: ,
                ns3:demoCommunity ,
                ns12: ,
                ns15:myAddressBook ,
                ns10:myblog ,
                ns9:mybookmarks ,
                ns7:mybriefcase ,
                ns5:mycalendar ,
                ns14:mypolls ,
                ns8:mywiki ,
                ns1:DemoFeeds ,
                ns8:ESBWiki ,
                ns6:MyGallery ,
                ns3:SP2 ,
                ns13: .
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
ns9:mybookmarks rdfs:label      "demo demo's Bookmarks" .
ns15:myAddressBook      rdfs:label      "demo demo's AddressBook" .
ns4:    rdfs:label      "demo demo's eCRM" .
ns12:   rdfs:label      "demo's Community" .
ns14:mypolls    rdfs:label      "demo demo's Polls" .
ns13:   rdfs:label      "demo demo's eCRM Description" .
ns8:mywiki      rdfs:label      "demo demo's Wiki" .
ns7:mybriefcase rdfs:label      "demo demo's Briefcase" .
ns1:DemoFeeds   rdfs:label      "demo demo's Feeds" .
ns10:myblog     rdfs:label      "demo's Weblog" .
ns5:mycalendar  rdfs:label      "demo demo's Calendar" .
ns11:   rdfs:label      "demo demo's AddressBook" .
ns6:MyGallery   rdfs:label      "demo demo's Gallery" .
ns8:ESBWiki     rdfs:label      "demo demo's Wiki" .
ns3:demoCommunity       rdfs:label      "demo demo's Community" .
ns3:SP2 rdfs:label      "demo demo's Community" .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ns1:    <https://demo.openlinksw.com/dataspace/person/demo#> .
@prefix ns2:    <http://www.w3.org/ns/auth/rsa#> .
ns1:cert        rdf:type        ns2:RSAPublicKey .
@prefix dc:     <http://purl.org/dc/elements/1.1/> .
@prefix ns4:    <https://demo.openlinksw.com/dataspace/person/demo/projects#ods%20project> .
ns4:    dc:title        "ods project" .
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .
ns4:    foaf:maker      ns1:this .
ns1:this        foaf:made       ns4: .
@prefix ns6:    <http://www.w3.org/ns/auth/cert#> .
ns1:cert        ns6:identity    ns1:this ;
        ns2:modulus     ns1:cert_mod .
ns1:cert_mod    ns6:hex "b8edefa13092d05e85257d6be0aca54218091278583f1d18759c4bced0007948fa6e920018abc3c3
e1bd08dca66b7737b744fd9e441ebefa425311363711714cd0fe3b334a79ce50be9eb3443193bcbf2f1486481e775382f1a1792a2
6be0169a1859d027170812a28914d158fb76a5933f11777a06c8db64d10f7c02900c4bb4bbf2d24c0e34c6ca135fdb5e05241bc02
09e1b7a3973122e850e96fcd0396544f82f0b11a46f0d868ba0f3d8efd957e7ef224871905a06c3c5d85ac9" .
ns1:cert        ns2:public_exponent     ns1:cert_exp .
ns1:cert_exp    ns6:decimal     "65537" .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ns1:    <https://demo.openlinksw.com/dataspace/person/demo#> .
@prefix ns2:    <http://vocab.org/bio/0.1/> .
ns1:event       rdf:type        ns2:Birth .
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .
@prefix ns4:    <mailto:demo@openlinksw.com> .
```

```
ns1:this        foaf:mbox       ns4: ;
        foaf:birthday   "01-01" .
@prefix dc:     <http://purl.org/dc/elements/1.1/> .
ns1:event       dc:date "1968-01-01" .
ns1:this        ns2:event       ns1:event .
```

Example with curl and SPARQL-OAuth endpoint

Note: this is just an example as token had expired already. You can go to this section to see how to interact with our Virtuoso UI.

```
$ curl "http://demo.openlinksw.com/oauth/sparql.vsp?debug=on&default-graph-uri=&format=text%2Fhtml&oauth_
8dacdd70bb5&oauth_nonce=a14d43339fcb2638&oauth_signature_method=HMAC-SHA1&oauth_timestamp=1242106643&oaut
ee95a2&oauth_version=1.0&query=select%20distinct%20%3FURI%20%3FObjectType%20where%20%7B%3FURI%20a%20%3FOb
w9yJ2saU1vgHuFxWcughai5cZY%3D"
<table class="sparql" border="1">
  <tr>
    <th>URI</th>
    <th>ObjectType</th>
  </tr>
  <tr>
    <td>http://www.openlinksw.com/virtrdf-data-formats#default-iid</td>
    <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
  </tr>
  <tr>
    <td>http://www.openlinksw.com/virtrdf-data-formats#default-iid-nullable</td>
    <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
  </tr>
  <tr>
    <td>http://www.openlinksw.com/virtrdf-data-formats#default-iid-nonblank</td>
    <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
  </tr>
  <tr>
    <td>http://www.openlinksw.com/virtrdf-data-formats#default-iid-nonblank-nullable</td>
    <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
  </tr>
  <tr>
    <td>http://www.openlinksw.com/virtrdf-data-formats#default</td>
    <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
  </tr>
  <tr>
    <td>http://www.openlinksw.com/virtrdf-data-formats#default-nullable</td>
    <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
  </tr>
  <tr>
    <td>http://www.openlinksw.com/virtrdf-data-formats#sql-varchar</td>
    <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
  </tr>
  <tr>
    <td>http://www.openlinksw.com/virtrdf-data-formats#sql-varchar-nullable</td>
    <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
  </tr>
  <tr>
    <td>http://www.openlinksw.com/virtrdf-data-formats#sql-longvarchar</td>
    <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
  </tr>
  <tr>
    <td>http://www.openlinksw.com/virtrdf-data-formats#sql-longvarchar-nullable</td>
    <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
  </tr>
  <tr>
    <td>http://www.openlinksw.com/virtrdf-data-formats#sql-longvarbinary</td>
    <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
  </tr>
  <tr>
    <td>http://www.openlinksw.com/virtrdf-data-formats#sql-longvarbinary-nullable</td>
    <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
  </tr>
  <tr>
    <td>http://www.openlinksw.com/virtrdf-data-formats#sql-varchar-uri</td>
    <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
  </tr>
  <tr>
    <td>http://www.openlinksw.com/virtrdf-data-formats#sql-varchar-uri-nullable</td>
```

```
  <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#sql-integer</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#sql-integer-nullable</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#sql-integer-uri</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#sql-integer-uri-nullable</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#sql-doubleprecision</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#sql-doubleprecision-nullable</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#sql-date</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#sql-date-nullable</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#sql-datetime</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#sql-datetime-nullable</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#multipart-uri</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#multipart-uri-nullable</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#multipart-uri-fn-nullable</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#multipart-literal-fn-nullable</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#sql-varchar-uri-fn</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#sql-varchar-uri-fn-nullable</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#sql-integer-uri-fn</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#sql-integer-uri-fn-nullable</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
```

```
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#sql-varchar-literal-fn</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#sql-varchar-literal-fn-nullable</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#sql-integer-literal-fn</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#sql-integer-literal-fn-nullable</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.w3.org/1999/02/22-rdf-syntax-ns#type</td>
  <td>http://www.w3.org/1999/02/22-rdf-syntax-ns#Property</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#default-iid-SuperFormats</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#array-of-QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#default-iid-nullable-SuperFormats</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#array-of-QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#default-iid-nonblank-SuperFormats</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#array-of-QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#default-iid-nonblank-nullable-SuperFormats</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#array-of-QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#default-SuperFormats</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#array-of-QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#default-nullable-SuperFormats</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#array-of-QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#sql-varchar-SuperFormats</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#array-of-QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#sql-varchar-nullable-SuperFormats</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#array-of-QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#sql-longvarchar-SuperFormats</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#array-of-QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#sql-longvarchar-nullable-SuperFormats</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#array-of-QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#sql-longvarbinary-SuperFormats</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#array-of-QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#sql-longvarbinary-nullable-SuperFormats</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#array-of-QuadMapFormat</td>
</tr>
<tr>
  <td>http://www.openlinksw.com/virtrdf-data-formats#sql-varchar-uri-SuperFormats</td>
  <td>http://www.openlinksw.com/schemas/virtrdf#array-of-QuadMapFormat</td>
</tr>
```

```
</table>
```

Example with CONSTRUCT

Go to the sparql endpoint UI: i.e. go to http://host:port/sparql

For the Default Graph URI enter: http://www.w3.org/2001/sw/DataAccess/proto-tests/data/construct/simple-data.rdf

Select "Retrieve remote RDF data for all missing source graphs".

For the query text enter:

```
SELECT * WHERE {?s ?p ?o}
```

Click the "Run Query" button.

The query results, shown below, are cached locally (network resources being fetched). The remote RDF data is saved in the local RDF quad store as graph http://www.w3.org/2001/sw/DataAccess/proto-tests/data/construct/simple-data.rdf

```
s                                       p                                                o
http://www.example/jose/foaf.rdf#jose     http://www.w3.org/1999/02/22-rdf-syntax-ns#type  http://xmlns.c
http://www.example/jose/foaf.rdf#jose     http://xmlns.com/foaf/0.1/nick                   Jo
http://www.example/jose/foaf.rdf#jose     http://xmlns.com/foaf/0.1/name                   Jose Jimen~ez
http://www.example/jose/foaf.rdf#jose     http://xmlns.com/foaf/0.1/knows                  http://www.exa
http://www.example/jose/foaf.rdf#jose     http://xmlns.com/foaf/0.1/homepage               http://www.exa
http://www.example/jose/foaf.rdf#jose     http://xmlns.com/foaf/0.1/workplaceHomepage      http://www.cor
http://www.example/jose/foaf.rdf#kendall  http://xmlns.com/foaf/0.1/knows                  http://www.exa
http://www.example/jose/foaf.rdf#julia    http://www.w3.org/1999/02/22-rdf-syntax-ns#type  http://xmlns.c
http://www.example/jose/foaf.rdf#julia    http://xmlns.com/foaf/0.1/mbox                   mailto:julia@m
http://www.example/jose/foaf.rdf#juan     http://www.w3.org/1999/02/22-rdf-syntax-ns#type  http://xmlns.c
http://www.example/jose/foaf.rdf#juan     http://xmlns.com/foaf/0.1/mbox                   mailto:juan@ma
```

Now let's take the CONSTRUCT query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX myfoaf: <http://www.example/jose/foaf.rdf#>
CONSTRUCT
  { myfoaf:jose foaf:depiction <http://www.example/jose/jose.jpg>.
    myfoaf:jose foaf:schoolHomepage <http://www.edu.example/>.
    ?s ?p ?o.
  }
FROM <http://www.w3.org/2001/sw/DataAccess/proto-tests/data/construct/simple-data.rdf>
WHERE
  {
    ?s ?p ?o. myfoaf:jose foaf:nick "Jo".
    FILTER ( ! (?s = myfoaf:kendall && ?p = foaf:knows && ?o = myfoaf:edd )
    && ! ( ?s = myfoaf:julia && ?p = foaf:mbox && ?o = <mailto:julia@mail.example> )
    && ! ( ?s = myfoaf:julia && ?p = rdf:type && ?o = foaf:Person))
  }
```

From an HTTP client, issue the GET command with the above query added as a URL-encoded parameter value:

```
GET -e -s http://host:port/sparql/?query=PREFIX+rdf%3A+%3Chttp%3A%2F%2Fwww.w3.org%2F1999%2F02%2F22-rdf-sy
```

The request response will be similar to:

```
200 OK
Connection: close
Date: Fri, 28 Dec 2007 10:06:14 GMT
Accept-Ranges: bytes
Server: Virtuoso/05.00.3023 (Win32) i686-generic-win-32  VDB
Content-Length: 2073
Content-Type: application/rdf+xml; charset=UTF-8
Client-Date: Fri, 28 Dec 2007 10:06:14 GMT
Client-Peer: 83.176.40.177:port
Client-Response-Num: 1
```

```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:rdfs="http://www.w3.org/2000/01/rd
<rdf:Description rdf:about="http://www.example/jose/foaf.rdf#juan"><ns0pred:mbox xmlns:ns0pred="http://xm
<rdf:Description rdf:about="http://www.example/jose/foaf.rdf#jose"><ns0pred:schoolHomepage xmlns:ns0pred=
<rdf:Description rdf:about="http://www.example/jose/foaf.rdf#jose"><ns0pred:type xmlns:ns0pred="http://ww
<rdf:Description rdf:about="http://www.example/jose/foaf.rdf#jose"><ns0pred:homepage xmlns:ns0pred="http:
<rdf:Description rdf:about="http://www.example/jose/foaf.rdf#juan"><ns0pred:type xmlns:ns0pred="http://ww
<rdf:Description rdf:about="http://www.example/jose/foaf.rdf#jose"><ns0pred:workplaceHomepage xmlns:ns0pr
<rdf:Description rdf:about="http://www.example/jose/foaf.rdf#jose"><ns0pred:nick xmlns:ns0pred="http://xm
<rdf:Description rdf:about="http://www.example/jose/foaf.rdf#jose"><ns0pred:depiction xmlns:ns0pred="http
<rdf:Description rdf:about="http://www.example/jose/foaf.rdf#jose"><ns0pred:name xmlns:ns0pred="http://xm
<rdf:Description rdf:about="http://www.example/jose/foaf.rdf#jose"><ns0pred:knows xmlns:ns0pred="http://x
</rdf:RDF>
Done
```

Example with extraction part of literal as variable

The following example shows how to extract a part of a literal as a variable for use in a numeric comparison using SPARQL

Suppose there are the following triples inserted:

```
SQL>SPARQL INSERT INTO GRAPH <http://mygraph.com> {  <:a>
                                                     <:p>
                                                     "123 abc" };
callret-0
VARCHAR
_______________________________________________________________________________

Insert into <http://mygraph.com>, 1 triples -- done

1 Rows. -- 30 msec.
SQL>SPARQL INSERT INTO GRAPH <http://mygraph.com> {  <:a>
                                                     <:p>
                                                     "234 abc" };
callret-0
VARCHAR
_______________________________________________________________________________

Insert into <http://mygraph.com>, 1 triples -- done

1 Rows. -- 0 msec.
```

In order to extract the numeric part, and then do a numeric (<.>,=), you can use atoi (), atol or atof in the filter:

```
SQL>SPARQL
SELECT *
FROM <http://mygraph.com>
WHERE
  {
    ?s ?p ?o . filter (bif:atoi (?o) > 130)
  };
s         p         o
VARCHAR   VARCHAR   VARCHAR
_____________________________________

:a        :p        234 abc

1 Rows. -- 10 msec.
```

Example how to define rule

See details here how to define rule context that is initialized from the contents of a given graph.

Implementation Notes

This service has been implemented using Virtuoso Server .

Virtuoso 'Semantic Bank' End Point

What is Piggy Bank?

Piggy Bank is an extension to the Firefox Web browser that turns it into a Semantic Web browser, letting you make use of existing information on the Web in more useful and flexible ways not offered by the original Web sites.

What is Semantic Bank?

Semantic Bank is the server companion of Piggy Bank that lets you persist, share and publish data collected by individuals, groups or communities. Here is a screen shot of one in action:

What can I do with this?

A Semantic Bank allows you to:

- Persist your information remotely on a server - This is useful, for example, if you want to share data between two of your computers or to avoid losing it due to mistakes or failure.
- Share information with other people - The ability to tag resources creates a powerful serendipitous categorization (as proven by things like del.icio.us or Flickr).
- Lets you publish your information - Both in the "pure" RDF form (for those who know how to make use of it) or to regular web pages, with the usual Longwell faceted browsing view of it

How can I help?

Semantic Bank is Open Source software and built around the spirit of open participation and collaboration.

There are several ways you can help:

- Install a Semantic Bank and let us know about it, so that we can update the list of available Semantic Banks.
- Subscribe to our mailing lists to show your interest and give us feedback
- Report problems and ask for new features through our issue tracking system.
- Send us patches or fixes to the code

Licensing and Legal Issues

Semantic Bank is open source software and is licensed under the BSD license.

Note , however, that this software ships with libraries that are not released under the same license; that we interpret their licensing terms to be compatible with ours and that we are redistributing them unmodified. For more information on the licensing terms of the libraries Semantic Bank depends on, please refer to the source code.

Download location:

"http://simile.mit.edu/dist/semantic-bank/

The Virtuoso Semantic Bank End Point

Before you can publish, you must register with one or more Semantic Banks:

- Invoke the menu command Tools > Piggy Bank > My Semantic Bank Accounts ...
- Click Add... in the Semantic Bank Accounts dialog box.
- In the popup dialog box, type in the URL to the Virtuoso Semantic Bank you want to register with. Example: http://server_name:server_port/bank
- Enter the account of a valid Virtuoso DAV user. (Note: currently we do not use encryption during authentication; do not use your precious password here.)
- Click OK, wait for the account to be registered, and then dismiss the Semantic Bank Accounts dialog box.
- To publish an item, just click the corresponding Publish button (much like how you save the item). To publish all the items being viewed, click the Publish All button.

What is the graph name used by Virtuoso for the triples from PiggyBank?

http://simile.org/piggybank/<piggybank-generated-name>

The piggybank-generated-name is a Virtuoso DAV user ID.

Making Linked Data Views Dereferenceable - Northwind Example

Consider an application that makes some relational data available for SPARQL requests, as described in the first part of the Northwind Linked Data View example . This may be sufficient for some clients but the IRIs of the described subjects are not dereferenceable. This means that external SPARQL processors cannot retrieve that data using the Virtuoso Sponger or the like. It also means that if some external resources refer to the IRI of some Northwind subject and a user browses that resource then he cannot look at the application's data by clicking on the subject link.

To make RDF access complete, applications can do the following:

1. Create a virtual directory
2. Instruct the server how to prepare RDF resources on demand
3. Configure rendering of RDF resources for non-RDF clients (including Web search engines)
4. Make the used ontology available
5. Provide an index or sitemap page to help users who try to browse published data but do not know the proper URLs

The following sequence of operations demonstrates how to implement the listed features without writing any special web pages. All requests (except the application-specific index/sitemap) will be handled by existing web service endpoints.

As a precaution, we erase any URL rewriting rule lists created by this example that may be in the database following a previous run of the script.

```
DB.DBA.URLREWRITE_DROP_RULELIST ('demo_nw_rule_list1', 1)
;
```

Do the same for individual rewrite rules:

```
DB.DBA.URLREWRITE_DROP_RULE ('demo_nw_rule1', 1)
;
DB.DBA.URLREWRITE_DROP_RULE ('demo_nw_rule2', 1)
;
DB.DBA.URLREWRITE_DROP_RULE ('demo_nw_rule3', 1)
;
DB.DBA.URLREWRITE_DROP_RULE ('demo_nw_rule4', 1)
;
```

As a sanity check we ensure that there are no other similarly named rules:

```
SQL>SELECT signal ('WEIRD', sprintf ('Rewrite rule "%s" found', URR_RULE))
FROM DB.DBA.URL_REWRITE_RULE WHERE URR_RULE like 'demo_nw%'
;
```

Next we create URI rewrite rules based on regular expressions by calling `DB.DBA.URLREWRITE_CREATE_REGEX_RULE` , so the same path will be redirected to different places depending on the MIME types the client can accept.

For a given input path, that is a URI identifying a particular Linked Data entity, the rewrite rule below generates an N3 or RDF/XML representation of the entity using a CONSTRUCT query. (Note: In the regular expression identifying the Accept: MIME types this rule applies to, i.e. in rdf.n3 and rdf.xml, each period (.) replaces a literal character because some SPARQL web clients published before the relevant W3C recommendations produce slightly incorrect "Accept:" strings.)

```
SQL>DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'demo_nw_rule2',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/sparql?query=CONSTRUCT+{+%3Chttp%3A//^{URIQADefaultHost}^%U%23this%3E+%3Fp+%3Fo+}+FROM+%3Cht
    vector('path', 'path', '*accept*'),
    null,
    '(text/rdf.n3)|(application/rdf.xml)',
```

```
    0,
    null
    );
```

Note

The request URL for the SPARQL web service looks terrible because it is URL-encoded; the sprintf format string for it is even worse! The easiest way of composing encoded strings of this sort is to use the Conductor UI for configuring the rewrite rules. Alternatively open the SPARQL endpoint page (assuming it supports a UI for entering queries, if no query string is specified), type in the desired CONSTRUCT or DESCRIBE statement into the web form (using some sample URI), execute it, cut the URL of the page with results from the address line of the browser window, paste it into the script and then replace the host name with *^{URIQADefaultHost}^* , every percent with double percent, the parts of the sample IRI to be substituted with *%U* ; finally adjust the vector of replacement parameters so that its length is equal to the number of *%U* or other format specifiers in the template.

The next rule redirects to the RDF browser service to display a description of the subject URI and let the user explore related subjects.

```
SQL>DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'demo_nw_rule1',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/rdfbrowser/index.html?uri=http%%3A//^{URIQADefaultHost}^%U%%23this',
    vector('path'),
    null,
    '(text/html)|(\\*/\\*)',
    0,
    303
    );
```

This next rule removes any trailing slash from the input path. Note that *\x24* is the hex character code for the end-of-line pattern *$* . It is written escaped because the dollar sign indicates the beginning of macro in ISQL.

```
SQL>DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'demo_nw_rule3',
    1,
    '(/[^#]*)/\x24',
    vector('path'),
    1,
    '%s',
    vector('path'),
    null,
    null,
    0,
    null
    );
```

To configure the server to furnish the ontology underpinning the example Northwind Linked Data View, the procedure LOAD_NW_ONTOLOGY_FROM_DAV, listed below, takes the ontology described in file /DAV/VAD/demo/sql/nw.owl and loads it into graph http://demo.openlinksw.com/schemas/NorthwindOntology/1.0/ in the local quad store. A rewrite rule is then created to query this graph when the input path identifies entities from this ontology.

```
SQL>create procedure DB.DBA.LOAD_NW_ONTOLOGY_FROM_DAV()
{
  declare content1, urihost varchar;
  SELECT cast (RES_CONTENT as varchar) INTO content1 from WS.WS.SYS_DAV_RES WHERE RES_FULL_PATH = '/DAV/V
  DB.DBA.RDF_LOAD_RDFXML (content1, 'http://demo.openlinksw.com/schemas/northwind#', 'http://demo.openlin
  urihost := cfg_item_value(virtuoso_ini_path(), 'URIQA','DefaultHost');
  if (urihost = 'demo.openlinksw.com')
  {
    DB.DBA.VHOST_REMOVE (lpath=>'/schemas/northwind');
    DB.DBA.VHOST_DEFINE (lpath=>'/schemas/northwind', ppath=>'/DAV/VAD/demo/sql/nw.owl', vsp_user=>'dba',
    DB.DBA.VHOST_REMOVE (lpath=>'/schemas/northwind#');
    DB.DBA.VHOST_DEFINE (lpath=>'/schemas/northwind#', ppath=>'/DAV/VAD/demo/sql/nw.owl', vsp_user=>'dba'
  }
};

DB.DBA.LOAD_NW_ONTOLOGY_FROM_DAV();
```

```
drop procedure DB.DBA.LOAD_NW_ONTOLOGY_FROM_DAV;

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'demo_nw_rule4',
    1,
    '/schemas/northwind#(.*)',
    vector('path'),
    1,
    '/sparql?query=DESCRIBE%20%3Chttp%3A//demo.openlinksw.com/schemas/northwind%23%U%3E%20FROM%20%3Chttp%
    vector('path'),
    null,
    '(text/rdf.n3)|(application/rdf.xml)',
    0,
    null
    );
```

Next we define virtual directory */Northwind* and associate with this a rulelist containing the URL rewriting rules defined above. Requests matching the rewriting rules should then be properly redirected to produce the requested data. Attempts to access the virtual directory root will execute the application's default VSP page, namely *sfront.vspx* .

```
SQL>DB.DBA.URLREWRITE_CREATE_RULELIST (
    'demo_nw_rule_list1',
    1,
    vector (
                'demo_nw_rule1',
                'demo_nw_rule2',
                'demo_nw_rule3',
                'demo_nw_rule4'
          ));

VHOST_REMOVE (lpath=>'/Northwind');
DB.DBA.VHOST_DEFINE (lpath=>'/Northwind', ppath=>'/DAV/home/demo/', vsp_user=>'dba', is_dav=>1, def_page=
          is_brws=>0, opts=>vector ('url_rewrite', 'demo_nw_rule_list1'));
```

Finally, to register the namespace prefix *northwind* as persistent we execute:

```
SQL>DB.DBA.XML_SET_NS_DECL ('northwind', 'http://demo.openlinksw.com/schemas/northwind#', 2);
```

Sponger Proxy URI Service

In certain cases, such as Ajax applications, it's prohibited to issue HTTP requests to a server other than the original server. In other cases it is necessary to transform the content of a target to an RDF format. To this end Virtuoso Server provides a Sponger Proxy URI Service. This service takes as an argument a target URL and may return the target's content "as is" or the Sponger may try to transform the content and return an RDF representation of the target. When transforming to RDF, the RDF format (RDF/XML, N3, TURTLE etc) of the output can be forced by a URL parameter or by content negotiation.

When the cartridges_dav.vad package is installed, Virtuoso reserves the path '/about/[id|html|data|rdf]/http/' for the RDF proxy service. In the current implementation, Virtuoso defines virtual directories for HTTP requests that come to the port specified as 'ServerPort' in the '[HTTPServer]' section of Virtuoso configuration file and refer to the above path string. So, if the Virtuoso installation on host example.com listens for HTTP requests on port 8080, client applications should use the 'service endpoint' string equal to 'http://example.com:8080/about/[id|html|data|rdf]/http/'.

If the cartridges_dav.vad VAD package is not installed, then the path '/proxy/rdf/' is used for the Sponger Proxy URI Service.

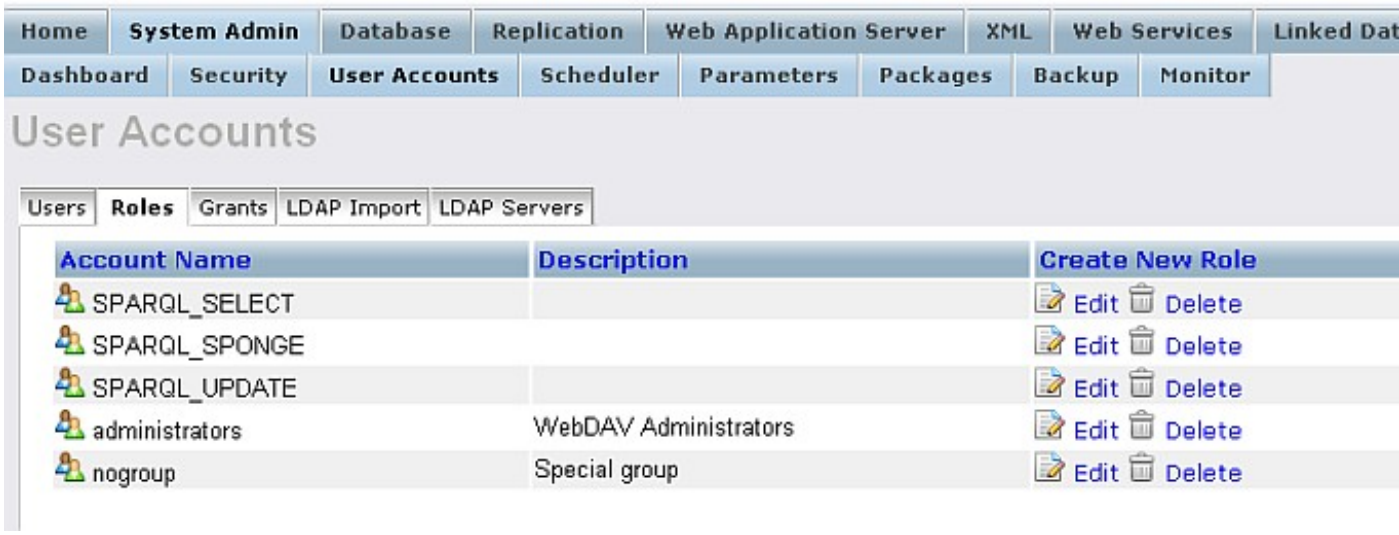
The old pattern for the Sponger Proxy URI Service, '/proxy/', is now deprecated.

Note: If you do not have the cartridges package installed, in order for the Sponger Proxy URI Service to work correctly, you must grant the SPARQL_UPDATE role to user SPARQL and grant execute permission on procedure RDF_SPONGE_UP.

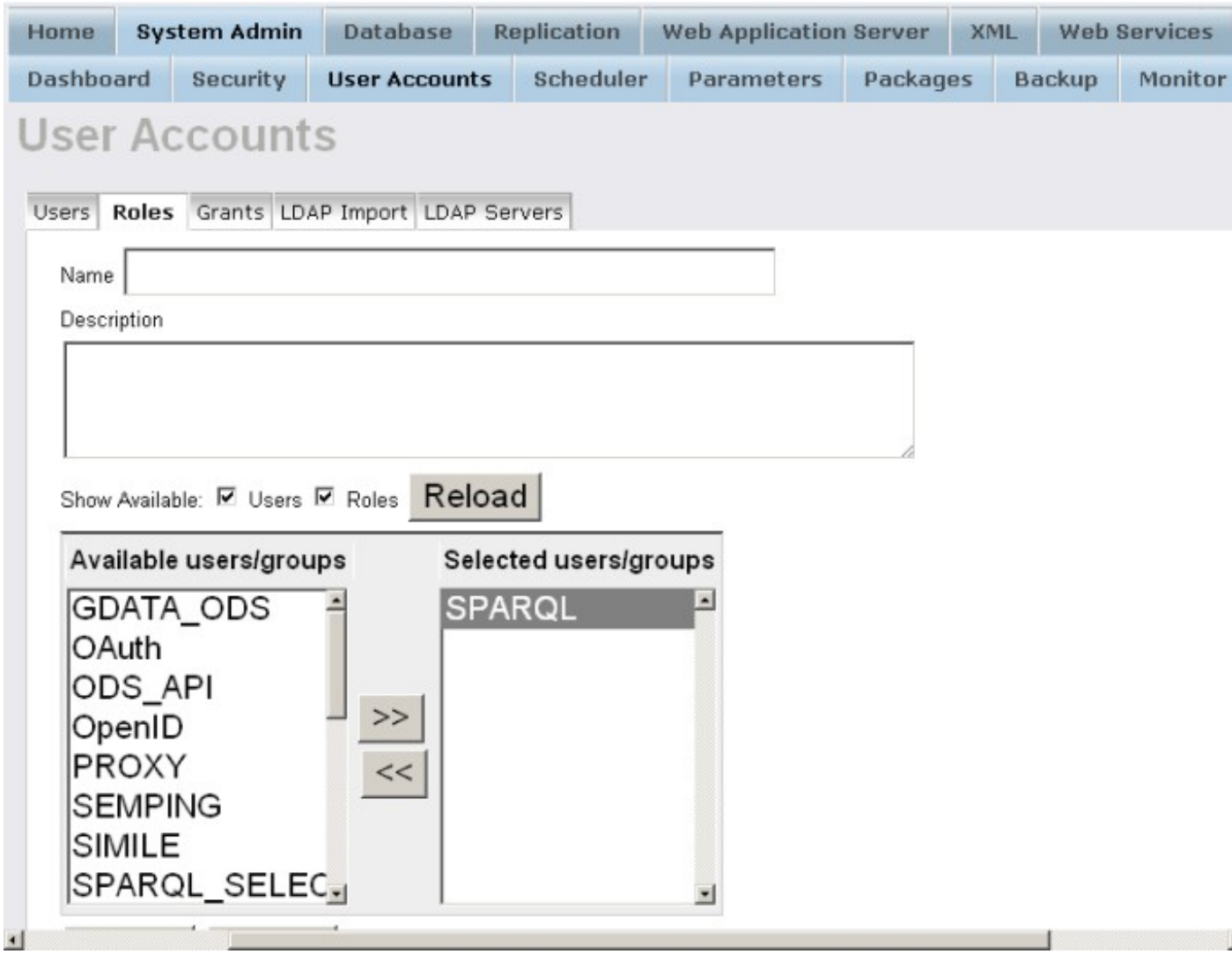
To enable SPARQL_UPDATE using the Conductor UI:

1. Go to the Virtuoso Administration Conductor i.e. http://host:port/conductor
2. Login as dba user
3. Go to System Admin->User Accounts->Roles

4. **Figure 16.36. Conductor UI**



5. Click the link "Edit" for "SPARQL_UPDATE
6. Select from the list of available user/groups "SPARQL" and click the ">>" button so to add it to the right-positioned list.
7. **Figure 16.37. Conductor UI**



8. Click the button "Update".

To grant execute permission on RDF_SPONGE_UP:

```
grant execute on DB.DBA.RDF_SPONGE_UP to "SPARQL";
```

When invoked with a URL of the form http://host:port/proxy?..., the Sponger Proxy URI Service accepts the following query string parameters:

- *force*

  - if 'rdf' is specified, the Sponger will try to extract RDF data from the target and return it
- *header*

  - HTTP headers to be sent to the target
- *output-format*

  - if 'force=rdf' is given, designates the desired output MIME type of the RDF data. The default is 'rdf+xml'. Other supported MIME types are 'n3', 'turtle' or 'ttl'.

When RDF data is requested and 'output-format' is not specified, the result will be serialized with a MIME type determined by the request 'Accept' headers i.e. the proxy service will do content negotiation.

Example: RDF file with URL: http://www.w3.org/People/Berners-Lee/card

```
-- Access the url in order to view the result in HTML format:
http://host:port/about/html/http/www.w3.org/People/Berners-Lee/card
-- Access the url in order to view the result in RDF:
http://host:port/about/rdf/http://www.w3.org/People/Berners-Lee/card
-- or use the following proxy invocation style:
http://host:port/proxy/rdf/http://www.w3.org/People/Berners-Lee/card
-- or this one:
http://host:port/proxy?url=http://www.w3.org/People/Berners-Lee/card&force=rdf
```

Note: It is not permitted, when using the style http://host:port/proxy/rdf, to pass URL query string parameters to the proxy.

Now go to the SPARQL endpoint, i.e. http://host:port/sparql

For the 'Default Graph URI' enter the URL of the RDF file: http://www.w3.org/People/Berners-Lee/card

For 'Query' enter:

```
SELECT *
WHERE
  {
    ?s ?p ?o
  }
```

Query result:

```
s                                                p                                                o
http://www.w3.org/People/Berners-Lee/card        http://www.w3.org/1999/02/22-rdf-syntax-ns#type  http:/
http://www.w3.org/People/Berners-Lee/card        http://purl.org/dc/elements/1.1/title            Tim Be
http://www.w3.org/People/Berners-Lee/card        http://creativecommons.org/ns#license            http:/
http://www.w3.org/People/Berners-Lee/card        http://xmlns.com/foaf/0.1/maker                  http:/
etc ...
```

SPARQL INI service

The [SPARQL] section of the virtuoso.ini configuration file sets parameters and limits for the SPARQL query web service. The values contained in the [SPARQL] section can be exposed in RDF form via the URL pattern http://cname/sparql?ini

Example: http://demo.openlinksw.com/sparql?ini

```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:rdfs="http://www.w3.org/2000/01/rd
<rdf:Description rdf:about="http://www.openlinksw.com/schemas/virtini#SPARQL"><ns0pred:MaxQueryCostEstima
<rdf:Description rdf:about="http://www.openlinksw.com/schemas/virtini#SPARQL"><ns0pred:ExternalXsltSource
<rdf:Description rdf:about="http://www.openlinksw.com/schemas/virtini#SPARQL"><ns0pred:DefaultQuery xmlns
```

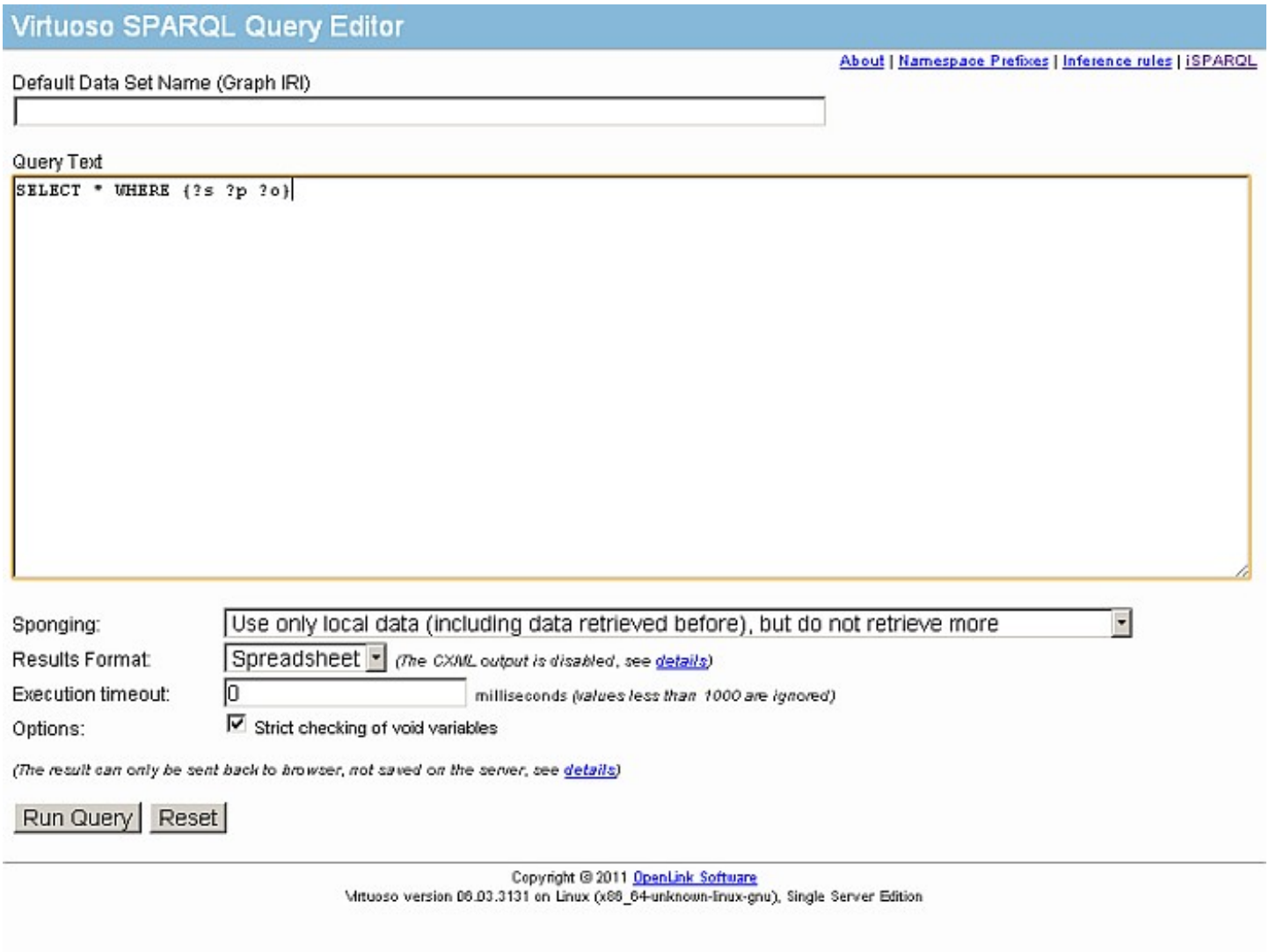
```
<rdf:Description rdf:about="http://www.openlinksw.com/schemas/virtini#SPARQL"><ns0pred:ResultSetMaxRows x
<rdf:Description rdf:about="http://www.openlinksw.com/schemas/virtini#SPARQL"><ns0pred:MaxQueryExecutionT
<rdf:Description rdf:about="http://www.openlinksw.com/schemas/virtini#SPARQL"><ns0pred:ExternalQuerySourc
<rdf:Description rdf:about="http://www.openlinksw.com/schemas/virtini#SPARQL"><ns0pred:DefaultGraph xmlns
<rdf:Description rdf:about="http://www.openlinksw.com/schemas/virtini#SPARQL"><ns0pred:PingService xmlns:
</rdf:RDF>
```

SPARQL Endpoint with Excel MIME Type Output Option

The SPARQL endpoint offers an Excel MIME type output option.

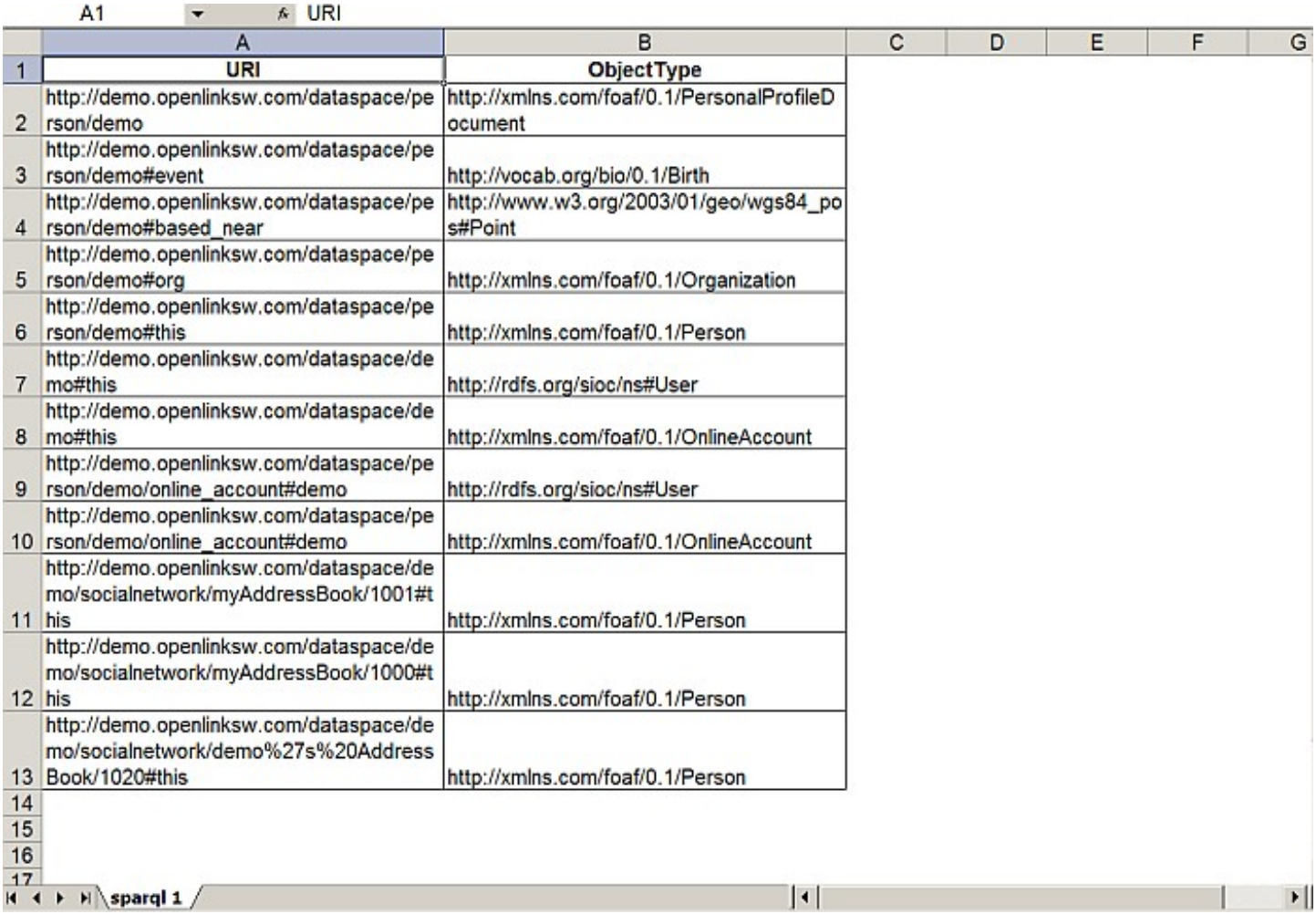
From http://cname:host/sparql, select "Spreadsheet" for the "Display Results As:" option and click the "Run Query" button.

Figure 16.38. SPARQL Endpoint with Excel MIME type output



The resulting query string contains a format parameter value of *"application/vnd.ms-excel"* . For example, A URL such as this one will be generated, and can be opened directly with Excel.

Figure 16.39. SPARQL Endpoint with Excel MIME type output



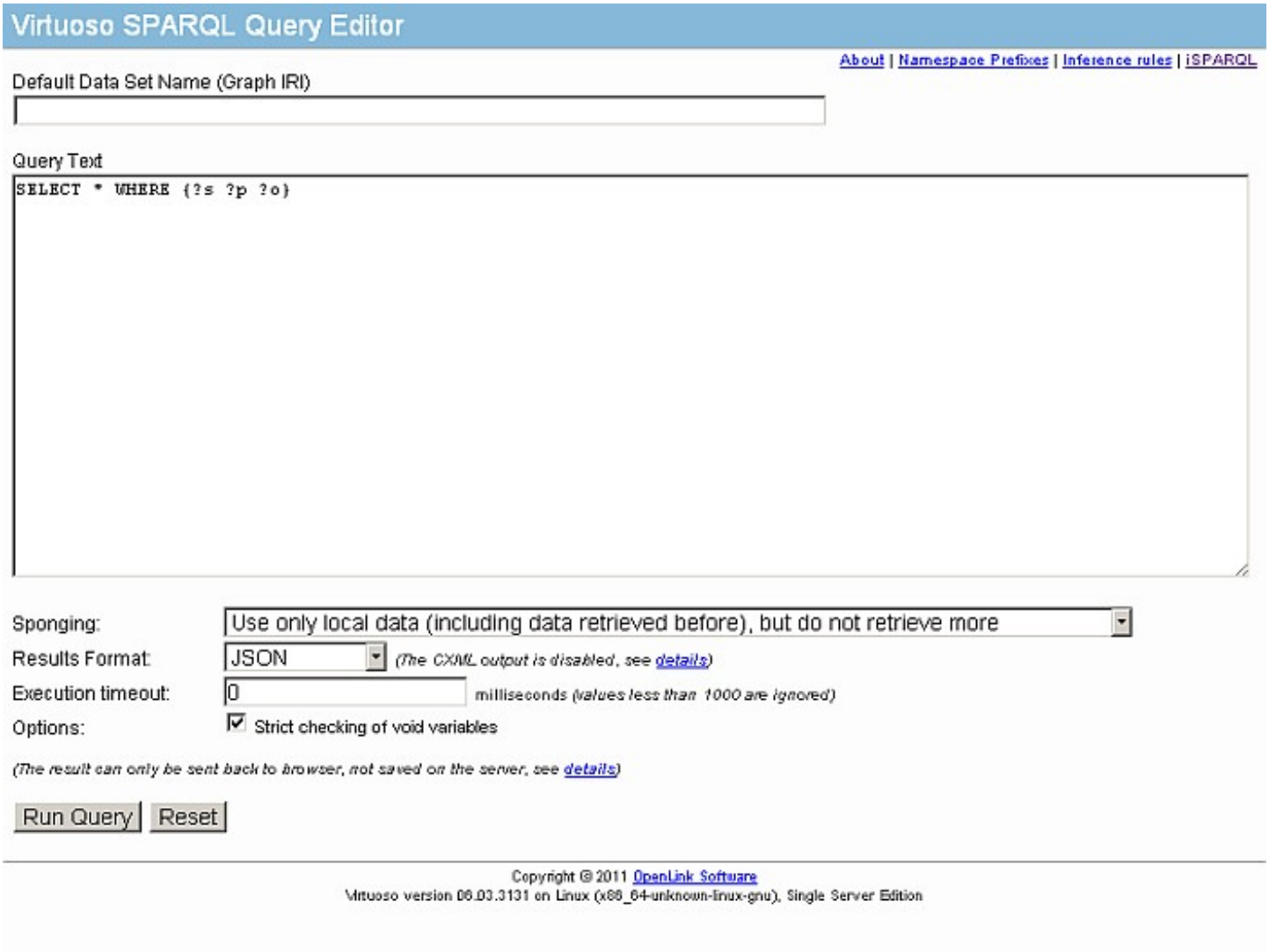
	A	B
1	URI	ObjectType
2	http://demo.openlinksw.com/dataspace/person/demo	http://xmlns.com/foaf/0.1/PersonalProfileDocument
3	http://demo.openlinksw.com/dataspace/person/demo#event	http://vocab.org/bio/0.1/Birth
4	http://demo.openlinksw.com/dataspace/person/demo#based_near	http://www.w3.org/2003/01/geo/wgs84_pos#Point
5	http://demo.openlinksw.com/dataspace/person/demo#org	http://xmlns.com/foaf/0.1/Organization
6	http://demo.openlinksw.com/dataspace/person/demo#this	http://xmlns.com/foaf/0.1/Person
7	http://demo.openlinksw.com/dataspace/demo#this	http://rdfs.org/sioc/ns#User
8	http://demo.openlinksw.com/dataspace/demo#this	http://xmlns.com/foaf/0.1/OnlineAccount
9	http://demo.openlinksw.com/dataspace/person/demo/online_account#demo	http://rdfs.org/sioc/ns#User
10	http://demo.openlinksw.com/dataspace/person/demo/online_account#demo	http://xmlns.com/foaf/0.1/OnlineAccount
11	http://demo.openlinksw.com/dataspace/demo/socialnetwork/myAddressBook/1001#this	http://xmlns.com/foaf/0.1/Person
12	http://demo.openlinksw.com/dataspace/demo/socialnetwork/myAddressBook/1000#this	http://xmlns.com/foaf/0.1/Person
13	http://demo.openlinksw.com/dataspace/demo/socialnetwork/demo%27s%20AddressBook/1020#this	http://xmlns.com/foaf/0.1/Person

SPARQL Endpoint with RDF+JSON Output: SPARQL UI Example

The SPARQL endpoint also offers a RDF+JSON output option.

From http://cname:host/sparql select "JSON" for "Display Results As:" and click the "Run Query" button.

Figure 16.40. SPARQL Endpoint with RDF+JSON output



As result URL containing as parameter the format *application/sparql-results+json* will be generated and the content should look like:

Figure 16.41. SPARQL Endpoint with JSON+RDF

```
{ "head": { "link": [], "vars": ["s", "p", "o"] },
  "results": { "distinct": false, "ordered": true, "bindings": [
    { "s": { "type": "uri", "value": "http://demo.openlinksw.com/dataspace/person/demo" }       , "p": { "type'
    { "s": { "type": "uri", "value": "http://demo.openlinksw.com/dataspace/person/demo" }       , "p": { "type'
    { "s": { "type": "uri", "value": "http://demo.openlinksw.com/dataspace/person/demo" }       , "p": { "type'
    { "s": { "type": "uri", "value": "http://demo.openlinksw.com/dataspace/person/demo" }       , "p": { "type'
    { "s": { "type": "uri", "value": "http://demo.openlinksw.com/dataspace/person/demo#event" } , "p": { "type'
    { "s": { "type": "uri", "value": "http://demo.openlinksw.com/dataspace/person/demo#event" } , "p": { "type'
    { "s": { "type": "uri", "value": "http://demo.openlinksw.com/dataspace/person/demo#based_near" }    , "p":
    { "s": { "type": "uri", "value": "http://demo.openlinksw.com/dataspace/person/demo#based_near" }    , "p":
    { "s": { "type": "uri", "value": "http://demo.openlinksw.com/dataspace/person/demo#based_near" }    , "p":
    { "s": { "type": "uri", "value": "http://demo.openlinksw.com/dataspace/person/demo#org" }   , "p": { "type'
```

SPARQL Endpoint with JSON/P Output Option: Curl Example

The SPARQL endpoint also offers a JSON/P output option.

The SPARQL endpoint accepts a 'callback' URL parameter and in this case when parameter 'format' is 'json', then it will produce JSON/P output.

```
$ curl "http://lod.openlinksw.com/sparql?query=select+*+where+\{+%3Fx+a+%3Fz+.+\}+limit+10&format=json&de
func(

{ "head": { "link": [], "vars": ["x", "z"] },
  "results": { "distinct": false, "ordered": true, "bindings": [
```

```
    { "x": { "type": "bnode", "value": "nodeID://b196899188" }  , "z": { "type": "uri", "value": "http://

    { "x": { "type": "uri", "value": "http://www.wasab.dk/morten/2005/04/sparqlette/#profile" } , "z": {
services/owl-s/1.1/Service.owl#ServiceProfile" }},
    { "x": { "type": "uri", "value": "http://www.wasab.dk/morten/2003/12/nearestAirport/#b-profile" }   ,
aml.org/services/owl-s/1.1/Service.owl#ServiceProfile" }},
    { "x": { "type": "uri", "value": "http://www.wasab.dk/morten/2003/12/nearestAirport/#a-profile" }   ,
aml.org/services/owl-s/1.1/Service.owl#ServiceProfile" }},
    { "x": { "type": "uri", "value": "http://www.wasab.dk/morten/2003/12/nearestAirport/index.rdf#a-profi
://www.daml.org/services/owl-s/1.1/Service.owl#ServiceProfile" }},
    { "x": { "type": "uri", "value": "http://www.wasab.dk/morten/2003/12/nearestAirport/index.rdf#b-profi
://www.daml.org/services/owl-s/1.1/Service.owl#ServiceProfile" }},
    { "x": { "type": "uri", "value": "http://www.wasab.dk/morten/2006/06/blogmatrix/#profile" } , "z": {
services/owl-s/1.1/Service.owl#ServiceProfile" }},
    { "x": { "type": "uri", "value": "http://www.wasab.dk/morten/2003/12/nearestAirport/#b-profile" }   ,
aml.org/services/owl-s/1.1/Service.owl#ServiceProfile" }},
    { "x": { "type": "uri", "value": "http://www.wasab.dk/morten/2003/12/nearestAirport/#a-profile" }   ,
aml.org/services/owl-s/1.1/Service.owl#ServiceProfile" }},
    { "x": { "type": "uri", "value": "http://www.wasab.dk/morten/2003/12/nearestAirport/#b-profile" }   ,
aml.org/services/owl-s/1.1/Service.owl#ServiceProfile" }} ] } })
```

16.2.4. Troubleshooting SPARQL Queries

A short SPARQL query can be compiled into a long SQL statement, especially if data comes from many quad map patterns. A moderately sized application with 50 tables and 10 columns per table may create thousands of quad map patterns for subjects spanning hundreds of different types. An attempt to "select everything" from Linked Data View of that complexity may easily create 5000 lines of SQL code. Thus it is to be expected that some queries will be rejected even if the same queries would work fine if the RDF data were held as physical quads in default storage, rather than synthesized through an Linked Data View.

In addition, the SQL compiler catches typos efficiently, signalling an error if a table or column name is unknown, efficiently catching typos. SPARQL uses IRIs that are long and sometimes unreadable, but there is no "closed world" schema of the data so a typo in an IRI is not an error; it is simply some other IRI. So a typo in an IRI or in a namespace prefix causes missing bindings of some triple patterns of the query and an incomplete result, but usually no errors are reported. A typo in graph or predicate IRI may cause the SPARQL compiler to generate code that accesses default (quad) storage instead of a relational source or generate empty code that accesses nothing.

The SQL compiler does not signal casting errors when it runs the statement generated from SPARQL, because the generated SQL code contains *option (QUIETCAST)* . This means that mismatches between expected and actual datatypes of values stay invisible and may cause rounding errors (e.g. integer division instead of floating-point) and even empty joins (due to join conditions that silently return NULL instead of returning a comparison error).

In other words, SPARQL queries are so laconic that there is no room for details that let the compiler distinguish between intent and a bug. This masks query complexity, misuse of names and type mismatches. One may make debugging easier by making queries longer.

Two very helpful debugging tools are automatic void variable recognition and plain old code inspection. "Automatic" means "cheap" so the very first step of debugging is to ensure that every triple pattern of the query may in principle return something. This helps in finding typos when the query gets data from Linked Data Views. It also helps when a query tries to join two disjoint sorts of subjects. If the *define sql:signal-void-variables 1* directive is placed in the preamble of the SPARQL query, the compiler will signal an error if it finds any triple pattern that cannot bind variables or any variable that is proved to be always unbound. This is especially useful when data are supposed to come from an *option (exclusive)* or *option (soft exclusive)* quad map. Without one of these options, the SPARQL compiler will usually bind variables using "physical quads"; the table of physical quads may contain any rows that match any given triple pattern; thus many errors will remain undiscovered. If the name of a quad map pattern is known then it is possible to force the SPARQL compiler to use only that quad map for the whole query or a part of the query. This is possible by using the following syntax:

```
QUAD MAP quad-map-name { group-pattern }
```

If some triple pattern inside *group-pattern* cannot be bound using *quad-map-name* or one of its descendants then *define sql:signal-void-variables 1* will force the compiler to signal the error.

Note

Although it is technically possible to use *QUAD MAP* to improve the performance of a query that tries to access redundant Linked Data Views, it is much better to achieve the same effect by providing a more restrictive query or by

Note

changing/extending the Linked Data View. If an application relies on this trick then interoperable third-party SPARQL clients may experience problems because they cannot use Virtuoso-specific extensions.

If the automated query checking gives nothing, function `sparql_to_sql_text` can be used in order to get the SQL text generated from the given query. Its only argument is the text of the SPARQL query to compile (without any leading SPARQL keyword or semicolon at the end). The returned value is the SQL text. The output may be long but it is the most authoritative source of diagnostic data.

When called from ISQL or an ODBC client, the return value of `sparql_to_sql_text` may be transferred as a BLOB so ISQL requires the "set blobs on" instruction to avoid data truncation. Even better, the SQL text can be saved to a file:

```
string_to_file ('debug.sql', sparql_to_sql_text ('SELECT * WHERE { graph ?g { ?s a ?type }}'), -2);
```

(The -2 is to overwrite the previous version of the file, as this function may be called many times).

Note

When passing the query text to sparql_to_sql_text, if the query contains single quotes, each embedded single quote must be doubled up. Use double quotes in SPARQL queries to avoid this inconvenience.

As an example, let's find out why the query

```
SQL>SPARQL
PREFIX northwind: <http://demo.openlinksw.com/schemas/northwind#>
SELECT DISTINCT ?emp
FROM <http://myhost.example.com/Northwind>
WHERE {
    ?order1 northwind:has_salesrep ?emp ; northwind:shipCountry ?country1 .
    ?order2 northwind:has_salesrep ?emp ; northwind:shipCountry ?country2 .
    filter (?country1 != ?country2) }
```

is much slower than a similar SQL statement. The call of `sparql_to_sql_text` returns the equivalent SQL statement:

```
SELECT DISTINCT sprintf_iri ( 'http://myhost.example.com/Northwind/Employee/%U%U%d#this' ,
    /*retval[*/ "s-6-1-t0"."b067b7d~FirstName~0" /* emp */ /*]retval*/ ,
    /*retval[*/  "s-6-1-t0"."b067b7d~FirstName~1" /*]retval*/ ,
    /*retval[*/  "s-6-1-t0"."b067b7d~FirstName~2" /*]retval*/ ) AS /*tmpl*/ "emp"
FROM (SELECT "s-6-1-t0-int~orders"."OrderID" AS /*tmpl*/ "20ffecc~OrderID",
         "s-6-1-t0-int~employees"."FirstName" AS /*as-name-N*/ "b067b7d~FirstName~0",
         "s-6-1-t0-int~employees"."LastName" AS /*as-name-N*/ "b067b7d~FirstName~1",
         "s-6-1-t0-int~employees"."EmployeeID" AS /*as-name-N*/ "b067b7d~FirstName~2"
         FROM Demo.demo.Employees AS "s-6-1-t0-int~employees", Demo.demo.Orders AS "s-6-1-t0-int~orders"
         WHERE /* inter-alias join cond */
       "s-6-1-t0-int~orders".EmployeeID = "s-6-1-t0-int~employees".EmployeeID) AS "s-6-1-t0",
    (SELECT "s-6-1-t1-int~orders"."OrderID" AS /*tmpl*/ "20ffecc~OrderID",
        "s-6-1-t1-int~orders"."ShipCountry" AS /*tmpl*/ "e45a7f~ShipCountry"
        FROM Demo.demo.Orders AS "s-6-1-t1-int~orders") AS "s-6-1-t1",
    (SELECT "s-6-1-t2-int~orders"."OrderID" AS /*tmpl*/ "20ffecc~OrderID",
        "s-6-1-t2-int~employees"."FirstName" AS /*as-name-N*/ "b067b7d~FirstName~0",
        "s-6-1-t2-int~employees"."LastName" AS /*as-name-N*/ "b067b7d~FirstName~1",
        "s-6-1-t2-int~employees"."EmployeeID" AS /*as-name-N*/ "b067b7d~FirstName~2"
        FROM Demo.demo.Employees AS "s-6-1-t2-int~employees", Demo.demo.Orders AS "s-6-1-t2-int~orders"
    WHERE /* inter-alias join cond */
       "s-6-1-t2-int~orders".EmployeeID = "s-6-1-t2-int~employees".EmployeeID) AS "s-6-1-t2",
    (SELECT "s-6-1-t3-int~orders"."OrderID" AS /*tmpl*/ "20ffecc~OrderID",
        "s-6-1-t3-int~orders"."ShipCountry" AS /*tmpl*/ "e45a7f~ShipCountry"
    FROM Demo.demo.Orders AS "s-6-1-t3-int~orders") AS "s-6-1-t3"
WHERE /* two fields belong to same equiv */
    /*retval[*/  "s-6-1-t0"."20ffecc~OrderID" /* order1 */ /*]retval*/  =
    /*retval[*/  "s-6-1-t1"."20ffecc~OrderID" /* order1 */ /*]retval*/
    AND /* two fields belong to same equiv */
    sprintf_iri ( 'http://myhost.example.com/Northwind/Employee/%U%U%d#this' ,
        /*retval[*/  "s-6-1-t0"."b067b7d~FirstName~0" /* emp */ /*]retval*/ ,
        /*retval[*/  "s-6-1-t0"."b067b7d~FirstName~1" /*]retval*/ ,
        /*retval[*/  "s-6-1-t0"."b067b7d~FirstName~2" /*]retval*/ ) =
    sprintf_iri ( 'http://myhost.example.com/Northwind/Employee/%U%U%d#this' ,
        /*retval[*/  "s-6-1-t2"."b067b7d~FirstName~0" /* emp */ /*]retval*/ ,
        /*retval[*/  "s-6-1-t2"."b067b7d~FirstName~1" /*]retval*/ ,
        /*retval[*/  "s-6-1-t2"."b067b7d~FirstName~2" /*]retval*/ )
    AND /* two fields belong to same equiv */
```

```
    /*retval[*/  "s-6-1-t2"."20ffecc~OrderID" /* order2 */ /*]retval*/  =
    /*retval[*/  "s-6-1-t3"."20ffecc~OrderID" /* order2 */ /*]retval*/
    AND /* filter */
   ( /*retval[*/  "s-6-1-t1"."e45a7f~ShipCountry" /* country1 */ /*]retval*/  <>
       /*retval[*/  "s-6-1-t3"."e45a7f~ShipCountry" /* country2 */ /*]retval*/ )
OPTION (QUIETCAST)
```

The query is next to unreadable but some comments split it into meaningful expressions. Every triple (or list of similar triples) becomes a subquery that returns fields needed to build the values of bound variables. The fields are printed wrapped by comments like */*retval[*/ expression /* original variable name */ /*]retval*/* . Names like*"s-6-1-t0"* contain the source line number where a group pattern begins (6) and the serial number of the triple (0). Comment */* inter-alias join cond */* means that the expression which follows is the condition as written in the declaration of the quad map pattern. Comment */* filter */* precedes expressions for FILTER expressions in the source SPARQL. The word "equiv" means "equivalence class", i.e. a group of occurrences of variables in the source query such that all occurrences are bound to the same value. E.g. when a name repeats in many triples of a group, all its occurrences form an equivalence class. In some cases the compiler can prove that two variables are always equal even if the names differ - these variables are also placed into an "equiv".

Looking at this query, you may notice equalities like *sprintf_iri (...) = sprintf_iri (...)* . That is sub-optimal because it indicates that no index will be used to optimize the join and that there will be one function call per row. When the variable *?emp* appears in two different triples, it means that the value of the variable is the same in both triples. The query compares IRIs instead of comparing the arguments of `sprintf_iri` because the format string is not proven to be a bijection. Indeed it cannot be a bijection for *arbitrary* strings, but the database must reflect the real world. If it is assumed that the real names of persons never start with a digit, within the *%d%U* format fragment, the digits will always be distinguishable from the name; so the IRI class can be declared as a bijection even if it is not true for arbitrary strings. The script can then include "suspicious" *option (bijection)* as follows:

```
create iri class sample:Employee "http://example.com/Employee/%d%U#this"
  (in employee_id integer not null, in employee_lastname varchar not null)
  option (bijection) .
```

Unfortunately, attempts to use the same trick with the declaration from the Northwind example will fail:

```
create iri class northwind:Employee "http://^{URIQADefaultHost}^/Northwind/Employee/%U%U%d#this"
  (in employee_firstname varchar not null, in employee_lastname varchar not null, in employee_id integer
  option (bijection) .
```

Bijection will allow the parsing, but it will never give the proper result, because the first *%U* will read the whole concatenation of *%U%U%d* , leaving nothing before the*#this* for the second *%U* (this is an error) and leaving nothing for the *%d* (that is an explicit parse error, becauses the integer field cannot be empty).

The string parser will process the string from left to right so it will be unable to parse the string. The compiler might sometimes report an error if it can prove that the format string is not appropriate for bijection.

The correct way of improving the Northwind example is to enable reliable bijection by adding strong delimiters:

```
create iri class northwind:Employee "http://^{URIQADefaultHost}^/Northwind/Employee/%U/%U/%d#this"
  (in employee_firstname varchar not null, in employee_lastname varchar not null, in employee_id integer
  option (bijection) .
```

After running the updated script, the query contains three comparisons of fields that were arguments of `sprintf_iri` in the previous version.

Example for casting string as IRI type

```
create function DB.DBA.RDF_DF_GRANTEE_ID_URI (in id integer)
{
  declare isrole integer;
  isrole := coalesce ((SELECT top 1 U_IS_ROLE FROM DB.DBA.SYS_USERS WHERE U_ID = id));
  if (isrole is null)
    return NULL;
  else if (isrole)
    return sprintf ('http://%s/sys/group?id=%d', registry_get ('URIQADefaultHost'), id);
  else
    return sprintf ('http://%s/sys/user?id=%d', registry_get ('URIQADefaultHost'), id);
}
;
```

```
grant execute on DB.DBA.RDF_DF_GRANTEE_ID_URI to SPARQL_SELECT
;

create function DB.DBA.RDF_DF_GRANTEE_ID_URI_INVERSE (in id_iri varchar)
{
  declare parts any;
  parts := sprintf_inverse (id_iri, sprintf ('http://%s/sys/user?id=%%d', registry_get ('URIQADefaultHost
  if (parts is not null)
    {
      if (exists (SELECT TOP 1 1 FROM DB.DBA.SYS_USERS WHERE U_ID = parts[0] and not U_IS_ROLE))
        return parts[0];
    }
  parts := sprintf_inverse (id_iri, sprintf ('http://%s/sys/group?id=%%d', registry_get ('URIQADefaultHos
  if (parts is not null)
    {
      if (exists (SELECT TOP 1 1 FROM DB.DBA.SYS_USERS WHERE U_ID = parts[0] and U_IS_ROLE))
        return parts[0];
    }
  return NULL;
}
;

grant execute on DB.DBA.RDF_DF_GRANTEE_ID_URI_INVERSE to SPARQL_SELECT
;

create iri class oplsioc:grantee_iri using
  function DB.DBA.RDF_DF_GRANTEE_ID_URI (in id integer) returns varchar ,
  function DB.DBA.RDF_DF_GRANTEE_ID_URI_INVERSE (in id_iri varchar) returns integer
  option ( bijection ,
    returns     "http://^{URIQADefaultHost}^/sys/group?id=%d"
    union       "http://^{URIQADefaultHost}^/sys/user?id=%d" ) .
```

16.2.5. SPARQL Inline in SQL

Virtuoso extends the SQL 92 syntax with SPARQL queries and subqueries. Instead of writing a SQL SELECT query or subquery, one can write the SPARQL keyword and a SPARQL query after the keyword.

```
SQL>SPARQL SELECT DISTINCT ?p WHERE { graph ?g { ?s ?p ?o } };
p
varchar
----------
http://example.org/ns#b
http://example.org/ns#d
http://xmlns.com/foaf/0.1/name
http://xmlns.com/foaf/0.1/mbox
...

SQL>SELECT distinct subseq ("p", strchr ("p", '#')) as fragment
  FROM (SPARQL SELECT DISTINCT ?p WHERE { graph ?g { ?s ?p ?o } } ) as all_predicates
  WHERE "p" like '%#%';
fragment
varchar
----------
#query
#data
#name
#comment
...
```

Note that names of variables returned from SPARQL are always case-sensitive and no case mode rules apply to them. Depending on the CaseMode parameter in the Virtuoso configuration file, double quotes should be used if necessary to refer to them in surrounding SQL code.

It is possible to pass parameters to a SPARQL query via a Virtuoso-specific syntax extension. *??* or *$?* indicates a positional parameter similar to *?* in plain SQL. *??* can be used in graph patterns or anywhere in place of a SPARQL variable. The value of a parameter should be passed in SQL form, i.e. this should be a number or a untyped string. An IRI ID can be passed in all cases where an absolute IRI can, except the obvious case of when the variable is an argument of a function that requires string. If the parameter is used in the 'graph', 'subject' or 'object' position of the SPARQL pattern, the string parameter is converted into an IRI automatically. In other cases an IRI string is indistinguishable from a string literal, so it is necessary to call the built-in SPARQL

function *iri()* , e.g. *iri (??)* . Using this notation, any dynamic SQL client (whether ODBC, JDBC or some other) can execute parameterized SPARQL queries, binding parameters just as with dynamic SQL.

```
SQL> create function param_passing_demo ()
{
  declare stat, msg varchar;
  declare mdata, rset any;
  exec ('SPARQL SELECT ?s WHERE { graph ?g { ?s ?? ?? }}',
    stat, msg,
    vector ( /* Vector of two parameters */
      'http://www.w3.org/2001/sw/DataAccess/tests/data/Sorting/sort-0#int1',
      4 ),
    10, /* Max no of rows */
    mdata, /* Variable to get metadata */
    rset ); /* Variable to get result-set */
  if (length (rset) = 0)
    signal ('23000',
      'No data found, try demo database with installed Virtuoso tutorials');
  return rset[0][0];
}

SQL> SELECT param_passing_demo ();
callret
VARCHAR
_______________________________________________________________________________

http://www.w3.org/2001/sw/DataAccess/tests/data/Sorting/sort-0#four

1 Rows. -- 00000 msec.
```

Another example:

```
INSERT INTO GRAPH <http://example.com/Northwind>
{ `iri($?)` <http://example.com/schemas/northwind#has_province> "Valencia" };
```

An inline SPARQL query can refer to SQL variables that are in scope in the SQL query or stored procedure containing it. Virtuoso extends the SPARQL syntax with a special notation to this effect. A reference to SQL variable X can be written as *?:X* or *$:X* . A reference to column *C* of a table or a sub-select with alias *T* can be written as *?:T.C* or *$:T.C* . Both notations can be used in any place where a variable name is allowed, except the 'AS' clause described below.

A column of a result set of a SPARQL SELECT can be used in SQL code inside a for statement just like any column from a SQL select.

SQL rules about double-quoted names are applicable to variables that are passed to a SPARQL query or selected from one. If a variable name contains unusual characters or should not be normalized according to SQL conventions then the name should use double quotes for escaping. e.g., the notation *?:"OrderLine"* will always refer to variable or column titled *OrderLine* whereas *?:OrderLine* can be converted to *ORDERLINE* or *orderline* .

It is safer to avoid using variable names that conflict with column names of RDF system tables, esp. *G* , *S* , *P* and *O* . These names are not reserved now but they may cause subtle bugs when an incorrect SPARQL subquery is compiled into SQL code that refers to identically named table columns. Some of these names may be rejected as syntax errors by future Virtuoso versions.

```
SQL> create procedure sql_vars_demo ()
{
#pragma prefix sort0: <http://www.w3.org/2001/sw/DataAccess/tests/data/Sorting/sort-0#>
  declare RES varchar;
  declare obj integer;
  result_names (RES);
  obj := 4;
  for (SPARQL SELECT ?subj WHERE { graph ?g { ?subj sort0:int1 ?:obj } } ) do
    result ("subj");
}

SQL> sql_vars_demo ();
RES
VARCHAR
_______________________________________________________________________________

http://www.w3.org/2001/sw/DataAccess/tests/data/Sorting/sort-0#four
```

```
1 Rows. -- 00000 msec.
```

The example also demonstrates the Virtuoso/PL pragma line for procedure-wide declarations of namespace prefixes. This makes the code more readable and eliminates duplicate declarations of namespace prefixes when the procedure contains many SPARQL fragments that refer to a common set of namespaces.

A SPARQL ASK query can be used as an argument of the SQL EXISTS predicate.

```
create function sparql_ask_demo () returns varchar
{
  if (exists (sparql ask where { graph ?g { ?s ?p 4}}))
    return 'YES';
  else
    return 'NO';
}

SQL> SELECT sparql_ask_demo ();
_____________________________________________________________________________

YES
```

Controlling SPARQL Output Data Types

The compilation of a SPARQL query may depend on an environment that is usually provided by the SPARQL protocol and which includes the default graph URI. Environment settings that come from the SPARQL protocol may override settings in the text of a SPARQL query. To let an application configure the environment for a query, SPARQL's syntax has been extended with the 'define' clause:

```
define parameter-qname parameter-value
```

Examples of supported parameters are *output:valmode* and *output:format*

output:valmode specifies which data types (i.e. representation) should be used for values in the result set. The default is "SQLVAL", meaning that a query returns result set values in SQL format and behaves as a typical SQL select - IRIs and string literals are returned as strings, making the output compatible with ODBC and the standard SQL routines. To compose triple vectors in Virtuoso PL code, an application may need data in long format. A valmode of "LONG" means that IRIs are returned as IRI_IDs and string literals may be returned as special "RDF boxes" even if they are actually plain strings. This may cause problems if these new datatypes are not known to the data recipient or if IRIs come from RDF Views (in which case IRI_IDs are created on the fly and 'pollute' the database), but it can result in fewer data conversions and thus better speed if used properly. "AUTO" disables all types of conversion for the result set, so the latter can comprise a mix of values across "SQLVAL" and "LONG" value modes, as well as some internal representations. It is better to avoid using this mode in user applications because the output may change from version to version.

If the query contains a

```
define output:valmode 'LONG'
```

clause then all returned values are in long format. e.g., the following query returns IRI_ID's instead of IRI strings.

```
SQL>SPARQL define output:valmode 'LONG' SELECT distinct ?p WHERE { graph ?g { ?s ?p ?o } };
p
----------
#i1000001
#i1000003
#i1000005
#i1000006
...
```

output:format instructs the SPARQL compiler that the result of the query should be serialized into an RDF document - that document will be returned as a single column of a single row result set. *output:format* is especially useful if a SPARQL CONSTRUCT or SPARQL DESCRIBE query is executed directly via an ODBC or JDBC database connection and the client cannot receive the resulting dictionary of triples (there's no way to transfer such an object via ODBC). Using this option, the client can receive the document that contains the whole result set of a SELECT or the dictionary of triples of a CONSTRUCT/DESCRIBE, and parse it locally.

Supported values for *output:format* are *RDF/XML* and *TURTLE* (or *TTL*). If both *output:valmode* and *output:format* are specified, *output:format* has higher priority, raising an error if *output:valmode* is set to a value other than *LONG* .

When a SPARQL query is compiled, the compiler checks whether the result set is to be sent to a remote ODBC/JDBC client or used in some other way. The compiler will automatically set *output:format* to *TURTLE* if compiling for execution by an SQL client.

The example below demonstrates how different values of *output:format* affect the result of SPARQL SELECT. Note 10 rows and 4 columns in the first result, and single LONG VARCHAR in the others. When using the ISQL client, use the 'set blobs on;' directive if fetching long texts to avoid receiving a 'data truncated' warning.

```
SQL> SPARQL SELECT * WHERE {graph ?g { ?s ?p ?o }} limit 10;
g                                             s                    p                              o
VARCHAR                                       VARCHAR              VARCHAR                        VARCHAR
_______________________________________________________________________________

http://local.virt/DAV/bound/manifest.rdf     nodeID://1000000000 http://example.com/test#query http://loc
. . .
http://local.virt/DAV/examples/manifest.rdf nodeID://1000000019 http://example.com/test#query http://loca

10 Rows. -- 00000 msec.

SQL> SPARQL define output:format "TTL" SELECT * WHERE {graph ?g { ?s ?p ?o }} limit 10;
callret-0
LONG VARCHAR
_______________________________________________________________________________

@prefix :rdf <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix :rs <http://www.w3.org/2005/sparql-results#> .
@prefix :xsd <http://www.w3.org/2001/XMLSchema#> .
[ rdf:type rs:results ;
  rs:result [
      rs:binding [ rs:name "g" ; rs:value <http://local.virt/DAV/bound/manifest.rdf> ] ;
      rs:binding [ rs:name "s" ; rs:value _:nodeID1000000000 ] ;
      rs:binding [ rs:name "p" ; rs:value <http://example.com/test#query> ] ;
      rs:binding [ rs:name "o" ; rs:value <http://local.virt/DAV/bound/bound1.rq> ] ;
      ] ;

. . .

  rs:result [
      rs:binding [ rs:name "g" ; rs:value <http://local.virt/DAV/examples/manifest.rdf> ] ;
      rs:binding [ rs:name "s" ; rs:value _:nodeID1000000019 ] ;
      rs:binding [ rs:name "p" ; rs:value <http://example.com/test#query> ] ;
      rs:binding [ rs:name "o" ; rs:value <http://local.virt/DAV/examples/ex11.2.3.1_1.rq> ] ;
      ] ;
    ] .

1 Rows. -- 00000 msec.

SQL> SPARQL define output:format "RDF/XML" SELECT * WHERE {graph ?g { ?s ?p ?o }} LIMIT 10;
callret-0
LONG VARCHAR
_______________________________________________________________________________

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rs="http://www.w3.org/2005/sparql-results#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#" >
   <rs:results rdf:nodeID="rset">
  <rs:result rdf:nodeID="sol206">
   <rs:binding rdf:nodeID="sol206-0" rs:name="g"><rs:value rdf:resource="http://local.virt/DAV/bound/mani
   <rs:binding rdf:nodeID="sol206-1" rs:name="s"><rs:value rdf:nodeID="1000000000"/></rs:binding>
   <rs:binding rdf:nodeID="sol206-2" rs:name="p"><rs:value rdf:resource="http://example.com/test#query"/>
   <rs:binding rdf:nodeID="sol206-3" rs:name="o"><rs:value rdf:resource="http://local.virt/DAV/bound/boun
  </rs:result>

. . .

  <rs:result rdf:nodeID="sol5737">
   <rs:binding rdf:nodeID="sol5737-0" rs:name="g"><rs:value rdf:resource="http://local.virt/DAV/examples/
```

```
    <rs:binding rdf:nodeID="sol5737-1" rs:name="s"><rs:value rdf:nodeID="1000000019"/></rs:binding>
    <rs:binding rdf:nodeID="sol5737-2" rs:name="p"><rs:value rdf:resource="http://example.com/test#query"/
    <rs:binding rdf:nodeID="sol5737-3" rs:name="o"><rs:value rdf:resource="http://local.virt/DAV/examples/
   </rs:result>
  </rs:results>
</rdf:RDF>

1 Rows. -- 00000 msec.
```

SPARQL CONSTRUCT and SPARQL DESCRIBE results are serialized as one would expect:

```
SQL> SPARQL
define output:format "TTL"
CONSTRUCT { ?s ?p "004" }
WHERE
  {
    graph ?g { ?s ?p 4 }
  };
callret-0
LONG VARCHAR
_______________________________________________________________________________

<http://www.w3.org/2001/sw/DataAccess/tests/data/Sorting/sort-0#four> <http://www.w3.org/2001/sw/DataAcce
_:b1000000913 <http://www.w3.org/2001/sw/DataAccess/tests/result-set#index> "004" .

1 Rows. -- 00000 msec.

SQL> SPARQL
define output:format "RDF/XML"
CONSTRUCT { ?s ?p "004" }
WHERE
  {
    graph ?g { ?s ?p 4 }
  };
callret-0
LONG VARCHAR
_______________________________________________________________________________

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<rdf:Description about="http://www.w3.org/2001/sw/DataAccess/tests/data/Sorting/sort-0#four"><ns0pred:int
<rdf:Description rdf:nodeID="b1000000913"><ns0pred:index xmlns:ns0pred="http://www.w3.org/2001/sw/DataAcc
</rdf:RDF>

1 Rows. -- 00000 msec.
```

SPARQL ASK returns a non-empty result set if a match is found for the graph pattern, an empty result set otherwise. If *output:format* is specified then the query makes a 'boolean result' document instead:

```
SQL> SPARQL ASK WHERE {graph ?g { ?s ?p 4 }};
__ask_retval
INTEGER
_______________________________________________________________________________

1

1 Rows. -- 00000 msec.

SQL> SPARQL ASK WHERE {graph ?g { ?s ?p "no such" }};
__ask_retval
INTEGER
_______________________________________________________________________________

0 Rows. -- 00000 msec.

SQL> SPARQL define output:format "TTL" ASK WHERE {graph ?g { ?s ?p 4 }};
callret
VARCHAR
_______________________________________________________________________________

@prefix :rdf <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
 @prefix :rs <http://www.w3.org/2005/sparql-results#> .
[ rdf:type rs:results ; rs:boolean TRUE ]
```

```
1 Rows. -- 00000 msec.

SQL> SPARQL define output:format "RDF/XML" ASK WHERE {graph ?g { ?s ?p 4 }};
callret
VARCHAR
_______________________________________________________________________________

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rs="http://www.w3.org/2005/sparql-results#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#" >
   <rs:results rdf:nodeID="rset">
    <rs:boolean rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">1</rs:boolean></results></rdf:RDF

1 Rows. -- 00000 msec.
```

16.2.6. API Functions

SPARQL can be used inline wherever SQL can be used. The only API functions that one needs to know are the ones for loading RDF data into the store. Dynamic SQL client applications can issue SPARQL queries against Virtuoso through the regular SQL client API, ODBC, JDBC or any other supported API, simply by prefixing the SPARQL query with the SPARQL keyword. Parameters work just as with dynamic SQL. Stored procedures can have SPARQL expressions inline and can declare cursors over SPARQL result sets.

Value conversions between SQL and SPARQL are most often automatic and invisible. In some cases one needs to be aware of the different SPARQL value representations (valmodes). SPARQL offers declarations for specifying whether returned graphs are to be serialized as XML or Turtle, or whether these will be hash tables of triples. See `dict_new()` and related functions for a description of the hash table SQL data type. The use of dict's is convenient for further programmatic processing of graphs.

RDF-related procedures use Virtuoso/PL vectors and dictionaries to represent RDF triples and sets of triples.

Valmode means the "format of values returned by an expression", i.e. 'short', 'long' or 'SQL value'.

Triple vector is a vector (array) of S, P and O, where all values are in 'long' formats, i.e. IRI_ID's for IRI values, numbers or datetimes for corresponding XMLSchema types, special "RDF box" objects if O is neither string nor IRI.

Dictionary of triples or *Hash table of triples* is an dictionary object made by the SQL function *dict_new ()* whose keys are triple vectors and values are not specified; this is a good storage format for an unordered set of distinct triples.

Dictionary of blank node names is a dictionary used for tricky processing of a number of TURTLE or RDF /XML descriptions of subgraphs that come from a common graph. Imagine a situation where different descriptions actually refer to the same blank nodes of the original graph and, moreover, the application that generates these descriptions always generates the same blank node id string for the same node. A reader of descriptions can correctly join described subgraphs into one big subgraph by filling in a dictionary that contains blank node id strings as keys and IRI_ID's assigned to those strings as dependent data. The sharing of the same node dictionary by all readers of an application will ensure that no blank node is duplicated.

Data Import

Using TTLP

DB.DBA.TTLP() parses TTL (TURTLE or N3 resource) and places its triples into DB.DBA.RDF_QUAD.

```
create procedure DB.DBA.TTLP (
    in strg any,       -- text of the resource
    in base varchar,   -- base IRI to resolve relative IRIs to absolute
    in graph varchar, -- target graph IRI, parsed triples will appear in that graph.
    in flags int)   -- bitmask of flags that permit some sorts of syntax errors in resource, use 0.
```

For loading a file of any great length, it is more practical to use the file_to_string_output function.

It is important the file be accessible to the Virtuoso server. You need to have set properly set the *DirsAllowed* parameter value in the section [Parameters] of the Virtuoso database INI file. For example on Windows it could be:

```
virtuoso.ini file:
```

```
[Parameters]
...
DirsAllowed =  .\tmp
...
```

So, in the example, the file you want to import from, should be in the tmp folder or in a subfolder. Note that this example folder is a subfolder of the Virtuoso Server working directory.

```
SQL> DB.DBA.TTLP (file_to_string_output ('.\tmp\data.ttl'), '', 'http://my_graph', 0);
```

Using TTLP_MT

The DB.DBA.TTLP_MT() procedure is like DB.DBA.TTLP() but loads the file on multiple threads, using parallel I/O and multiprocessing if available. The function does not leave a transaction log. Hence, after a successful load, one should execute the checkpoint statement to make sure that a server restart does not wipe out the results.

```
create procedure DB.DBA.TTLP_MT (
    in strg any,       -- text of the resource
    in base varchar,   -- base IRI to resolve relative IRIs to absolute
    in graph varchar,  -- target graph IRI, parsed triples will appear in that graph.
    in flags int) -- flags, use 0
```

Using RDF_LOAD_RDFXML_MT

For loading large resources when transactional integrity is not important (loading of a single resource may take more than one transaction) you can use also the *DB.DBA.RDF_LOAD_RDFXML_MT()* procedure:

```
create procedure DB.DBA.RDF_LOAD_RDFXML_MT (
    in strg varchar,  -- text of the resource
    in base varchar,  -- base IRI to resolve relative IRIs to absolute
    in graph varchar) -- target graph IRI, parsed triples will appear in that graph.
```

The following example demonstrates importing data from the RDF resource with URI: http://www.w3.org/People/Berners-Lee/card

```
SQL>create procedure MY_LOAD_FILE (in full_uri varchar, in in_resultset integer := 0)
{
  declare REPORT varchar;
  declare graph_uri, dattext varchar;
  declare app_env any;
  app_env := null;
  whenever sqlstate '*' goto err_rep;
  if (not in_resultset)
    result_names (REPORT);
  dattext := cast (XML_URI_GET_AND_CACHE (full_uri) as varchar);
  MY_SPARQL_REPORT (sprintf ('Downloading %s: %d bytes',
      full_uri, length (dattext) ) );
  graph_uri := full_uri;
  DELETE FROM RDF_QUAD WHERE G = DB.DBA.RDF_MAKE_IID_OF_QNAME (graph_uri);
  DB.DBA.RDF_LOAD_RDFXML_MT (dattext, full_uri, graph_uri);
  return graph_uri;
err_rep:
  result (sprintf ('%s: %s', __SQL_STATE, __SQL_MESSAGE));
  return graph_uri;
}
;

Done. -- 0 msec.

SQL>create procedure MY_SPARQL_REPORT(in strg varchar)
{
  if (__tag(strg) <> 182)
    strg := cast (strg as varchar) || sprintf (' -- not a string, tag=%d', __tag(strg));
  strg := replace (strg, 'SPARQL_DAV_DATA_URI()', '\044{SPARQL_DAV_DATA_URI()}');
  strg := replace (strg, 'SPARQL_DAV_DATA_PATH()', '\044{SPARQL_DAV_DATA_PATH()}');
  strg := replace (strg, 'SPARQL_FILE_DATA_ROOT()', '\044{SPARQL_FILE_DATA_ROOT()}');
  result (strg);
}
;
```

```
Done. -- 0 msec.

SQL> MY_LOAD_FILE('http://www.w3.org/People/Berners-Lee/card');
REPORT
VARCHAR
_______________________________________________________________________________

Downloading http://www.w3.org/People/Berners-Lee/card: 17773 bytes

1 Rows. -- 4046 msec.

SQL>SPARQL
SELECT *
FROM <http://www.w3.org/People/Berners-Lee/card>
WHERE {?s ?p ?o} ;

s                                          p                                          o
VARCHAR                                    VARCHAR                                    VARCHAR
_______________________________________________________________________________________________

http://bblfish.net/people/henry/card#me       http://xmlns.com/foaf/0.1/name                  Henry Story
http://www.w3.org/People/Berners-Lee/card#i   http://www.w3.org/1999/02/22-rdf-syntax-ns#type http://xmln
http://www.w3.org/People/Berners-Lee/card#i   http://www.w3.org/1999/02/22-rdf-syntax-ns#type http://www.
http://www.w3.org/People/Berners-Lee/card#i   http://xmlns.com/foaf/0.1/nick                 TimBL
http://www.w3.org/People/Berners-Lee/card#i   http://xmlns.com/foaf/0.1/nick                 timbl
http://www.w3.org/People/Berners-Lee/card#i   http://xmlns.com/foaf/0.1/mbox                 mailto:timb
http://www.w3.org/People/Berners-Lee/card#i   http://xmlns.com/foaf/0.1/mbox_sha1sum         965c47c5a70
http://www.w3.org/People/Berners-Lee/card#i   http://xmlns.com/foaf/0.1/knows                http://bblf
http://www.w3.org/People/Berners-Lee/card#i   http://xmlns.com/foaf/0.1/knows                http://home
http://www.w3.org/People/Berners-Lee/card#i   http://xmlns.com/foaf/0.1/knows                http://danb
http://www.w3.org/People/Berners-Lee/card#i   http://xmlns.com/foaf/0.1/knows                http://norm
http://www.w3.org/People/Berners-Lee/card#i   http://xmlns.com/foaf/0.1/knows                http://www.
http://www.w3.org/People/Berners-Lee/card#i   http://xmlns.com/foaf/0.1/knows                http://www.
http://www.w3.org/People/Berners-Lee/card#i   http://xmlns.com/foaf/0.1/knows                http://www.
http://www.w3.org/People/Berners-Lee/card#i   http://xmlns.com/foaf/0.1/knows                http://dig.
..........
```

Using RDF_TTL2HASH

The DB.DBA.RDF_TTL2HASH() does not load TTL content, instead it returns a dictionary of triples in 'long valmode'.

```
create function DB.DBA.RDF_TTL2HASH (
    in strg any,
    in base varchar,
    in graph varchar ) returns any
```

Parameter *flags* is useful when the syntax of the resource is TURTLE-like, but not correct TURTLE. By default, use zero value. Add 1 to let string literals contain end-of-line characters. Add 2 to suppress error messages on blank node verbs. Add 4 to allow variables instead of blank nodes. Add 8 to silently skip triples with literal subjects.

Using RDF_LOAD_RDFXML

The DB.DBA.RDF_LOAD_RDFXML() procedure parses RDF/XML and places its triples into DB.DBA.RDF_QUAD.

```
create procedure DB.DBA.RDF_LOAD_RDFXML (
    in strg any,           -- text of and XML document
    in base_iri varchar,   -- base IRI to resolve relative IRIs
    in graph_iri varchar ) -- the IRI of destination graph
```

See example .

Using RDF_QUAD_URI, RDF_QUAD_URI_L and RDF_QUAD_URI_L_TYPED

To insert a single quad into DB.DBA.RDF_QUAD() table, use one of these procedures:

```
-- Simple insertion of a quad where the object is a node
create procedure DB.DBA.RDF_QUAD_URI (
  in g_uri varchar, in s_uri varchar, in p_uri varchar,
  in o_uri varchar ) -- IRI string or IRI_ID
```

```
-- Simple insertion of a quad where the object is a literal value in 'SQL valmode'
create procedure DB.DBA.RDF_QUAD_URI_L (
  in g_uri varchar, in s_uri varchar, in p_uri varchar,
  in o_lit any ) -- string, number or datetime, NULL is not allowed

create procedure DB.DBA.RDF_QUAD_URI_L_TYPED (
  in g_uri varchar, in s_uri varchar, in p_uri varchar,
  in o_lit any,     -- string value of the literal
  in dt any,        -- datatype as IRI string or IRI_ID, can be NULL
  in lang varchar ) -- language as string or NULL
```

Arguments g_uri, s_uri and p_uri of these three functions should be IRI strings or IRI_IDs. All string arguments should be in UTF-8 encoding, otherwise they will be stored but are not queryable via SPARQL.

Data Export

These two procedures serialize a vector of triples into a session, in TURTLE or RDF/XML syntax. In their current versions, every triple is printed in a separate top-level record (say, in an rdf:Description tag), without any pretty-printing or nesting optimization.

```
create procedure DB.DBA.RDF_TRIPLES_TO_TTL (
    inout triples any, -- vector of triples in 'long valmode'.
    inout ses any )    -- an output stream in server default encoding

create procedure DB.DBA.RDF_TRIPLES_TO_RDF_XML_TEXT (
    inout triples any,          -- vector of triples in 'long valmode'.
    in print_top_level integer, -- zero if only rdf:Description tags should be written,
                                -- non-zero if the rdf:RDF top-level element should also be written
    inout ses any )             -- an output stream in server default encoding
```

Data query

```
-- Local execution of SPARQL via SPARQL protocol, produces a result set of SQL values.
create procedure DB.DBA.SPARQL_EVAL (
    in query varchar,      -- text of SPARQL query to execute
    in dflt_graph varchar, -- default graph IRI, if not NULL then this overrides what's specified in quer
    in maxrows integer )   -- limit on numbers of rows that should be returned.

-- Similar to SPARQL_EVAL, but returns a vector of vectors of SQL values.
create function DB.DBA.SPARQL_EVAL_TO_ARRAY (
    in query varchar,      -- text of SPARQL query to execute
    in dflt_graph varchar, -- default graph IRI, if not NULL then this overrides what's specified in quer
    in maxrows integer )   -- limit on numbers of rows that should be returned.
returns any

-- Remote execution of SPARQL via SPARQL protocol, produces a result set of SQL values.
create procedure DB.DBA.SPARQL_REXEC (
    in service varchar,    -- service URI to call via HTTP
    in query varchar,      -- text of SPARQL query to execute
    in dflt_graph varchar, -- default graph IRI, if not NULL then this overrides what's specified in quer
    in named_graphs any,   -- vector of named graph IRIs, if not NULL then this overrides what's specifie
    in req_hdr any,        -- additional HTTP header lines that should be passed to the service; 'Host: .
    in maxrows integer,    -- limit on numbers of rows that should be returned.
    in bnode_dict any )    -- dictionary of bnode ID references.

-- Similar to SPARQL_REXEC (), but returns a vector of vectors of SQL values.
-- All arguments are the same.
create function DB.DBA.SPARQL_REXEC_TO_ARRAY (
    in service varchar, in query varchar, in dflt_graph varchar, in named_graphs any,
    in req_hdr any, in maxrows integer, in bnode_dict any)
returns any

-- Similar to SPARQL_REXEC (), but fills in output parameters with metadata (like exec metadata) and a ve
s of 'long valmode' values.
-- First seven arguments are the same.
create procedure DB.DBA.SPARQL_REXEC_WITH_META (
    in service varchar, in query varchar, in dflt_graph varchar, in named_graphs any,
    in req_hdr any, in maxrows integer, in bnode_dict any,
    out metadata any,  -- metadata like exec () returns.
    out resultset any) -- results as 'long valmode' value.
```

If the query is a CONSTRUCT or DESCRIBE then the result set consists of a single row and column, the value inside is a dictionary of triples in 'long valmode'.

16.2.7. Useful Internal Functions

Conversion Functions for XMLSchema/RDF Data Serialization Syntax

These functions emulate constructor functions from XQuery Core Function Library.

```
create function DB.DBA."http://www.w3.org/2001/XMLSchema#boolean" (in strg any) returns integer
create function DB.DBA."http://www.w3.org/2001/XMLSchema#dateTime" (in strg any) returns datetime
create function DB.DBA."http://www.w3.org/2001/XMLSchema#double" (in strg varchar) returns double precisi
create function DB.DBA."http://www.w3.org/2001/XMLSchema#float" (in strg varchar) returns float
create function DB.DBA."http://www.w3.org/2001/XMLSchema#integer" (in strg varchar) returns integer
```

RDF-specific Predicates

```
-- Returns 1 if string s matches pattern p, 0 otherwise
create function DB.DBA.RDF_REGEX (
    in s varchar,            -- source string to check
    in p varchar,            -- regular expression pattern string
    in coll varchar := null) -- unused for now (modes are not yet implemented)

-- Returns 1 if language identifier r matches lang pattern t
create function DB.DBA.RDF_LANGMATCHES (
  in r varchar, -- language identifies (string or NULL)
  in t varchar) -- language pattern (exact name, first two letters or '*')
```

16.2.8. Default and Named Graphs

Sometimes the default graph IRI is not known when the SPARQL query is composed. It can be added at the very last moment by providing the IRI in a 'define' clause as follows:

```
define input:default-graph-uri &lt;http://example.com&gt
```

Such a definition overrides the default graph URI set in query by the 'FROM ...' clause (if any).

The query may contain more than one *define input:default-graph-uri* . The set of values of *input:default-graph-uri* has the highest possible priority and cannot be redefined in the rest of the text of the query by FROM clauses.

FROM NAMED clauses can be used multiple times in one query:

```
SPARQL
  SELECT ?id
  FROM NAMED <http://example.com/user1.ttl>
  OPTION (get:soft "soft", get:method "GET")
  FROM NAMED <http://example.com/user2.ttl>
  OPTION (get:soft "soft", get:method "GET")
  WHERE { GRAPH ?g { ?id a ?o } }
```

Similarly, *define input:named-graph-uri <http://example.com>* is a replacement for a FROM NAMED clause

When Virtuoso receives a SPARQL request via HTTP, the value of the default graph can be set in the protocol using a *default-graph-uri* HTTP parameter. Multiple occurrences of this parameter are allowed. This HTTP parameter is converted into *define input:default-graph-uri* . There's similar support for *named-graph-uri* HTTP parameter. For debugging purposes, graph names set in the protocol are sent back in the reply header as *X-SPARQL-default-graph: ...* and *X-SPARQL-named-graph: ...* header lines, one line per graph.

A web service endpoint may provide different default configurations for different host names mentioned in HTTP requests. This facility is configured via table *DB.DBA.SYS_SPARQL_HOST* .

```
create table DB.DBA.SYS_SPARQL_HOST (
  SH_HOST       varchar not null primary key, -- host mask
  SH_GRAPH_URI varchar,                 -- default graph uri
  SH_USER_URI   varchar,                  -- reserved for any use in applications
```

```
  SH_BASE_URI varchar,                  -- for future use (not used currently) to set BASE in sparql quer
  SH_DEFINES long varchar,              -- additional defines for requests
  PRIMARY KEY (SH_HOST)
)
```

When the SPARQL web service endpoint receives a request it checks the *Host* HTTP header line. This line contains zero or more target host names, delimited by commas. For every host name in the line, the service scans the *DB.DBA.SYS_SPARQL_HOST* table in search of a row containing a matching host name in *SH_HOST* . The *SH_HOST* field acts as 'pattern' argument for the SQL string operator LIKE. If a matching row is found, the text of SPARQL request is extended.

If a default graph is not explicitly set by the HTTP parameters and *SH_GRAPH_URI* is not null then the default graph is set to *SH_GRAPH_URI* .

If *SH_DEFINES* is not null then it is added in front of the query; so this field is a good place for the text for any DEFINE options. See various DEFINE examples usage.

SH_USER_URI is for arbitrary user data and can be used in any way by the application that is "responsible" for the declared host.

The search of *DB.DBA.SYS_SPARQL_HOST* stops at the first found row, other possible matches are silently ignored.

Example Usage:

```
INSERT INTO DB.DBA.SYS_SPARQL_HOST (SH_HOST, SH_GRAPH_URI, SH_USER_URI, SH_BASE_URI, SH_DEFINES) VALUES
('example.com', 'urn:example:com', 'urn:example:user', NULL, 'define input:inference "http://mygraph.com"
```

16.2.9. Calling SQL from SPARQL

A SPARQL expression can contain calls to Virtuoso/PL functions and built-in SQL functions in both the WHERE clause and in the result set. Two namespace prefixes, *bif* and *sql* are reserved for these purposes. When a function name starts with the *bif:* namespace prefix, the rest of the name is treated as the name of a SQL BIF (Built-In Function). When a function name starts with the *sql:* namespace prefix, the rest of the name is treated as the name of a Virtuoso/PL function owned by DBA with database qualifier DB, e.g. *sql:example(...)* is converted into *DB.DBA."example"(...)* .

In both cases, the function receives arguments in SQL format ('SQL valmode') and also returns the result in SQL format. The SPARQL compiler will automatically add code for format conversion into the resulting SQL code so SQL functions can be used even if *define output:valmode 'LONG'* forces the use of RDF representation in the result set.

Example with sql: namespace prefix

```
SQL>create procedure DB.DBA.ComposeInfo (
  in pname varchar,
  in pnick varchar := '',
  in pbox  varchar := '')
{
   declare ss varchar;
   ss := concat(pname, ' ', pnick, ' ', pbox);
   ss := rtrim (ss, ' ');
   return ss;

};
Done. -- 0 msec.

SQL>SPARQL
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT (sql:ComposeInfo (?name, ?nick, ?box))
FROM <http://www.w3.org/People/Berners-Lee/card>
WHERE
  {
    ?s rdf:type foaf:Person .
    optional{?s foaf:name ?name }.
    optional{?s foaf:nick ?nick }.
    optional{?s foaf:box ?box }.
    filter (?nick like '%TimBL%') .
  };
callret-0
VARCHAR
```

```
Timothy Berners-Lee TimBL

1 Rows. -- 30 msec.
```

See Also:

- Example "Things around highly populated places"
- Virtuoso Faceted Web Service Examples
- Virtuoso Faceted Usage Statistics Examples

Example with sql: namespace prefix and bif:contains

```
SQL>SPARQL
SELECT DISTINCT ?cityUri ?cityName (sql:BEST_LANGMATCH (?cityName, 'en, en-gb;q=0.8, fr;q=0.7, *;q=0.1',
WHERE
  {
    ?cityUri ?predicate ?value.
    ?cityUri a <http://dbpedia.org/ontology/City>.
    ?value bif:contains "London".
    OPTIONAL
      {
        ?cityUri rdfs:label ?cityName
      }
  };

cityUri                                                cityName                      bestCityName
ANY                                                    ANY                                 ANY
_____________________________________________________________________________________________________
http://dbpedia.org/resource/Anerley                        Anerley                           Anerley
http://dbpedia.org/resource/Felixstowe                 Felixstowe                    Felixstowe
http://dbpedia.org/resource/Chesham                        Chesham                           Chesham
http://dbpedia.org/resource/Stratford%2C_London        Stratford, London             Stratford, London
http://dbpedia.org/resource/Ashford%2C_Surrey            Ashford (Surrey)  A             shford (Surrey)
http://dbpedia.org/resource/Newmarket%2C_Suffolk             Newmarket (Suffolk)           Newmarket (Suffo
http://dbpedia.org/resource/North_Rhine-Westphalia         Renania d'o Norte-Westfalia     Renania d'o Nort
http://dbpedia.org/resource/West_Bromwich                    West Bromwich                       West Bromw
....
```

Example with bif: namespace prefix

```
SQL>SPARQL
SELECT *
FROM <http://www.w3.org/people#>
WHERE { ?s ?p ?o . ?o bif:contains '"Timo*"'};
s                                                  p                                  o
VARCHAR                                            VARCHAR                            VARCHAR
_____________________________________________________________________________________________

 http://www.w3.org/People/Berners-Lee/card#i    http://xmlns.com/foaf/0.1/name        Timothy Berners-Lee
 http://www.w3.org/People/Berners-Lee/card#i    http://xmlns.com/foaf/0.1/givenname   Timothy

2 Rows. -- 2 msec.
```

See Also:

- Example filtering RDF objects triples by a given predicate
- Example with extraction part of literal as variable
- Example for Usage of Expressions inside CONSTRUCT, INSERT and DELETE {...} Templates
- Example for various expressions usage
- Example for generating RDF information resource URI

16.2.10. SPARQL DESCRIBE

The SPARQL specification does not define the precise output of DESCRIBE, so different applications may need different results for the same subject. Some applications need quick generation of short and incomplete results whereas others may need detailed reports composed from multiple sources.

The supported option values for *sql:describe-mode* are:

- *(default)*

  -- for subject-predicate-object triples of given IRI as subject plus subject-predicate-object triples of given IRI as object. No "sql:describe-mode" or define sql:describe-mode "" will set default.
- *SPO*

  -- for subject-predicate-object triples of given IRI as subject;
- *CBD*

  -- for concise bound description of given subject (i.e., SPO + CBD of each blank node object found by SPO, recursively);
- *OBJCBD*

  -- like CBD but traverses from objects to subjects, not from subjects to objects;
- In addition, user may write his/her own "postporocessing" function that will get the result of describe, lists of "good" and "bad" graphs, name of storage, options and alter it in any way he/she wishes. The notation is e.g. define sql:describe-mode "SPO+XYZ", meaning SPO describe and call DB.DBA.SPARQL_DESC_POSTPROC_XYZ on top of it. See below example usage.

If define *sql:describe-mode "xxx"* is specified then the generated SQL code will use the procedures named:

```
DB.DBA.SPARQL_DESC_DICT_xxx (in subj_dict any, in consts any, in graphs
any, in storage_name any, in options any)
```

and

```
DB.DBA.SPARQL_DESC_DICT_xxx_PHYSICAL (in subj_dict any, in consts any,
in graphs any, in storage_name any, in options any)
```

In a new blank database, only two such pairs of procedures are created. Procedures *DB.DBA.SPARQL_DESC_DICT_SPO* and *DB.DBA.SPARQL_DESC_DICT_SPO_PHYSICAL* are for *sql:describe-mode "SPO"* . This pair of procedures searches for all triples where the input IRIs are used as subjects; they are faster than the default routine which searches for all triples where the input IRIs are used as subjects or objects. Similarly, *DB.DBA.SPARQL_DESC_DICT_CBD* and *DB.DBA.SPARQL_DESC_DICT_CBD_PHYSICAL* are for *sql:describe-mode "CBD"* . CBD stands for Concise Bounded Description of given subject (i.e., SPO + CBD of each blank node object found by SPO, recursively).

In each pair, both procedures have the same semantics but the second one is used if and only if the SPARQL compiler can prove that all subjects to process are from physical storage *(DB.DBA.RDF_QUAD)* . Thus the second procedure will not search for subjects in Linked Data Views.

Each procedure should return a dictionary with triples as keys and integer 1 as values. So the dictionary is filled by calls like:

```
dict_put (resulting_dict,
          vector (subj_iri_id, pred_iri_id, obj_iri_id_or_rdf_box),
          1);
```

Procedure arguments are as follows:

- *subj_dict*

  - a dictionary whose keys are IRI IDs and maybe values of other types, esp. RDF boxes. Keys are subjects to be described, so values other than IRI IDs should usually be ignored. Values should be ignored.
- *consts*

- a vector of IRI IDs and values of other types. The items contained in the vector are subjects to be described, as with the keys of subj_dict.

- *graphs*

  - a vector of IRI IDs of graphs that can be used for DESCRIBE. The vector may contain garbage, like in the two previous cases. A NULL can be passed instead of a vector indicating that the source graphs are not specified in the source query.

- *storage_name*

  - the value of "define input:storage" from the original SPARQL query, NULL if missing.

- *options*

  - reserved for future use and can be ignored.

One should grant execute permission on both procedures to SPARQL_SELECT before referring to them in SPARQL.

SPARQL DESCRIBE Examples

Assume the following statements are executed:

```
__rdf_set_bnode_t_treshold();
SET blobs ON;
SET echo ON;

SPARQL PREFIX xmp: <http://example.com/xmp/>
CLEAR GRAPH xmp:good1;

SPARQL PREFIX xmp: <http://example.com/xmp/>
CLEAR GRAPH xmp:good2;

SPARQL PREFIX xmp: <http://example.com/xmp/>
CLEAR GRAPH xmp:bad1;

SPARQL PREFIX xmp: <http://example.com/xmp/>
CLEAR GRAPH xmp:bad2;

SPARQL PREFIX xmp: <http://example.com/xmp/>
INSERT IN xmp:good1
  {
    xmp:Top1 xmp:item xmp:TheSubject .
    xmp:TheSubject xmp:details
            xmp:ChildObject ,
            ( xmp:car xmp:cadr xmp:caddr ) .
  };

SPARQL PREFIX xmp: <http://example.com/xmp/>
INSERT IN xmp:good2
  {
    xmp:Top2 xmp:items [ rdf:_1 xmp:TheSubject ; rdf:_2 xmp:OtherSubject ] .
  };
```

Examples SPARQL DESCRIBE -- No Option

```
SPARQL
DEFINE output:format "NICE_TTL"
PREFIX xmp: <http://example.com/xmp/>
DESCRIBE xmp:TheSubject FROM xmp:good1 FROM xmp:good2;

fmtaggret-NICE_TTL
LONG VARCHAR
 @prefix ns0: <http://example.com/xmp/> .
 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
 ns0:Top1 ns0:item ns0:TheSubject .
 ns0:TheSubject ns0:details ns0:ChildObject , [ ] .
 _:vb78520012 rdf:_1 ns0:TheSubject .
```

Examples SPARQL DESCRIBE Option "SPO"

Example 1

```
SPARQL
DEFINE output:format "NICE_TTL"
DEFINE sql:describe-mode "SPO"
PREFIX xmp: <http://example.com/xmp/>
DESCRIBE xmp:TheSubject FROM xmp:good1 FROM xmp:good2;

fmtaggret-NICE_TTL
LONG VARCHAR
 @prefix ns0: <http://example.com/xmp/> .
 ns0:TheSubject ns0:details ns0:ChildObject , [ ] .
```

Example 2

```
SQL>set blobs on;
SQL>SPARQL
define sql:describe-mode "SPO"
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX sioct: <http://rdfs.org/sioc/types#>

DESCRIBE ?forum
FROM <http://demo.openlinksw.com/dataspace>
WHERE {
  ?forum rdf:type sioct:Weblog .
}
LIMIT 1;

callret-0
LONG VARCHAR
_______________________________________________________________________________

<http://demo.openlinksw.com/dataspace/bloguser/weblog/bloguser_blog> <http://www.w3.org/1999/02/22-rdf-sy
                <http://atomowl.org/ontologies/atomrdf#Feed> ;
        <http://rdfs.org/sioc/ns#description> "XML templates demo's Weblog" ;
        <http://rdfs.org/sioc/ns#has_space> <http://demo.openlinksw.com/dataspace/bloguser/space#this> ;
        <http://rdfs.org/sioc/ns#container_of> <http://demo.openlinksw.com/dataspace/bloguser/weblog/blog
                <http://demo.openlinksw.com/dataspace/bloguser/weblog/bloguser_blog/21> ;
        <http://rdfs.org/sioc/ns#id> "bloguser_blog" ;
        <http://xmlns.com/foaf/0.1/maker> <http://demo.openlinksw.com/dataspace/person/bloguser#this> ;
        <http://rdfs.org/sioc/ns#link> <http://demo.openlinksw.com/dataspace/bloguser/weblog/bloguser_blo
        <http://atomowl.org/ontologies/atomrdf#entry> <http://demo.openlinksw.com/dataspace/bloguser/webl
                <http://demo.openlinksw.com/dataspace/bloguser/weblog/bloguser_blog/21> ;
        <http://atomowl.org/ontologies/atomrdf#contains> <http://demo.openlinksw.com/dataspace/bloguser/w
                <http://demo.openlinksw.com/dataspace/bloguser/weblog/bloguser_blog/20> ;
        <http://atomowl.org/ontologies/atomrdf#title> "bloguser_blog" ;
        <http://www.w3.org/2000/01/rdf-schema#label> "XML templates demo's Weblog" ;
        <http://rdfs.org/sioc/ns#scope_of> <http://demo.openlinksw.com/dataspace/bloguser/weblog/bloguser
        <http://rdfs.org/sioc/ns#has_owner> <http://demo.openlinksw.com/dataspace/bloguser#this> ;
        <http://www.w3.org/2000/01/rdf-schema#isDefinedBy> <http://demo.openlinksw.com/dataspace/bloguser
        <http://purl.org/dc/elements/1.1/identifier> "62"^^<http://www.w3.org/2001/XMLSchema#integer> ;
        <http://rdfs.org/sioc/services#has_service> <http://demo.openlinksw.com/RPC2> ,
                <http://demo.openlinksw.com/mt-tb> ,
                <http://demo.openlinksw.com/Atom/bloguser-blog-0> ,
                <http://demo.openlinksw.com/GData/bloguser-blog-0> .
<http://demo.openlinksw.com/RPC2> <http://rdfs.org/sioc/services#service_of> <http://demo.openlinksw.com/
<http://demo.openlinksw.com/mt-tb> <http://rdfs.org/sioc/services#service_of> <http://demo.openlinksw.com
<http://demo.openlinksw.com/dataspace/bloguser/weblog/bloguser_blog#owner> <http://rdfs.org/sioc/ns#has_s
<http://demo.openlinksw.com/dataspace/bloguser/weblog/bloguser_blog/20> <http://rdfs.org/sioc/ns#has_cont
        <http://atomowl.org/ontologies/atomrdf#source> <http://demo.openlinksw.com/dataspace/bloguser/web
<http://demo.openlinksw.com/dataspace/bloguser/weblog/bloguser_blog/21> <http://rdfs.org/sioc/ns#has_cont
        <http://atomowl.org/ontologies/atomrdf#source> <http://demo.openlinksw.com/dataspace/bloguser/web
<http://demo.openlinksw.com/dataspace/bloguser#this> <http://rdfs.org/sioc/ns#owner_of> <http://demo.open
<http://demo.openlinksw.com/dataspace/bloguser/space#this> <http://rdfs.org/sioc/ns#space_of> <http://dem
<http://demo.openlinksw.com/dataspace/person/bloguser#this> <http://xmlns.com/foaf/0.1/made> <http://demo
<http://demo.openlinksw.com/Atom/bloguser-blog-0> <http://rdfs.org/sioc/services#service_of> <http://demo
<http://demo.openlinksw.com/GData/bloguser-blog-0> <http://rdfs.org/sioc/services#service_of> <http://dem

1 Rows. -- 240 msec.
```

Examples SPARQL DESCRIBE -- Option "CBD"

Example 1

```
SPARQL
DEFINE output:format "NICE_TTL"
DEFINE sql:describe-mode "CBD"
PREFIX xmp: <http://example.com/xmp/>
DESCRIBE xmp:TheSubject FROM xmp:good1 FROM xmp:good2;

Query result:
fmtaggret-NICE_TTL
LONG VARCHAR
 @prefix ns0: <http://example.com/xmp/> .
 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
 ns0:TheSubject ns0:details ns0:ChildObject , ( ns0:car ns0:cadr ns0:caddr ) .
```

Example 2

```
SPARQL
DEFINE output:format "NICE_TTL"
DEFINE sql:describe-mode "CBD"
PREFIX xmp: <http://example.com/xmp/>
DESCRIBE xmp:TheSubject  xmp:good1 from xmp:good2;
```

Example 2

```
SQL>SPARQL
DEFINE sql:describe-mode "CBD"
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
DESCRIBE ?friend
WHERE
  {
    ?s foaf:knows ?friend  .
    ?friend foaf:nick ?nick.
    filter (?s=<http://www.advogato.org/person/rmorgan/foaf.rdf#me>)
}
;

@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ns1:    <http://www.advogato.org/person/chrisd/foaf.rdf#> .
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .
ns1:me  rdf:type        foaf:Person .
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
ns1:me  rdfs:seeAlso    <http://www.advogato.org/person/chrisd/foaf.rdf> ;
        foaf:name       "Chris DiBona" ;
        foaf:nick       "chrisd" ;
        foaf:homepage   <http://www.dibona.com> ;
        foaf:mbox_sha1sum       "e8231d19ac0d11ccbdc565485054461e5d71f0d3" .
@prefix ns4:    <http://www.advogato.org/person/schoen/foaf.rdf#> .
ns1:me  foaf:knows      ns4:me .
@prefix ns5:    <http://www.advogato.org/person/jpick/foaf.rdf#> .
ns1:me  foaf:knows      ns5:me .
@prefix ns6:    <http://www.advogato.org/person/benson/foaf.rdf#> .
ns1:me  foaf:knows      ns6:me .
@prefix ns7:    <http://www.advogato.org/person/conrad/foaf.rdf#> .
ns1:me  foaf:knows      ns7:me .
@prefix ns8:    <http://www.advogato.org/person/starshine/foaf.rdf#> .
ns1:me  foaf:knows      ns8:me .
@prefix ns9:    <http://www.advogato.org/person/chip/foaf.rdf#> .
ns1:me  foaf:knows      ns9:me .
@prefix ns10:   <http://www.advogato.org/person/crackmonkey/foaf.rdf#> .
.....
```

Example SPARQL DESCRIBE -- Option "OBJCBD"

Example 1

```
SPARQL
DEFINE output:format "NICE_TTL"
DEFINE sql:describe-mode "OBJCBD"
```

```
PREFIX xmp: <http://example.com/xmp/>
DESCRIBE xmp:TheSubject FROM xmp:good1 FROM xmp:good2;

Query result:
fmtaggret-NICE_TTL
LONG VARCHAR
 @prefix ns0: <http://example.com/xmp/> .
 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
 ns0:Top1 ns0:item ns0:TheSubject .
 ns0:Top2 ns0:items [ rdf:_1 ns0:TheSubject ] .
```

Example SPARQL DESCRIBE -- Custom "Post Porocessing" function

```
create function DB.DBA.SPARQL_DESC_POSTPROC_SOURCES (inout triples any array, in good_graphs any array, i
{
  declare triple, val, so_dict, so_lst, addon_dict, dlst any;
  dbg_obj_princ ('DB.DBA.SPARQL_DESC_POSTPROC_SOURCES (', triples, good_graphs, bad_graphs, storage, opts
  so_dict := dict_new (dict_size (triples));
  dict_iter_rewind (triples);
  while (dict_iter_next (triples, triple, val))
    {
      declare v any;
      v := triple[0];
      dict_put (so_dict, iri_to_id_nosignal (v), 1);
      v := triple[2];
      if (isiri_id (v) or (isstring (v) and box_tag (v) = 1))
        dict_put (so_dict, iri_to_id_nosignal (v), 1);
    }
  so_lst := dict_list_keys (so_dict, 1);
  foreach (any s_itm in so_lst) do
    {
      for (sparql select distinct ?g where { graph ?g {{ `iri(?:s_itm)` ?p ?o } union { ?s ?p  `iri(?:s_i
        dict_put (triples, vector (s_itm, iri_to_id('Source'), "g"), 1);
    }
  return triples;
}
;

SPARQL
DEFINE output:format "NICE_TTL"
DEFINE sql:describe-mode "CBD+SOURCES"
PREFIX xmp: <http://example.com/xmp/>
DESCRIBE xmp:TheSubject FROM xmp:good1 FROM xmp:good2;

SELECT sparql_to_sql_text ('
DEFINE output:format "NICE_TTL"
DEFINE sql:describe-mode "CBD+SOURCES"
PREFIX xmp: <http://example.com/xmp/>
DESCRIBE xmp:TheSubject FROM xmp:good1 FROM xmp:good2') as x long varchar;
```

16.2.11. Transitivity in SPARQL

Virtuoso SPARQL allows access to Virtuoso's SQL transitivity extension. Read the SQL section for a definition of the options.

The SPARQL syntax is slightly different from the SQL, although the option names and meanings are the same.

In SPARQL, the transitive options occur after a subquery enclosed in braces:

The below produces all the IRI's that are the same as <http://dbpedia.org/resource/New_York>.

```
SPARQL
SELECT ?syn
WHERE
  {
    {
      SELECT ?x ?syn
      WHERE
        {
          { ?x owl:sameAs ?syn }
          UNION
          { ?syn owl:sameAs ?x }
```

```
        }
      }
      OPTION ( TRANSITIVE, t_in (?x), t_out (?syn), t_distinct, t_min (0) )
      FILTER (?x = <http://dbpedia.org/resource/New_York>) .
    }
```

In this case, we provide a binding for ?x in the filter outside of the transitive subquery. The subquery therefore is made to run from in to out. The same effect would be accomplished if we bound ?syn and SELECT ?x, the designations of in and out are arbitrary and for transitive steps that can be evaluated equally well in both directions this makes no difference.

The transitive subquery in the above is

```
{SELECT ?syn
 WHERE
  {
    { SELECT ?x ?syn
      WHERE
       {
         { ?x owl:sameAs ?syn }
         UNION
         { ?syn owl:sameAs ?x}
       }
    } OPTION (TRANSITIVE, t_in (?x), t_out (?syn), t_distinct, t_min (0) )
  }
} .
```

Leaving out the option would just look for one step of owl:sameAs. Making it transitive will apply the subquery to all bindings it produces until all are visited at least once (the t_distinct modifier).

If the transitive step consists of a single triple pattern, there is a shorthand:

```
<alice> foaf:knows ?friend option (transitive t_min (1))
```

will bind ?friend to all directly and indirectly found foaf:known individuals. If t_min had been 0, Malice> would have also been in the generated bindings.

The syntax is

```
option (transitive transitivity_option[,...])

transitivity_option ::=  t_in (<variable_list>)
| t_out (<variable_list>)
| t_distinct
| t_shortest_only
| t_no_cycles
| t_cycles_only
| t_min (INTNUM)
| t_max (INTNUM)
| t_end_flag (<variable>)
| t_step (<variiable_or_step>)
| t_direction INTNUM

variable_list ::= <variable> [,...]

variable_or_step ::= <variable> | path_id' | 'step_no'
```

Unlike SQL, variable names are used instead of column numbers. Otherwise all the options have the same meaning.

Some examples of the use of transitivity are:

Collection of Transitivity Option Demo Queries for SPARQL

Example for finding out what graphs contain owl:sameAs for "New York"

To find out what graphs contain owl:sameAs for Dan York, we do

```
SELECT ?g ?x count (*) as ?count
WHERE {
        {
          SELECT ?x ?alias ?g
          WHERE {
                  {
                    GRAPH ?g {?x owl:sameAs ?alias }
                  }
          UNION
                  {
                   GRAPH ?g {?alias owl:sameAs ?x}
                  }
                }
        }
        OPTION ( TRANSITIVE,
                 t_in (?x),
                 t_out (?alias),
                 t_distinct,
                 t_min (1)) .
        FILTER (?x = <http://dbpedia.org/resource/New_York> ) .
      }
```

Here we select all paths that start with the initial URI and pass through one or more sameAs statements. Each step produces a result of the transitive subquery. The graph where the sameAs triple was found is returned and used as the grouping column. In this way we see how many times each graph is used. Note that graphs are counted many times since the graphs containing immediate sameAs statements are counted for paths of length 1, then again as steps on paths that reach to their aliases and so on.

Example for query that takes all the people known by Tim Berners-Lee, to a depth between 1 and 4 applications of the subquery

This query takes all the people known by kidehen, to a depth between 1 and 4 applications of the subquery. It then sorts them by the distance and the descending count of connections of each found connection. This is equivalent to the default connections list shown by LinkedIn.

```
SPARQL
SELECT ?o ?dist ((SELECT COUNT (*) WHERE {?o foaf:knows ?xx}))
WHERE
  {
    {
      SELECT ?s ?o
      WHERE
        {
          ?s foaf:knows ?o
        }
    } OPTION ( TRANSITIVE,
               t_distinct,
               t_in(?s),
               t_out(?o),
               t_min (1),
               t_max (4),
               t_step ('step_no') as ?dist ) .
    FILTER (?s= <http://www.w3.org/People/Berners-Lee/card#i>)
  }
ORDER BY ?dist DESC 3
LIMIT 50
```

Example for query that takes all the people known by Tim Berners-Lee, to a depth between 2 and 4 applications of the subquery

This query takes all the people known by kidehen, to a depth between 2 and 4 applications of the subquery. It then sorts them by the distance and the descending count of connections of each found connection. This is equivalent to the default connections list shown by LinkedIn.

```
SPARQL
SELECT ?o ?dist ((SELECT COUNT (*) WHERE {?o foaf:knows ?xx}))
```

```
WHERE
  {
    {
      SELECT ?s ?o
      WHERE
        {
          ?s foaf:knows ?o
        }
    } OPTION ( TRANSITIVE,
               t_distinct,
               t_in(?s),
               t_out(?o),
               t_min (2),
               t_max (4),
               t_step ('step_no') as ?dist) .
    FILTER (?s= <http://www.w3.org/People/Berners-Lee/card#i>)
  }
ORDER BY ?dist DESC 3
LIMIT 50
```

Example for finding how two people know each other and what graphs are involved in the connection

To find how two people know each other and what graphs are involved in the connection, we do:

```
SPARQL
SELECT ?link ?g ?step ?path
WHERE
  {
    {
      SELECT ?s ?o ?g
      WHERE
        {
          graph ?g {?s foaf:knows ?o }
        }
    } OPTION ( TRANSITIVE,
               t_distinct,
               t_in(?s),
               t_out(?o),
               t_no_cycles,
               T_shortest_only,
               t_step (?s) as ?link,
               t_step ('path_id') as ?path,
               t_step ('step_no') as ?step,
               t_direction 3) .
    FILTER (?s= <http://www.w3.org/People/Berners-Lee/card#i>
    && ?o = <http://www.advogato.org/person/mparaz/foaf.rdf#me>)
  }
  LIMIT 20
```

This query binds both the t_in and t_out variables. The ?g is left as a free variable. Also, specifying ?s and the system defined constants step_no and path_id as with t_step, we get for each transitive step a row of results with the intermediate binding of ?s, the count of steps from the initial ?s and a distinct identifier for the individual path, since there can be many distinct paths that link the ?s and ?o specified in the filter.

See the SQL transitive option section for details on the meaning of step_no and path_id.

Example for TBox Subsumption

Subsumption Demo Using Transitivity Clause

Yago Class Hierarchy (TBox) Subsumption

AlphaReceptors

```
# all subjects with IRI: <http://dbpedia.org/class/yago/AlphaReceptor105609111>,
# that are sub-classes of anything (hence ?y)
# without restrictions on tree levels
SELECT ?y
```

```
FROM <http://dbpedia.org/resource/classes/yago#>
WHERE
  {
    {
      SELECT *
      WHERE
        {
          ?x rdfs:subClassOf ?y .
        }
    }
    OPTION (TRANSITIVE, t_distinct, t_in (?x), t_out (?y) ) .
    FILTER (?x = <http://dbpedia.org/class/yago/AlphaReceptor105609111>)
  }
```

Example for Receptors

```
SELECT ?x
FROM <http://dbpedia.org/resource/classes/yago#>
WHERE
  {
    {
      SELECT *
      WHERE
        {
          ?x rdfs:subClassOf ?y .
        }
    } OPTION (transitive, t_distinct, t_in (?x), t_out (?y) ) .
  FILTER (?y = <http://dbpedia.org/class/yago/Receptor105608868>)
}
```

Inference Rule example using transitive properties from SKOS vocabulary

The following example demostrates the steps how to retrieve the skos ontology, add triples for skos:broaderTransitiveinto the graph, define inference rule, and at the and execute sparql query with inference rule and transitivity option. The queries were executed against the LOD instance (http://lod.openlinksw.com):

1. Make the Context graph, assuming you don't want to load entire SKOS vocabulary into our Quad Store:

```
SQL>SPARQL
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
INSERT INTO GRAPH <urn:rules.skos> { skos:broader rdfs:subPropertyOf skos:broaderTransitive .
                                     skos:narrower rdfs:subPropertyOf skos:narrowerTransitive };
```

2. OR Load entire SKOS ontology into Quad Store via iSQL interface (commandline or HTML based Conductor):

```
SQL>DB.DBA.RDF_LOAD_RDFXML (http_get ('http://www.w3.org/2009/08/skos-reference/skos-owl1-dl.rdf')
Done.
```

3. Make Context Rule:

```
SQL>rdfs_rule_set ('skos-trans', 'urn:rules.skos');
Done.
```

4. Go to SPARQL endpoint, for ex. http://lod.openlinksw.com/sparql
5. Use inference rule pragma to set context rule for SPARQL query, i.e:

```
SPARQL
DEFINE input:inference "skos-trans"
PREFIX p: <http://dbpedia.org/property/>
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX category: <http://dbpedia.org/resource/Category:>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX geo: <http://www.georss.org/georss/>

SELECT DISTINCT ?m ?n ?p ?d
WHERE
  {
    ?m rdfs:label ?n.
    ?m skos:subject ?c.
    ?c skos:broaderTransitive category:Churches_in_Paris OPTION (TRANSITIVE) .
    ?m p:abstract ?d.
```

```
    ?m geo:point ?p
    FILTER ( lang(?n) = "fr" )
    FILTER ( lang(?d) = "fr" )
  }
```

6. You will get 22 rows returned from the query. Note that for comparison, if the option (transitive) is ommitted, then only 2 rows will be returned in our example query:

Figure 16.42. Transitive option

m	n	p	d
http://dbpedia.org/resource/Basilique_Saint-Denis	Basilique Saint-Denis	48.9355555556 2.35972222222	La basilique Saint-Denis est une église de style gothique située à Saint-Denis, en Seine-Saint-Denis (93). Elle a le statut de cathédrale depuis 1966, mais elle est aussi une abbatiale.
http://dbpedia.org/resource/Saint-Julien-le-Pauvre	Église Saint-Julien-le-Pauvre	48.8520833333 2.34708333333	L'église Saint-Julien-le-Pauvre est l'église grecque-melkite catholique de Paris, située rive gauche d le quartier latin, proche de la Seine. Batie en 975 par ordre d'Henri Ier. Eglise Saint-Julien-le-Pauv Julien-le-Pauvre Eglise
http://dbpedia.org/resource/%C3%89glise_de_la_Ste.-Trinit%C3%A9%2C_Paris	Église de la Sainte-Trinité de Paris	48.8772222222 2.33138888889	L'église de la Sainte-Trinité, dans le 9e arrondissement de Paris, est située place d'Estienne-d'Orve au départ de la rue Blanche, dans le prolongement de la rue Saint-Lazare et de la rue de la Chauss d'Antin (elle était conçue pour être vue depuis l'Opéra).
http://dbpedia.org/resource/Saint-Nicolas-du-Chardonnet	Église Saint-Nicolas-du-Chardonnet	48.8491666667 2.35027777778	L'église Saint-Nicolas-du-Chardonnet est située à Paris, 23, rue des Bernardins, dans le V arrondissement, dans le quartier Saint-Victor, à côté de la Maison de la Mutualité. Depuis le 27 février 1977, date de son occupation par la force par des proches de la Fraternité sacerdotale Saint-Pie-X dont elle dépend officieusement depuis, elle constitue le principal lieu de culte parisien mouvement catholique traditionaliste — parfois qualifié d'« intégrisme » — ainsi que, dans une certaine mesure, de la frange catholique de l'extrême droite française.
http://dbpedia.org/resource/Saint-%C3%89tienne-du-Mont	Église Saint-Étienne-du-Mont	48.8466666667 2.34805555556	L'église Saint-Étienne-du-Mont est une église située sur la montagne Sainte-Geneviève, dans le V arrondissement de Paris, à proximité du lycée Henri-IV et du Panthéon.
http://dbpedia.org/resource/Saint-S%C3%A9verin_%28Paris%29	Église Saint-Séverin	48.8520388889 2.34567222222	L'église Saint-Séverin est une petite église du Quartier Latin de Paris, dans le quartier de Saint-Séverin . C'est la plus ancienne église située sur la rive gauche de Paris. L'ensemble formé par l'ég la « maison paroissiale - presbytère », le square André-Lefèbvre et le cloître est délimité par la rue des Prêtres-Saint-Séverin, la rue de la Parcheminerie, la rue Saint-Jacques et la rue Saint-Séverin. est utilisée comme lieu de culte de la paroisse Saint-Séverin-Saint-Nicolas. En effet, depuis 1977, de l'occupation de l'église Saint-Nicolas-du-Chardonnet par des fidèles de la Fraternité sacerdotale Saint-Pie-X, le territoire de Saint-Nicolas dépend également de Saint-Séverin. La paroisse a recouv l'usage d'un immeuble sis 39, boulevard Saint-Germain, à proximité de Saint-Nicolas.
http://dbpedia.org/resource/Piti%C3%A9-Salp%C3%AAtri%C3%A8re_Hospital	Hôpital de la Salpêtrière	48.837 2.365	L'hôpital de la Salpêtrière est un hôpital de l'Assistance publique - hôpitaux de Paris (AP-HP) situé dans le 13 arrondissement de Paris.
http://dbpedia.org/resource/Abbey_of_Saint-Germain-des-Pr%C3%A9s	Abbaye de Saint-Germain-des-Prés	48.8538888889 2.33444444444	L'abbaye de Saint-Germain-des-Prés était la plus ancienne et la plus prestigieuse abbaye de Paris, située dans le quartier de Saint-Germain dans le 6e arrondissement de Paris. Elle fut supprimée à l Révolution. Son abbatiale sert depuis le Concordat d'église paroissiale.
http://dbpedia.org/resource/%C3%89glise_Saint-Sulpice%2C_Paris	Église Saint-Sulpice (Paris)	48.8511111111 2.33472222222	L'église Saint-Sulpice est une grande église de Paris, située dans le VI arrondissement.

Inference Rule example using transitive properties from SKOS vocabulary: Variant II

This example shows how to find entities that are subcategories of Protestant Churches, no deeper than 3 levels within the concept scheme hierarchy, filtered by a specific subcategory. It demonstrates use of inference rules, sub-queries, and filter to obtain entities associated with category: Protestant_churches combined with the use of the transitivitve closure, sets to a maximum of 3 steps down a SKOS based concept scheme hierarchy:

1. Make sure the inference rule "skos-trans" is created as described in the previous example
2. Go to SPARQL endpoint, for ex. http://lod.openlinksw.com/sparql
3. Use inference rule pragma to set context rule for SPARQL query, i.e:

```
DEFINE input:inference "skos-trans"
PREFIX p: <http://dbpedia.org/property/>
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX category: <http://dbpedia.org/resource/Category:>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX geo: <http://www.georss.org/georss/>

SELECT DISTINCT ?c AS ?skos_broader
       ?trans AS ?skos_narrower
       ?dist AS ?skos_level
       ?m ?n ?p AS ?geo_point
WHERE
```

```
{
  {
    SELECT ?c  ?m ?n ?p ?trans ?dist
    WHERE
      {
            ?m rdfs:label ?n.
            ?m skos:subject ?c.
            ?c skos:broaderTransitive category:Protestant_churches .
            ?c skos:broaderTransitive ?trans
              OPTION ( TRANSITIVE,
                       t_distinct,
                       t_in (?c),
                       t_out (?trans),
                       t_max (3),
                       t_step ( 'step_no' ) as ?dist ) .
            ?m p:abstract ?d.
            ?m geo:point ?p
            FILTER ( lang(?n) = "en" )
            FILTER ( lang(?d) = "en" )
      }
  }
  FILTER ( ?trans = <http://dbpedia.org/resource/Category:Churches_in_London> )
}
ORDER BY ASC (?dist)
```

4. You will get 22 rows returned from the query.

Figure 16.43. Transitive option

File Help Login

QBE Advanced Results

Machine-readable

Result SPARQL Params Response Query

Execute Permalink

skos_broader	skos_narrower	skos_level	m	n
http://dbpedia.org/resource/Category:Methodist_churches_in_London	http://dbpedia.org/resource/Category:Churches_in_London	2	http://dbpedia.org/resource/West_London_Methodist_Mission	West London Methodist Mission
http://dbpedia.org/resource/Category:Methodist_churches_in_London	http://dbpedia.org/resource/Category:Churches_in_London	2	http://dbpedia.org/resource/West_Street_Chapel	West Street Chapel
http://dbpedia.org/resource/Category:Baptist_churches_in_London	http://dbpedia.org/resource/Category:Churches_in_London	2	http://dbpedia.org/resource/Morden_Baptist_Church	Morden Baptist Church
http://dbpedia.org/resource/Category:Pentecostal_churches_in_London	http://dbpedia.org/resource/Category:Churches_in_London	2	http://dbpedia.org/resource/Living_Word_Christian_Fellowship	Living Word Christian Fellowship
http://dbpedia.org/resource/Category:Methodist_churches_in_London	http://dbpedia.org/resource/Category:Churches_in_London	2	http://dbpedia.org/resource/Westminster_Central_Hall	Westminster Central Hall
http://dbpedia.org/resource/Category:Methodist_churches_in_London	http://dbpedia.org/resource/Category:Churches_in_London	2	http://dbpedia.org/resource/Westminster_Central_Hall	Westminster Central Hall
http://dbpedia.org/resource/Category:Pentecostal_churches_in_London	http://dbpedia.org/resource/Category:Churches_in_London	2	http://dbpedia.org/resource/Jubilee_International_Church	Jubilee International Church
http://dbpedia.org/resource/Category:Baptist_churches_in_London	http://dbpedia.org/resource/Category:Churches_in_London	2	http://dbpedia.org/resource/Metropolitan_Tabernacle	Metropolitan Tabernacle
http://dbpedia.org/resource/Category:Baptist_churches_in_London	http://dbpedia.org/resource/Category:Churches_in_London	2	http://dbpedia.org/resource/Metropolitan_Tabernacle	Metropolitan Tabernacle
http://dbpedia.org/resource/Category:Pentecostal_churches_in_London	http://dbpedia.org/resource/Category:Churches_in_London	2	http://dbpedia.org/resource/Kingsway_International_Christian_Centre	Kingsway International Christian Centre
http://dbpedia.org/resource/Category:Pentecostal_churches_in_London	http://dbpedia.org/resource/Category:Churches_in_London	2	http://dbpedia.org/resource/New_Wine_Church	New Wine Church
http://dbpedia.org/resource/Category:Baptist_churches_in_London	http://dbpedia.org/resource/Category:Churches_in_London	2	http://dbpedia.org/resource/Poverest_Road_Baptist_Church	Poverest Road Baptist Church
http://dbpedia.org/resource/Category:Lutheran_churches_in_London	http://dbpedia.org/resource/Category:Churches_in_London	2	http://dbpedia.org/resource/St_Anne_and_St_Agnes	St Anne and St Agnes
http://dbpedia.org/resource/Category:Church_of_Scotland_churches_in_London	http://dbpedia.org/resource/Category:Churches_in_London	2	http://dbpedia.org/resource/Crown_Court_Church	Crown Court Church

16.2.12. Supported SPARQL-BI "define" pragmas

SPARQL-BI compiler and run-time support are not isolated from environment and some used heuristics are not perfect and sometimes different use cases require different behavior within same standard. These reasons are seen frequently in the industry, and the solution is well known: compiler pragmas. So we allow them at the beginning of any SPARQL query in form:

```
define QName value
```

Pragmas to control the input data set for the query

```
input:default-graph-uri works like "FROM" clause;
input:named-graph-uri works like "FROM NAMED" clause;
input:default-graph-exclude works like "NOT FROM" clause;
input:named-graph-exclude works like "NOT FROM NAMED" clause.
```

The difference is that these pragmas have higher priority and they can be used for security restrictions in combination with `define input:freeze` that blocks further changes in the list of source graphs. The web service endpoint (or similar non-web application) can edit the incoming query by placing list of pragmas ended with `input:freeze` in front of query text. Even if the intruder will try to place some graph names, it will get compilation error, not an access to the data. input:freeze disables all `input:grab-...` pragmas as well.

- `input:ifp` : adds IFP keyword in OPTION (QUIETCAST, ...) clause in the generated SQL. The value of this define is not used ATM, an empty string is safe for future extensions.
- `input:same-as` : works like input:ifp but adds SAME_AS keyword.
- `input:storage` : selects quad map storage to use. The value is an IRI of storage, the default value is, of course, virtrdf:DefaultQuadStorage. If the value is an empty string then only quads from RDF_VIEW are used. This is a good choice for low-level admin procedures, for two reasons: they will not interfere with any changes in virtrdf:DefaultQuadStorage and they will continue to work even if all compiler's metadata are corrupted, including the description of virtrdf:DefaultQuadStorage (define input:storage "" switches the SPARQL compiler to a small set of metadata that are built in server C code and thus are very hard to corrupt by users).
- `input:inference` : specifies the name of inference rule set to use.
- `input:param (and synonyms input:params, sql:param, sql:params)` : declares a variable name as a protocol parameter. The SPARQL query can refer to protocol parameter X via variable with special syntax of name ?::X . If a query text should be made by query builder that does not understand SPARQL-BI extensions then the text may contain variable ?X and define input:param "X" . This does not work for positional parameters, one can not replace a reference to ?::3 with ?3 and define input:param "3".
- `input:grab-var` : Network Resource Fetch values of variable;
- `input:grab-iri` : Network Resource Fetch the constant IRI;
- `input:grab-all` : Network Resource Fetch all constants and variables of the query;
- `input:grab-seealso (and synonym input:grab-follow-predicate)` : sets predicate that tells where to Fetch more Network Resource data about a subject;
- `input:grab-limit` : how many resources can be fetched;
- `input:grab-depth` : how many iterations can be done, sponging additional data on each iteration;
- `input:grab-base` : base to resolve relative IRIs before passing to Sponger;
- `input:grab-resolver` : IRI resolving procedure (i.e., one that turns base and relative IRI to an absolute IRI);
- `input:grab-destination` : single resource that should be filled in with results of all fetchings;
- `input:grab-loader` : a name of procedure that retrieve the resource via HTTP, parse it and store it.

All these pragmas are described in more details here , but in addition there are some experimental:

- `input:grab-intermediate` : extends the set of IRIs to sponge, useful in combination with input:grab-seealso. If present then for a given subject, Network Resource Fetch will retrieve not only values of see-also predicates for that subject but the subject itself. The define value is not used in current implementation.
- `input:grab-group-destination` : resembles input:grab-destination but sponges will create individual graphs for Network Resource Fetch results, and in additional to this common routine, a copy of each Network Resource Fetch result is added to the resource specified by the value of input:grab-group-destination. input:grab-destination redirects loadings, input:grab-group-destination duplicates them.
- `get:soft` : "soft" or "replacing", depending on mode of loading source graph;
- `get:uri` : an URI of web resource where the graph should come from (e.g., a local mirror);
- `get:method` : "GET" or "MGET", depending on loading the resource itself or loading metadata about the resource;
- `get:refresh` : limits the lifetime of a local cached copy of the source, the value is in seconds. Should be used in combination with get:soft;

- `get:proxy` : the proxy server to use, as "host:port" string.

These defines are described also here . Note that all of them can be used in option list of "FROM ... OPTION (get:...)" extended SPARQL-BI syntax for FROM/FROM NAMED clause.

Note that all of them can be used in option list of "FROM ... OPTION (get:...)" extended SPARQL-BI syntax for FROM/FROM NAMED clause.

Pragmas to control code generation

- `sql:assert-user` : defines the user who is supposed to be the single "proper" use for the query. If the compiler is launched by other user, an error is signaled. The typical use is define sql:assist-user "dba". This is too weak to be a security measure, but may help in debugging of security issues.
- `sql:gs-app-callback` : application-specific callback that returns permission bits of a given graph;
- `sql:gs-app-uid` : application-specific user id to use in callback.

See Also:

RDF Graph Security

- `sql:globals-mode` : tells how to print names of global variables, supported values are "XSLT" (print colon before name of global variable and "SQL" (print as usual).
- `sql:log-enable` : value that will be passed to SPARUL procedures and there it will be passed to log_enable() BIF. Thus define sql:log-enable N will result in log_enable(N, 1) at the beginning of the operation and other log_enable() call will restore previous mode of transaction log at exit from the procedure or at any error signalled from it.
- `sql:table-option` : value will be added as an option to each triple in the query and later it will be printed in TABLE OPTION (...) clause of source table clause. This works only for SQL code for plain triples from RDF_QUAD, fragments of queries related to RDF Views will remain unchanged.
- `sql:select-option` : value will be added as an global OPTION () clause of the generated SQL SELECT. This clause is always printed, it is always at least OPTION (QUIETCAST, ...). The most popular use case is define sql:table-option "ORDER" to tell the SQL compiler execute joins in the order of their use in the query (this can make query compilation much faster but the compilation result can be terrible if you do not know precisely what you're doing and not inspected execution plan of the generated SQL query).
- `sql:describe-mode` : sets procedures that will produce the result of a DESCRIBE query. The pragma is ignored for other types of SPARQL queries. In the default mode, the result contains all X ?p ?o and all ?s ?p X triples for each given X. In "SPO" mode, the result contains X ?p ?o triples only. In "CBD" mode, the result contains concise bound descriptions of given subjects. Application developers may add more modes.
- `sql:signal-void-variables` : the most useful debugging variable if Linked Data Views are in use. It tells the SPARQL compiler to signal an error if it can prove that some variable can never be bound. Usually it means error in query, like typo in IRI or totally wrong triple pattern.

Pragmas to control the type of the result

- `output:valmode` : tells the compiler which SQL datatypes should be used for output values. ODBC clients and the like known nothing about RDF and expect plain SQL values, so the appropriate value for them is "SQLVAL" and that's the default. When a Virtuoso/PL procedure is RDF-aware and keeps results for further passing to other SPARQL queries or some low-level RDF routines, the value "LONG" tells the compiler to preserve RDF boxes as is and to return IRI IDs instead of IRI string value. Third possible value, "AUTO", is for dirty hackers that do not want any conversion of any sort at the output to read the SQL output of SPARQL front-end, find the format of each column and add the needed conversions later. You will probably never need it.
- `output:format` : tells the compiler that the query should produce a string output with the serialization of the result, not a result set. There are three of them because the caller, like SPARQL web service endpoint, may not know the actual type of the query that should be executed. The value of output:format is used for SELECT and data manipulation queries, if specified, it can also be used for CONSTRUCT, DESCRIBE or ASK, if it is specified but related output:dict-format or output:scalar-format is not.
- `output:scalar-format` : tells the compiler that the query should produce a string output with the serialization of the result, not a result set. There are three of them because the caller, like SPARQL web service endpoint, may not know the actual type of the query that should be executed. The value of output:scalar-format is used for ASK queries only, if specified.
- `output:dict-format` : tells the compiler that the query should produce a string output with the serialization of the result, not a result set. There are three of them because the caller, like SPARQL web service endpoint, may not know the actual type of the query that should be executed. The value of output:dict-format is used for CONSTRUCT and

DESCRIBE queries only, if specified.

Supported formats that return a string session

- "RDF/XML",
- "TURTLE" (and "TTL" is a synonym),
- "JSON" (canonical JSON for result sets, Talis-style JSON for CONSTRUCT and DESCRIBE),
- "JSON;ODATA" (oData-style JSON for CONSTRUCT and DESCRIBE, error otherwise),
- "RDFA;XHTML" (only for CONSTRUCT and DESCRIBE, error otherwise),
- "ATOM;XML" (only for CONSTRUCT and DESCRIBE as well).

Supported formats that do not return a string session to the caller

Supported formats that do not return a string session to the caller, but form an HTTP response instead and send it directly to the client HTTP connection with an appropriate HTTP header:

- "HTTP+XML mime/type",
- "HTTP+TTL mime/type",
- "HTTP+NT mime/type". A MIME type in value will be placed in the returned header, it should be separated from the starting keyword with one white space.

Supported Special formats

A special format "_JAVA_" is for SPARQL queries sent via JDBC. It changes only the output of ASK queries.

The "_JAVA_" and "_UDBC_" are aliases in Virtuoso Version 6.1.5. Till Virtuoso 6.1.5 the default behaves as "TTL". For Virtuoso version 6.1.5 and higher it is ODBC/JDBC oriented e.g. "_UDBC_" is the default format for ODBC/JDBC clients.

Note : If you want to revert to old TTL behaviour, you should specify it explicitly via:

```
define output:format "TTL"
```

Note: Pragmas output:valmode and output:format may conflict if used together, and if they're not in conflict then output:valmode is redundant: the compiler knows for sure which output:valmode-s are needed by various output:format-s.

- `output:route` : works only for SPARUL operators and tells the SPARQL compiler to generate procedure names that differ from default. As a result, the effect of operator will depend on application. That is for tricks. E.g., consider an application that extracts metadata from DAV resources stored in the Virtuoso and put them to RDF storage to make visible from outside. When a web application has permissions and credentials to execute a SPARUL query, the changed metadata can be written to the DAV resource (and after that the trigger will update them in the RDF storage), transparently for all other parts of application.
- `output:maxrows` : limits the number of rows in the returned result set. The integer value is expected, the positive integer value is obviously recommended.

Minor notes

Values of most pragmas are strings. Exceptions are:

- input:grab-depth,
- input:grab-limit,
- output:maxrows,
- sql:log-enable,
- sql:signal-void-variables

that have integer values.

Values of some pragmas a passed through the compiler to the run-time so they are seen in the generated SQL code as arguments of procedures:

- get:method,
- get:proxy,

- get:query,
- get:refresh,
- get:soft,
- get:uri

so sometimes you may meet them in SQL debuggers output and the like.

16.2.13. Built-in bif functions

- *bif:__rdf_long_from_batch_params(i_nt integer, st_value, st2_value)*

  - For value URI, the params values should be: 1, value.stringValue(), NULL
  - For value BNODE, the params values should be: 1, "_:"+((BNode)value).getID(), NULL
  - For value Literal with Language!=NULL, the params values should be: 5, lit.stringValue(), lit.getLanguage()
  - For value Literal with Datatype!=NULL, the params values should be: 4, lit.stringValue(), lit.getDatatype().toString()
  - For value Literal with Datatype==NULL && Language==NULL, the params values should be: 3, lit.stringValue(), NULL
  - For value any value exclude above, the params values should be: 3, value.stringValue(), NULL
  - For string value (without Datatype and Language), the params values should be: 3, value.stringValue(), NULL

Example:

```
SPARQL SELECT *
WHERE
  { graph ?g { `iri(??)` `iri(??)`
    `bif:__rdf_long_from_batch_params(3,value.stringValue(),NULL)` }
  }
```

16.2.14. Sending SOAP Requests to Virtuoso SPARQL Endpoint

This section presents a sample scenario on how to execute a SPARQL query as a SOAP request to the Virtuoso SPARQL Endpoint.

1. Assume the following sample SOAP request containing simple SPARQL query:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.
   <soapenv:Body>
      <query-request xmlns="http://www.w3.org/2005/09/sparql-protocol-types/#">
         <query xmlns="">SELECT DISTINCT ?z FROM virtrdf: {?x ?y ?z .} LIMIT 10</query>
      </query-request>
   </soapenv:Body>
</soapenv:Envelope>
```

2. Save locally the content from above for ex. to file with the name "soap.xml".
3. To pass the SOAP request to a Virtuoso SPARQL Endpoint, execute the following curl command:

```
$ curl -d@soap.xml -H "Content-Type:text/xml" -H "SOAPAction: ''" http://example.com/sparql
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <query-result xmlns="http://www.w3.org/2005/09/sparql-protocol-types/#">
      <sparql xmlns="http://www.w3.org/2005/sparql-results#" xmlns:xsi="http://www.w3.org/2001/XML
        <head>
         <variable name="z"/>
        </head>
        <results distinct="false" ordered="true">
          <result>
            <binding name="z">
             <uri>http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat</uri>
            </binding>
          </result>
          <result>
            <binding name="z">
             <uri>http://www.openlinksw.com/schemas/virtrdf#QuadStorage</uri>
            </binding>
          </result>
          <result>
            <binding name="z">
```

```
          <uri>http://www.openlinksw.com/schemas/virtrdf#array-of-QuadMapFormat</uri>
        </binding>
      </result>
      <result>
        <binding name="z">
         <uri>http://www.openlinksw.com/schemas/virtrdf#QuadMap</uri>
        </binding>
      </result>
      <result>
        <binding name="z">
         <uri>http://www.openlinksw.com/schemas/virtrdf#QuadMapValue</uri>
        </binding>
      </result>
      <result>
        <binding name="z">
         <uri>http://www.openlinksw.com/schemas/virtrdf#array-of-QuadMapColumn</uri>
        </binding>
      </result>
      <result>
        <binding name="z">
         <uri>http://www.openlinksw.com/schemas/virtrdf#QuadMapColumn</uri>
        </binding>
      </result>
      <result>
        <binding name="z">
         <uri>http://www.openlinksw.com/schemas/virtrdf#array-of-QuadMapATable</uri>
        </binding>
      </result>
      <result>
        <binding name="z">
         <uri>http://www.openlinksw.com/schemas/virtrdf#QuadMapATable</uri>
        </binding>
      </result>
      <result>
        <binding name="z">
         <uri>http://www.openlinksw.com/schemas/virtrdf#QuadMapFText</uri>
        </binding>
      </result>
    </results>
  </sparql>
 </query-result>
</soapenv:Body>
</soapenv:Envelope>
```

16.2.15. Use of Hash Join With RDF

For queries that touch large quantities of RDF data and have many selection conditions, use of HASH JOIN is often desirable. For short lookup queries, HASH JOIN is usually not desirable.

Depending on the version, the query optimizer may or may not do HASH JOINs with RDF. This is controlled by the *hash_join_enable* flag.

To check the flag, do:

```
sys_stat ('hash_join_enable');
```

- 0 - means that hash joins are never made
- 1 - means that these are for SQL only
- 2 - means that these can also be used with RDF

The flag is set in the ini file in the [Flags] section:

```
[Flags]
hash_join_enable = 2
```

The flag may be transiently set with the SQL statement:

```
__dbf_set ('hash_join_enable', 2);
```

To check the effectiveness of HASH JOINs and whether the optimizer introduces these in the first place, it is most convenient to use the *profile* function.

In the following example, we look at the star schema benchmark Q4:

```
SPARQL PREFIX rdfh: <http://lod2.eu/schemas/rdfh#>
SELECT SUM(?rdfh_lo_revenue) AS ?lo_revenue ?d_year ?p_brand1
FROM <http://lod2.eu/schemas/rdfh-inst#ssb1_ttl_qb>
WHERE
  {
    ?li a rdfh:lineorder ;
        rdfh:lo_orderdate ?lo_orderdate ;
        rdfh:lo_partkey ?lo_partkey ;
        rdfh:lo_suppkey ?lo_suppkey ;
              rdfh:lo_revenue ?rdfh_lo_revenue .

    ?lo_orderdate rdfh:d_year ?d_year .
    ?lo_partkey rdfh:p_brand1 ?p_brand1 .
    ?lo_partkey rdfh:p_category "MFGR#12" .
    ?lo_suppkey rdfh:s_region "AMERICA" .
  }
GROUP BY ?d_year ?p_brand1
ORDER BY ?d_year ?p_brand1
;
```

The query aggregates rows from a large fact table and selects based on a date range, a brand, and the location of the supplier. To run this, it is best to put the query in a file and have profile ('sparql... ') wrapped around the text. Then, in isql:

```
SET SET BLOBS ON;
LOAD q4.sql;
```

Without HASH JOIN, the profile is:

```
{
time   1.9e-06% fanout         1 input         1 rows

Precode:
      0: __rdflit := Call __rdflit (rdflit170373)
      5: __rdflit := Call __rdflit (rdflit16802)
      10: BReturn 0
Subquery 31
{
time     1e-06% fanout         1 input         1 rows
{ fork
time   0.00035% fanout         1 input         1 rows
{ fork
time       3.6% fanout     1e+06 input         1 rows
RDF_QUAD     1e+06 rows(s_18_9_t6.S, s_18_9_t6.O)
 inlined  P =  #dfh#p_brand1  G =  #nst#ssb1_ttl_qb
time       1.7% fanout   0.03979 input     1e+06 rows
RDF_QUAD_POGS unq      0.04 rows (s_18_9_t7.S)
 P =  #dfh#p_category  ,  O = rdflit170373 ,  S = s_18_9_t6.S ,  G =  #nst#ssb1_ttl_qb
time       2.5% fanout   180.179 input     39790 rows

Precode:
      0: __ro2sq := Call __ro2sq (s_18_9_t6.O)
      5: BReturn 0
RDF_QUAD_POGS   4.4e+02 rows(s_18_9_t2.S)
 P =  #dfh#lo_partkey  ,  O = k_s_18_9_t6.S G =  #nst#ssb1_ttl_qb
time        35% fanout         1 input 7.16932e+06 rows
RDF_QUAD         1 rows(s_18_9_t3.O, s_18_9_t3.S)
 inlined  P =  #dfh#lo_suppkey  ,  S = s_18_9_t2.S G =  #nst#ssb1_ttl_qb
time       4.5% fanout  0.201214 input 7.16932e+06 rows
RDF_QUAD_POGS unq       0.2 rows ()
 P =  #dfh#s_region  ,  O = rdflit16802 ,  S = cast ,  G =  #nst#ssb1_ttl_qb
time        21% fanout         1 input 1.44256e+06 rows
RDF_QUAD         1 rows(s_18_9_t4.S, s_18_9_t4.O)
 inlined  P =  #dfh#lo_revenue  ,  S = k_s_18_9_t2.S G =  #nst#ssb1_ttl_qb
time        12% fanout         1 input 1.44256e+06 rows
RDF_QUAD_POGS unq       0.8 rows (s_18_9_t0.S)
 P =  #-ns#type  ,  O =  #dfh#lineorder  ,  S = k_s_18_9_t2.S ,  G =  #nst#ssb1_ttl_qb
```

```
time        14% fanout         1 input 1.44256e+06 rows
RDF_QUAD         1 rows(s_18_9_t1.O)
 inlined  P =  #dfh#lo_orderdate  ,  S = s_18_9_t0.S G =  #nst#ssb1_ttl_qb
time       3.5% fanout         1 input 1.44256e+06 rows
RDF_QUAD         1 rows(s_18_9_t5.O)
 inlined  P =  #dfh#d_year  ,  S = cast G =  #nst#ssb1_ttl_qb
time       1.9% fanout         0 input 1.44256e+06 rows
Sort (s_18_9_t5.O, s_18_9_t6.O) -> (s_18_9_t4.O, __ro2sq)

}
time   4.1e-05% fanout       280 input         1 rows
group by read node
(s_18_9_t5.O, s_18_9_t6.O, aggregate, __ro2sq)
time   0.00043% fanout         0 input       280 rows

Precode:
      0: __ro2sq := Call __ro2sq (s_18_9_t5.O)
      5: BReturn 0
Sort (__ro2sq, __ro2sq) -> (aggregate)

}
time   2.9e-05% fanout       280 input         1 rows
Key from temp (aggregate, __ro2sq, __ro2sq)

After code:
      0: lo_revenue :=  := artm aggregate
      4: d_year :=  := artm __ro2sq
      8: p_brand1 :=  := artm __ro2sq
      12: BReturn 0
time   7.6e-07% fanout         0 input       280 rows
Subquery Select(lo_revenue, d_year, p_brand1)
}

After code:
      0: lo_revenue := Call __ro2sq (lo_revenue)
      5: d_year := Call __ro2sq (d_year)
      10: p_brand1 := Call __ro2sq (p_brand1)
      15: BReturn 0
time   6.3e-07% fanout         0 input       280 rows
Select (lo_revenue, d_year, p_brand1)
}

 5542 msec 2420% cpu, 2.11877e+07 rnd 8.13668e+06 seq   85.6039% same seg   13.6018% same pg
Compilation: 10 msec 0 reads         0% read 0 messages         0% clw

  <para>With hash join the profile is:</para>
<programlisting><![CDATA[
{
time   1.4e-05% fanout         1 input         1 rows
time         7% fanout         1 input         1 rows

Precode:
      0: __rdflit := Call __rdflit (rdflit170373)
      5: __rdflit := Call __rdflit (rdflit16802)
      10: BReturn 0
{ hash filler
time     0.088% fanout     1e+06 input         1 rows
RDF_QUAD     1e+06 rows(s_18_9_t6.S, s_18_9_t6.O)
 inlined  P =  #dfh#p_brand1  G =  #nst#ssb1_ttl_qb
time      0.15% fanout         0 input     1e+06 rows
Sort hf 39 (s_18_9_t6.S, s_18_9_t6.S) -> (s_18_9_t6.O)

}
time   0.00046% fanout         1 input         1 rows
{ hash filler
time    0.0004% fanout      2556 input         1 rows
RDF_QUAD_POGS   2.6e+03 rows(s_18_9_t5.S, s_18_9_t5.O)
 inlined  P =  #dfh#d_year  G =  #nst#ssb1_ttl_qb
time   0.00056% fanout         0 input      2556 rows
Sort hf 56 (s_18_9_t5.S) -> (s_18_9_t5.O)

}
time    0.0036% fanout         1 input         1 rows
```

```
{ hash filler
time   0.00094% fanout     12068 input         1 rows
RDF_QUAD_POGS   1.2e+04 rows(s_18_9_t8.S)
 P =  #dfh#s_region  ,  O = rdflit16802 G =  #nst#ssb1_ttl_qb
time   0.00046% fanout         0 input     12068 rows
Sort hf 69 (s_18_9_t8.S)
}
time     0.012% fanout         1 input         1 rows
{ hash filler
time    0.0026% fanout     39790 input         1 rows
RDF_QUAD_POGS     4e+04 rows(s_18_9_t7.S)
 P =  #dfh#p_category  ,  O = rdflit170373 G =  #nst#ssb1_ttl_qb
time    0.0013% fanout         0 input     39790 rows
Sort hf 82 (s_18_9_t7.S)
}
Subquery 88
{
time   1.5e-05% fanout         1 input         1 rows
{ fork
time   1.3e-05% fanout         1 input         1 rows
{ fork
time        52% fanout 7.16932e+06 input         1 rows
RDF_QUAD   1.8e+08 rows(s_18_9_t2.O, s_18_9_t2.S)
 inlined  P =  #dfh#lo_partkey  G =  #nst#ssb1_ttl_qb
hash partition+bloom by 86 (tmp)hash join merged always card      0.04 -> ()
time       6.1% fanout         1 input 7.16932e+06 rows

Precode:
      0: s_18_9_t7.S :=  := artm s_18_9_t2.O
      4: BReturn 0
Hash source 82 merged into ts       0.04 rows(cast) -> ()
time         7% fanout  0.201214 input 7.16932e+06 rows
RDF_QUAD         1 rows(s_18_9_t3.O, s_18_9_t3.S)
 inlined  P =  #dfh#lo_suppkey  ,  S = s_18_9_t2.S G =  #nst#ssb1_ttl_qb
hash partition+bloom by 73 (tmp)hash join merged always card       0.2 -> ()
time    0.0018% fanout         1 input 1.44256e+06 rows
Hash source 69 merged into ts        0.2 rows(cast) -> ()
time       2.3% fanout         1 input 1.44256e+06 rows
RDF_QUAD_POGS unq       0.8 rows (s_18_9_t0.S)
 P =  #-ns#type  ,  O =  #dfh#lineorder  ,  S = k_s_18_9_t2.S ,  G =  #nst#ssb1_ttl_qb
time       2.3% fanout         1 input 1.44256e+06 rows
RDF_QUAD         1 rows(s_18_9_t1.O, s_18_9_t1.S)
 inlined  P =  #dfh#lo_orderdate  ,  S = s_18_9_t0.S G =  #nst#ssb1_ttl_qb
hash partition+bloom by 60 ()
time      0.38% fanout         1 input 1.44256e+06 rows
Hash source 56           1 rows(cast) -> (s_18_9_t5.O)
time       2.2% fanout         1 input 1.44256e+06 rows
RDF_QUAD         1 rows(s_18_9_t4.O)
 inlined  P =  #dfh#lo_revenue  ,  S = k_s_18_9_t0.S G =  #nst#ssb1_ttl_qb
time        20% fanout         1 input 1.44256e+06 rows
Hash source 39         1.6 rows(k_s_18_9_t2.O, k_s_18_9_t7.S) -> (s_18_9_t6.O)
time      0.86% fanout         0 input 1.44256e+06 rows
Sort (set_no, s_18_9_t5.O, s_18_9_t6.O) -> (s_18_9_t4.O)

}
time   0.00023% fanout       280 input         1 rows
group by read node
(gb_set_no, s_18_9_t5.O, s_18_9_t6.O, aggregate)
time       0.1% fanout         0 input       280 rows

Precode:
      0: __ro2sq := Call __ro2sq (s_18_9_t6.O)
      5: __ro2sq := Call __ro2sq (s_18_9_t5.O)
      10: BReturn 0
Sort (__ro2sq, __ro2sq) -> (aggregate)

}
time    0.0002% fanout       280 input         1 rows
Key from temp (aggregate, __ro2sq, __ro2sq)

After code:
      0: lo_revenue :=  := artm aggregate
      4: d_year :=  := artm __ro2sq
```

```
      8: p_brand1 :=  := artm __ro2sq
      12: BReturn 0
time   5.3e-06% fanout         0 input       280 rows
Subquery Select(lo_revenue, d_year, p_brand1)
}

After code:
      0: lo_revenue := Call __ro2sq (lo_revenue)
      5: d_year := Call __ro2sq (d_year)
      10: p_brand1 := Call __ro2sq (p_brand1)
      15: BReturn 0
time   5.5e-06% fanout         0 input       280 rows
Select (lo_revenue, d_year, p_brand1)
}

 3101 msec 993% cpu, 1.14967e+07 rnd 1.81041e+08 seq   99.5619% same seg  0.417643% same pg
Compilation: 23 msec 0 reads         0% read 0 messages         0% clw
```

These are runs on warm cache on a dataset of scale factor 30, about 3 billion triples.

We notice that the HASH-based plan completes faster and has a lower CPU percentage. This is to be expected since HASH JOINs are especially useful for JOINs between a large table and a smaller one.

The INDEX-based plan does 21M random INDEX lookups whereas the HASH-based one does only 11M. We also note that the INDEX access pattern is more local with the HASH plan, with 99% of lookups hitting the same segment as the previous, against only 85%.

These numbers are in the summary at the bottom of each profile:

- rnd -- means index access
- seq -- means rows retrieved in sequential scan; same seg is the percentage of index accesses that hit the same segment as the previous access.

The INDEX-based plan starts with the smallest selection; in this case, the days parts with the given brand. From this, it joins to the lineorder and gets the supplier. It fetches the region of the supplier and leaves out the ones not in America.

The HASH-based plan makes a hash table of all the parts with the brand, all the suppliers in America, and all the days in the time dimension. It then scans lineorder, and first drops the rows whose part is not in the hash, then drops the rows where the supplier is not in the hash, and then gets the year of each date. This last operation does not drop any rows, but is still done by hash, because there are relatively few days and the day-to-year translation is done a very large number of times.

The number of rows in and out of each operator is given after the time percent, above the operator. Fanout is the number of rows of output per one row of input.

Given the long-running queries of any workload, you can perform this same comparison to determine if HASH JOIN is useful in the case at hand. Looking at the real time and CPU% is usually enough.

- Using the sql:select-option pragma: One can specify the HASH JOIN is not to be used.

  ```
  define sql:select-option "loop"
  ```

  will exclude use of HASH JOIN in the specific query.
- The table_option construct can be used to select the JOIN type for any triple pattern.

  ```
  {
    ?lo rdfh:lo_suppkey ?supp .
    ?supp rdfh:s_region  "AMERICA" option (table_option "hash")
  }
  ```

  would have the effect of building a hash from the suppliers in America.

You may wish to experiment with these options and look at the profile output for each.

For some analytics workloads, enabling HASH JOIN may give a factor of 2 or 3 better performance. For lookup workloads, there may be no gain.

Sometimes a HASH JOIN may be used when an INDEX lookup would be better, thus in some cases it makes sense to turn off HASH JOINs either per query or globally.

16.3. Extensions

16.3.1. Using Full Text Search in SPARQL

Virtuoso's triple store supports optional full text indexing of RDF object values since version 5.0. It is possible to declare that objects of triples with a given predicate or graph get indexed. The graphs and triples may be enumerated or a wildcard may be used.

The triples for which a full text index entry exists can be found using the *bif:contains* or related filters and predicates.

For example, the query:

```
SQL>SELECT *
FROM <people>
WHERE
  {
    ?s foaf:Name ?name . ?name bif:contains "'rich*'".
  }
```

would match all subjects whose *foaf:Name* contain a word Rich. This would match Richard, Richie etc.

Note that words and phrases should be enclosed in quotes if they contain spaces or other non-alphanumeric chars.

If the *bif:contains* or related predicate is applied to an object that is not a string or is not the object of an indexed triple, no match will be found.

The syntax for text patterns is identical to the syntax for the SQL contains predicate.

The SPARQL/SQL optimizer determines whether the text pattern will be used to drive the query or whether it will filter results after other conditions are applied first. In contrast to *bif:contains* , regexp matching never drives the query or makes use of an index, thus in practice regexps are checked after other conditions.

Specifying What to Index

Whether the object of a given triple is indexed in the text index depends on indexing rules. If at least one indexing rule matches the triple, the object gets indexed if the object is a string. An indexing rule specifies a graph and a predicate. Either may be an IRI or NULL, in which case it matches all IRI's.

Rules also have a 'reason', which can be used to group rules into application-specific sets. A triple will stop being indexed only after all rules mandating its indexing are removed. When an application requires indexing a certain set of triples, rules are added for that purpose. These rules are tagged with the name of the application as their reason. When an application no longer requires indexing, the rules belonging to this application can be removed. This will not turn off indexing if another application still needs certain triples to stay indexed.

Indexing is enabled/disabled for specific graph/predicate combinations with:

```
create function DB.DBA.RDF_OBJ_FT_RULE_ADD
  (in rule_g varchar, in rule_p varchar, in reason varchar) returns integer

create function DB.DBA.RDF_OBJ_FT_RULE_DEL
  (in rule_g varchar, in rule_p varchar, in reason varchar) returns integer
```

Example: The first function adds a rule. The first two arguments are the text representation of the IRI's for the graph and predicate. If NULL is given then all graph's or predicates match. Specifying both as NULL means that all string valued objects will be added to a text index.

Example:

```
DB.DBA.RDF_OBJ_FT_RULE_ADD (null, null, 'All');
```

The second function reverses the effect of the first. Only a rule that has actually been added can be deleted. Thus one cannot say that all except a certain enumerated set should be indexed.

```
DB.DBA.RDF_OBJ_FT_RULE_DEL (null, null, 'All');
```

The reason argument is an arbitrary string identifying the application that needs this rule. Two applications can add the same rule. Removing one of them will still keep the rule in effect. If an object is indexed by more than one rule, the index data remain free from duplicates, neither index size nor speed is affected.

If *DB.DBA.RDF_OBJ_FT_RULE_ADD* detects that *DB.DBA.RDF_QUAD* contains quads whose graphs and/or predicates match to the new rule but which have not been indexed before then these quads are indexed automatically. However the function *DB.DBA.RDF_OBJ_FT_RULE_DEL* does not remove indexing data about related objects. Thus the presence of indexing data about an object does not imply that it is necessarily used in some quad that matches to some rule.

The above functions return one if the rule is added or deleted and zero if the call was redundant (the rule has been added before or there's no rule to delete).

Example

```
-- We load Tim Berners-Lee's FOAF file into a graph called 'people'.

SQL>DB.DBA.RDF_LOAD_RDFXML (http_get ('http://www.w3.org/People/Berners-Lee/card#i'), 'no', 'http://www.w
Done. -- 172 msec.

-- We check how many triples we got.

SQL>SPARQL SELECT COUNT (*) FROM <http://www.w3.org/people#> WHERE {?s ?p ?o};
callret-0
INTEGER
 266
No. of rows in result: 1

-- We check the GRAPH: <http://www.w3.org/people#> for objects like "Tim":

SQL>SPARQL
SELECT *
FROM <http://www.w3.org/people#>
WHERE
  {
    ?s ?p ?o . FILTER (?o LIKE '%Tim%')
  };
s                                                 p                                                 o
VARCHAR                                           VARCHAR                                           VARCHAR
_______________________________________________________________________________________________

http://www.w3.org/People/Berners-Lee/card#i       http://xmlns.com/foaf/0.1/name                    Timothy Berne
http://www.w3.org/People/Berners-Lee/card#i       http://xmlns.com/foaf/0.1/nick                    TimBL
http://www.w3.org/People/Berners-Lee/card#i       http://www.w3.org/2002/07/owl#sameAs              http://www4.w
http://www.w3.org/People/Berners-Lee/card#i       http://xmlns.com/foaf/0.1/knows                   http://dbpedi
http://www.w3.org/People/Berners-Lee/card#i       http://www.w3.org/2000/01/rdf-schema#label        Tim Berners-L
http://www.w3.org/People/Berners-Lee/card#i       http://xmlns.com/foaf/0.1/givenname               Timothy
http://dbpedia.org/resource/Tim_Bray              http://xmlns.com/foaf/0.1/name                    Tim Bray
no                                                http://purl.org/dc/elements/1.1/title             Tim Berners-L

8 Rows. -- 230 msec.

-- We specify that all string objects in the graph 'people' should be text indexed.

SQL>DB.DBA.RDF_OBJ_FT_RULE_ADD('http://www.w3.org/people#', null, 'people');
Done. -- 130 msec.

-- We update the text index.

SQL>DB.DBA.VT_INC_INDEX_DB_DBA_RDF_OBJ ();
Done. -- 140 msec.

-- See impact of the index  by querying the subjects and predicates
-- of all triples in the GRAPH: <http://www.w3.org/people#>,
-- where the object is a string which contains a word beginning with "TIM".
```

```
SQL>SPARQL SELECT * FROM <http://www.w3.org/people#> WHERE { ?s ?p ?o . ?o bif:contains '"Timo*"'};
s                                                 p                                   o
VARCHAR                                           VARCHAR                             VARCHAR
_______________________________________________________________________________

 http://www.w3.org/People/Berners-Lee/card#i    http://xmlns.com/foaf/0.1/name        Timothy Berners-Lee
 http://www.w3.org/People/Berners-Lee/card#i    http://xmlns.com/foaf/0.1/givenname   Timothy

2 Rows. -- 2 msec.
```

The query below is identical with that above but uses a different syntax. The filter syntax is more flexible in that it allows passing extra options to the *contains* predicate. These may be useful in the future.

```
SQL>SPARQL SELECT * FROM <people> WHERE { ?s ?p ?o . FILTER (bif:contains(?o,  '"Timo*"')) };
```

Note:

It is advisable to upgrade to the latest version of Virtuoso before adding free-text rules for the first time. This is especially the case if large amounts of text are to be indexed. The reason is that the free-text index on RDF may be changed in future versions and automatic upgrading of an existing index data into the new format may take much more time than indexing from scratch.

The table *DB.DBA.RDF_OBJ_FT_RULES* stores list of free-text index configuration rules.

```
create table DB.DBA.RDF_OBJ_FT_RULES (
  ROFR_G varchar not null,       -- specific graph IRI or NULL for "all graphs"
  ROFR_P varchar not null,       -- specific predicate IRI or NULL for "all predicates"
  ROFR_REASON varchar not null,  -- identification string of a creator, preferably human-readable
  primary key (ROFR_G, ROFR_P, ROFR_REASON) );
```

Applications may read from this table but they should not write to it directly. Duplications in the rules do not affect the speed of free-text index operations because the content of the table is cached in memory in a special form. Unlike the use of configuration functions, directly writing to the table will not update the in-memory cache.

The table is convenient to search for rules added by a given application. If a unique identification string is used during installation of an application when rules are added then it's easy to remove those rules as part of any uninstall routine.

Time of Indexing

The triple store's text index is in manual batch mode by default. This means that changes in triples are periodically reflected in the text index but are not maintained in strict synchrony. This is much more efficient than keeping the indices in constant synchrony. This setting may be altered with the *db.dba.vt_batch_update* stored procedure.

To force synchronization of the RDF text index, use:

```
DB.DBA.VT_INC_INDEX_DB_DBA_RDF_OBJ ();
```

To set the text index to follow the triples in real time, use:

```
DB.DBA.VT_BATCH_UPDATE ('DB.DBA.RDF_OBJ', 'OFF', null);
```

To set the text index to be updated every 10 minutes, use:

```
DB.DBA.VT_BATCH_UPDATE ('DB.DBA.RDF_OBJ', 'ON', 10);
```

To make the update always manual, specify NULL as the last argument above.

One problem related to free-text indexing of *DB.DBA.RDF_QUAD* is that some applications (e.g. those that import billions of triples) may set off triggers. This will make free-text index data incomplete. Calling procedure *DB.DBA.RDF_OBJ_FT_RECOVER ()* will insert all missing free-text index items by dropping and re-inserting every existing free-text index rule.

Free-Text Indexes on Linked Data Views

If an *O* field of a quad map pattern gets its value from a database column that has a free text index then this index can be used in SPARQL for efficient text searching. As a variation of this facility, the free-text index of another table may be used.

If a statement of a quad map pattern declaration starts with a declaration of table aliases, the table alias declaration may include the name of a table column that should have a text index. For example, consider the possibility of using a free-text index on the content of DAV resources stored in the DAV system tables of Virtuoso:

```
prefix mydav: <...>
create quad storage mydav:metadata
FROM WS.WS.SYS_DAV_RES as dav_resource text literal RES_CONTENT
...
  {
    ...
    mydav:resource-iri (dav_resource.RES_FULL_PATH)
        a mydav:resource ;
        mydav:resource-content dav_resource.RES_CONTENT ;
        mydav:resource-mime-type dav_resource.RESTYPE ;
    ...
  }
```

The clause *text literal RES_CONTENT* grants the SPARQL compiler permission to use a free-text index for objects that are literals composed from column *dav_resource.RES_CONTENT* . This clause also allows choosing between *text literal* (supports only the *contains()* predicate) and *text xml literal* (supports both *contains()* and *xcontains()*) text indexes. It is important to understand that the free-text index will produce results using raw relational data. If a literal class transformation changes the text stored in the column then these changes are ignored by free-text search. e.g. if a transformation concatenates a word to the value of the column, but the free-text search will not find this word.

The free-text index may be used in a more sophisticated way. Consider a built-in table *DB.DBA.RDF_QUAD* that does not have a free-text index. Moreover, the table does not contain the full values of all objects; the *O* column contains "short enough" values inlined, but long and special values are represented by links to the *DB.DBA.RDF_OBJ* table. The RDF_OBJ table, however, has free-text index that can be used. The full declaration of the built-in default mapping for default storage could be written this way:

```
-- Important! Do not try to execute on live system
-- without first changing the quad storage and quad map pattern names!

SPARQL
create virtrdf:DefaultQuadMap as
graph rdfdf:default-iid-nonblank (DB.DBA.RDF_QUAD.G)
subject rdfdf:default-iid (DB.DBA.RDF_QUAD.S)
predicate rdfdf:default-iid-nonblank (DB.DBA.RDF_QUAD.P)
object rdfdf:default (DB.DBA.RDF_QUAD.O)

create quad storage virtrdf:DefaultQuadStorage
FROM DB.DBA.RDF_QUAD as physical_quad
FROM DB.DBA.RDF_OBJ as physical_obj text xml literal RO_DIGEST of (physical_quad.O)
WHERE (^{physical_quad.}^.O = ^{physical_obj.}^.RO_DIGEST)
  {
    create virtrdf:DefaultQuadMap as
      graph rdfdf:default-iid-nonblank (physical_quad.G)
      subject rdfdf:default-iid (physical_quad.S)
      predicate rdfdf:default-iid-nonblank (physical_quad.P)
      object rdfdf:default (physical_quad.O) .
  }
;
```

The reference to the free-text index is extended by clause *of (physical_quad.O)* . This means that the free-text on *DB.DBA.RDF_OBJ.RO_DIGEST* will be used when the object value comes from *physical_quad.O* as if *physical_quad.O* were indexed itself. If a SPARQL query invokes *virtrdf:DefaultQuadMap* but contains no free-text criteria then only *DB.DBA.RDF_QUAD* appears in the final SQL statement and no join with *DB.DBA.RDF_OBJ* is made. Adding a free-text predicate will add *DB.DBA.RDF_OBJ* to the list of source tables and a join condition for *DB.DBA.RDF_QUAD.O* and *DB.DBA.RDF_OBJ.RO_DIGEST* ; and it will add *contains (RO_DIGEST, ...)* predicate, rather than *contains (O, ...)* . As a result, "you pay only for what you use": adding free-text index to the declaration does not add tables to the query unless the index is actually used.

Boolean functions `bif:contains` and `bif:xcontains` are used for objects that come from Linked Data Views as well as for regular "physical" triples. Every function takes two arguments and returns a boolean value. The first argument is an local variable. The argument variable should be used as an object field in the group pattern where the filter condition is placed. Moreover, the occurrence of the variable in an object field should be placed *before* the filter. If there are many occurrences of the variable in object fields then the free-text search is associated with the rightmost occurrence that is still to the left of the filter. The triple pattern that contains the rightmost occurrence is called the "intake" of the free-text search. When the SPARQL compiler chooses the appropriate quad map patterns that may generate data matching the intake triple pattern, it skips quad map patterns that have no declared free-text indexes, because nothing can be found by free-text search in data that have no free-text index. Every quad map pattern that has a free-text pattern will ultimately produce an invocation of the SQL contains or xcontains predicate, so the final result of a free-text search may be a union of free-text searches from different quad map patterns.

The described logic is important only in very complicated cases, whereas simple queries are self-evident:

```
SELECT * FROM <my-dav-graph>
WHERE {
    ?resource a mydav:resource ;
        mydav:resource-content ?text .
    FILTER (bif:contains (?text, "hello and world")) }
```

or, more succinctly,

```
SELECT * FROM <my-dav-graph>
WHERE {
    ?resource a mydav:resource ;
        mydav:resource-content ?text .
    ?text bif:contains "hello and world" . }
```

Example Using Score

```
SQL>
SPARQL
SELECT *
WHERE
  {
    ?s ?p ?o .
    ?o bif:contains 'NEW AND YORK'
    OPTION (score ?sc) .
  }
ORDER BY DESC (?sc)
LIMIT 10

s                                                                    p
ANY                                                                  ANY
_____________________________________________________________________________

http://dbpedia.org/resource/New_York%2C_New_York_%28disambiguation%29    http://www.w3.org/2000/01/rdf-sc
http://dbpedia.org/resource/New_York%2C_New_York_%28disambiguation%29    http://dbpedia.org/property/abst
http://newyorkjobs.wordpress.com/2006/07/10/new-york-jobs-71006          http://purl.org/dc/elements/1.1/
http://dbpedia.org/resource/Global_Communication                         http://dbpedia.org/property/cont
http://dbpedia.org/resource/New_York_%28disambiguation%29                http://www.w3.org/2000/01/rdf-sc
http://dbpedia.org/resource/New_York_%28disambiguation%29                http://dbpedia.org/property/abst
http://dbpedia.org/resource/New_York_%28disambiguation%29                http://www.w3.org/2000/01/rdf-sc
http://dbpedia.org/resource/New_York_%28disambiguation%29                http://dbpedia.org/property/abst
http://dbpedia.org/resource/New_York_College                             http://www.w3.org/2000/01/rdf-sc
http://dbpedia.org/resource/New_York_College                             http://dbpedia.org/property/abst
No. of rows in result: 10
```

16.3.2. SPARUL -- an Update Language For RDF Graphs

Introduction

Starting with version 5.0, Virtuoso supports the SPARQL/Update extension to SPARQL. This is sufficient for most of routine data manipulation operations. If the *SPARQL_UPDATE* role is granted to user *SPARQL* user then data manipulation statements may be executed via the SPARQL web service endpoint as well as by data querying.

Manage RDF Storage

Two functions allow the user to alter RDF storage by inserting or deleting all triples listed in some vector. Both functions receive the IRI of the graph that should be altered and a vector of triples that should be added or removed. The graph IRI can be either an IRI ID or a string. The third optional argument controls the transactional behavior - the parameter value is passed to the `log_enable` function. The return values of these functions are not defined and should not be used by applications.

```
create function DB.DBA.RDF_INSERT_TRIPLES (in graph_iri any, in triples any, in log_mode integer := null)
create function DB.DBA.RDF_DELETE_TRIPLES (in graph_iri any, in triples any, in log_mode integer := null)
```

Simple operations may be faster if written as low-level SQL code instead of using SPARUL. The use of SPARQL DELETE is unnecessary in cases where the better alternative is for the application to delete from RDF_QUAD using simple SQL filters like:

```
DELETE FROM DB.DBA.RDF_QUAD
WHERE G = DB.DBA.RDF_MAKE_IID_OF_QNAME (
    'http://local.virt/DAV/sparql_demo/data/data-xml/source-simple2/source-data-01.rdf' );
```

On the other hand, simple filters does not work when the search criteria refer to triples that are affected by the modification. Consider a function that deletes all triples whose subjects are nodes of type 'http://xmlns.com/foaf/0.1/Person'. Type information is stored in triples that will be deleted, so the simplest function is something like this:

```
create procedure DELETE_PERSONAL_DATA (in foaf_graph varchar)
{
  declare pdata_dict, pdata_array any;
-- Step 1: select everything that should be deleted
  pdata_dict := ((
      sparql construct { ?s ?p ?o }
      WHERE { graph ?:foaf_graph {
              ?s ?p ?o . ?s rdf:type <http://xmlns.com/foaf/0.1/Person>
            } }
      ));
-- Step 2: delete all found triples
  pdata_array := dict_list_keys (pdata_dict, 1);
  RDF_DELETE_TRIPLES (foaf_graph, pdata_array);
};

DELETE_PERSONAL_DATA (
  'http://local.virt/DAV/sparql_demo/data/data-xml/source-simple2/source-data-01.rdf' );
```

From Virtuoso 5.0 onwards, applications can use SPARUL to do the same in a more convenient way:

```
create procedure DELETE_PERSONAL_DATA (in foaf_graph varchar)
{
  sparql delete { ?s ?p ?o }
      WHERE { graph ?:foaf_graph {
              ?s ?p ?o . ?s rdf:type <http://xmlns.com/foaf/0.1/Person>
            } }
};
```

Examples

Example for changing the graph

The graph to be changed may be specified by an option preceding of query, instead of being specified in the 'insert into graph' clause.

```
SQL>SPARQL DEFINE input:default-graph-uri <http://mygraph.com>
INSERT INTO <http://mygraph.com> { <http://example.com/dataspace/Kingsley#this> <http://www.w3.org/1999/0
callret-0
VARCHAR
_______________________________________________________________________________

Insert into <http://mygraph.com>, 1 triples -- done

1 Rows. -- 20 msec.
```

Example for delete graph equivalence

The following two statements are equivalent but the latter may work faster, especially if there are many Linked Data Views in the system or if the graph in question contains triples from Linked Data Views. Note that neither of these two statements affects data coming from Linked Data Views.

```
SQL> SPARQL DELETE FROM GRAPH <http://mygraph.com> { ?s ?p ?o } FROM <http://mygraph> WHERE { ?s ?p ?o };
callret-0
VARCHAR
_______________________________________________________________________________

Delete from <http://mygraph.com>, 1 triples -- done

1 Rows. -- 10 msec.

SQL> SPARQL CLEAR GRAPH <http://mygraph.com>;
callret-0
VARCHAR
_______________________________________________________________

Clear <http://mygraph.com> -- done

1 Rows. -- 10 msec.
```

DROP GRAPH is equivalent of CLEAR GRAPH. It requires initially the graph to be created explicitly.

Note: All SPARQL commands should work via SPARUL (i.e. executed from the /sparql endpoint) as soon as "SPARQL" user has "SPARQL_UPDATE" privilege.

Assume the following sequence of commands to be executed from the /sparql endpoint:

- Create explicitly a graph:

  ```
  CREATE GRAPH <qq>

  callret-0
  Create graph <qq> -- done
  ```
- If you don't know whether the graph is created explicitly or not, use

  DROP SILENT GRAPH

  :

  ```
  DROP SILENT     GRAPH <qq>

  callret-0
  Drop silent graph <qq> -- done
  ```
- If you know the graph is created explicitly, use

  DROP GRAPH

  :

  ```
  DROP GRAPH <qq>

  callret-0
  Drop graph <qq> -- done
  ```

Example for deleting all triples for given subject

The following statement deletes all records with <http://example.com/dataspace/Kingsley#this> as the subject:

```
SQL>SPARQL
DELETE FROM GRAPH <http://mygraph.com> { ?s ?p ?o }
FROM <http://mygraph.com>
WHERE { ?s ?p ?o . filter ( ?s = <http://example.com/dataspace/Kingsley#this>) };
callret-0
```

```
VARCHAR
_______________________________________________________________________________

Delete from <http://mygraph.com>, 1 triples -- done

1 Rows. -- 10 msec.
```

Alternatively, the statement can be written in this way:

```
SQL>SPARQL
DELETE FROM GRAPH <http://mygraph.com> { <http://example.com/dataspace/Kingsley#this> ?p ?o }
FROM <http://mygraph.com>
WHERE { <http://example.com/dataspace/Kingsley#this> ?p ?o };
callret-0
VARCHAR
_______________________________________________________________________________

Delete from <http://mygraph.com>, 1 triples -- done

1 Rows. -- 10 msec.
```

Example for INSERT statements equivalent

Keywords 'insert in' and 'insert into' are interchangeable in Virtuoso for backward compatibility, but the SPARUL specification lists only 'insert into'. For example, the statements below are equivalent:

```
SQL>SPARQL INSERT INTO GRAPH <http://mygraph.com> {  <http://example.com/dataspace/Kingsley#this>
                                                     <http://rdfs.org/sioc/ns#id>
                                                     <Kingsley> };
callret-0
VARCHAR
_______________________________________________________________________________

Insert into <http://mygraph.com>, 1 triples -- done

1 Rows. -- 0 msec.
SQL>SPARQL INSERT INTO GRAPH <http://mygraph.com> {  <http://example.com/dataspace/Caroline#this>
                                                     <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
                                                     <http://rdfs.org/sioc/ns#User> };
callret-0
VARCHAR
_______________________________________________________________________________

Insert into <http://mygraph.com>, 1 triples -- done

1 Rows. -- 0 msec.

-- and

SQL>SPARQL INSERT IN GRAPH <http://mygraph.com> {  <http://example.com/dataspace/Kingsley#this>
                                                   <http://rdfs.org/sioc/ns#id>
                                                   <Kingsley> };
callret-0
VARCHAR
_______________________________________________________________________________

Insert into <http://mygraph.com>, 1 triples -- done

1 Rows. -- 10 msec.
SQL>SPARQL INSERT IN GRAPH <http://mygraph.com> {  <http://example.com/dataspace/Caroline#this>
                                                   <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
                                                   <http://rdfs.org/sioc/ns#User> };
callret-0
VARCHAR
_______________________________________________________________________

Insert into <http://mygraph.com>, 1 triples -- done

1 Rows. -- 0 msec.
```

Example for various expressions usage

It is possible to use various expressions to calculate fields of new triples. This is very convenient, even if not a part of the original specification.

```
SQL>SPARQL INSERT INTO GRAPH <http://mygraph.com> { ?s <http://rdfs.org/sioc/ns#id> `iri (bif:concat (str
WHERE { ?s <http://rdfs.org/sioc/ns#id> ?o };
callret-0
VARCHAR
_______________________________________________________________________________

Insert into <http://mygraph.com>, 4 triples -- done

1 Rows. -- 0 msec.
```

Example for operator IN usage

The example shows how to find which predicate/object pairs the following subjects have in common and count the occurances:

```
http://dbpedia.org/resource/Climate_change
http://dbpedia.org/resource/Disaster_risk_reduction
http://dbpedia.org/resource/Tanzania
http://dbpedia.org/resource/Capacity_building
http://dbpedia.org/resource/Poverty
http://dbpedia.org/resource/Construction
http://dbpedia.org/resource/Vulnerability
http://dbpedia.org/resource/Mount_Kilimanjaro
http://dbpedia.org/resource/Social_vulnerability
```

The following query returns the desired results:

```
SPARQL
SELECT ?s1 ?s2 COUNT (1)
WHERE
  {
    ?s1 ?p ?o .
    FILTER (?s1 IN (<http://dbpedia.org/resource/Climate_change>,
    <http://dbpedia.org/resource/Disaster_risk_reduction>,
    <http://dbpedia.org/resource/Tanzania>,
    <http://dbpedia.org/resource/Capacity_building>,
    <http://dbpedia.org/resource/Poverty>,
    <http://dbpedia.org/resource/Construction>,
    <http://dbpedia.org/resource/Vulnerability>,
    <http://dbpedia.org/resource/Mount_Kilimanjaro>,
    <http://dbpedia.org/resource/Social_vulnerability> ))
    ?s2 ?p ?o .
    FILTER (?s2 IN (<http://dbpedia.org/resource/Climate_change>,
    <http://dbpedia.org/resource/Disaster_risk_reduction>,
    <http://dbpedia.org/resource/Tanzania>,
    <http://dbpedia.org/resource/Capacity_building>,
    <http://dbpedia.org/resource/Poverty>,
    <http://dbpedia.org/resource/Construction>,
    <http://dbpedia.org/resource/Vulnerability>,
    <http://dbpedia.org/resource/Mount_Kilimanjaro>,
    <http://dbpedia.org/resource/Social_vulnerability> ))
    FILTER (?s1 != ?s2)
    FILTER (str(?s1) < str (?s2))
  }
LIMIT 20
```

The result of executing the query:

```
s1                                          s2                                                cal
http://dbpedia.org/resource/Climate_change       http://dbpedia.org/resource/Tanzania                 2
http://dbpedia.org/resource/Social_vulnerability http://dbpedia.org/resource/Vulnerability            1
http://dbpedia.org/resource/Mount_Kilimanjaro    http://dbpedia.org/resource/Poverty                  1
http://dbpedia.org/resource/Mount_Kilimanjaro    http://dbpedia.org/resource/Tanzania                 3
http://dbpedia.org/resource/Capacity_building    http://dbpedia.org/resource/Disaster_risk_reduction  1
http://dbpedia.org/resource/Poverty              http://dbpedia.org/resource/Tanzania                 1
```

You can also find live demo query results here

See Also:

Example usage of IN operator for retrieving all triples for each entity.

Example for Modify used as Update

'Modify graph' may be used as a form of 'update' operation.

```
-- update a graph with insert scoped on the same graph
SQL>SPARQL
MODIFY GRAPH <http://mygraph.com>
DELETE { ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?o }
INSERT { ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type1> ?o }
FROM <http://mygraph.com>
WHERE { ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?o };

-- update a graph with insert scoped on another graph
SQL>SPARQL
MODIFY GRAPH <http://mygraph.com>
DELETE { ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?o }
INSERT { ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type1> ?o }
FROM <http://example.com>
WHERE { ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?o };

-- update a graph with insert scoped over the whole RDF Quad Store
SQL>SPARQL
MODIFY GRAPH <http://mygraph.com>
DELETE { ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?o }
INSERT { ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type1> ?o }
WHERE { ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?o };

-- update a graph with delete of particular tripple
SQL>SPARQL
DELETE FROM GRAPH <http://mygraph.com> { <http://example.com/dataspace/Caroline#this> <http://www.w3.org/
```

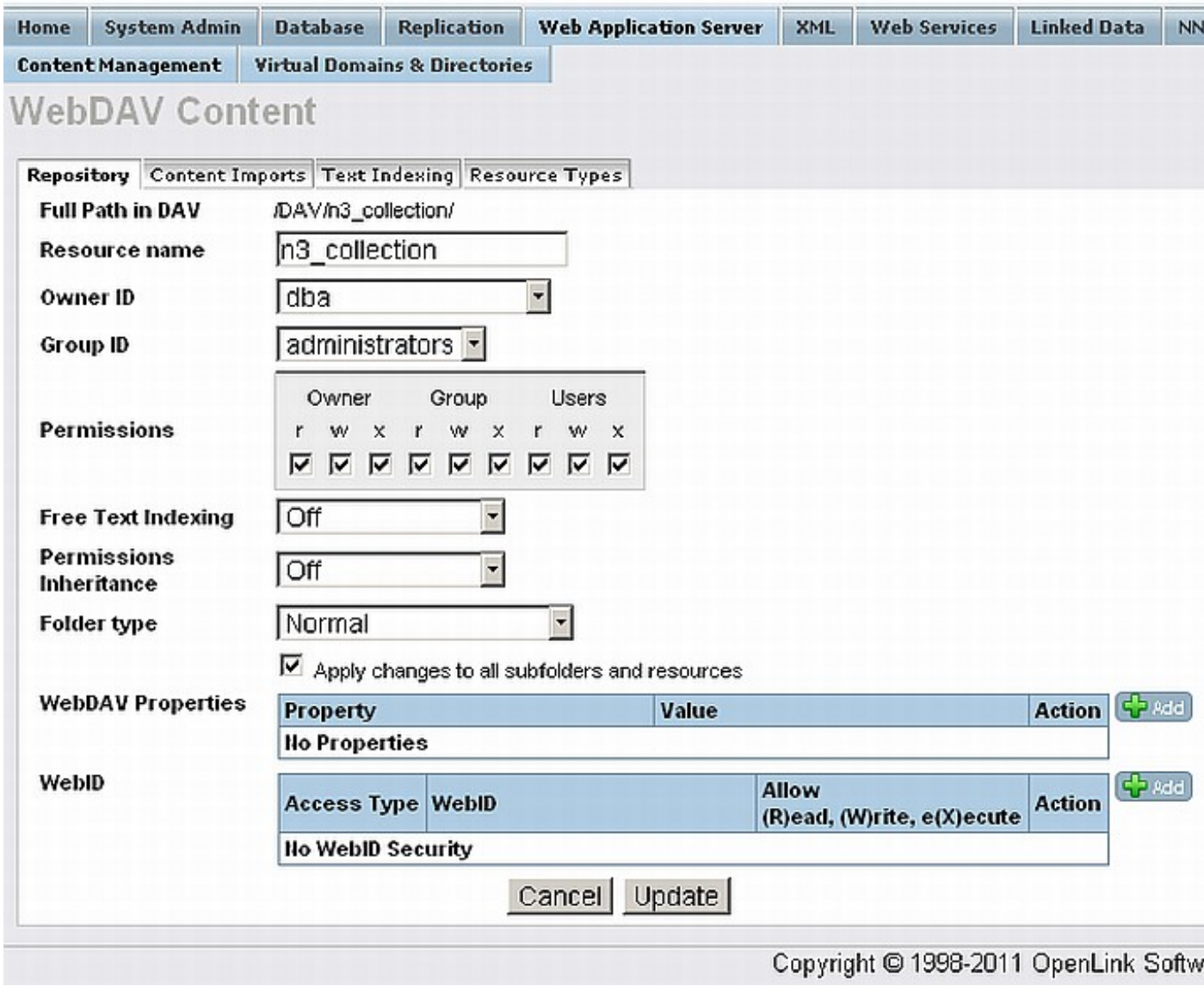
Example for generating RDF information resource URI

The RDF information resource URI can be generated via a string expression.

1. Suppose there is a sample file kidehen.n3:

```
<http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#this> <http://www.w3.org/1999/0
<http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#this> <http://www.w3.org/2000/0
<http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#this> <http://rdfs.org/sioc/ns#
```

2. Using Conductor UI go to Web Application Server and create folder, for ex. with name "n3_collection"
3. Upload the "n3_collection" folder the file kidehen.n3
4. Click properties for the folder "n3_collection" and set to be public readable in the Permissions section
5. Check "Apply changes to all subfolders and resources".
6. Finally Click "Update"
7. **Figure 16.44. Generating RDF information resource URI**



8. To clear the graph execute:

```
SQL>SPARQL CLEAR GRAPH <http://mygraph.com>;
callret-0
VARCHAR
_______________________________________________________________________

Clear <http://mygraph.com> -- done

1 Rows. -- 10 msec.
```

9. To load the kidehen.n3 file execute:

```
SQL>SPARQL
load bif:concat ("http://", bif:registry_get("URIQADefaultHost"), "/DAV/n3_collection/kidehen.n3")
INTO GRAPH <http://mygraph.com>;
callret-0
VARCHAR
_______________________________________________________________________________

Load <http://example.com/DAV/n3_collection/kidehen.n3> into graph <http://mygraph.com> -- done

1 Rows. -- 30 msec.
```

10. In order to check the new inserted triples execute:

```
SQL>SPARQL
SELECT *
FROM <http://mygraph.com>
WHERE
```

```
  {
    ?s ?p ?o
  }
;
s                                                                    p
VARCHAR                                                              VARCHAR
_____________________________________________________________________________

http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#this    http://www.w3.org/1999/0
http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#this    http://rdfs.org/sioc/ns#
http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#this    http://www.w3.org/2000/0

3 Rows. -- 10 msec.
```

Example for operations over a web service endpoint

Several operations can be sent to a web service endpoint as a single statement and executed in sequence.

```
SQL>SPARQL
INSERT IN GRAPH <http://mygraph.com> { <http://example.com/dataspace/Kingsley#this>
                                       <http://rdfs.org/sioc/ns#id>
                                       <Kingsley> }
INSERT INTO GRAPH <http://mygraph.com> { <http://example.com/dataspace/Caroline#this>
                                         <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
                                         <http://rdfs.org/sioc/ns#User> }
INSERT INTO GRAPH <http://mygraph.com> { ?s <http://rdfs.org/sioc/ns#id> `iri (bif:concat (str (?o), "Ide
WHERE { ?s <http://rdfs.org/sioc/ns#id> ?o };
callret-0
VARCHAR
_____________________________________________________________________________

Insert into <http://mygraph.com>, 1 triples -- done
Insert into <http://mygraph.com>, 1 triples -- done
Insert into <http://mygraph.com>, 8 triples -- done
Commit -- done

1 Rows. -- 10 msec.

SQL>SPARQL
MODIFY GRAPH <http://mygraph.com>
DELETE { ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?o }
INSERT { ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type1> ?o }
FROM <http://mygraph.com>
WHERE { ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?o };

SQL>DELETE FROM GRAPH <http://mygraph.com> { <http://example.com/dataspace/Caroline#this> <http://www.w3.

SQL>SPARQL
load bif:concat ("http://", bif:registry_get("URIQADefaultHost"), "/DAV/n3_collection/kidehen.n3") INTO G
```

See Also:

Examples for Modify used as Update

Example for Dropping graph

When handling very large RDF data collections (e.g. 600 million triples) loaded into Virtuoso server as a single graph, the fastest operation to drop the graph is:

```
SQL>SPARQL CLEAR GRAPH <http://mygraph.com>;
callret-0
VARCHAR
_____________________________________________________________________________

Clear <http://mygraph.com> -- done

1 Rows. -- 10 msec.
```

The operation can be speeded up by executing log_enable (0) or even log_enable (2) beforehand, and log_enable(1) after it completes.

Example for testing Graph Equality

The procedure below keeps simple cases of graphs with bnodes:

1. First it compares all triples without bnodes
2. Then it iteratively establishes equivalences between bnodes that are directly and unambiguously connected to equivalent vertexes by identical predicates.

```
-- Fast Approximate RDF Graph Equivalence Test
-- (C) 2009-2020 OpenLink Software
-- License: GNU General Public License (only version 2 of the license).
-- No warranty, even implied warranty

-- This compares the content of triple dictionaries \c dict1 and \c dict2,
-- returns NULL if no difference found (with bnode equivalence in mind),
-- returns description of a difference otherwise.
-- The function is experimental (note suffix _EXP), so no accurate QA is made.
-- Some version of the function may be inserted later in OpenLink Virtuoso Server under some different na
create function DB.DBA.RDF_TRIPLE_DICTS_DIFFER_EXP (
  in dict1 any, --- Triple dictionary, traditional, (vectors of S, P, O are keys, any non-nulls are value
  in dict2 any, --- Second triple dictionary, like to \c dict1
  in accuracy integer,  --- Accuracy, 0 if no bnodes expected, 1 if "convenient" trees with intermediate
  in equiv_map any := null, --- If specified then it contain mapping from IRI_IDs of bnodes of \c dict1 t
-- It can be extended during the run so use dict_duplicate() before call if needed.
  in equiv_rev any := null --- If specified then it is an inverted dictionary of \c equiv_map (this time
  )
{
  declare dict_size1, dict_size2 integer;
  declare old_dirt_level, dirt_level integer;
  declare ctr, tailctr, sp_made_new_equiv integer;
  declare array1, array2, dict2_sp, dict1_op, dict2_op, array1_op any;
  dict_size1 := dict_size (dict1);
  dict_size2 := dict_size (dict2);
  dict2 := dict_duplicate (dict2);
  if (dict_size1 <> dict_size2)
    return 'Sizes differ';
  if (equiv_map is null)
    {
      equiv_map := dict_new (dict_size1);
      equiv_rev := dict_new (dict_size1);
    }
  old_dirt_level := dict_size1 - dict_size (equiv_map);
  array1 := dict_list_keys (dict1, 0);
next_loop:
-- Step 1: removing triples with all three items matched
  ctr := dict_size1-1;
  while (ctr >= 0)
    {
      declare s_in_1, o_in_1, s_in_2, o_in_2, triple_in_2 any;
      s_in_1 := array1[ctr][0];
      o_in_1 := array1[ctr][2];
      if (is_bnode_iri_id (s_in_1))
        {
          s_in_2 := dict_get (equiv_map, s_in_1, null);
          if (s_in_2 is null)
            goto next_full_eq_check;
        }
      else
        s_in_2 := s_in_1;
      if (is_bnode_iri_id (o_in_1))
        {
          o_in_2 := dict_get (equiv_map, o_in_1, null);
          if (o_in_2 is null)
            goto next_full_eq_check;
        }
      else
        o_in_2 := o_in_1;
      triple_in_2 := vector (s_in_2, array1[ctr][1], o_in_2);
      if (dict_get (dict2, triple_in_2, null) is null)
        return vector (array1[ctr], ' is in first, ', triple_in_2, ' is missing in second');
      dict_remove (dict2, triple_in_2);
      if (ctr < dict_size1-1)
```

```
        array1[ctr] := array1[dict_size1-1];
      dict_size1 := dict_size1-1;
next_full_eq_check:
      ctr := ctr-1;
    }
-- Step 1 end, garbage truncated:
  if ((0 = dict_size1) or (0 = accuracy))
    return null;
  if (dict_size1 < length (array1))
    array1 := subseq (array1, 0, dict_size1);
  if (dict_size (dict2) <> dict_size1)
    signal ('OBLOM', 'Internal error: sizes of graphs suddenly differ');
-- Step 2: establishing equivs between not-yet-coupled bnodes that are values of functional predicates of
  sp_made_new_equiv := 0;
  dict2_sp := dict_new (dict_size1);
  array2 := dict_list_keys (dict2, 0);
  for (ctr := dict_size1-1; ctr >= 0; ctr := ctr-1)
    {
      declare sp2, o2, prev_uniq_o2 any;
      sp2 := vector (array2[ctr][0], array2[ctr][1]);
      prev_uniq_o2 := dict_get (dict2_sp, sp2, null);
      if (prev_uniq_o2 is null)
        {
          o2 := array2[ctr][2];
          if (is_bnode_iri_id (o2))
            dict_put (dict2_sp, sp2, o2);
          else
            dict_put (dict2_sp, sp2, #i0);
        }
      else if (prev_uniq_o2 <> #i0)
        dict_put (dict2_sp, sp2, #i0);
    }
  rowvector_subj_sort (array1, 0, 1);
  rowvector_subj_sort (array1, 1, 1);
  rowvector_subj_sort (array2, 1, 1);
  ctr := 0;
  while (ctr < dict_size1)
    {
      declare s_in_1, o_in_1, s_in_2, o_in_2, o_in_dict2_sp, o_in_dict2_sp_in_1 any;
      tailctr := ctr+1;
      if (array1[ctr][1] <> array2[ctr][1])
        {
          if (array1[ctr][1] > array2[ctr][1])
            return vector ('Cardinality of predicate ', array2[ctr][1], ' is greater in second than in fi
          else
            return vector ('Cardinality of predicate ', array1[ctr][1], ' is greater in first than in sec
        }
      while ((tailctr < dict_size1) and
        (array1[tailctr][0] = array1[ctr][0]) and
        (array1[tailctr][1] = array1[ctr][1]) )
        tailctr := tailctr+1;
      if ((tailctr - ctr) > 1)
        goto next_sp_check;
      o_in_1 := array1[ctr][2];
      if (not is_bnode_iri_id (o_in_1))
        goto next_sp_check;
      o_in_2 := dict_get (equiv_map, o_in_1, null);
      if (o_in_2 is not null)
        goto next_sp_check;
      s_in_1 := array1[ctr][0];
      if (is_bnode_iri_id (s_in_1))
        {
          s_in_2 := dict_get (equiv_map, s_in_1, null);
          if (s_in_2 is null)
            goto next_sp_check;
        }
      else
        s_in_2 := s_in_1;
      o_in_dict2_sp := dict_get (dict2_sp, vector (s_in_2, array1[ctr][1]), null);
      if (o_in_dict2_sp is null)
        return vector (vector (s_in_1, array1[ctr][1], o_in_1), ' is unique SP in first, ', vector (s_in_
      if (o_in_dict2_sp = #i0)
        return vector (vector (s_in_1, array1[ctr][1], o_in_1), ' is unique SP in first, ', vector (s_in_
```

```
      o_in_dict2_sp_in_1 := dict_get (equiv_rev, o_in_dict2_sp, null);
      if (o_in_dict2_sp_in_1 is not null)
        {
          if (o_in_dict2_sp_in_1 = o_in_1)
            goto next_sp_check;
          return vector (vector (s_in_1, array1[ctr][1], o_in_1), ' is unique SP in first, ', vector (s_i
        }
      dict_put (equiv_map, o_in_1, o_in_dict2_sp);
      dict_put (equiv_rev, o_in_dict2_sp, o_in_1);
      sp_made_new_equiv := sp_made_new_equiv + 1;
next_sp_check:
      ctr := tailctr;
    }
  dict_list_keys (dict2_sp, 2);
-- Step 2 end
  if (sp_made_new_equiv * 10 > dict_size1)
    goto next_loop; -- If dictionary is noticeably extended then it's worth to remove more triples before
-- Step 3: establishing equivs between not-yet-coupled bnodes that are subjects of inverse functional pro
  dict1_op := dict_new (dict_size1);
  for (ctr := dict_size1-1; ctr >= 0; ctr := ctr-1)
    {
      declare op1, s1, prev_uniq_s1 any;
      op1 := vector (array1[ctr][2], array1[ctr][1]);
      prev_uniq_s1 := dict_get (dict1_op, op1, null);
      if (prev_uniq_s1 is null)
        {
          s1 := array1[ctr][0];
          if (is_bnode_iri_id (s1))
            dict_put (dict1_op, op1, s1);
          else
            dict_put (dict1_op, op1, #i0);
        }
      else if (prev_uniq_s1 <> #i0)
        dict_put (dict1_op, op1, #i0);
    }
  array1_op := dict_to_vector (dict1_op, 2);
  dict2_op := dict_new (dict_size1);
  for (ctr := dict_size1-1; ctr >= 0; ctr := ctr-1)
    {
      declare op2, s2, prev_uniq_s2 any;
      op2 := vector (array2[ctr][2], array2[ctr][1]);
      prev_uniq_s2 := dict_get (dict2_op, op2, null);
      if (prev_uniq_s2 is null)
        {
          s2 := array2[ctr][0];
          if (is_bnode_iri_id (s2))
            dict_put (dict2_op, op2, s2);
          else
            dict_put (dict2_op, op2, #i0);
        }
      else if (prev_uniq_s2 <> #i0)
        dict_put (dict2_op, op2, #i0);
    }
  ctr := length (array1_op) - 2;
  while (ctr >= 0)
    {
      declare o_in_1, s_in_1, o_in_2, s_in_2, s_in_dict2_op, s_in_dict2_op_in_1 any;
      s_in_1 := array1_op[ctr+1];
      if (not is_bnode_iri_id (s_in_1))
        goto next_op_check;
      s_in_2 := dict_get (equiv_map, s_in_1, null);
      if (s_in_2 is not null)
        goto next_op_check;
      o_in_1 := array1_op[ctr][0];
      if (is_bnode_iri_id (o_in_1))
        {
          o_in_2 := dict_get (equiv_map, o_in_1, null);
          if (o_in_2 is null)
            goto next_op_check;
        }
      else
        o_in_2 := o_in_1;
      s_in_dict2_op := dict_get (dict2_op, vector (o_in_2, array1_op[ctr][1]), null);
```

```
      if (s_in_dict2_op is null)
        return vector (vector (s_in_1, array1_op[ctr][1], o_in_1), ' is unique OP in first, ', vector (o_
      if (s_in_dict2_op = #i0)
        return vector (vector (s_in_1, array1_op[ctr][1], o_in_1), ' is unique OP in first, ', vector (o_
      s_in_dict2_op_in_1 := dict_get (equiv_rev, s_in_dict2_op, null);
      if (s_in_dict2_op_in_1 is not null)
        {
          if (s_in_dict2_op_in_1 = s_in_1)
            goto next_op_check;
          return vector (vector (s_in_1, array1_op[ctr][1], o_in_1), ' is unique OP in first, ', vector (
        }
      dict_put (equiv_map, s_in_1, s_in_dict2_op);
      dict_put (equiv_rev, s_in_dict2_op, s_in_1);
next_op_check:
      ctr := ctr - 2;
    }
  dict_list_keys (dict2_op, 2);
-- Step 3 end
  dirt_level := dict_size1 - dict_size (equiv_map);
  if (dirt_level >= old_dirt_level)
    return vector (vector (array1[0][0], array1[0][1], array1[0][2]), ' has no matches in second with the
  old_dirt_level := dirt_level;
  goto next_loop;
}
;

create function DB.DBA.RDF_GRAPHS_DIFFER_EXP (in g1_uri varchar, in g2_uri varchar, in accuracy integer)
{
  return DB.DBA.RDF_TRIPLE_DICTS_DIFFER_EXP (
    (sparql define output:valmode "LONG" construct { ?s ?p ?o } where { graph `iri(?:g1_uri)` { ?s ?p ?o
    (sparql define output:valmode "LONG" construct { ?s ?p ?o } where { graph `iri(?:g2_uri)` { ?s ?p ?o
    accuracy );
}
;

-- The rest of file contains some minimal tests.

set verbose off;
set banner off;
set types off;

create function DB.DBA.DICT_EXTEND_WITH_KEYS (in dict any, in keys any)
{
  if (dict is null)
    dict := dict_new (length (keys));
  foreach (any k in keys) do
    dict_put (dict, k, 1);
  return dict;
}
;

create function DB.DBA.TEST_RDF_TRIPLE_DICTS_DIFFER_EXP (in title varchar, in should_differ integer, in v
{
  declare d1, d2, eqm, eqr, differ_status any;
  d1 := DB.DBA.DICT_EXTEND_WITH_KEYS (null, v1);
  d2 := DB.DBA.DICT_EXTEND_WITH_KEYS (null, v2);
  eqm := dict_new (10);
  eqr := dict_new (10);
  dbg_obj_princ ('===== ' || title);
  differ_status := DB.DBA.RDF_TRIPLE_DICTS_DIFFER_EXP (d1, d2, accuracy, eqm, eqr);
  dbg_obj_princ ('Result: ', differ_status);
  if (0 < dict_size (eqm))
  dbg_obj_princ ('Equivalence map: ', dict_to_vector (eqm, 0));
  dbg_obj_princ ('Equivalence rev: ', dict_to_vector (eqr, 0));
  return sprintf ('%s: %s',
    case when (case when should_differ then equ (0, isnull (differ_status)) else isnull (differ_status) e
    title );
}
;

create function DB.DBA.TEST_RDF_GRAPHS_DIFFER_EXP (in title varchar, in should_differ integer, in g1_uri
{
  declare differ_status any;
```

```
  differ_status := DB.DBA.RDF_GRAPHS_DIFFER_EXP (g1_uri, g2_uri, accuracy);
  dbg_obj_princ ('Result: ', differ_status);
  return sprintf ('%s: %s',
    case when (case when should_differ then equ (0, isnull (differ_status)) else isnull (differ_status) e
    title );
}
;

select DB.DBA.TEST_RDF_TRIPLE_DICTS_DIFFER_EXP ( 'Identical graphs', 0,
  vector (
    vector (#i100, #i200, #i300),
    vector (#i100, #i200, 1) ),
  vector (
    vector (#i100, #i200, #i300),
    vector (#i100, #i200, 1) ),
  100
);

select DB.DBA.TEST_RDF_TRIPLE_DICTS_DIFFER_EXP ( 'Sizes differ', 1,
  vector (
    vector (#i100, #i200, #i300),
    vector (#i100, #i200, 1) ),
  vector (
    vector (#i100, #i200, #i300),
    vector (#i100, #i200, 1),
    vector (#i101, #i201, #i301) ),
  100
);

select DB.DBA.TEST_RDF_TRIPLE_DICTS_DIFFER_EXP ( 'Cardinality of a pred differ', 1,
  vector (
    vector (#i100, #i200, #ib300),
    vector (#i101, #i200, #ib302),
    vector (#i103, #i201, #ib304),
    vector (#ib109, #i200, #ib109) ),
  vector (
    vector (#i100, #i200, #ib301),
    vector (#i101, #i200, #ib303),
    vector (#i103, #i201, #ib305),
    vector (#ib109, #i201, #ib109) ),
  100
);

select DB.DBA.TEST_RDF_TRIPLE_DICTS_DIFFER_EXP ( 'Bnodes in O with unique SP (equiv)', 0,
  vector (
    vector (#i100, #i200, #i300),
    vector (#i100, #i201, #ib301),
    vector (#i101, #i201, #ib301),
    vector (#i102, #i202, #ib303),
    vector (#ib303, #i204, #i306),
    vector (#ib303, #i205, #ib305),
    vector (#i100, #i200, 1) ),
  vector (
    vector (#i100, #i200, #i300),
    vector (#i100, #i201, #ib302),
    vector (#i101, #i201, #ib302),
    vector (#i102, #i202, #ib304),
    vector (#ib304, #i204, #i306),
    vector (#ib304, #i205, #ib306),
    vector (#i100, #i200, 1) ),
  100
);

select DB.DBA.TEST_RDF_TRIPLE_DICTS_DIFFER_EXP ( 'Bnodes in O with unique SP (diff 1)', 1,
  vector (
    vector (#i100, #i200, #i300),
    vector (#i100, #i201, #ib301),
    vector (#i102, #i202, #ib303),
    vector (#ib303, #i204, #i306),
    vector (#ib303, #i205, #ib305),
    vector (#i100, #i200, 1) ),
  vector (
    vector (#i100, #i200, #i300),
```

```
    vector (#i100, #i201, #ib302),
    vector (#i102, #i202, #ib304),
    vector (#ib304, #i204, #i306),
    vector (#ib304, #i205, #i306),
    vector (#i100, #i200, 1) ),
  100
);

select DB.DBA.TEST_RDF_TRIPLE_DICTS_DIFFER_EXP ( 'Bnodes in O with unique SP (diff 2)', 1,
  vector (
    vector (#i100, #i200, #i300),
    vector (#i100, #i201, #ib301),
    vector (#i102, #i202, #ib303),
    vector (#ib303, #i204, #i306),
    vector (#ib303, #i205, #ib305),
    vector (#i100, #i200, 1) ),
  vector (
    vector (#i100, #i200, #i300),
    vector (#i100, #i201, #ib302),
    vector (#i102, #i202, #ib304),
    vector (#ib304, #i204, #i306),
    vector (#ib304, #i205, #ib304),
    vector (#i100, #i200, 1) ),
  100
);

select DB.DBA.TEST_RDF_TRIPLE_DICTS_DIFFER_EXP ( 'foaf-like-mix (equiv)', 0,
  vector (
    vector (#i100, #i200, #i300),
    vector (#i100, #i201, #ib301),
    vector (#i100, #i201, #ib303),
    vector (#i100, #i201, #ib305),
    vector (#i100, #i201, #ib307),
    vector (#ib301, #i202, 'Anna'),
    vector (#ib303, #i202, 'Anna'),
    vector (#ib305, #i202, 'Brigit'),
    vector (#ib307, #i202, 'Clara'),
    vector (#ib301, #i203, 'ann@ex.com'),
    vector (#ib303, #i203, 'ann@am.com'),
    vector (#ib305, #i203, 'root@ple.com'),
    vector (#ib307, #i203, 'root@ple.com') ),
  vector (
    vector (#i100, #i200, #i300),
    vector (#i100, #i201, #ib302),
    vector (#i100, #i201, #ib304),
    vector (#i100, #i201, #ib306),
    vector (#i100, #i201, #ib308),
    vector (#ib302, #i202, 'Anna'),
    vector (#ib304, #i202, 'Anna'),
    vector (#ib306, #i202, 'Brigit'),
    vector (#ib308, #i202, 'Clara'),
    vector (#ib302, #i203, 'ann@ex.com'),
    vector (#ib304, #i203, 'ann@am.com'),
    vector (#ib306, #i203, 'root@ple.com'),
    vector (#ib308, #i203, 'root@ple.com') ),
  100
);

select DB.DBA.TEST_RDF_TRIPLE_DICTS_DIFFER_EXP ( 'foaf-like-mix (swapped names)', 1,
  vector (
    vector (#i100, #i200, #i300),
    vector (#i100, #i201, #ib301),
    vector (#i100, #i201, #ib303),
    vector (#i100, #i201, #ib305),
    vector (#i100, #i201, #ib307),
    vector (#ib301, #i202, 'Anna'),
    vector (#ib303, #i202, 'Anna'),
    vector (#ib305, #i202, 'Brigit'),
    vector (#ib307, #i202, 'Clara'),
    vector (#ib301, #i203, 'ann@ex.com'),
    vector (#ib303, #i203, 'ann@am.com'),
    vector (#ib305, #i203, 'root@ple.com'),
    vector (#ib307, #i203, 'root@ple.com') ),
```

```
  vector (
    vector (#i100, #i200, #i300),
    vector (#i100, #i201, #ib302),
    vector (#i100, #i201, #ib304),
    vector (#i100, #i201, #ib306),
    vector (#i100, #i201, #ib308),
    vector (#ib302, #i202, 'Anna'),
    vector (#ib304, #i202, 'Brigit'),
    vector (#ib306, #i202, 'Anna'),
    vector (#ib308, #i202, 'Clara'),
    vector (#ib302, #i203, 'ann@ex.com'),
    vector (#ib304, #i203, 'ann@am.com'),
    vector (#ib306, #i203, 'root@ple.com'),
    vector (#ib308, #i203, 'root@ple.com') ),
  100
);

select DB.DBA.TEST_RDF_TRIPLE_DICTS_DIFFER_EXP ( 'foaf-like-mix (swapped names)', 1,
  vector (
    vector (#i100, #i200, #i300),
    vector (#i100, #i201, #ib301),
    vector (#i100, #i201, #ib303),
    vector (#i100, #i201, #ib305),
    vector (#i100, #i201, #ib307),
    vector (#ib301, #i202, 'Anna'),
    vector (#ib303, #i202, 'Anna'),
    vector (#ib305, #i202, 'Brigit'),
    vector (#ib307, #i202, 'Clara'),
    vector (#ib301, #i203, 'ann@ex.com'),
    vector (#ib303, #i203, 'ann@am.com'),
    vector (#ib305, #i203, 'root@ple.com'),
    vector (#ib307, #i203, 'root@ple.com') ),
  vector (
    vector (#i100, #i200, #i300),
    vector (#i100, #i201, #ib302),
    vector (#i100, #i201, #ib304),
    vector (#i100, #i201, #ib306),
    vector (#i100, #i201, #ib308),
    vector (#ib302, #i202, 'Anna'),
    vector (#ib304, #i202, 'Brigit'),
    vector (#ib306, #i202, 'Anna'),
    vector (#ib308, #i202, 'Clara'),
    vector (#ib302, #i203, 'ann@ex.com'),
    vector (#ib304, #i203, 'ann@am.com'),
    vector (#ib306, #i203, 'root@ple.com'),
    vector (#ib308, #i203, 'root@ple.com') ),
  100
);

select DB.DBA.TEST_RDF_TRIPLE_DICTS_DIFFER_EXP ( 'bnodes only (equiv that can not be proven)', 1,
  vector (
    vector (#ib101, #i200, #ib103),
    vector (#ib103, #i201, #ib101) ),
  vector (
    vector (#ib102, #i200, #ib104),
    vector (#ib104, #i201, #ib102) ),
  100
);

sparql clear graph <http://GraphCmp/One>;

TTLP ('@prefix foaf: <http://i-dont-remember-it> .
_:me
        a foaf:Person ;
        foaf:knows      [ foaf:nick "oerling" ; foaf:title "Mr." ; foaf:sha1 "abra" ] ;
        foaf:knows      [ foaf:nick "kidehen" ; foaf:title "Mr." ; foaf:sha1 "bra" ] ;
        foaf:knows      [ foaf:nick "aldo" ; foaf:title "Mr." ; foaf:sha1 "cada" ] .',
'', 'http://GraphCmp/One' );

sparql clear graph <http://GraphCmp/Two>;
TTLP ('@prefix foaf: <http://i-dont-remember-it> .
_:iv
        foaf:knows      [ foaf:title "Mr." ; foaf:sha1 "cada" ; foaf:nick "aldo" ] ;
```

```
        foaf:knows      [ foaf:sha1 "bra" ; foaf:title "Mr." ; foaf:nick "kidehen" ] ;
        foaf:knows      [ foaf:nick "oerling" ; foaf:sha1 "abra" ; foaf:title "Mr." ] ;
        a foaf:Person .',
'', 'http://GraphCmp/Two' );

select DB.DBA.TEST_RDF_GRAPHS_DIFFER_EXP ( 'nonexisting graphs (equiv, of course)', 0,
  'http://GraphCmp/NoSuch', 'http://GraphCmp/NoSuch',
  100 );

select DB.DBA.TEST_RDF_GRAPHS_DIFFER_EXP ( 'throughout test on foafs (equiv)', 0,
  'http://GraphCmp/One', 'http://GraphCmp/Two',
  100 );
```

Example for Adding triples to graph

```
SQL>SPARQL
INSERT INTO GRAPH <http://BookStore.com>
{ <http://www.dajobe.org/foaf.rdf#i> <http://purl.org/dc/elements/1.1/title>  "SPARQL and RDF" .
  <http://www.dajobe.org/foaf.rdf#i> <http://purl.org/dc/elements/1.1/date> <1999-01-01T00:00:00>.
  <http://www.w3.org/People/Berners-Lee/card#i> <http://purl.org/dc/elements/1.1/title> "Design notes" .
  <http://www.w3.org/People/Berners-Lee/card#i> <http://purl.org/dc/elements/1.1/date> <2001-01-01T00:00:
  <http://www.w3.org/People/Connolly/#me> <http://purl.org/dc/elements/1.1/title> "Fundamentals of Compil
  <http://www.w3.org/People/Connolly/#me> <http://purl.org/dc/elements/1.1/date> <2002-01-01T00:00:00>. }
callret-0
VARCHAR
_______________________________________________________________________

Insert into <http://BookStore.com>, 6 triples -- done

1 Rows. -- 0 msec.
```

Example for Updating triples from graph

A SPARQL/Update request that contains a triple to be deleted and a triple to be added (used here to correct a book title).

```
SQL>SPARQL
MODIFY GRAPH <http://BookStore.com>
DELETE
 { <http://www.w3.org/People/Connolly/#me>  <http://purl.org/dc/elements/1.1/title>  "Fundamentals of Com
INSERT
 { <http://www.w3.org/People/Connolly/#me>  <http://purl.org/dc/elements/1.1/title>  "Fundamentals" };
callret-0
VARCHAR
_______________________________________________________________________________

Modify <http://BookStore.com>, delete 1 and insert 1 triples -- done

1 Rows. -- 20 msec.
```

See Also:

Examples for Modify used as Update

Example for Deleting triples from graph

The example below has a request to delete all records of old books (dated before year 2000)

```
SQL>SPARQL
PREFIX dc:  <http://purl.org/dc/elements/1.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
DELETE FROM GRAPH <http://BookStore.com> { ?book ?p ?v }
WHERE
  { GRAPH  <http://BookStore.com>
   { ?book dc:date ?date
     FILTER ( xsd:dateTime(?date) < xsd:dateTime("2000-01-01T00:00:00")).
    ?book ?p ?v.
   }
  };
_______________________________________________________________________________

Delete from <http://BookStore.com>, 6 triples -- done
```

```
1 Rows. -- 10 msec.
```

Example for Copying triples from one graph to another

The next snippet copies records from one named graph to another based on a pattern:

```
SQL>SPARQL clear graph <http://BookStore.com>;
SQL>SPARQL clear graph <http://NewBookStore.com>;
SQL>SPARQL
insert in graph <http://BookStore.com>
  {
    <http://www.dajobe.org/foaf.rdf#i> <http://purl.org/dc/elements/1.1/date> <1999-04-01T00:00:00> .
    <http://www.w3.org/People/Berners-Lee/card#i> <http://purl.org/dc/elements/1.1/date> <1998-05-03T00:0
    <http://www.w3.org/People/Connolly/#me> <http://purl.org/dc/elements/1.1/date> <2001-02-08T00:00:00>
  };
SQL>SPARQL
PREFIX dc:  <http://purl.org/dc/elements/1.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
INSERT INTO GRAPH <http://NewBookStore.com> { ?book ?p ?v }
WHERE
  { GRAPH  <http://BookStore.com>
   { ?book dc:date ?date
     FILTER ( xsd:dateTime(?date) > xsd:dateTime("2000-01-01T00:00:00")).
     ?book ?p ?v.
   }
  };
callret-0
VARCHAR
_____________________________________________________________________________

Insert into <http://NewBookStore.com>, 6 triples -- done

1 Rows. -- 30 msec.
```

Example for Moving triples from one graph to another

This example moves records from one named graph to another named graph based on a pattern:

```
SQL>SPARQL
PREFIX dc:  <http://purl.org/dc/elements/1.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

INSERT INTO GRAPH <http://NewBookStore.com>
 { ?book ?p ?v }
WHERE
  { GRAPH  <http://BookStore.com>
     { ?book dc:date ?date .
       FILTER ( xsd:dateTime(?date) > xsd:dateTime("2000-01-01T00:00:00")).
       ?book ?p ?v.
     }
  };
_____________________________________________________________________________

Insert into <http://NewBookStore.com>, 6 triples -- done

1 Rows. -- 10 msec.

SQL>SPARQL
PREFIX dc:  <http://purl.org/dc/elements/1.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
DELETE FROM GRAPH <http://BookStore.com>
 { ?book ?p ?v }
WHERE
  { GRAPH  <http://BookStore.com>
      { ?book dc:date ?date .
        FILTER ( xsd:dateTime(?date) > xsd:dateTime("2000-01-01T00:00:00")).
        ?book ?p ?v.
      }
  };
_____________________________________________________________________________
```

```
Delete from <http://BookStore.com>, 3 triples -- done

1 Rows. -- 10 msec.
```

Example for BBC SPARQL Collection

```
## All programmes related to James Bond:
PREFIX po: <http://purl.org/ontology/po/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?uri ?label
WHERE
  {
    ?uri po:category
      <http://www.bbc.co.uk/programmes/people/bmFtZS9ib25kLCBqYW1lcyAobm8gcXVhbGlmaWVyKQ#person> ;
    rdfs:label ?label.
   }

## Find all Eastenders broadcasta after 2009-01-01,
## along with the broadcast version & type
PREFIX event: <http://purl.org/NET/c4dm/event.owl#>
PREFIX tl: <http://purl.org/NET/c4dm/timeline.owl#>
PREFIX po: <http://purl.org/ontology/po/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?version_type ?broadcast_start
WHERE
  {
    <http://www.bbc.co.uk/programmes/b006m86d#programme> po:episode ?episode .
    ?episode po:version ?version .
    ?version a ?version_type .
    ?broadcast po:broadcast_of ?version .
    ?broadcast event:time ?time .
    ?time tl:start ?broadcast_start .
    FILTER ( (?version_type != <http://purl.org/ontology/po/Version>)
    && (?broadcast_start > "2009-01-01T00:00:00Z"^^xsd:dateTime) )
  }

## Find all programmes that featured both the Foo Fighters and Al Green
PREFIX po: <http://purl.org/ontology/po/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX mo: <http://purl.org/ontology/mo/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX event: <http://purl.org/NET/c4dm/event.owl#>
PREFIX tl: <http://purl.org/NET/c4dm/timeline.owl#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
SELECT DISTINCT ?programme ?label
WHERE
  {
    ?event1 po:track ?track1 .
    ?track1 foaf:maker ?maker1 .
    ?maker1 owl:sameAs
       <http://www.bbc.co.uk/music/artists/67f66c07-6e61-4026-ade5-7e782fad3a5d#artist> .
    ?event2 po:track ?track2 .
    ?track2 foaf:maker ?maker2 .
    ?maker2 owl:sameAs
       <http://www.bbc.co.uk/music/artists/fb7272ba-f130-4f0a-934d-6eeea4c18c9a#artist> .
    ?event1 event:time ?t1 .
    ?event2 event:time ?t2 .
    ?t1 tl:timeline ?tl .
    ?t2 tl:timeline ?tl .
    ?version po:time ?t .
    ?t tl:timeline ?tl .
    ?programme po:version ?version .
    ?programme rdfs:label ?label .
  }

## Get short synopsis' of EastEnders episodes
PREFIX po: <http://purl.org/ontology/po/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?t ?o
WHERE
  {
    <http://www.bbc.co.uk/programmes/b006m86d#programme> po:episode ?e .
```

```
    ?e a po:Episode .
    ?e po:short_synopsis ?o .
    ?e dc:title ?t
  }

## Get short synopsis' of EastEnders episodes (with graph)
PREFIX po: <http://purl.org/ontology/po/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?g ?t ?o
WHERE
  {
    graph ?g
      {
         <http://www.bbc.co.uk/programmes/b006m86d#programme> po:episode ?e .
         ?e a po:Episode .
         ?e po:short_synopsis ?o .
         ?e dc:title ?t
      }
  }

## Get reviews where John Paul Jones' has been involved

PREFIX mo: <http://purl.org/ontology/mo/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX rev: <http://purl.org/stuff/rev#>
PREFIX po: <http://purl.org/ontology/po/>
SELECT DISTINCT ?r_name, ?rev
WHERE
  {
    {
      <http://www.bbc.co.uk/music/artists/4490113a-3880-4f5b-a39b-105bfceaed04#artist> foaf:made ?r1 .
      ?r1 a mo:Record .
      ?r1 dc:title ?r_name .
      ?r1 rev:hasReview ?rev
    }
    UNION
    {
      <http://www.bbc.co.uk/music/artists/4490113a-3880-4f5b-a39b-105bfceaed04#artist> mo:member_of ?b1 .
      ?b1 foaf:made ?r1 .
      ?r1 a mo:Record .
      ?r1 dc:title ?r_name .
      ?r1 rev:hasReview ?rev
    }
  }
```

Example usage of IN operator for retrieving all triples for each entity

To retrieve all triples for each entity for a given list of entities uris, one might use the following syntax:

```
SELECT ?p ?o
WHERE
  {
    ?s ?p ?o .
    FILTER ( ?s IN (<someGraph#entity1>, <someGraph#entity2>, ...<someGraph#entityN> ) )
  }
```

So to demonstrate this feature, execute the following query:

```
SQL>SPARQL
SELECT DISTINCT ?p ?o
WHERE
  {
    ?s ?p ?o .
    FILTER ( ?s IN (<http://dbpedia.org/resource/Climate_change>, <http://dbpedia.org/resource/Social_vul
  }
LIMIT 100

p                                                       o
ANY                                                     ANY
_______________________________________________________________________________
```

```
http://www.w3.org/1999/02/22-rdf-syntax-ns#type        http://s.zemanta.com/ns#Target
http://s.zemanta.com/ns#title                          Climate change
http://s.zemanta.com/ns#targetType                     http://s.zemanta.com/targets#rdf

3 Rows. -- 10 msec.
```

See Also:

Example usage of IN operator.

Example for extending SPARQL via SQL for Full Text search: Variant I

To find all albums looked up by album name, one might use the following syntax:

```
SQL>SPARQL
SELECT ?s ?o ?an ( bif:search_excerpt ( bif:vector ( 'In', 'Your' ) , ?o ) )
WHERE
  {
    ?s rdf:type mo:Record .
    ?s foaf:maker ?a .
    ?a foaf:name ?an .
    ?s dc:title ?o .
    FILTER ( bif:contains ( ?o, '"in your"' ) )
  }
LIMIT 10;

http://musicbrainz.org/music/record/30f13688-b9ca-4fa5-9430-f918e2df6fc4  China in Your Hand
http://musicbrainz.org/music/record/421ad738-2582-4512-b41e-0bc541433fbc        China in Your Hand
http://musicbrainz.org/music/record/01acff2a-8316-4d4b-af93-97289e164379        China in Your Hand
http://musicbrainz.org/music/record/4fe99b06-ac73-40dd-8be7-bdaefb014981        China in Your Hand
http://musicbrainz.org/music/record/ac1cb011-6040-4515-baf2-59551a9884ac        In Your Hands
http://dbtune.org/magnatune/album/mercy-inbedinst                             In Your Bed - instrumental
http://musicbrainz.org/music/record/a09ae12e-3694-4f68-bf25-f6ff4f790962        A Word in Your Ear      A
http://dbtune.org/magnatune/album/mercy-inbedremix                            In Your Bed - the remixes
http://musicbrainz.org/music/record/176b6626-2a25-42a7-8f1d-df98bec092b4        Smoke Gets in Your Eyes
http://musicbrainz.org/music/record/e617d90e-4f86-425c-ab97-efdf4a8a452b        Smoke Gets in Your Eyes
```

Note that the query will not show anything when there are triples like:

```
<x> <y> "In"
<z> <q> "Your"
```

Example for extending SPARQL via SQL for Full Text search: Variant II

To get movies from DBpedia, where the query can contain terms from the title, one might use the following syntax:

```
SQL>SPARQL
 SELECT ?s ?an ?dn ?o( bif:search_excerpt ( bif:vector ( 'Broken', 'Flowers' ) , ?o ) )
 WHERE
  {
    ?s rdf:type dbpedia-owl:Film .
    ?s dbpprop:name ?o .
    FILTER ( bif:contains ( ?o, '"broken flowers"' ) )
    OPTIONAL { ?s dbpprop:starring ?starring .}
    OPTIONAL { ?s dbpprop:director ?director . }
    OPTIONAL { ?starring dbpprop:name ?an . }
    OPTIONAL { ?director dbpprop:name ?dn . }
  };

http://dbpedia.org/resource/Broken_Flowers  Tilda Swinton       Jim Jarmusch    Broken Flowers
http://dbpedia.org/resource/Broken_Flowers      Swinton, Tilda  Jim Jarmusch      Broken Flowers
....
http://dbpedia.org/resource/Broken_Flowers  Bill Murray           Jim Jarmusch          Music from Broken
....
```

Note that the query will not show anything when there are triples like:

```
<x> <y> "Broken"
<z> <q> "Flowers"
```

Example for date manipulation of xsd types within SPARQL

This example shows usage of dateTime column truncation to date only and performs a group by on this column:

```
-- prepare the data by inserting triples in a graph:
SQL>SPARQL
INSERT INTO GRAPH <http://BookStore.com>
  {
    <http://www.dajobe.org/foaf.rdf#i> <http://purl.org/dc/elements/1.1/title>  "SPARQL and RDF" .
    <http://www.dajobe.org/foaf.rdf#i> <http://purl.org/dc/elements/1.1/date> <1999-01-01T00:00:00>.
    <http://www.w3.org/People/Berners-Lee/card#i> <http://purl.org/dc/elements/1.1/title> "Design notes"
    <http://www.w3.org/People/Berners-Lee/card#i> <http://purl.org/dc/elements/1.1/date> <2001-01-01T00:0
    <http://www.w3.org/People/Connolly/#me> <http://purl.org/dc/elements/1.1/title> "Fundamentals of Comp
    <http://www.w3.org/People/Connolly/#me> <http://purl.org/dc/elements/1.1/date> <2002-01-01T00:00:00>.
    <http://www.ivan-herman.net/foaf.rdf#me> <http://purl.org/dc/elements/1.1/title>  "RDF Store" .
    <http://www.ivan-herman.net/foaf.rdf#me> <http://purl.org/dc/elements/1.1/date> <2001-03-05T00:00:00>
    <http://bblfish.net/people/henry/card#me> <http://purl.org/dc/elements/1.1/title> "Design RDF notes"
    <http://bblfish.net/people/henry/card#me> <http://purl.org/dc/elements/1.1/date> <2001-01-01T00:00:00
    <http://hometown.aol.com/chbussler/foaf/chbussler.foaf#me> <http://purl.org/dc/elements/1.1/title> "R
    <http://hometown.aol.com/chbussler/foaf/chbussler.foaf#me> <http://purl.org/dc/elements/1.1/date> <20
  };

_______________________________________________________

Insert into <http://BookStore.com>, 12 triples -- done

-- Find Count of Group by Dates
SQL>SPARQL
SELECT (xsd:date(bif:subseq(str(?a_dt), 0, 10))), count(*)
FROM <http://BookStore.com>
WHERE
  {
    ?s <http://purl.org/dc/elements/1.1/date> ?a_dt
  }
GROUP BY (xsd:date(bif:subseq(str(?a_dt), 0, 10)));

callret-0                                             callret-1
VARCHAR                                               VARCHAR
_______________________________________________________
1999-01-01                                            1
2001-01-01                                            2
2002-01-01                                            2
2001-03-05                                            1

4 Rows. -- 15 msec.
SQL>
```

Example for executing INSERT/DELETE (SPARUL) statements against a WebID protected SPARQL endpoint

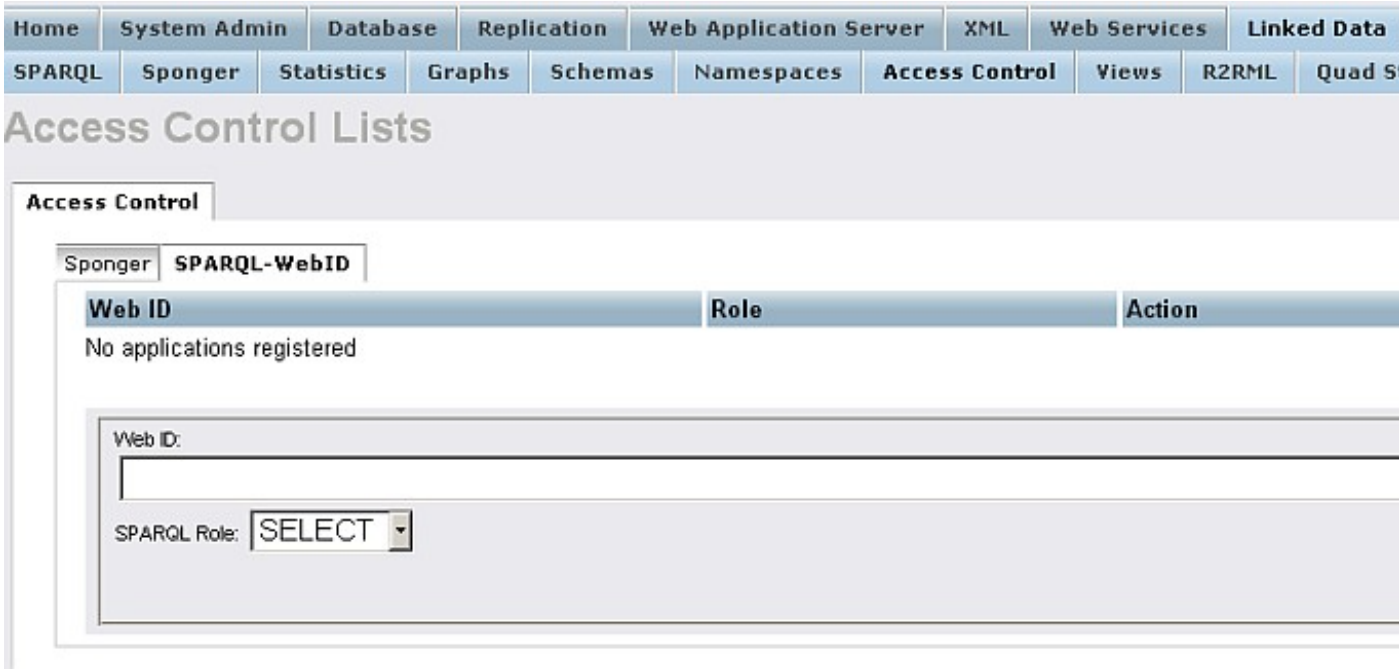
The following sample scenario demonstrates how to perform INSERT/DELETE (SPARUL) statements against a protected SPARQL Endpoint by setting WebID Protocol ACLs using the Virtuoso Authentication Server UI:

1. Obtain a WebID:
    a. Download and install the ods_framework_dav.vad .
        - Note: an existing ODS DataSpace user instance can also be used, for example at http://id.myopenlink.net/ods/
    b. Register an ODS Data Space user, for example with name "demo".
    c. The generated WebID will be for example:

        ```
        http://id.myopenlink.net/dataspace/person/demo#this
        ```

    d. Generate a Personal HTTP based Identifier for the "demo" user and then bind the personal Identifier to an X.509 Certificate, thereby giving assigning the user a WebID.
2. Download and install the conductor_dav.vad package, if not already installed.
3. Go to http://<cname>:<port>/conductor, where <cname>:<port> are replaced by your local server values.
4. Go to System Admin -> Linked Data -> Access Control -> SPARQL-WebID

Figure 16.45. Conductor SPARQL-WebID



5. In the displayed form:

   a. Enter the Web ID for the user registered above, for example:

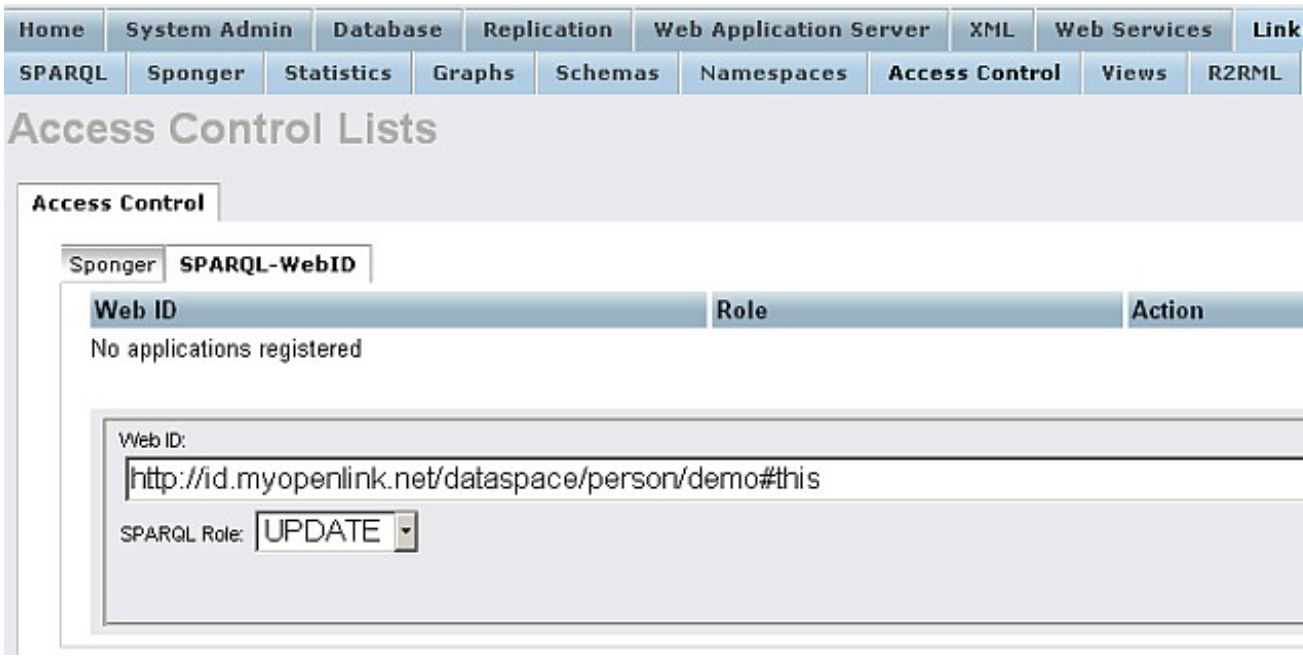
      ```
      http://id.myopenlink.net/dataspace/person/demo#this
      ```

   b. Select "SPARQL Role": "

      UPDATE

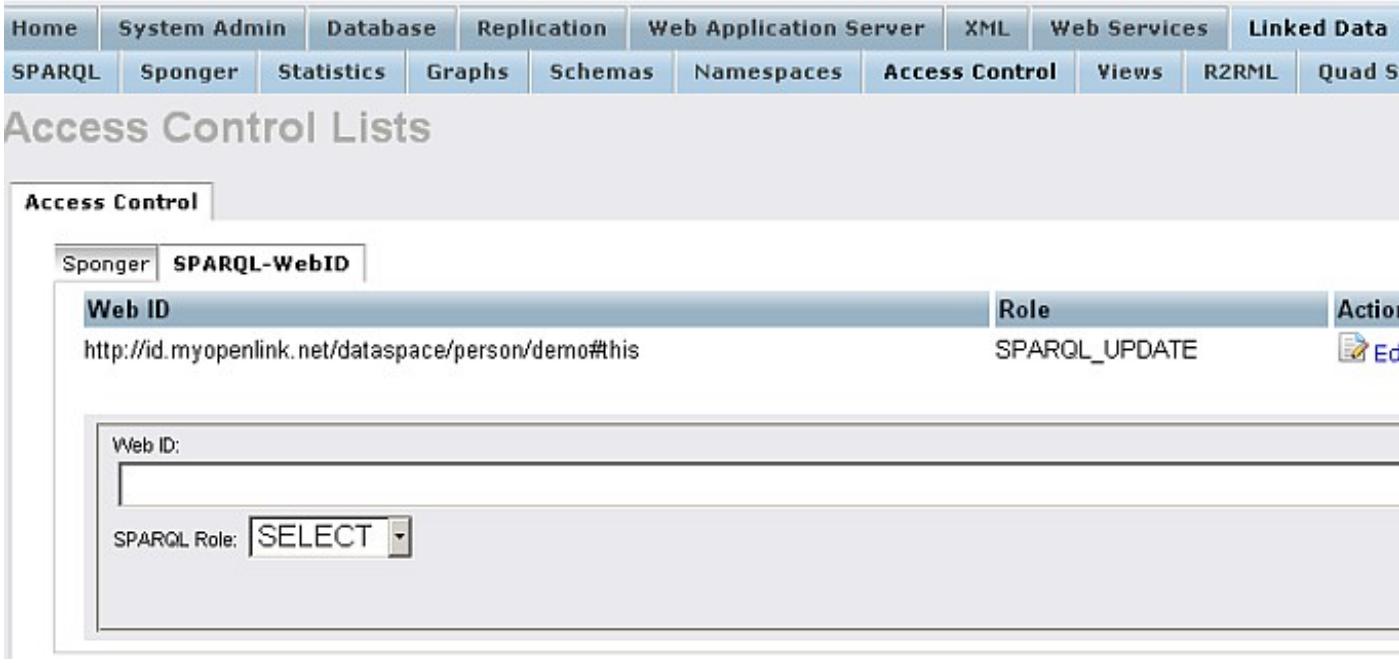
      ".

Figure 16.46. Conductor SPARQL-WebID



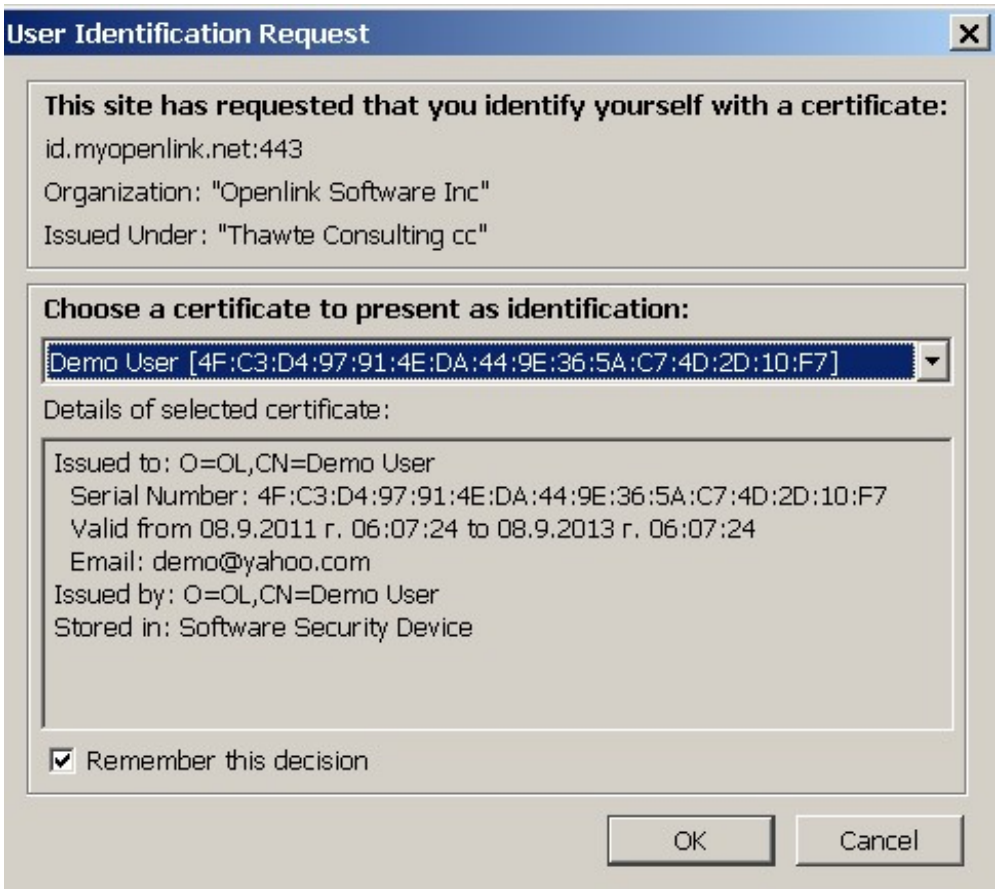
6. Click the "Register" button.
7. The WebID Protocol ACL will be created:

Figure 16.47. Conductor SPARQL-WebID



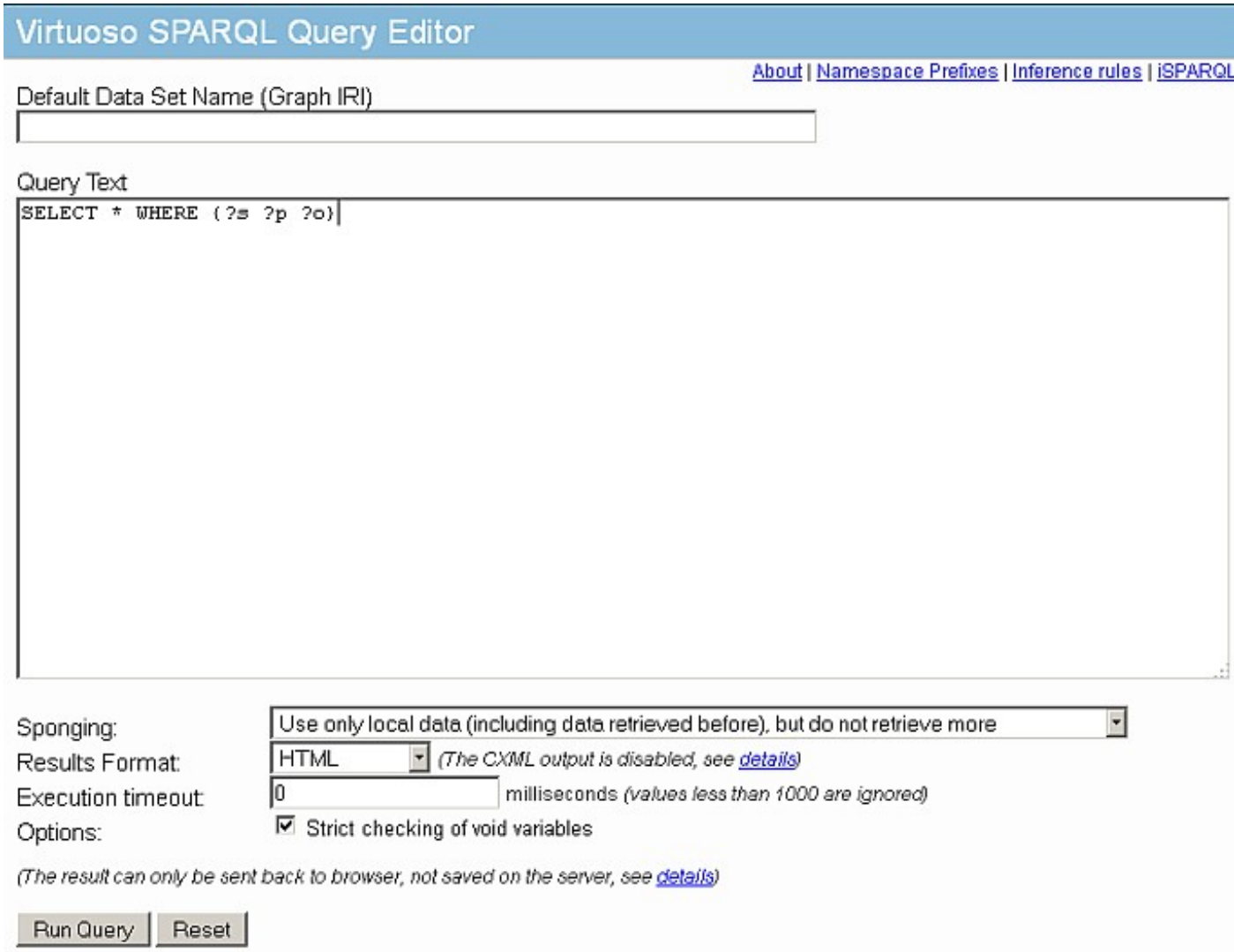
8. Go to the SPARQL-WebID endpoint, https://<cname>:<port>/sparql-webid, where <cname>:<port> are replaced by your local server values.
9. Select the user's certificate:

Figure 16.48. Conductor SPARQL-WebID



10. The SPARQL Query UI will be displayed:

Figure 16.49. Conductor SPARQL-WebID



11. Execute the query:

```
INSERT INTO GRAPH <http://example.com> {
  <s1> <p1> <o1> .
  <s2> <p2> <o2> .
  <s3> <p3> <o3>
}
```

Figure 16.50. Conductor SPARQL-WebID

Figure 16.51. Conductor SPARQL-WebID

callret-0
Insert into <http://mytest.com>, 3 (or less) triples -- done

Note: If the SPARQL Role "Sponge" is set instead, in order to be able to execute DELETE/INSERT statements over the protected SPARQL Endpoint, the following grants need to be performed for the user, associated with the WebID ACL Role:

```
grant execute on DB.DBA.SPARQL_INSERT_DICT_CONTENT to "demo";
grant execute on DB.DBA.SPARQL_DELETE_DICT_CONTENT to "demo";
```

Example usage of deleting Triple Patterns that are Not Scoped to a Named Graph

Presuming this triple exists in one or more graphs in the store:

```
{
  <http://kingsley.idehen.net/dataspace/person/kidehen#this>
    <http://xmlns.com/foaf/0.1/knows>
      <http://id.myopenlink.net/dataspace/person/KingsleyUyiIdehen#this>
}
```

The SQL query below will delete that triple from all graphs in the store:

```
DELETE
  FROM DB.DBA.RDF_QUAD
```

```
 WHERE p = iri_to_id
             ('http://xmlns.com/foaf/0.1/knows')
   AND s = iri_to_id
             ('http://kingsley.idehen.net/dataspace/person/kidehen#this')
   AND o = iri_to_id
             ('http://id.myopenlink.net/dataspace/person/KingsleyUyiIdehen#this')
;
```

According to SPARQL 1.1 Update , the FROM clause which scopes the query to a single graph is optional. Thus, the SQL query above can be rewritten to the SPARQL query below, again deleting the matching triple from all graphs in the store:

```
DELETE
  {
    GRAPH ?g
      {
        <http://kingsley.idehen.net/dataspace/person/kidehen#this>
          <http://xmlns.com/foaf/0.1/knows>
            <http://id.myopenlink.net/dataspace/person/KingsleyUyiIdehen#this>
      }
  }
WHERE
  {
    GRAPH ?g
      {
        <http://kingsley.idehen.net/dataspace/person/kidehen#this>
          <http://xmlns.com/foaf/0.1/knows>
            <http://id.myopenlink.net/dataspace/person/KingsleyUyiIdehen#this>
      }
  }
```

Example usage of deleting triples containing blank nodes

There are two ways to delete a particular blank node:

1. To refer to it via some properties or:
2. To convert it to it's internal "serial number", a long integer, and back.

Assume the following sample scenario:

1. Clear the graph:

```
SPARQL CLEAR GRAPH <http://sample/>;

Done. -- 4 msec.
```

2. Insert three blank nodes with two related triples each:

```
SPARQL
  INSERT IN GRAPH <http://sample/>
    {
      [] <p> <o1a> , <o1b> .
      [] <p> <o2a> , <o2b> .
      [] <p> <o3a> , <o3b>
    }

Done. -- 15 msec.
```

3. Delete one pair of triples:

```
SPARQL WITH <http://sample/>
  DELETE { ?s ?p ?o }
  WHERE
    {
      ?s ?p ?o ;
      <p> <o1a> .
    }

Done. -- 7 msec.
```

4. Ensure that we still have two bnodes, two triple per bnode:

```
SPARQL
```

```
  SELECT *
  FROM <http://sample/>
  WHERE
    {
      ?s ?p ?o
    }
s               p       o
VARCHAR         VARCHAR VARCHAR
_______________

nodeID://b10006 p       o3a
nodeID://b10006 p       o3b
nodeID://b10007 p       o2a
nodeID://b10007 p       o2b

4 Rows. -- 4 msec.
```

6. Each bnode, as well as any "named" node, is identified internally as an integer:

```
SPARQL
  SELECT (<LONG::bif:iri_id_num>(?s)) AS ?s_num, ?p, ?o
  FROM <http://sample/>
  WHERE
    {
      ?s ?p ?o
    };
s_num                p       o
INTEGER              VARCHAR VARCHAR
_____________________________

4611686018427397910  p       o3a
4611686018427397910  p       o3b
4611686018427397911  p       o2a
4611686018427397911  p       o2b

4 Rows. -- 5 msec.
```

7. The integer can be converted back to internal identifier. Say, here we try to delete a triple that does not exist (even if the ID integer is valid):

```
SPARQL
  DELETE FROM <http://sample/>
    {
      `bif:iri_id_from_num(4611686018427397911)` <p> <o3a>
    };

Done. -- 5 msec.
```

8. Should have no effect, because the "46..11" IRI has <o2a> and <o2b>, and was not requested <o3a>:

```
SPARQL
  SELECT *
  FROM <http://sample/>
  WHERE
    {
      ?s ?p ?o
    };
s               p        o
VARCHAR         VARCHAR  VARCHAR
_______________

nodeID://b10006 p        o3a
nodeID://b10006 p        o3b
nodeID://b10007 p        o2a
nodeID://b10007 p        o2b

4 Rows. -- 5 msec.
```

9. Now let's try to delete a triple that does actually exist. Note the use of backquotes to insert an expression into template:

```
SPARQL
  DELETE FROM <http://sample/>
    {
      `bif:iri_id_from_num(4611686018427397911)` <p> <o2a>
    };
```

```
Done. -- 4 msec.
```

10. So there's an effect:

```
SPARQL
  SELECT *
  FROM <http://sample/>
  WHERE
    {
      ?s ?p ?o
    };
s                 p        o
VARCHAR           VARCHAR  VARCHAR
_________________

nodeID://b10006   p        o3a
nodeID://b10006   p        o3b
nodeID://b10007   p        o2b

3 Rows. -- 2 msec.
```

11. Now delete everything related to `nodeID://b10006` subject:

```
SPARQL
  WITH <http://sample/>
  DELETE
    {
      ?s ?p ?o
    }
  WHERE
    {
      ?s ?p ?o .
      FILTER (?s = bif:iri_id_from_num(4611686018427397910))
    };

Done. -- 18 msec.
```

12. Three minus two gives one triple remaining:

```
SQL> SPARQL
  SELECT *
  FROM <http://sample/>
  WHERE
    {
      ?s ?p ?o
    };
s                 p        o
VARCHAR           VARCHAR  VARCHAR
_________________

nodeID://b10007   p        o2b

1 Rows. -- 4 msec.
```

Note : IDs of bnodes will vary from server to server and even from run to run on the same server, so the application should identify bnodes by properties before doing `bif:iri_id_XXX` tricks.

Example usage of expressions inside CONSTRUCT, INSERT and DELETE {...} Templates

When one wants to use expressions inside CONSTRUCT {...}, INSERT {...} or DELETE {...} construction templates, the expressions should be in back-quotes i.e:

```
`expression`
```

What?

How to construct RDF triples via SPARQL CONSTRUCT queries that include expressions.

Why?

The are times when you need to post-process existing RDF triples en route to creating enhanced data views. For instance, enhancing the literal values associated with annotation properties such as rdfs:label and rdfs:comment .

Why?

Here some SPARQL 1.1 Update Language examples showcasing how this is achieved using Virtuoso.

Example usage of expression inside CONSTRUCT

```
CONSTRUCT
  {
    ?inst  rdfs:label `bif:concat ( ?inst_label,
                                    " Instance with up to ",
                                    str(?core_val),
                                    " logical processor cores and " ,
                                    str(?sess_val) , " concurrent ODBC sessions from licensed host" )`
  }
FROM <http://uda.openlinksw.com/pricing/>
WHERE
  {
    ?inst a gr:Individual, oplweb:ProductLicense ;
                           rdfs:label ?inst_label ;
           oplweb:hasMaximumProcessorCores ?core ;
                       oplweb:hasSessions ?sess .

    ?core a gr:QuantitativeValueInteger ;
        gr:hasMaxValueInteger ?core_val .

    ?sess a gr:QuantitativeValueInteger ;
                  gr:hasValue ?sess_val .
}
```

See live results of the query.

Example usage of expression inside INSERT

```
SPARQL
INSERT INTO GRAPH <urn:mygraph>
  {
    ?inst  rdfs:label `bif:concat ( ?inst_label,
                                    " Instance with up to ",
                                    str(?core_val),
                                    " logical processor cores and " ,
                                    str(?sess_val) ,
                                    " concurrent ODBC sessions from licensed host" )`
  }
FROM <http://uda.openlinksw.com/pricing/>
WHERE
  {
    ?inst a gr:Individual, oplweb:ProductLicense ;
                           rdfs:label ?inst_label ;
           oplweb:hasMaximumProcessorCores ?core ;
                       oplweb:hasSessions ?sess .

    ?core a gr:QuantitativeValueInteger ;
        gr:hasMaxValueInteger ?core_val .

    ?sess a gr:QuantitativeValueInteger ;
                  gr:hasValue ?sess_val .
};

Done. -- 406 msec.

SQL> SPARQL
SELECT ?label
FROM <urn:mygraph>
WHERE
  {
    ?inst  rdfs:label ?label
  };

label
VARCHAR
_______________________________________________________________________________
```

```
ODBC Driver (Single-Tier Lite Express Edition) Instance with up to 16 logical processor cores and 5 concu
ODBC Driver (Single-Tier Lite Express Edition) Instance with up to 16 logical processor cores and 5 concu
ODBC Driver (Single-Tier Lite Edition) Instance with up to 16 logical processor cores and 5 concurrent OD
ODBC Driver (Single-Tier Lite Edition) Instance with up to 16 logical processor cores and 5 concurrent OD
ODBC Driver (Single-Tier Lite Edition) Instance with up to 16 logical processor cores and 5 concurrent OD
ODBC Driver (Single-Tier Lite Edition) Instance with up to 16 logical processor cores and 5 concurrent OD
JDBC Driver (Single-Tier Lite Edition) Instance with up to 16 logical processor cores and 5 concurrent OD
OLEDB Driver (Single-Tier Lite Edition) Instance with up to 16 logical processor cores and 5 concurrent O
ADO.NET Driver (Single-Tier Lite Edition) Instance with up to 16 logical processor cores and 5 concurrent

9 Rows. -- 31 msec.
```

Example usage of expression inside DELETE

```
SPARQL
DELETE FROM GRAPH <urn:mygraph>
{ ?inst rdfs:label `bif:concat ( "JDBC Driver (Single-Tier Lite Edition) Instance with up to ", str(?core
FROM <http://uda.openlinksw.com/pricing/>
WHERE
{
    ?inst a gr:Individual, oplweb:ProductLicense ;
                           rdfs:label ?inst_label ;
          oplweb:hasMaximumProcessorCores ?core ;
                        oplweb:hasSessions ?sess .
    filter (regex(?inst_label,"JDBC Driver")) .

    ?core a gr:QuantitativeValueInteger ;
        gr:hasMaxValueInteger ?core_val .

    ?sess a gr:QuantitativeValueInteger ;
                  gr:hasValue ?sess_val .
}
;

Done. -- 32 msec.

SQL> SPARQL
SELECT ?label
FROM <urn:mygraph>
WHERE
  {
    ?inst  rdfs:label ?label
  };

label
VARCHAR
_______________________________________________________________________________

ODBC Driver (Single-Tier Lite Express Edition) Instance with up to 16 logical ...
ODBC Driver (Single-Tier Lite Express Edition) Instance with up to 16 logical ...
ODBC Driver (Single-Tier Lite Edition) Instance with up to 16 logical processor ...
ODBC Driver (Single-Tier Lite Edition) Instance with up to 16 logical processor ...
ODBC Driver (Single-Tier Lite Edition) Instance with up to 16 logical processor ...
ODBC Driver (Single-Tier Lite Edition) Instance with up to 16 logical processor ...
OLEDB Driver (Single-Tier Lite Edition) Instance with up to 16 logical processor ...
ADO.NET Driver (Single-Tier Lite Edition) Instance with up to 16 logical processor ...

8 Rows. -- 16 msec.
```

16.3.3. Business Intelligence Extensions for SPARQL

Virtuoso extends SPARQL with expressions in results, subqueries, aggregates and grouping. These extensions allow a straightforward translation of arbitrary SQL queries to SPARQL. This extension is called "SPARQL BI", because the primary objective is to match needs of Business Intelligence. The extended features apply equally to querying physical quads or relational tables mapped through Linked Data Views.

Note

In this section, many examples use the TPC-H namespace. You may test them on your local demo database. They use data from the TPC-H dataset that is mapped into a graph with an IRI of the form http://example.com/tpch. When testing, you should replace the fake host name "example.com" with the host name of your own installation verbatim, that is as specified

Note

in the "DefaultHost" parameter in the [URIQA] section of the Virtuoso configuration file.

Aggregates in SPARQL

Virtuoso extends SPARQL with SQL like aggregate and "group by" functionality. This functionality is also available by embedding SPARQL text inside SQL, but the SPARQL extension syntax has the benefit of also working over the SPARQL protocol and of looking more SPARQL-like.

The supported aggregates are *COUNT* , *MIN* , *MAX* , *AVG* and *SUM* . These can take an optional *DISTINCT* keyword. These are permitted only in the selection part of a select query. If a selection list consists of a mix of variables and aggregates, the non-aggregate selected items are considered to be grouping columns and a *GROUP BY* over them is implicitly added at the end of the generated SQL query. Virtuoso also supports explicit syntax for *GROUP BY* , *ORDER BY* , *LIMIT* and *OFFSET* . There is no explicit syntax for *HAVING* in Virtuoso SPARQL.

If a selection consists of aggregates exclusively, the result set has one row with the values of the aggregates. If there are aggregates and variables in the selection, the result set has as many rows as there are distinct combinations of the variables; the aggregates are then calculated over each such distinct combination, as if there were a SQL GROUP BY over all non-aggregates. The implicit grouping pays attention to all subexpressions in the return list; say, if a result column expression is `(?x * max (?y))` then `?y` is aggregated and `?x` is not so it is grouped by ?x. This also means that if a result column expression is `(bif:year (?shipdate))` then a group is made for each distinct `?shipdate` , i.e. up to 366 groups for each distinct year. If you need one group per year, write explicit `GROUP BY (bif:year (?shipdate))` .

With the count aggregate the argument may be either * , meaning counting all rows, or a variable name, meaning counting all the rows where this variable is bound. If there is no implicit *GROUP BY* , there can be an optional *DISTINCT* keyword before the variable that is the argument of an aggregate.

There is a special syntax for counting distinct combinations of selected variables. This is:

```
SELECT COUNT DISTINCT ?v1 ... ?vn
  FROM ....
```

User-defined aggregate functions are not supported in current version of the SPARQL compiler.

Path Expressions

Virtuoso has support for paths consisting of dereferencing properties in SPARQL. Virtuoso allows simple paths in expressions and has a separate feature for transitivity:

- S+>P: for "one or many values of P of S"
- S*>P: for "zero or many values of P of S", so *> may form a LEFT OUTER JOIN whereas +> forms an INNER JOIN.
- S|>P: is reserved for potential "single value of P of S or an error if there are many values"

If this property is set (for example by an Linked Data View) then +> should be used.

Simple Example

```
SELECT ?f+>foaf:name ?f|>foaf:mbox WHERE { ?x foaf:name "Alice" . ?x foaf:knows ?f . FILTER (?f+>foaf:nam
```

means:

```
SELECT ?fname ?mbox
WHERE
  {
    ?x foaf:knows ?f .
    ?x foaf:knows ?f .
    OPTIONAL {?f foaf:mbox ?mbox} .
    ?f foaf:name ?fname .
    ?x foaf:name "Alice" .
    ?x foaf:knows ?f2 .
    ?f2 foaf:name "John" .
  }
```

Other Examples

```
SPARQL
DEFINE sql:signal-void-variables 1
PREFIX tpcd: <http://www.openlinksw.com/schemas/tpcd#>
PREFIX oplsioc: <http://www.openlinksw.com/schemas/oplsioc#>
PREFIX sioc: <http://rdfs.org/sioc/ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT
  ?l+>tpcd:returnflag,
  ?l+>tpcd:linestatus,
  sum(?l+>tpcd:linequantity) as ?sum_qty,
  sum(?l+>tpcd:lineextendedprice) as ?sum_base_price,
  sum(?l+>tpcd:lineextendedprice*(1 - ?l+>tpcd:linediscount)) as ?sum_disc_price,
  sum(?l+>tpcd:lineextendedprice*(1 - ?l+>tpcd:linediscount)*(1+?l+>tpcd:linetax)) as ?sum_charge,
  avg(?l+>tpcd:linequantity) as ?avg_qty,
  avg(?l+>tpcd:lineextendedprice) as ?avg_price,
  avg(?l+>tpcd:linediscount) as ?avg_disc,
  count(1) as ?count_order
FROM <http://example.com/tpcd>
WHERE {
    ?l a tpcd:lineitem .
    FILTER (?l+>tpcd:shipdate <= bif:dateadd ("day", -90, '1998-12-01'^^xsd:date)) }
ORDER BY ?l+>tpcd:returnflag ?l+>tpcd:linestatus


SPARQL
DEFINE sql:signal-void-variables 1
PREFIX tpcd: <http://www.openlinksw.com/schemas/tpcd#>
SELECT
  ?supp+>tpcd:acctbal,
  ?supp+>tpcd:name,
  ?supp+>tpcd:has_nation+>tpcd:name as ?nation_name,
  ?part+>tpcd:partkey,
  ?part+>tpcd:mfgr,
  ?supp+>tpcd:address,
  ?supp+>tpcd:phone,
  ?supp+>tpcd:comment
FROM <http://example.com/tpcd>
WHERE {
  ?ps a tpcd:partsupp; tpcd:has_supplier ?supp; tpcd:has_part ?part .
  ?supp+>tpcd:has_nation+>tpcd:has_region tpcd:name 'EUROPE' .
  ?part tpcd:size 15 .
  ?ps tpcd:supplycost ?minsc .
  { SELECT ?part min(?ps+>tpcd:supplycost) as ?minsc
    WHERE {
        ?ps a tpcd:partsupp; tpcd:has_part ?part; tpcd:has_supplier ?ms .
        ?ms+>tpcd:has_nation+>tpcd:has_region tpcd:name 'EUROPE' .
      } }
    FILTER (?part+>tpcd:type like '%BRASS') }
ORDER BY
  desc (?supp+>tpcd:acctbal)
  ?supp+>tpcd:has_nation+>tpcd:name
  ?supp+>tpcd:name
  ?part+>tpcd:partkey
```

Examples

Example for count of physical triples in http://mygraph.com

```
SPARQL
SELECT COUNT (*)
  FROM <http://mygraph.com>
 WHERE {?s ?p ?o}
```

Example for count of O's for each distinct P

```
SPARQL define input:inference "http://mygraph.com"
SELECT ?p COUNT (?o)
  FROM <http://mygraph.com>
 WHERE {?s ?p ?o}
```

Example for count of triples, including inferred triples and the count of distinct O values

```
SPARQL define input:inference "http://mygraph.com"
SELECT COUNT (?p) COUNT (?o) COUNT (DISTINCT ?o)
 FROM <http://mygraph.com>
WHERE {?s ?p ?o}
```

Example for get number of distinct bindings of ?s ?p ?o

```
SPARQL define input:inference "http://mygraph.com"
SELECT count distinct ?s ?p ?o
  FROM <http://mygraph.com>
 WHERE {?s ?p ?o}
```

Example for get counts and total prices of ordered items, grouped by item status

```
SPARQL
prefix tpch: <http://www.openlinksw.com/schemas/tpch#>
SELECT ?status count(*) sum(?extendedprice)
FROM <http://localhost.localdomain:8310/tpch>
WHERE {
    ?l a tpch:lineitem ;
      tpch:lineextendedprice ?extendedprice ;
      tpch:linestatus ?status .
  }
```

Example for get counts and total prices of ordered items, grouped by item status

Example: A dataset of people, some duplicated

Suppose there is a dataset with many people, some of them sharing the same name. To list them we would, ideally, execute the query:

```
SPARQL
SELECT DISTINCT
 (?name) ?person ?mail
 WHERE {
   ?person rdf:type foaf:Person .
   ?person foaf:name ?name .
   ?person foaf:mbox_sha1sum ?mail
 }
```

Unfortunately, the facility to apply DISTINCT to a part of the result set row (i.e. to ?name) does not currently exist. (Although the above form is permitted, it's interpreted as being identical to 'SELECT DISTINCT ?name, ?person, ?mail WHERE ...') If there's demand for such a feature then we may introduce an aggregate called, say, SPECIMEN, that will return the very first of the aggregated values. e.g.:

```
SPARQL
SELECT ?name (specimen(?person)) (specimen(?mail))
WHERE
  {
    ?person rdf:type foaf:Person .
    ?person foaf:name ?name .
    ?person foaf:mbox_sha1sum ?mail
  }
```

As a workaround to this limitation, the MIN aggregate can be used, provided duplicates are few and there's no requirement that ?person should correspond to ?mail (i.e. the result should contain some person node and some mail node but they don't have to be connected by foaf:mbox_sha1sum):

```
SPARQL
SELECT ?name (min(?person)) (min(?mail))
WHERE
  {
    ?person rdf:type foaf:Person .
    ?person foaf:name ?name .
    ?person foaf:mbox_sha1sum ?mail
  }
```

Otherwise, a complicated query is needed:

```
SPARQL
SELECT
 ?name
 ((SELECT (min (?person3))
     WHERE {
         ?person3 rdf:type foaf:Person .
         ?person3 foaf:name ?name .
         ?person3 foaf:mbox_sha1sum ?mail } )) as ?person
 ?mail
 WHERE {
     { SELECT distinct ?name
       WHERE {
           ?person1 rdf:type foaf:Person .
           ?person1 foaf:name ?name .
           ?person1 foaf:mbox_sha1sum ?mail1 } }
     { SELECT ?name (min(?mail2)) as ?mail
       WHERE {
           ?person2 rdf:type foaf:Person .
           ?person2 foaf:name ?name .
           ?person2 foaf:mbox_sha1sum ?mail2 } }
 }
```

Example quering dbpedia

The following example demonstrate how to query dbpedia. Suppose there is local onotlogy, which has a datatype property hasLocation with a string containing city names. The query below finds which of those cities are in dbpedia:

```
SPARQL
PREFIX dbpprop: <http://dbpedia.org/property/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX vocab:<http://myexample.com/localOntology.rdf>
PREFIX dbpedia: <http://dbpedia.org/>
PREFIX dbpres: <http://dbpedia.org/resource/>
SELECT ?city
WHERE
  {
    ?sub :location ?city .
    FILTER(bif:exists(( ASK { ?subdb a dbo:City . ?subdb dbpprop:officialName ?city })))
  }
```

Example for MAX with HAVING and GROUP BY

```
## Example "Find which town or city in
## the UK has the largest proportion of students.

PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
PREFIX dbpedia-owl-uni: <http://dbpedia.org/ontology/University/>
PREFIX dbpedia-owl-inst: <http://dbpedia.org/ontology/EducationalInstitution/>

SELECT ?town COUNT(?uni)
       ?pgrad ?ugrad
       MAX(?population)
       ( ((?pgrad+?ugrad)/ MAX(?population))*100 ) AS ?percentage
WHERE
  {
    ?uni dbpedia-owl-inst:country dbpedia:United_Kingdom ;
                         dbpedia-owl-uni:postgrad ?pgrad ;
                        dbpedia-owl-uni:undergrad ?ugrad ;
                             dbpedia-owl-inst:city ?town .
    OPTIONAL
        {
          ?town dbpedia-owl:populationTotal ?population .
          FILTER (?population > 0 )
        }
  }
GROUP BY ?town ?pgrad ?ugrad
HAVING ( ( ( (?pgrad+?ugrad)/ MAX(?population) )*100 ) > 0 )
ORDER BY DESC 6
```

Example Aggregating Distance Values Over Years

The following example demonstrate how to aggregate Distance Values Over Years:

First we insert some data in a graph with name for ex. <urn:dates:distances>:

```
SQL> SPARQL INSERT INTO GRAPH <urn:dates:distances>
  {
    <:a1> <http://purl.org/dc/elements/1.1/date> <2010-12-23T00:00:00> .
    <:a1> <http://linkedgeodata.org/vocabulary#distance> <0.955218675> .
    <:a2> <http://purl.org/dc/elements/1.1/date> <2010-12-24T00:00:00> .
    <:a2> <http://linkedgeodata.org/vocabulary#distance> <0.798155989> .
    <:a3> <http://purl.org/dc/elements/1.1/date> <2010-12-25T00:00:00> .
    <:a3> <http://linkedgeodata.org/vocabulary#distance> <0.064686628> .
    <:a4> <http://purl.org/dc/elements/1.1/date> <2010-12-26T00:00:00> .
    <:a4> <http://linkedgeodata.org/vocabulary#distance> <0.279800332> .
    <:a5> <http://purl.org/dc/elements/1.1/date> <2010-12-27T00:00:00> .
    <:a5> <http://linkedgeodata.org/vocabulary#distance> <0.651255995> .
    <:a6> <http://purl.org/dc/elements/1.1/date> <2010-12-28T00:00:00> .
    <:a6> <http://linkedgeodata.org/vocabulary#distance> <0.094410557> .
    <:a7> <http://purl.org/dc/elements/1.1/date> <2010-12-29T00:00:00> .
    <:a7> <http://linkedgeodata.org/vocabulary#distance> <0.43461913> .
    <:a8> <http://purl.org/dc/elements/1.1/date> <2010-12-30T00:00:00> .
    <:a8> <http://linkedgeodata.org/vocabulary#distance> <0.264862918> .
    <:a9> <http://purl.org/dc/elements/1.1/date> <2010-12-31T00:00:00> .
    <:a9> <http://linkedgeodata.org/vocabulary#distance> <0.770588658> .
    <:a10> <http://purl.org/dc/elements/1.1/date> <2011-01-01T00:00:00> .
    <:a10> <http://linkedgeodata.org/vocabulary#distance> <0.900997627> .
    <:a11> <http://purl.org/dc/elements/1.1/date> <2011-01-02T00:00:00> .
    <:a11> <http://linkedgeodata.org/vocabulary#distance> <0.324972375> .
    <:a12> <http://purl.org/dc/elements/1.1/date> <2011-01-03T00:00:00> .
    <:a12> <http://linkedgeodata.org/vocabulary#distance> <0.937221226> .
    <:a13> <http://purl.org/dc/elements/1.1/date> <2011-01-04T00:00:00> .
    <:a13> <http://linkedgeodata.org/vocabulary#distance> <0.269511925> .
    <:a14> <http://purl.org/dc/elements/1.1/date> <2011-01-05T00:00:00> .
    <:a14> <http://linkedgeodata.org/vocabulary#distance> <0.726014538> .
    <:a15> <http://purl.org/dc/elements/1.1/date> <2011-01-06T00:00:00> .
    <:a15> <http://linkedgeodata.org/vocabulary#distance> <0.843581439> .
    <:a16> <http://purl.org/dc/elements/1.1/date> <2011-01-07T00:00:00> .
    <:a16> <http://linkedgeodata.org/vocabulary#distance> <0.835685559> .
    <:a17> <http://purl.org/dc/elements/1.1/date> <2011-01-08T00:00:00> .
    <:a17> <http://linkedgeodata.org/vocabulary#distance> <0.673213742> .
    <:a18> <http://purl.org/dc/elements/1.1/date> <2011-01-09T00:00:00> .
    <:a18> <http://linkedgeodata.org/vocabulary#distance> <0.055026879> .
    <:a19> <http://purl.org/dc/elements/1.1/date> <2011-01-10T00:00:00> .
    <:a19> <http://linkedgeodata.org/vocabulary#distance> <0.987475424> .
    <:a20> <http://purl.org/dc/elements/1.1/date> <2011-01-11T00:00:00> .
    <:a20> <http://linkedgeodata.org/vocabulary#distance> <0.167315598> .
    <:a21> <http://purl.org/dc/elements/1.1/date> <2011-01-12T00:00:00> .
    <:a21> <http://linkedgeodata.org/vocabulary#distance> <0.545317103> .
    <:a22> <http://purl.org/dc/elements/1.1/date> <2011-01-13T00:00:00> .
    <:a22> <http://linkedgeodata.org/vocabulary#distance> <0.75137005> .
    <:a23> <http://purl.org/dc/elements/1.1/date> <2011-01-14T00:00:00> .
    <:a23> <http://linkedgeodata.org/vocabulary#distance> <0.123649985> .
    <:a24> <http://purl.org/dc/elements/1.1/date> <2011-01-15T00:00:00> .
    <:a24> <http://linkedgeodata.org/vocabulary#distance> <0.750214251> .
  };
callret-0
VARCHAR
_______________________________________________________________________________

Insert into <urn:dates:distances>, 48 (or less) triples -- done

1 Rows. -- 94 msec.
```

Then we execute the following query:

```
SQL> SPARQL
PREFIX dst: <http://linkedgeodata.org/vocabulary#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT (bif:year( bif:stringdate(?sdate)) AS ?syear) (bif:sum( bif:number(?dist)) AS ?distance)
FROM <urn:dates:distances>
```

```
WHERE
  {
    ?row dc:date ?sdate .
    ?row dst:distance ?dist
  }
GROUP BY (bif:year(bif:stringdate(?sdate)))
ORDER BY ASC(bif:year(bif:stringdate(?sdate)));

syear                                   distance
VARCHAR                                 VARCHAR
_______________________________________________

2010                                    4.313598882
2011                                    8.891567721

2 Rows. -- 31 msec.
```

Note on Aggregates and Inference

Inferencing is added to a SPARQL query only for those variables whose value is actually used. Thus,

```
SELECT COUNT (*)
 FROM <http://mygraph.com>
WHERE {?s ?p ?o}
```

will not return inferred values since s, p, and o are not actually used. In contrast,

```
SPARQL
SELECT COUNT (?s) COUNT (?p) COUNT (?o)
 FROM <http://mygraph.com>
WHERE {?s ?p ?o}
```

will also return all the inferred triples.

Note: This difference in behaviour may lead to confusion and will, therefore, likely be altered in the future.

Pointer Operator (*+>* and **>*)

When expressions occur in result sets, many variables are often introduced only for the purpose of passing a value from a triple pattern to the result expression. This is inconvenient because many triple patterns are trivial. The presence of large numbers of variable names masks "interesting" variables that are used in more than once in pattern and which establish logical relationships between different parts of the query. As a solution we introduce pointer operators.

The +> (pointer) operator allows referring to a property value without naming it as a variable and explicitly writing a triple pattern. We can shorten the example above to:

```
SPARQL
prefix tpch: <http://www.openlinksw.com/schemas/tpch#>
SELECT ?l+>tpch:linestatus count(*) sum(?l+>tpch:lineextendedprice)
FROM <http://localhost.localdomain:8310/tpch>
WHERE { ?l a tpch:lineitem }
```

The *?subject+>propertyname* notation is equivalent to having a triple pattern *?subject propertyname ?aux_var* binding an auxiliary variable to the mentioned property of the subject, within the group pattern enclosing the reference. For a SELECT, the enclosing group pattern is considered to be the top level pattern of the where clause or, in the event of a union, the top level of each term of the union. Each distinct pointer adds exactly one triple pattern to the enclosing group pattern. Multiple uses of +> with the same arguments do not each add a triple pattern. (Having multiple copies of an identical pattern might lead to changes in cardinality if multiple input graphs were being considered. If a lineitem had multiple discounts or extended prices, then we would get the cartesian product of both.)

If a property referenced via +> is absent, the variable on the left side of the operator is not bound in the enclosing group pattern because it should be bound in all triple patterns where it appears as a field, including implicitly added patterns.

The *?subject*>propertyname* notation is introduced in order to access optional property values. It adds an OPTIONAL group *OPTIONAL { ?subject propertyname ?aux_var }* , not a plain triple pattern, so the binding of ?subject is not changed even if the object variable is not bound. If the property is set for all subjects in question then the results of *> and +> are the same. All other

things being equal, the +> operator produces better SQL code than *> so use *> only when it is really needed.

Subqueries in SPARQL

Pure SPARQL does not allow binding a value that is not retrieved through a triple pattern. We lift this restriction by allowing expressions in the result set and providing names for result columns. We also allow a SPARQL SELECT statement to appear in another SPARQL statement in any place where a group pattern may appear. The names of the result columns form the names of the variables bound, using values from the returned rows. This resembles derived tables in SQL.

For instance, the following statement finds the prices of the 1000 order lines with the biggest discounts:

```
SPARQL
define sql:signal-void-variables 1
prefix tpch: <http://www.openlinksw.com/schemas/tpch#>
SELECT ?line ?discount (?extendedprice * (1 - ?discount)) as ?finalprice
FROM <http://localhost.localdomain:8310/tpch>
WHERE
  {
    ?line a tpch:lineitem ;
    tpch:lineextendedprice ?extendedprice ;
    tpch:linediscount ?discount .
  }
ORDER BY DESC (?extendedprice * ?discount)
LIMIT 1000
```

After ensuring that this query works correctly, we can use it to answer more complex questions. Imagine that we want to find out how big the customers are who have received the biggest discounts.

```
SPARQL
define sql:signal-void-variables 1
prefix tpch: <http://www.openlinksw.com/schemas/tpch#>
SELECT ?cust sum(?extendedprice2 * (1 - ?discount2)) max (?bigdiscount)
FROM <http://localhost.localdomain:8310/tpch>
WHERE
  {
    {
      SELECT ?line (?extendedprice * ?discount) as ?bigdiscount
      WHERE {
        ?line a tpch:lineitem ;
          tpch:lineextendedprice ?extendedprice ;
          tpch:linediscount ?discount . }
      ORDER BY DESC (?extendedprice * ?discount)
      LIMIT 1000
    }
    ?line tpch:has_order ?order .
    ?order tpch:has_customer ?cust .
    ?cust tpch:customer_of ?order2 .
    ?order2 tpch:order_of ?line2 .
    ?line2 tpch:lineextendedprice ?extendedprice2 ;
      tpch:linediscount ?discount2 .
  }
ORDER BY (SUM(?extendedprice2 * (1 - ?discount2)) / MAX (?bigdiscount))
```

The inner select finds the 1000 biggest (in absolute value) discounts and their order lines. For each line we find orders of it, and the customer. For each customer found, we find all the orders he made and all the lines of each of the orders (variable ?line2).

Note that the inner select does not contain FROM clauses. It is not required because the inner select inherits the access permissions of all the outer queries. It is also important to note that the internal variable bindings of the subquery are not visible in the outer query; only the result set variables are bound. Similarly, variables bound in the outer query are not accessible to the subquery.

Note also the declaration *define sql:signal-void-variables 1* that forces the SPARQL compiler to signal errors if some variables cannot be bound due to misspelt names or attempts to make joins across disjoint domains. These diagnostics are especially important when the query is long.

Expressions in Triple Patterns

In addition to expressions in filters and result sets, Virtuoso allows the use of expressions in triples of a CONSTRUCT pattern or WHERE pattern - an expression can be used instead of a constant or a variable name for a subject, predicate or object. When used in this context, the expression is surrounded by backquotes.

Example: With a WHERE Clause:

The following example returns all the distinct 'fragment' parts of all subjects in all graphs that have some predicate whose value is equal to 2+2.

```
SQL>SPARQL SELECT distinct (bif:subseq (?s, bif:strchr (?s, '#')))
   WHERE {
     graph ?g {
       ?s ?p `2+2` .
       FILTER (! bif:isnull (bif:strchr (?s, '#') ) )
     } };

callret
VARCHAR
----------
#four
```

Inside a WHERE part, every expression in a triple pattern is replaced with new variable and a filter expression is added to the enclosing group. The new filter is an equality of the new variable and the expression. Hence the sample above is identical to:

```
SPARQL
SELECT distinct (bif:subseq (?s, bif:strchr (?s, '#')))
   WHERE {
     graph ?g {
       ?s ?p ?newvariable .
       FILTER (! bif:isnull (bif:strchr (?s, '#') ) )
       FILTER (?newvariable = (2+2)) .
     } }
```

Example: With CONSTRUCT

```
CONSTRUCT {
  <http://bio2rdf.org/interpro:IPR000181>
<http://bio2rdf.org/ns/bio2rdf#hasLinkCount>
`(SELECT (count(?s)) as ?countS
   WHERE { ?s ?p <http://bio2rdf.org/interpro:IPR000181> })` }
WHERE { ?s1 ?p1 ?o1 } limit 1
```

The result should be:

```
<http://bio2rdf.org/interpro:IPR000181> <http://bio2rdf.org/ns/bio2rdf#hasLinkCount> "0"^^<http://www.w3.
```

Example: Inserting into a graph using an expression

```
SQL>SPARQL insert into graph <http://MyNewGraph.com/> {
<http://bio2rdf.org/interpro:IPR000181>
<http://bio2rdf.org/ns/bio2rdf#hasLinkCount>
  `(SELECT (count(?s)) as ?countS
    WHERE { ?s ?p <http://bio2rdf.org/interpro:IPR000181> })` }
 WHERE { ?s1 ?p1 ?o1 } limit 1  ;

callret-0
VARCHAR
_______________________________________________________________________________

Insert into <http://MyNewGraph.com/>, 1 triples -- done

1 Rows. -- 30 msec.
```

16.4. RDF Graphs Security

16.4.1. RDF Graph Groups

In some cases, the data-set of a SPARQL query is not known at compile time. It is possible to pass IRIs of source graphs via parameters, but the method is not perfect as:

- not all protocols are suitable for parameter passing, and no one is an interoperable standard
- passing list of IRIs as a parameter will usually require the use of Virtuoso-specific functions in the text of SPARQL query, that's bad for some query builders.
- lack of knowledge about actual graphs may damage query optimization

It would be nice to create named lists of graphs and a clause like "SELECT from all graph names of the specified list". *"Graph groups"* serve for this purpose. That is Virtuoso-specific SPARQL extension that let create a named list of IRIs such that if name of the list is used in *FROM* clause like *IRI* of default graph then it is equivalent to list of *FROM* clauses, one clause for each item of the list.

Internally, descriptions of graph groups are kept in two tables: *Table of graph groups:*

```
create table DB.DBA.RDF_GRAPH_GROUP (
  RGG_IID IRI_ID not null primary key,  -- IRI ID of RGG_IRI field
  RGG_IRI varchar not null,             -- Name of the group
  RGG_MEMBER_PATTERN varchar,           -- Member IRI pattern
  RGG_COMMENT varchar                   -- Comment
  )
create index RDF_GRAPH_GROUP_IRI on DB.DBA.RDF_GRAPH_GROUP (RGG_IRI)
;
```

Table of contents of groups:

```
create table DB.DBA.RDF_GRAPH_GROUP_MEMBER (
  RGGM_GROUP_IID IRI_ID not null,       -- IRI_ID of the group
  RGGM_MEMBER_IID IRI_ID not null,      -- IRI_ID of the group member
  primary key (RGGM_GROUP_IID, RGGM_MEMBER_IID)
  )
;
```

Fields *RGG_MEMBER_PATTERN* and *RGG_COMMENT* are not used by system internals but applications may wish to write their data there for future reference. *RGG_COMMENT* is supposed to be human-readable description of the group and *RGG_MEMBER_PATTERN* may be useful for functions that automatically add IRIs of a given graph to all graph groups such that the graph IRI string match *RGG_MEMBER_PATTERN* regexp pattern.

A dictionary of all groups and their members is cached in memory for fast access. Due to this reason, applications may read these tables and modify *RGG_MEMBER_PATTERN* and *RGG_COMMENT* if needed but not change other fields directly. The following API procedures makes changes in a safe way:

```
DB.DBA.RDF_GRAPH_GROUP_CREATE (
  in group_iri varchar,
  in quiet integer,
  in member_pattern varchar := null,
  in comment varchar := null)
```

That creates a new empty graph group. An error is signaled if the group exists already and quiet parameter is zero.

```
DB.DBA.RDF_GRAPH_GROUP_INS (in group_iri varchar, in memb_iri varchar)
DB.DBA.RDF_GRAPH_GROUP_DEL (in group_iri varchar, in memb_iri varchar)
```

These two are to add or remove member to an existing group. Double insert or removal of not a member will not signal errors, but missing group will.be signaled.

```
DB.DBA.RDF_GRAPH_GROUP_DROP (
  in group_iri varchar,
  in quiet integer)
```

That removes graph group. An error is signaled if the group did not exist before the call and quiet parameter is zero.

Graph groups are *"macro-expanded"* only in FROM clauses and have no effect on FROM NAMED or on GRAPH <IRI> {...} . Technically, it is not prohibited to use an IRI as both plain graph IRI and graph group IRI in one storage but this is confusing and is not recommended.

Graph groups can not be members of other graph groups, i.e. the IRI of a graph group can appear in the list of members of some group but it will be treated as plain graph IRI and will not cause recursive expansion of groups.

16.4.2. NOT FROM and NOT FROM NAMED Clauses

In addition to standard FROM and FROM NAMED clauses, Virtuoso extends SPARQL with NOT FROM and NOT FROM NAMED clauses of "opposite" meaning.

```
SELECT ... NOT FROM <x> ... WHERE {...}
```

means "SELECT FROM other graphs, but not from the given one". This is especially useful because NOT FROM supports graph groups (NOT FROM NAMED supports only plain graphs). So if

```
<http://example.com/users/private>
```

is a graph group of all graphs with confidential data about users then

```
SELECT * NOT FROM <http://example.com/users/private> WHERE {...}
```

will be restricted only to insecure data.

NOT FROM overrides any FROM and NOT FROM NAMED overrides any FROM NAMED, the order of clauses in the query text is not important.

The SPARQL web service endpoint configuration string may contain pragmas *input:default-graph-exclude* and *input:named-graph-exclude* that become equivalent to NOT FROM and NOT FROM NAMED clauses like *input:default-graph-uri* and *input:named-graph-uri* mimics FROM and FROM NAMED.

16.4.3. Graph-Level Security

Virtuoso supports graph-level security for "physical" RDF storage. That is somewhat similar to table access permissions in SQL. However, the difference between SPARQL and SQL data models results in totally different style of security administration. In SQL, when new application is installed it comes with its own set of tables and every query in its code explicitly specifies tables in use. Security restrictions of two applications interfere only if applications knows each other and are supposedly designed to cooperate. It is possible to write an application that will get list of available tables and retrieve data from any given table but that is a special case and it usually requires DBA privileges.

In SPARQL, data of different applications shares one table and the query language allows to select data of all applications at once. This feature makes SPARQL convenient for cross-application data integration. At the same time, that become a giant security hole if any sensitive data are stored.

A blind copying SQL security model to SPARQL domain would result in significant loss of performance or weak security or even both problems at the same time. That is why SPARQL model is made much more restrictive, even if it becomes inconvenient for some administration tasks.

Graph-level security does not replace traditional SQL security. A user should become member of appropriate group (`SPARQL_SELECT` , `SPARQL_SPONGE` or `SPARQL_UPDATE`) in order to start using its graph-level privileges.

16.4.4. Graph-Level Security and SQL

SPARQL-level graph security is sufficient for SPARQL client operating over HTTP. It is not sufficient for SQL clients due to the fact that graph level security is baked into the SPARQL compiler, not by an SQL compiler.

The Virtuoso SPARQL compiler analyzes the graph-level permissions of a user (an identity principal named using an identifier e.g., WebID or NetID). For each triple pattern or graph group pattern the compiler adds an implicit FILTER () that ensures that

appropriate privileges are granted on target named graphs to a given user. Ultimately, these FILTERs becomes part of the generated SQL code processed against the RDF_QUAD and related RDF data management system tables.

SQL users accessing Virtuoso via ODBC, JDBC, ADO.NET, and OLE-DB connections have the ability to execute arbitrary SQL code via stored procedures, subject to SQL level privileges on target Tables and Views which provides a point of vulnerability to the RDF system tables (RDF_QUAD and others). To close this vulnerability, the SQL compiler restricts SQL connection access, in regards to RDF system tables, to members of `the SPARQL_SELECT_RAW` group.

Note: `SPARQL_SELECT_RAW` group is a feature applicable to Virtuoso 7.5 or higher.

Graph-Level Security and SQL Usage Example

The following example demonstrates how to grant `SPARQL_SELECT_RAW` to a Virtuoso SQL user:

```
SQL> DB.DBA.USER_CREATE ('John', 'John');
Done. -- 0 msec.
SQL> GRANT SPARQL_SELECT to "John";
Done. -- 0 msec.
SQL> GRANT SPARQL_SELECT_RAW to "John";
Done. -- 0 msec.
```

16.4.5. Understanding Default Permissions

In relational database, default permissions are trivial. DBA is usually the only account that can access any table for both read and write. Making some table public or private does not affect applications that do not refer that table in the code. Tables are always created before making security restrictions on them.

Chances are very low that an application will unintentionally create some table and fill in with confidential data. There are no unauthenticated users, any client has some user ID and no one user is "default user" so permissions of any two users are always independent.

SPARQL access can be anonymous and graphs can be created during routine data manipulation. For anonymous user, only public resources are available. Thus "default permissions" on some or all graphs are actually permissions of "nobody" user, (the numeric ID of this user can be obtained by http_nobody_uid() function call). As a consequence, there's a strong need in "default permission" for a user, this is the only way to specify what to do with all graphs that does not exist now it might exist in some future.

An attempt to make default permissions wider than specific is always potential security hole in SPARQL, so this is strictly prohibited.

Four sorts of access are specified by four bits of an integer "permission bit-mask", plain old UNIX style:

- Bit 1 permits read access.
- Bit 2 permits write access via SPARUL and it's basically useless without bit 1 set.
- Bit 4 permits write access via "RDF sponge" methods and it's basically useless without bits 1 and 2 set.
- Bit 8 allows to obtain list of members of graph group; an IRI can be used as graph IRI and as graph group IRI at the same time so bit 8 can be freely combined with any of bits 1, 2 or 4.

Note that obtaining the list of members of a graph group does not grant any access permissions to triples from member graphs. It is quite safe to mix secure and public graphs in one graph group.

When a SPARQL query should check whether a given user have permission to access a given graph then the order of checks is as follows:

1. permissions of the user on the specific graph;
2. default permissions of the user on all graphs;
3. public permissions on the specific graph;
4. public permissions on all graphs

If no one above mentioned permission is set then the access is "read/write/sponge/list".

For "nobody" user, steps 3 and 4 become exact copies of steps 1 and 2 so they are skipped.

16.4.6. Initial Configuration of SPARQL Security

It is convenient to configure the RDF storage security by adding restrictions in the order inverse to the order of checks:

1. Step 1: Set public permissions on all graphs to the most restricted level of any application that will be installed. So if any single graph will be unreadable for public, then public permissions on all graphs should be set to 0 or 8.
2. Step 2: Public permissions on "insecure" graphs should be set. So if the database contains DBpedia or WordNet or some other data of Linking Open Data project then public permissions for that graphs may be set to 1.
3. Step3: Configure trusted users, such as administrative DBA-like accounts, and to specify their permissions on all graphs.
4. Step 4: Some additional right can be granted to some specific users on some specific graphs.

Note that there's no need to permit something to DBA itself, because DBA's default permissions are set automatically.

Configuring New User

1. Step 1: Grant SPARQL_SELECT, SPARQL_SPONGE or SPARQL_UPDATE to the user.
2. Step 2: Set user's permissions on all graphs.
3. Step 3: Grant rights on some specific graphs.

Example: Blogs and Resource Sharing

The following example demonstrates usage of the following functions:

- `DB.DBA.RDF_DEFAULT_USER_PERMS_SET (uname, permissions, set_private);`
- `DB.DBA.RDF_DEFAULT_USER_PERMS_DEL (uname, set_private);`
- `DB.DBA.RDF_GRAPH_USER_PERMS_SET (graph_iri, uname, permissions);`
- `DB.DBA.RDF_GRAPH_USER_PERMS_DEL (graph_iri, uname);`
- `DB.DBA.RDF_ALL_USER_PERMS_DEL (uname, uid);`

Consider a "groupware" application that let users create personal resources with access policies.

```
-- First, create few users, in alphabetical order.
SQL> DB.DBA.USER_CREATE ('Anna', 'Anna');
Done. -- 0 msec.
SQL> DB.DBA.USER_CREATE ('Brad', 'Brad');
Done. -- 0 msec.
SQL> DB.DBA.USER_CREATE ('Carl', 'Carl');
Done. -- 16 msec.
SQL> GRANT SPARQL_UPDATE to "Anna";
Done. -- 0 msec.
SQL> GRANT SPARQL_UPDATE to "Brad";
Done. -- 0 msec.
SQL> GRANT SPARQL_UPDATE to "Carl";
Done. -- 0 msec.

-- At least some data are supposed to be confidential, thus the whole storage becomes confidential.
SQL> DB.DBA.RDF_DEFAULT_USER_PERMS_SET ('nobody', 0);
Done. -- 16 msec.

-- Moreover, no one of created users have access to all graphs (even for reading).
SQL> DB.DBA.RDF_DEFAULT_USER_PERMS_SET ('Anna', 0);
Done. -- 0 msec.
SQL> DB.DBA.RDF_DEFAULT_USER_PERMS_SET ('Brad', 0);
Done. -- 0 msec.
SQL> DB.DBA.RDF_DEFAULT_USER_PERMS_SET ('Carl', 0);
Done. -- 0 msec.

-- Add Anna's and Brad's private graph to the group http://www.openlinksw.com/schemas/virtrdf#PrivateGrap
SQL> DB.DBA.RDF_GRAPH_GROUP_INS ('http://www.openlinksw.com/schemas/virtrdf#PrivateGraphs', 'http://examp
SQL> DB.DBA.RDF_GRAPH_GROUP_INS ('http://www.openlinksw.com/schemas/virtrdf#PrivateGraphs', 'http://examp

-- Anna's graphs:
--insert simple data in Anna's personal system graph:
SQL> SPARQL INSERT IN <http://example.com/Anna/system> { <Anna-system> a <secret> };
Done. -- 31 msec.

--insert simple data in Anna's private graph:
```

```
SQL> SPARQL INSERT IN <http://example.com/Anna/private> { <Anna-private> a <secret> };
Done. -- 0 msec.

-- Anna can only read her personal system data graph.
SQL> DB.DBA.RDF_GRAPH_USER_PERMS_SET ('http://example.com/Anna/system', 'Anna', 1);
Done. -- 0 msec

-- Anna can read and write her private data graph.
SQL> DB.DBA.RDF_GRAPH_USER_PERMS_SET ('http://example.com/Anna/private', 'Anna', 3);
Done. -- 0 msec

-- Brad's graphs:
-- insert simple data in Brad's personal system graph:
SQL> SPARQL INSERT IN <http://example.com/Brad/system> { <Brad-system> a <secret> };
Done. -- 0 msec

-- insert simple data in Brad's private graph:
SQL> SPARQL INSERT IN <http://example.com/Brad/private> { <Brad-private> a <secret> };
Done. -- 0 msec

-- Brad can only read his personal system data graph.
SQL> DB.DBA.RDF_GRAPH_USER_PERMS_SET ('http://example.com/Brad/system', 'Brad', 1);
Done. -- 0 msec

-- Brad can read and write his private data graph.
SQL> DB.DBA.RDF_GRAPH_USER_PERMS_SET ('http://example.com/Brad/private', 'Brad', 3);
Done. -- 0 msec

-- Friends graphs:
SQL> SPARQL INSERT IN <http://example.com/Anna/friends> { <Anna-friends> foaf:knows 'Brad' };
Done. -- 14 msec
SQL> SPARQL INSERT IN <http://example.com/Brad/friends> { <Brad-friends> foaf:knows 'Anna' };
Done. -- 15 msec

-- Anna and Brad are friends and can read each others notes for friends.
SQL> DB.DBA.RDF_GRAPH_USER_PERMS_SET ('http://example.com/Anna/friends', 'Anna', 3);
SQL> DB.DBA.RDF_GRAPH_USER_PERMS_SET ('http://example.com/Anna/friends', 'Brad', 1);
SQL> DB.DBA.RDF_GRAPH_USER_PERMS_SET ('http://example.com/Brad/friends', 'Brad', 3);
SQL> DB.DBA.RDF_GRAPH_USER_PERMS_SET ('http://example.com/Brad/friends', 'Anna', 1);

-- BubbleSortingServicesInc graph
SQL> SPARQL INSERT IN <http://example.com/BubbleSortingServicesInc> { <BubbleSortingServicesInc-info> a <
Done. -- 31 msec

-- Brad and Carl share write access to graph of his company.
SQL> DB.DBA.RDF_GRAPH_USER_PERMS_SET ('http://example.com/BubbleSortingServicesInc', 'Brad', 3);
SQL> DB.DBA.RDF_GRAPH_USER_PERMS_SET ('http://example.com/BubbleSortingServicesInc', 'Carl', 3);

-- Anna's blog
SQL> SPARQL INSERT IN <http://example.com/Anna/blog> { <Anna-blog> a <my-blog> };

-- Anna writes a blog for public.
SQL> DB.DBA.RDF_GRAPH_USER_PERMS_SET ('http://example.com/Anna/blog', 'Anna', 3);
SQL> DB.DBA.RDF_GRAPH_USER_PERMS_SET ('http://example.com/Anna/blog', 'nobody', 1);

-- DBpedia is public read and local discussion wiki is readable and writable.
SQL> DB.DBA.RDF_GRAPH_USER_PERMS_SET ('http://dbpedia.org/', 'nobody', 1);
SQL> DB.DBA.RDF_GRAPH_USER_PERMS_SET ('http://example.com/wiki', 'nobody', 3);
SQL> DB.DBA.RDF_GRAPH_USER_PERMS_SET ('http://example.com/public', 'nobody', 3);

-- Graph groups have its own security.
SQL> DB.DBA.RDF_GRAPH_GROUP_CREATE ('http://example.com/Personal', 1);
SQL> DB.DBA.RDF_GRAPH_GROUP_INS ('http://example.com/Personal', 'http://example.com/Anna/system');
SQL> DB.DBA.RDF_GRAPH_GROUP_INS ('http://example.com/Personal', 'http://example.com/Anna/private');
SQL> DB.DBA.RDF_GRAPH_GROUP_INS ('http://example.com/Personal', 'http://example.com/Brad/system');
SQL> DB.DBA.RDF_GRAPH_GROUP_INS ('http://example.com/Personal', 'http://example.com/Brad/private');
SQL> DB.DBA.RDF_GRAPH_USER_PERMS_SET ('http://example.com/Personal', 'Anna', 8);
SQL> DB.DBA.RDF_GRAPH_USER_PERMS_SET ('http://example.com/Personal', 'Brad', 8);

-- See as dba user what is in the <http://example.com/Personal> graph:
SQL> SPARQL
SELECT *
FROM <http://example.com/Personal>
```

```
WHERE { ?s ?p ?o } ;
s               p                                                   o
VARCHAR         VARCHAR                                             VARCHAR
_______________________________________________________________________________

Anna-system     http://www.w3.org/1999/02/22-rdf-syntax-ns#type     secret
Anna-private    http://www.w3.org/1999/02/22-rdf-syntax-ns#type     secret
Brad-system     http://www.w3.org/1999/02/22-rdf-syntax-ns#type     secret
Brad-private    http://www.w3.org/1999/02/22-rdf-syntax-ns#type     secret

4 Rows. -- 32 msec.

-- See as Anna user what is in the <http://example.com/Personal> graph:
SQL> reconnect Anna;
Connected to OpenLink Virtuoso
Driver: 06.04.3132 OpenLink Virtuoso ODBC Driver

SQL> SPARQL
SELECT *
FROM <http://example.com/Personal>
WHERE { ?s ?p ?o };
s               p                                                   o
VARCHAR         VARCHAR                                             VARCHAR
_______________________________________________________________________________

Anna-system     http://www.w3.org/1999/02/22-rdf-syntax-ns#type     secret
Anna-private    http://www.w3.org/1999/02/22-rdf-syntax-ns#type     secret

-- See as Brad user what is in the <http://example.com/Personal> graph:
SQL> reconnect Brad;
Connected to OpenLink Virtuoso
Driver: 06.04.3132 OpenLink Virtuoso ODBC Driver

SQL> SPARQL SELECT *
FROM <http://example.com/Personal>
WHERE { ?s ?p ?o };
s               p                                                   o
VARCHAR         VARCHAR                                             VARCHAR
_______________________________________________________________________________

Brad-system     http://www.w3.org/1999/02/22-rdf-syntax-ns#type     secret
Brad-private    http://www.w3.org/1999/02/22-rdf-syntax-ns#type     secret

-- See as Anna user what is in Brad's friends graph <http://example.com/Brad/friends>:
SQL> reconnect Anna;
Connected to OpenLink Virtuoso

SQL> SPARQL SELECT *
FROM <http://example.com/Brad/friends>
WHERE { ?s ?p ?o };
s               p                                   o
VARCHAR         VARCHAR                             VARCHAR
_____________________________________________________________

Brad-friends    http://xmlns.com/foaf/0.1/knows     Anna
1 Rows. -- 0 msec.

-- Remove Anna's read permissions on Brad's notes:
SQL> reconnect dba;
SQL> RDF_GRAPH_USER_PERMS_DEL('http://example.com/Brad/friends','Anna');

-- See again as Anna user what is in Brad's friends graph <http://example.com/Brad/friends>:
SQL> reconnect Anna;
Connected to OpenLink Virtuoso

SQL> SPARQL SELECT *
FROM <http://example.com/Brad/friends>
WHERE { ?s ?p ?o };
s               p                                   o
VARCHAR         VARCHAR                             VARCHAR
_____________________________________________________________

0 Rows. -- 0 msec.
```

```
SQL> SPARQL
SELECT *
FROM <http://example.com/Anna/blog>
WHERE { ?s ?p ?o } ;

s             p                                                 o
VARCHAR       VARCHAR                                           VARCHAR
_______________________________________________________________________________

Anna-blog     http://www.w3.org/1999/02/22-rdf-syntax-ns#type  my-blog

1 Rows. -- 16 msec.

SQL> SPARQL
SELECT *
FROM <http://example.com/Anna/friends>
WHERE { ?s ?p ?o } ;

s             p                               o
VARCHAR       VARCHAR                         VARCHAR
_______________________________________________________________________________

Anna-friends  http://xmlns.com/foaf/0.1/knows  Brad
1 Rows. -- 16 msec.

-- Remove all the setting of Brad's permissions, both default permissions and
-- permissions on specific graphs.
-- Note: 142 is example id of Brads U_ID in DB.DBA.SYS_USERS table

SQL> reconnect dba;
Connected to OpenLink Virtuoso
Driver: 06.04.3132 OpenLink Virtuoso ODBC Driver

SQL> DB.DBA.RDF_ALL_USER_PERMS_DEL('Brad',142);
Done. -- 0 msec.

SQL> reconnect Brad;
Connected to OpenLink Virtuoso
Driver: 06.04.3132 OpenLink Virtuoso ODBC Driver

SQL> SPARQL
SELECT *
FROM <http://example.com/Anna/friends>
WHERE { ?s ?p ?o } ;
s          p          o
VARCHAR    VARCHAR    VARCHAR
_______________________________________________________________________________

0 Rows. -- 16 msec.

-- Check what Carl can see --
SQL> reconnect Carl;
Connected to OpenLink Virtuoso
Driver: 06.04.3132 OpenLink Virtuoso ODBC Driver

SQL> SPARQL
SELECT *
FROM <http://example.com/BubbleSortingServicesInc>
WHERE { ?s ?p ?o } ;

s                            p                                                 o
VARCHAR                      VARCHAR                                           VARCHAR
_______________________________________________________________________________

BubbleSortingServicesInc-info  http://www.w3.org/1999/02/22-rdf-syntax-ns#type  info

1 Rows. -- 0 msec.

SQL> SPARQL
SELECT *
FROM <http://example.com/Anna/private>
WHERE { ?s ?p ?o } ;
```

```
s               p                                                   o
VARCHAR         VARCHAR                                             VARCHAR
_______________________________________________________________________________

0 Rows. -- 16 msec.

-- let Carl read everything:
SQL> reconnect dba;
Connected to OpenLink Virtuoso
Driver: 06.04.3132 OpenLink Virtuoso ODBC Driver

SQL> DB.DBA.RDF_DEFAULT_USER_PERMS_SET ('Carl', 1, 1);
Done. -- 0 msec

SQL> reconnect Carl;
Enter password for Carl :
Connected to OpenLink Virtuoso
Driver: 06.04.3132 OpenLink Virtuoso ODBC Driver

SQL> SPARQL
SELECT *
FROM <http://example.com/Anna/private>
WHERE { ?s ?p ?o } ;
s               p                                                   o
VARCHAR         VARCHAR                                             VARCHAR
_______________________________________________________________________________

Anna-private    http://www.w3.org/1999/02/22-rdf-syntax-ns#type    secret

1 Rows. -- 16 msec.

-- Remove Carl's default permissions:
SQL> reconnect dba;
Connected to OpenLink Virtuoso
Driver: 06.04.3132 OpenLink Virtuoso ODBC Driver

SQL> DB.DBA.RDF_DEFAULT_USER_PERMS_DEL('Carl', 1);
Done. -- 0 msec.

SQL> reconnect Carl;
Enter password for Carl :
Connected to OpenLink Virtuoso
Driver: 06.04.3132 OpenLink Virtuoso ODBC Driver

SQL> SPARQL
SELECT *
FROM <http://example.com/Anna/private>
WHERE { ?s ?p ?o } ;
s               p                                                   o
VARCHAR         VARCHAR                                             VARCHAR
_______________________________________________________________________________

0 Rows. -- 16 msec.
```

16.4.7. Application Callbacks for Graph Level Security

In some cases, different applications should provide different security for different users. Two SPARQL pragmas are provided for this purpose:

- Pragma sql:gs-app-callback is to specify Virtuoso/PL callback function that return permission bits for given graph.
- Pragma sql:gs-app-uid is to specify application-specific user ID that is some string that is passed to the callback "as is".

The name of callback is always DB.DBA.SPARQL_GS_APP_CALLBACK_nnn, where nnn is value of sql:gs-app-callback.

The callback is called only if the application has access to the graph in question so it may restrict the caller's account but not grant more permissions.

Example

Let user of application get full access to graphs whose IRIs contain user's name in path. In addition, let all of them permission to use all graph groups and let the "moderator" user read everything.

```
reconnect "dba";

create function DB.DBA.SPARQL_GS_APP_CALLBACK_TEST (in g_iid IRI_ID, in app_uid varchar) returns integer
{
  declare g_uri varchar;
-- A fake IRI ID #i0 is used to mention account's default permissions for all graphs.
  if (#i0 = g_iid)
    {
      if ('moderator' = app_uid)
        return 9; -- Moderator can read and list everything.
      return 8; -- Other users can list everything.
    }
  g_uri := id_to_iri (g_iid);
  if (strstr (g_uri, '/' || app_uid || '/'))
    return 15; -- User has full access to "his" graph.
  return 8; -- User can list any given graph group.
}
;

SPARQL
define sql:gs-app-callback "TEST"
define sql:gs-app-uid "Anna"
SELECT ?g ?s WHERE { ?s <p> ?o }
;
```

16.4.8. Graph-level security and sponging

In some cases the sponged data contains private information for instances cartridges like Facebook, etc. To perform private sponging, Virtuoso offers *get:private* pragma:

```
define get:private ""
or
define get:private <graph_group_IRI>
```

When used for sponging graph X, it adjusts graph-level security of graph X (and of graph_group_IRI, if specified) so that X becomes a privately accessible graph of the user who sponges the X and if graph_group_IRI is specified then X becomes accessible to users that can access graph_group_IRI with permissions like permissions they have on graph_group_IRI.

The exact rules are as following:

- If graph is virtrdf: then an error is signaled.
- If graph name is an IRI of handshaked web service endpoint or "public IRI" of a handshaked web service endpoint then an error is signaled.
- If access is public by default even for private graphs then an error is signaled and sponging is not tried.
- If default is "no access" but someone (other than current user) has specifically granted read access to the graph in question AND current user is not dba AND current user has no bit 32 permission on this graph then an error is signaled.
- If read access is public by default for world and disabled for private graphs then the graph to be sponged is added to the group of private graphs.
- If current user is not DBA, current user gets granted read+write+sponge+admin access to the graph to be sponged. In addition, current user gets special permission bit 32, indicating that the graph is made by private sponge of this specific user.
- If the value of get:private is an IRI then:
    - The IRI is supposed to be an IRI of "plain" graph group, error is signaled in case of nonexising graph group, group of private graphs or group of graphs to be replicated.
    - The graph is added to that group
    - Each non-dba user that can get list of files of the group will get permissions for the loaded graph equal to permissions they have on graph group minus "list" permission.

Example Performing Sponging on a entirely confidential database using get:private pragma

The following example demonstrates how private sponging using get:private pragma works for entirely confidential database.

Note : Please take in mind that the steps from below will change the security of any existing database, thus the example scenario should be performed on a empty db.

1. Create few users in alphabetical order:

```
DB.DBA.USER_CREATE ('Anna', 'Anna');
DB.DBA.USER_CREATE ('Brad', 'Brad');
DB.DBA.USER_CREATE ('Carl', 'Carl');
```

2. Set to Anna, Brad and Carl SPARQL SELECT, UPDATE and SPONGE permissions:

```
grant SPARQL_SELECT to "Anna";
grant SPARQL_SELECT to "Brad";
grant SPARQL_SELECT to "Carl";

grant SPARQL_UPDATE to "Anna";
grant SPARQL_UPDATE to "Brad";
grant SPARQL_UPDATE to "Carl";

grant SPARQL_SPONGE to "Anna";
grant SPARQL_SPONGE to "Brad";
grant SPARQL_SPONGE to "Carl";
```

3. Set specific privileges to given graphs for specifics users: Catering for the fact that some datasets are supposed to be confidential, thus the whole quad storage is set to confidential. Then specific privileges can be assigned to specific graphs for specific users:

```
DB.DBA.RDF_DEFAULT_USER_PERMS_SET ('nobody', 0);
```

4. Set specific privileges: assuming for users Anna, Brad and Carl none of these individual has any kind of global access to graphs:

```
DB.DBA.RDF_DEFAULT_USER_PERMS_SET ('Anna', 0);
DB.DBA.RDF_DEFAULT_USER_PERMS_SET ('Brad', 0);
DB.DBA.RDF_DEFAULT_USER_PERMS_SET ('Carl', 0);
```

5. Assuming the following four sorts of access that are specified by four bits of an integer "permission bit-mask", following plain old UNIX style:

    - Bit 1 permits read access.
    - Bit 2 permits write access via SPARUL and is basically useless without bit 1 set.
    - Bit 4 permits write access via "RDF Network Resource Fetch" methods and is basically useless without bits 1 and 2 set.
    - Bit 8 allows retrieval of the list of members of a graph group. An IRI can be used as a graph IRI and as a graph group IRI at the same time, so bit 8 can be freely combined with any of bits 1, 2 or 4.
    - In the statements from below should be considered:
        - "15 = 8+4+2+1" -- i.e. combining all the four sorts of access FROM above
        - "9 = 8 + 1" -- i.e. read access + access to retrieve the list of members for a given graph group

```
-- Create Graph Group for Anna and set privileges:
DB.DBA.RDF_GRAPH_GROUP_CREATE ('urn:Anna:Sponged:Data', 1);
DB.DBA.RDF_GRAPH_USER_PERMS_SET ('urn:Anna:Sponged:Data', 'Anna', 15);
DB.DBA.RDF_GRAPH_USER_PERMS_SET ('urn:Anna:Sponged:Data', 'Brad', 9);
DB.DBA.RDF_GRAPH_USER_PERMS_SET ('urn:Anna:Sponged:Data', 'Carl', 9);

-- Create Graph Group for Brad and set privileges:
DB.DBA.RDF_GRAPH_GROUP_CREATE ('urn:Brad:Sponged:Data', 1);
DB.DBA.RDF_GRAPH_USER_PERMS_SET ('urn:Brad:Sponged:Data', 'Anna', 9);
DB.DBA.RDF_GRAPH_USER_PERMS_SET ('urn:Brad:Sponged:Data', 'Brad', 15);
DB.DBA.RDF_GRAPH_USER_PERMS_SET ('urn:Brad:Sponged:Data', 'Carl', 9);

-- Create Graph Group for Carl and set privileges:
DB.DBA.RDF_GRAPH_GROUP_CREATE ('urn:Carl:Sponged:Data', 1);
DB.DBA.RDF_GRAPH_USER_PERMS_SET ('urn:Carl:Sponged:Data', 'Anna', 9);
DB.DBA.RDF_GRAPH_USER_PERMS_SET ('urn:Carl:Sponged:Data', 'Brad', 9);
DB.DBA.RDF_GRAPH_USER_PERMS_SET ('urn:Carl:Sponged:Data', 'Carl', 15);
```

6. Examples with invalid graph group names:

a. Example with Non-existing Graph Group:

```
-- An error for non-existing Graph group <http://nosuch/> will be raised.

SPARQL
  DEFINE get:soft "replacing"
  DEFINE get:private <http://nosuch/>
  SELECT *
  FROM <http://example.com/>
  WHERE
    { ?s ?p ?o };
```

b. Example with "virtrdf:PrivateGraphs" graph group which is reserved for system usage:

```
-- An error for attempt to add a graph to special
-- graph group <http://www.openlinksw.com/schemas/virtrdf#PrivateGraphs> will be raised.

SPARQL
  DEFINE get:soft "replacing"
  DEFINE get:private virtrdf:PrivateGraphs
  SELECT * FROM <http://example.com/>
  WHERE
    { ?s ?p ?o };
```

c. Example with "virtrdf:rdf_repl_graph_group" graph group which is reserved for system usage:

```
-- An error for attempt to add a graph to special
-- graph group <http://www.openlinksw.com/schemas/virtrdf#rdf_repl_graph_group> will be rai

SPARQL
  DEFINE get:soft "replacing"
  DEFINE get:private virtrdf:rdf_repl_graph_group
  SELECT * FROM <http://example.com/>
  WHERE
    { ?s ?p ?o };
```

7. Examples to check Anna's sponging permissions on different graph groups:

a. Example for adding graph to Anna's graph group <urn:Anna:Sponged:Data>:

```
-- No error will be raised as Anna has the efficient rights for graph group <urn:Anna:Spong

reconnect "Anna";

SPARQL
  DEFINE get:soft "replacing"
  DEFINE get:private <urn:Anna:Sponged:Data>
  SELECT *
  FROM <http://example.com/anna/>
  WHERE
    { ?s ?p ?o };
```

b. Example for adding graph to Brad's graph group <urn:Brad:Sponged:Data>:

```
-- An error will be raised because "Anna" has not enough rights on that group

reconnect "Anna";

SPARQL
  DEFINE get:soft "replacing"
  DEFINE get:private <urn:Brad:Sponged:Data>
  SELECT * FROM <http://example.com/>
  WHERE
    { ?s ?p ?o };
```

c. Example for adding graph to Carl's graph group <urn:Carl:Sponged:Data>:

```
-- An error will be raised because "Anna" has not enough rights on that group

reconnect "Anna";
```

```
SPARQL
  DEFINE get:soft "replacing"
  DEFINE get:private <urn:Carl:Sponged:Data>
  SELECT *
  FROM <http://example.com/>
  WHERE
    { ?s ?p ?o };
```

8. Examples check Brad's sponging permissions on different graph groups:

   a. Example for adding graph to Anna's graph group <urn:Anna:Sponged:Data>:

```
-- An error will be raised because "Brad" has not enough rights on that group

reconnect "Brad";

SPARQL
  DEFINE get:soft "replacing"
  DEFINE get:private <urn:Anna:Sponged:Data>
  SELECT *
  FROM <http://example.com/>
  WHERE
    { ?s ?p ?o };
```

   b. Example for adding graph to Brad's graph group <urn:Brad:Sponged:Data>:

```
-- No error will be raised as Brad has the efficient rights for graph group <urn:Brad:Spong

reconnect "Brad";

SPARQL
  DEFINE get:soft "replacing"
  DEFINE get:private <urn:Brad:Sponged:Data>
  SELECT *
  FROM <http://example.com/brad/>
  WHERE
    { ?s ?p ?o };
```

   c. Example for adding graph to Carl's graph group <urn:Carl:Sponged:Data>:

```
-- An error will be raised because "Brad" has not enough rights on that group

reconnect "Brad";

SPARQL
  DEFINE get:soft "replacing"
  DEFINE get:private <urn:Carl:Sponged:Data>
  SELECT *
  FROM <http://example.com/>
  WHERE
    { ?s ?p ?o };
```

9. Examples check Carl's sponging permissions on different graph groups:

   a. Example for adding graph to Anna's graph group <urn:Anna:Sponged:Data>:

```
-- An error will be raised because "Carl" has not enough rights on that group

reconnect "Carl";

SPARQL
  DEFINE get:soft "replacing"
  DEFINE get:private <urn:Anna:Sponged:Data>
  SELECT *
  FROM <http://example.com/>
  WHERE
    { ?s ?p ?o };
```

   b. Example for adding graph to Brad's graph group <urn:Brad:Sponged:Data>:

```
-- An error will be rased because "Carl" has not enough rights on that group

reconnect "Carl";

SPARQL
  DEFINE get:soft "replacing"
```

```
  DEFINE get:private <urn:Brad:Sponged:Data>
  SELECT *
  FROM <http://example.com/>
  WHERE
    { ?s ?p ?o };
```

c. Example for adding graph to Carl's graph group <urn:Carl:Sponged:Data>:

```
-- No error will be raised as Carl has the efficient rights for graph group <urn:Brad:Spong

reconnect "Carl";

SPARQL
  DEFINE get:soft "replacing"
  DEFINE get:private <urn:Carl:Sponged:Data>
  SELECT *
  FROM <http://example.com/carl/>
  WHERE
    { ?s ?p ?o };
```

10. User Carl performs private sponging:

```
reconnect "Carl";

SPARQL
  DEFINE get:soft "replacing"
  DEFINE get:private <urn:Carl:Sponged:Data>
  SELECT *
  FROM <http://www.openlinksw.com/data/turtle/products.ttl>
  WHERE
    { ?s ?p ?o };

-- Should return for ex. 365 rows.
SPARQL
  DEFINE get:soft "replacing"
  DEFINE get:private <urn:Carl:Sponged:Data>
  SELECT COUNT(*)
  FROM <http://www.openlinksw.com/data/turtle/products.ttl>
  WHERE
    { ?s ?p ?o };

SPARQL
  DEFINE get:soft "replacing"
  DEFINE get:private <urn:Carl:Sponged:Data>
  SELECT *
  FROM NAMED <http://www.openlinksw.com/data/turtle/software.ttl>
  FROM NAMED <http://www.openlinksw.com/data/turtle/licenses.ttl>
  WHERE
    {
      graph ?g
        { ?s ?p ?o
      }
    };

-- Should return for ex. 1317 rows.
SPARQL
  DEFINE get:soft "replacing"
  DEFINE get:private <urn:Carl:Sponged:Data>
  SELECT COUNT(*)
  FROM NAMED <http://www.openlinksw.com/data/turtle/software.ttl>
  FROM NAMED <http://www.openlinksw.com/data/turtle/licenses.ttl>
  WHERE
    {
      graph ?g
        { ?s ?p ?o
      }
    };
```

11. Viewing Graph Groups shows Carl's graph group <urn:Carl:Sponged:Data> contains total 4 graphs:

```
SQL> SELECT id_to_iri (RGGM_GROUP_IID), id_to_iri(RGGM_MEMBER_IID)
     FROM DB.DBA.RDF_GRAPH_GROUP_MEMBER
     ORDER BY 1,2;

id_to_iri                id_to_iri__1
```

```
VARCHAR                  VARCHAR
_________________________________________________________________

....
urn:Anna:Sponged:Data    http://example.com/anna/
urn:Brad:Sponged:Data    http://example.com/brad/
urn:Carl:Sponged:Data    http://example.com/carl/
urn:Carl:Sponged:Data    http://www.openlinksw.com/data/turtle/licenses.ttl
urn:Carl:Sponged:Data    http://www.openlinksw.com/data/turtle/products.ttl
urn:Carl:Sponged:Data    http://www.openlinksw.com/data/turtle/software.ttl
```

Example Performing Sponging with Private Graphs Using get:private pragma

The following example demonstrates how private sponging using get:private pragma works for database with private graphs.

1. Create few users in alphabetical order:

```
DB.DBA.USER_CREATE ('Anna', 'Anna');
DB.DBA.USER_CREATE ('Brad', 'Brad');
DB.DBA.USER_CREATE ('Carl', 'Carl');
```

2. Set to Anna, Brad and Carl SPARQL SELECT, UPDATE and SPONGE permissions:

```
grant SPARQL_SELECT to "Anna";
grant SPARQL_SELECT to "Brad";
grant SPARQL_SELECT to "Carl";

grant SPARQL_UPDATE to "Anna";
grant SPARQL_UPDATE to "Brad";
grant SPARQL_UPDATE to "Carl";

grant SPARQL_SPONGE to "Anna";
grant SPARQL_SPONGE to "Brad";
grant SPARQL_SPONGE to "Carl";
```

3. Set specific privileges: Setup assuming 3 users: Anna, Brad and Carl where each of these individual users has read access to graphs:

```
-- Close any public access to "private" graphs
DB.DBA.RDF_DEFAULT_USER_PERMS_SET ('nobody', 0, 1);

-- Set Read Only for public on graphs not listed as "private".
DB.DBA.RDF_DEFAULT_USER_PERMS_SET ('nobody', 1);

DB.DBA.RDF_DEFAULT_USER_PERMS_SET ('Anna', 0, 1);
DB.DBA.RDF_DEFAULT_USER_PERMS_SET ('Brad', 0, 1);
DB.DBA.RDF_DEFAULT_USER_PERMS_SET ('Carl', 0, 1);

DB.DBA.RDF_DEFAULT_USER_PERMS_SET ('Anna', 1);
DB.DBA.RDF_DEFAULT_USER_PERMS_SET ('Brad', 1);
DB.DBA.RDF_DEFAULT_USER_PERMS_SET ('Carl', 1);
```

4. Assuming the following four sorts of access that are specified by four bits of an integer "permission bit-mask", following plain old UNIX style:

    - Bit 1 permits read access.
    - Bit 2 permits write access via SPARUL and is basically useless without bit 1 set.
    - Bit 4 permits write access via "RDF Network Resource Fetch" methods and is basically useless without bits 1 and 2 set.
    - Bit 8 allows retrieval of the list of members of a graph group. An IRI can be used as a graph IRI and as a graph group IRI at the same time, so bit 8 can be freely combined with any of bits 1, 2 or 4.
    - In the statements from below should be considered:

        - "15 = 8+4+2+1" -- i.e. combining all the four sorts of access FROM above
        - "9 = 8 + 1" -- i.e. read access + access to retrieve the list of members for a given graph group

```
-- Create Graph Group for Anna and set privileges:
DB.DBA.RDF_GRAPH_GROUP_CREATE ('urn:Anna:Sponged:Data', 1);
DB.DBA.RDF_GRAPH_USER_PERMS_SET ('urn:Anna:Sponged:Data', 'Anna', 15);
DB.DBA.RDF_GRAPH_USER_PERMS_SET ('urn:Anna:Sponged:Data', 'Brad', 9);
DB.DBA.RDF_GRAPH_USER_PERMS_SET ('urn:Anna:Sponged:Data', 'Carl', 9);
```

```
-- Create Graph Group for Brad and set privileges:
DB.DBA.RDF_GRAPH_GROUP_CREATE ('urn:Brad:Sponged:Data', 1);
DB.DBA.RDF_GRAPH_USER_PERMS_SET ('urn:Brad:Sponged:Data', 'Anna', 9);
DB.DBA.RDF_GRAPH_USER_PERMS_SET ('urn:Brad:Sponged:Data', 'Brad', 15);
DB.DBA.RDF_GRAPH_USER_PERMS_SET ('urn:Brad:Sponged:Data', 'Carl', 9);

-- Create Graph Group for Carl and set privileges:
DB.DBA.RDF_GRAPH_GROUP_CREATE ('urn:Carl:Sponged:Data', 1);
DB.DBA.RDF_GRAPH_USER_PERMS_SET ('urn:Carl:Sponged:Data', 'Anna', 9);
DB.DBA.RDF_GRAPH_USER_PERMS_SET ('urn:Carl:Sponged:Data', 'Brad', 9);
DB.DBA.RDF_GRAPH_USER_PERMS_SET ('urn:Carl:Sponged:Data', 'Carl', 15);

-- Set Anna's, Brad's and Carl's graphs by inserting them into the <b>virtrdf:PrivateGraphs</b> gr
DB.DBA.RDF_GRAPH_GROUP_INS ('http://www.openlinksw.com/schemas/virtrdf#PrivateGraphs', 'http://exa
DB.DBA.RDF_GRAPH_GROUP_INS ('http://www.openlinksw.com/schemas/virtrdf#PrivateGraphs', 'http://exa
DB.DBA.RDF_GRAPH_GROUP_INS ('http://www.openlinksw.com/schemas/virtrdf#PrivateGraphs', 'http://exa
```

5. Examples with invalid graph group names:

   a. Example with Non-existing Graph Group:

```
-- An error for non-existing Graph group <http://nosuch/> will be raised.

SPARQL
  DEFINE get:soft "replacing"
  DEFINE get:private <http://nosuch/>
  SELECT *
  FROM <http://example.com/>
  WHERE
    { ?s ?p ?o };
```

   b. Example with "virtrdf:PrivateGraphs" graph group which is reserved for system usage:

```
-- An error for attempt to add a graph to special
-- graph group <http://www.openlinksw.com/schemas/virtrdf#PrivateGraphs> will be raised.

SPARQL
  DEFINE get:soft "replacing"
  DEFINE get:private virtrdf:PrivateGraphs
  SELECT * FROM <http://example.com/>
  WHERE
    { ?s ?p ?o };
```

   c. Example with "virtrdf:rdf_repl_graph_group" graph group which is reserved for system usage:

```
-- An error for attempt to add a graph to special
-- graph group <http://www.openlinksw.com/schemas/virtrdf#rdf_repl_graph_group> will be rai

SPARQL
  DEFINE get:soft "replacing"
  DEFINE get:private virtrdf:rdf_repl_graph_group
  SELECT * FROM <http://example.com/>
  WHERE
    { ?s ?p ?o };
```

6. Examples to check Anna's sponging permissions on different graph groups:

   a. Example for adding graph to Anna's graph group <urn:Anna:Sponged:Data>:

```
-- No error will be raised as Anna has the efficient rights for graph group <urn:Anna:Spong

reconnect "Anna";

SPARQL
  DEFINE get:soft "replacing"
  DEFINE get:private <urn:Anna:Sponged:Data>
  SELECT *
  FROM <http://example.com/anna/>
  WHERE
    { ?s ?p ?o };
```

   b. Example for adding graph to Brad's graph group <urn:Brad:Sponged:Data>:

```
-- An error will be raised because "Anna" has not enough rights on that group

reconnect "Anna";

SPARQL
  DEFINE get:soft "replacing"
  DEFINE get:private <urn:Brad:Sponged:Data>
  SELECT * FROM <http://example.com/>
  WHERE
    { ?s ?p ?o };
```

c. Example for adding graph to Carl's graph group <urn:Carl:Sponged:Data>:

```
-- An error will be raised because "Anna" has not enough rights on that group

reconnect "Anna";

SPARQL
  DEFINE get:soft "replacing"
  DEFINE get:private <urn:Carl:Sponged:Data>
  SELECT *
  FROM <http://example.com/>
  WHERE
    { ?s ?p ?o };
```

7. Examples check Brad's sponging permissions on different graph groups:

a. Example for adding graph to Anna's graph group <urn:Anna:Sponged:Data>:

```
-- An error will be raised because "Brad" has not enough rights on that group

reconnect "Brad";

SPARQL
  DEFINE get:soft "replacing"
  DEFINE get:private <urn:Anna:Sponged:Data>
  SELECT *
  FROM <http://example.com/>
  WHERE
    { ?s ?p ?o };
```

b. Example for adding graph to Brad's graph group <urn:Brad:Sponged:Data>:

```
-- No error will be raised as Brad has the efficient rights for graph group <urn:Brad:Spong

reconnect "Brad";

SPARQL
  DEFINE get:soft "replacing"
  DEFINE get:private <urn:Brad:Sponged:Data>
  SELECT *
  FROM <http://example.com/brad/>
  WHERE
    { ?s ?p ?o };
```

c. Example for adding graph to Carl's graph group <urn:Carl:Sponged:Data>:

```
-- An error will be raised because "Brad" has not enough rights on that group

reconnect "Brad";

SPARQL
  DEFINE get:soft "replacing"
  DEFINE get:private <urn:Carl:Sponged:Data>
  SELECT *
  FROM <http://example.com/>
  WHERE
    { ?s ?p ?o };
```

8. Examples check Carl's sponging permissions on different graph groups:

a. Example for adding graph to Anna's graph group <urn:Anna:Sponged:Data>:

```
-- An error will be raised because "Carl" has not enough rights on that group
```

```
reconnect "Carl";

SPARQL
  DEFINE get:soft "replacing"
  DEFINE get:private <urn:Anna:Sponged:Data>
  SELECT *
  FROM <http://example.com/>
  WHERE
    { ?s ?p ?o };
```

b. Example for adding graph to Brad's graph group <urn:Brad:Sponged:Data>:

```
-- An error will be rased because "Carl" has not enough rights on that group

reconnect "Carl";

SPARQL
  DEFINE get:soft "replacing"
  DEFINE get:private <urn:Brad:Sponged:Data>
  SELECT *
  FROM <http://example.com/>
  WHERE
    { ?s ?p ?o };
```

c. Example for adding graph to Carl's graph group <urn:Carl:Sponged:Data>:

```
-- No error will be raised as Carl has the efficient rights for graph group <urn:Brad:Spong

reconnect "Carl";

SPARQL
  DEFINE get:soft "replacing"
  DEFINE get:private <urn:Carl:Sponged:Data>
  SELECT *
  FROM <http://example.com/carl/>
  WHERE
    { ?s ?p ?o };
```

9. User Carl performs private sponging:

```
reconnect "Carl";

SPARQL
  DEFINE get:soft "replacing"
  DEFINE get:private <urn:Carl:Sponged:Data>
  SELECT *
  FROM <http://www.openlinksw.com/data/turtle/products.ttl>
  WHERE
    { ?s ?p ?o };

-- Should return for ex. 365 rows.
SPARQL
  DEFINE get:soft "replacing"
  DEFINE get:private <urn:Carl:Sponged:Data>
  SELECT COUNT(*)
  FROM <http://www.openlinksw.com/data/turtle/products.ttl>
  WHERE
    { ?s ?p ?o };

SPARQL
  DEFINE get:soft "replacing"
  DEFINE get:private <urn:Carl:Sponged:Data>
  SELECT *
  FROM NAMED <http://www.openlinksw.com/data/turtle/software.ttl>
  FROM NAMED <http://www.openlinksw.com/data/turtle/licenses.ttl>
  WHERE
    {
      graph ?g
        { ?s ?p ?o
      }
    };

-- Should return for ex. 1317 rows.
SPARQL
  DEFINE get:soft "replacing"
```

```
DEFINE get:private <urn:Carl:Sponged:Data>
SELECT COUNT(*)
FROM NAMED <http://www.openlinksw.com/data/turtle/software.ttl>
FROM NAMED <http://www.openlinksw.com/data/turtle/licenses.ttl>
WHERE
  {
    graph ?g
      { ?s ?p ?o
    }
  };
```

10. User Anna reads Carl's data:

```
SQL> reconnect "Anna";
SQL> SPARQL
  SELECT COUNT(*)
  FROM <http://www.openlinksw.com/data/turtle/products.ttl>
  WHERE
    { ?s ?p ?o };

callret-0
INTEGER
_______________________________________________________________________________

365

1 Rows. -- 15 msec.
```

16.5. Linked Data Views over RDBMS Data Source

Linked Data Views map relational data into RDF and allow customizing RDF representation of locally stored RDF data. To let SPARQL clients access relational data as well as physical RDF graphs in a single query, we introduce a declarative Meta Schema Language for mapping SQL Data to RDF Ontologies. As a result, all types of clients can efficiently access all data stored on the server. The mapping functionality dynamically generates RDF Data Sets for popular ontologies such as SIOC, SKOS, FOAF, and ATOM/OWL without disruption to the existing database infrastructure of Web 1.0 or Web 2.0 solutions. Linked Data Views are also suitable for declaring custom representation for RDF triples, e.g. property tables, where one row holds many single-valued properties.

16.5.1. Introduction

The Virtuoso Linked Data Views meta schema is a built-in feature of Virtuoso's SPARQL to SQL translator. It recognizes triple patterns that refer to graphs for which an alternate representation is declared and translates these into SQL accordingly. The main purpose of this is evaluating SPARQL queries against existing relational databases. There exists previous work from many parties for rendering relational data as RDF and opening it to SPARQL access. We can mention D2RQ, SPASQL, Squirrel RDF, DBLP and others. The Virtuoso effort differs from these mainly in the following:

- Integration with a triple store. Virtuoso can process a query for which some triple patterns will go to local or remote relational data and some to local physical RDF triples.
- SPARQL query can be used in any place where SQL can. Database connectivity protocols are neutral to the syntax of queries they transmit, thus any SQL client, e.g. JDBC, ODBC or XMLA application, can send SPARQL queries and fetch result sets. Moreover, a SQL query may contain SPARQL subqueries and SPARQL expressions may use SQL built-in functions and stored procedures.
- Integration with SQL. Since SPARQL and SQL share the same run time and query optimizer, the query compilation decisions are always made with the best knowledge of the data and its location. This is especially important when mixing triples and relational data or when dealing with relational data distributed across many outside databases.
- No limits on SPARQL. It remains possible to make queries with unspecified graph or predicate against mapped relational data, even though these may sometimes be inefficient.
- Coverage of the whole relational model. Multi-part keys etc. are supported in all places.

16.5.2. Rationale

Since most of the data that is of likely use for the emerging semantic web is stored in relational databases, the argument for exposing this to SPARQL access is clear. We note that historically, SQL access to relational data has essentially never been given to the public outside of the organization. If programmatic access to corporate IS has been available to partners or the public, it has

been through dynamic web pages or more recently web services. There are reasons of performance, security, maintainability and so forth for this.

The culture of the emerging semantic web is however taking a different turn. Since RDF and OWL offer a mergeable and queryable model for heterogeneous data, it is more meaningful and maintainable to expose selected data for outside query than it would be with SQL. Advances in hardware make this also less of a performance issue than it would have been in the client-server database era.

In the context of Virtuoso, since Virtuoso is originally a virtual/federated database, incorporating SPARQL to relational mapping is an evident extension of the product's mission as a multi-protocol, multi-platform connector between information systems.

16.5.3. Quad Map Patterns, Values and IRI Classes

In the simplest sense, any relational schema can be rendered into RDF by converting all primary keys and foreign keys into IRI's, assigning a predicate IRI to each column, and an rdf:type predicate for each row linking it to a RDF class IRI corresponding to the table. Then a triple with the primary key IRI as subject, the column IRI as predicate and the column's value as object is considered to exist for each column that is neither part of a primary or foreign key.

Strictly equating a subject value to a row and each column to a predicate is often good but is too restrictive for the general case.

- Multiple triples with the same subject and predicate can exist.
- A single subject can get single-valued properties from multiple tables or in some cases stored procedures.
- An IRI value of a subject or other field of a triple can be composed from more than one SQL value, these values may reside in different columns, maybe in different joined tables.
- Some table rows should be excluded from mapping.

Thus in the most common case the RDF meta schema should consist of independent transformations; the domain of each transformation is a result-set of some SQL *SELECT* statement and range is a set of triples. The *SELECT* that produce the domain is quite simple: it does not use aggregate functions, joins and sorting, only inner joins and *WHERE* conditions. There is no need to support outer joins in the RDF meta schema because NULLs are usually bad inputs for functions that produce IRIs. In the rare cases when NULLs are OK for functions, outer joins can be encapsulated in SQL views. The range of mapping can be described by a SPARQL triple pattern: a pattern field is a variable if it depends on table columns, otherwise it is a constant. Values of variables in the pattern may have additional restrictions on datatypes, when datatypes of columns are known.

This common case of an RDF meta schema is implemented in Virtuoso, with one adjustment. Virtuoso stores quads, not triples, using the graph field (G) to indicate that a triple belongs to some particular application or resource. A SPARQL query may use quads from different graphs without large difference between G and the other three fields of a quad. E.g., variable *?g* in expression *GRAPH ?g {...}* can be unbound. SPARQL has special syntax for "graph group patterns" that is convenient for sets of triple patterns with a common graph, but it also has shorthands for common subject and predicate, so the difference is no more than in syntax. There is only one feature that is specific for graphs but not for other fields: the SPARQL compiler can create restrictions on graphs according to *FROM* and *FROM NAMED* clauses.

Virtuoso Linked Data Views should offer the same flexibility with the graphs as SPARQL addressing physical triples. A transformation cannot always be identified by the graph used for ranges because graph may be composed from SQL data. The key element of the meta schema is a "*quad map pattern* ". A simple quad map pattern fully defines one particular transformation from one set of relational columns into triples that match one SPARQL graph pattern. The main part of quad map pattern is four declarations of "*quad map values* ", each declaration specifies how to calculate the value of the corresponding triple field from the SQL data. The pattern also lists boolean SQL expressions that should be used to filter out unwanted rows of source data (and to join multiple tables if source columns belong to different tables). There are also quad map patterns that group together similar quad patterns but do not specify any real transformation or even prevent unwanted transformations from being used, they are described in "Grouping Map Patterns" below.

Quad map values refer to schema elements of two further types: "IRI classes" and "literal classes".

Note

In SQL, adding a new view can not break anything. This is because SQL lacks the ability of querying "everything" so data sources are always specified. This is not true for SPARQL, so please treat *any* metadata manipulation as potentially destructive operation. If an RDF storage is supposed to be used by more than one application then these applications should be tested together, not one after other, and they should be installed/upgraded on live database in the very same order as they were installed/upgraded on instrumental machine during testing. Always remember that these applications share RDF tables

Note

so they may interfere.

IRI Classes

An IRI class declares that a column or set of columns gets converted into a IRI in a certain way. The conversion of this sort can be declared revertible (bijection) so an IRI can be parsed into original SQL values; this is useful when some equality of an IRI constant and a calculated IRI can be replaced with an equality of a parse result of a constant and an SQL column that is index criteria or simply faster. In addition, the SPARQL optimizer will eliminate redundant conversions if one IRI class is explicitly declared as a subclass of another. The most flexible declaration for conversion consists of specifying functions that assemble and disassemble from IRI into its constituent parts. This is overkill for typical conversions so it is possible to specify only one sprintf-style format string such that *sprintf()* SQL function will print an IRI using this format and *sprintf_inverse()* will be able to parse it back.

The use of *sprintf_inverse()* assumes that the format does not contain fragments like *'%s%s'* that make it impossible to separate parts of IRI from each other.

In the following, we shall map the Virtuoso users and user roles system tables into the SIOC ontology.

```
create iri class oplsioc:user_iri "http://myhost/sys/user?id=%d"
  (in uid integer not null) .
create iri class oplsioc:group_iri "http://myhost/sys/group?id=%d"
  (in gid integer not null) .
create iri class oplsioc:membership_iri
  "http://myhost/sys/membership?super=%d&sub=%d"
  (in super integer not null, in sub integer not null) .
create iri class oplsioc:dav_iri "http://myhost%s"
  (in path varchar) .
```

These IRI classes are used for mapping data from the *DB.DBA.SYS_USERS* and *DB.DBA.SYS_ROLE_GRANTS* system tables that are defined in Virtuoso as follows:

```
create table DB.DBA.SYS_USERS (
  U_ID                integer not null unique,
  U_NAME              char (128) not null primary key,
  U_IS_ROLE           integer default 0,
  U_FULL_NAME         char (128),
  U_E_MAIL            char (128) default &amp;quot;,
  U_ACCOUNT_DISABLED  integer default 1,
  U_DAV_ENABLE        integer default 0,
  U_SQL_ENABLE        integer default 1,
  U_HOME              varchar (128),
. . .
 );
```

Single record in *DB.DBA.SYS_USERS* corresponds to a plain user or a group (role). Users and roles are collectively named "grantees". Thus a role may be granted to another role or to a user account. A role grant may be direct (explicit) or assigned by recursion.

```
create table SYS_ROLE_GRANTS (
  GI_SUPER   integer,
  GI_SUB     integer,
  GI_DIRECT  integer default 1,
. . .
  primary key (GI_SUPER, GI_SUB, GI_DIRECT));
```

One IRI class usually corresponds to one ontology class, because similar things are usually called similarly. One may wish to use identifiers of ontology classes as identifiers of related IRI classes, to not remember double number of names, e.g. *create IRI class mybank:XpressXfer* for subjects that will have *rdf:type* property *mybank:XpressXfer* made by mapping. That is technically possible but proven to become inconvenient and misleading as application evolves. While RDF types tend to persist, IRI classes may change over time or same subject may get more than one name via more than one IRI class, say, for exports to different systems. It is found to be more convenient to compose names of IRI classes by adding some common prefixes or suffixes to RDF classes (or to table names), say, write *create IRI class mybank:XpressXfer_iri* .

Literal Classes

A "literal class" declares that a column or set of columns gets converted into a literal instead of an IRI. More precisely, the result of conversion can be *IRI_ID* so it represents an IRI, but in current version of Virtuoso this is supported only for some internal built-in literal classes, not for classes declared by the user. So for user-defined literal class the result of the conversion is an RDF literal even if it is a string representation of a valid IRI.

In any case, a literal class can be used only in quad map values of O fields, because Virtuoso does not support literal values as subjects.

A special case of literal class is the identity class that converts a value from *varchar* column into an untyped literal and value from column of any other SQL datatype into a typed literal with type from XMLSchema set, i.e. *xsd:integer* , *xsd:dateTime* and so on. Columns of types *ANY* and *IRI_ID* are not supported.

The SPARQL optimizer knows that RDF literal types are pairwise disjoint so literal classes that produce literals of different types are known to be pairwise disjoint. The optimizer will replace a join on two disjoint literal classes with an empty statement, to simplify the resulting query.

Simple Quad Map Patterns

The following declaration of quad map pattern is self-explanatory. The line for *object* uses identity literal class so there's no need to specify its name.

```
graph      <http://myhost/sys>
subject    oplsioc:user_iri (DB.DBA.SYS_USERS.U_ID)
predicate  foaf:email
object     DB.DBA.SYS_USERS.U_E_MAIL
```

The description language also supports SPARQL-style notation that contains less keywords and eliminates duplicate graphs, subjects and predicates. The following add two patterns with constant graph IRI <*http://myhost/sys*> and subjects are made from column *DB.DBA.SYS_USERS.U_ID* by *oplsioc:user_iri* .

```
graph <http://myhost/sys>
  {
    oplsioc:user_iri (DB.DBA.SYS_USERS.U_ID)
      a sioc:user ;
      oplsioc:name DB.DBA.SYS_USERS.U_FULL_NAME .
  }
```

Assigning Names To Quad Map Patterns

In real applications, quad map patterns should be named, for schema manipulation and keeping debug info readable. Thus it is much better to rewrite the previous example as

```
create virtrdf:SysUsers as graph <http://myhost/sys>
  {
    oplsioc:user_iri (DB.DBA.SYS_USERS.U_ID)
      a sioc:user
          as virtrdf:SysUserType-User;
      oplsioc:name DB.DBA.SYS_USERS.U_FULL_NAME
          as virtrdf:SysUsersFullName .
  }
```

Using these names, one may later write, say, *drop quad map virtrdf:SysUserType-User* .

One name, *virtrdf:DefaultQuadMap* is reserved. It is an internal quad map pattern used to access "native-form" quads from *DB.DBA.RDF_QUAD* :

```
create virtrdf:DefaultQuadMap as
graph rdfdf:default-iid-nonblank (DB.DBA.RDF_QUAD.G)
subject rdfdf:default-iid (DB.DBA.RDF_QUAD.S)
predicate rdfdf:default-iid-nonblank (DB.DBA.RDF_QUAD.P)
object rdfdf:default (DB.DBA.RDF_QUAD.O)
```

IRI classes from *rdfdf:...* namespace are also reserved.

Grouping Map Patterns

The previous example actually contains three map patterns, not two. The name *virtrdf:SysUsers* refers to a "*group map pattern* " that does not define any real transformation of relational data into RDF but helps organize quad map patterns into a tree. Group may contain both quad map patterns and other groups. A group can be manipulated as a whole, e.g. *drop quad map virtrdf:SysUsers* will remove all three map patterns.

16.5.4. Configuring RDF Storages

"*Quad Storage* " is a named set of quad map patterns. The declaration *define input:storage storage-name* states that a SPARQL query will be executed using only quad patterns of the given quad storage. Declarations of IRI classes, literal classes and quad patterns are shared between all quad storages of an RDF meta schema but every quad storage contains only a subset of all available quad patterns. Two quad storages are always defined:

- A

  virtrdf:default

  one usually consists of everything (all user-relational mappings plus

  virtrdf:DefaultQuadMap

  for "native-form" quads from

  DB.DBA.RDF_QUAD

)
- A

  virtrdf:empty

  storage refers solely to

  DB.DBA.RDF_QUAD

  and can not be altered.

Three statements for manipulating storages are

- *create quad storage storage-name { quad-map-decls } .*
- *alter quad storage storage-name { quad-map-decls-or-drops } .*
- *drop quad storage storage-name .*

A map pattern can be created only as a part of *create quad storage* or *alter quad storage* statement, so initially it is used by exactly one storage. It can be imported to some other storage using directive *create map-id using storage source-storage* . E.g., declarations of many storages create *virtrdf:DefaultQuadMap* using storage *virtrdf:DefaultQuadStorage* .

Only a "top-level" quad map pattern (standalone or a whole group with descendants) can be imported, member of a group can not. The import directive also can not be a part of some group declaration.

The directive *drop quad map map-name* removes a map from one storage when it appears inside *alter quad storage* statement. Otherwise it removes the map from all storages. There exists garbage collection for quad map patterns, so any unused map is immediately deleted. A group is deleted with all its descendants.

16.5.5. Translation Of SPARQL Triple Patterns To Quad Map Patterns

When a SPARQL query is compiled into SQL using a quad storage, every triple pattern should become a subquery that retrieves data from relational tables. This subquery is an *UNION ALL* of joins generated from appropriate quad map patterns. The complete SQL query is composed from these basic subqueries. Thus the first operation of the SQL generation for a triple pattern is searching for quad map patterns that may in principle produce triples that match the triple pattern.

The more restrictions contained in the triple pattern the fewer quad map patterns will be used. A triple pattern *graph ?g { ?s ?p ?o }* is common enough to invoke all data transformations of the storage. A triple pattern *graph <g> { ?s <p> <o> }* will usually intersect with the range of only one quad map. Sometimes it is possible to prove that the storage can not contain any data that matches the given triple pattern, hence zero number of members of *UNION ALL* will result in constantly empty result-set.

The search for quad maps for a given pair of triple pattern and quad map storage is quite simple. The storage is treated as a tree of map patterns where quad map patterns are leafs, grouping patterns are inner nodes and the whole storage is also treated as a grouping pattern that specify no fields and contains all top-level map patterns of the storage.

The tree is traversed from the root, left to right, non-leaf vertex are checked before their children. The check of a vertex consists of up to four field checks, for G, S, P and O. Every field check compares the field definition in the vertex and the corresponding field in the triple pattern, G and G, S and S and so on. Note that a non-leaf vertex defines less than four of its fields, e.g., the root vertex does not define any of its fields and top-level *graph map { ... }* defines only graph. Checks are performed only for defined fields and return one of three values: "failed", "passed", "full match", according to the following rules:

Table 16.8. Matching Triple Field and Vertex Field

Field of vertex	Field in triple pattern	Result
constant	same constant	full match
constant	different constant	failed
constant	variable of same type	passed
constant	variable of different type	failed
quad map value	constant of same type	full match
quad map value	constant of different type	failed
quad map value of type X	variable, X or subtype of X	full match
quad map value of type X	variable, supertype of X	passed
quad map value of type X	variable, type does not intersect with X	failed

If any of the checks fails, the vertex and all its children are excluded from the rest of processing. Otherwise, if all four fields are defined for the quad map pattern, the map is added to the list of matching map patterns. The difference between "passed" and "full match" is significant only if the map is declared with *option (exclusive)* If all performed checks return "full match" and *option (exclusive)* is set then the traverse of the tree is stopped as soon as all children of the vertex are traversed. The most typical use of this option is when the application developer is sure that all triples of a graph belong to his application and they come from his own quad map patterns, not from *DB.DBA.RDF_QUAD* . This is to prevent the SPARQL compiler from generating redundant subqueries accessing *DB.DBA.RDF_QUAD* . The declaration may look like

```
create quad storage <mystorage>
  {
    graph <mygraph> option (exclusive) { . . . }
    create virtrdf:DefaultQuadMap
      using storage virtrdf:DefaultQuadStorage .
  }
```

Exclusive patterns make the order of declarations important, because an exclusive declaration may "throw a shadow" on declarations after it. Consider a database that have a special table RDF_TYPE that caches all RDF types of all subjects in all graphs. Consider two declarations: all triples from graph <*http://myhost/sys*> and all triples with *rdf:type* predicate, both exclusive:

```
graph <http://myhost/sys> option (exclusive)
  {
    . . . # mapping of DB.DBA.SYS_USERS as in previous examples.
  }
graph rdfdf:default-iid-nonblank (DB.DBA.RDF_TYPE.G)
subject rdfdf:default-iid (DB.DBA.RDF_TYPE.S)
predicate rdf:type
object rdfdf:default (DB.DBA.RDF_TYPE.O)
option (exclusive)
```

The order of these declarations dictates that triple pattern

```
graph <http://myhost/sys> {?s rdf:type ?o}
```

is compiled using only quad map patterns of the graph declaration, ignoring second declaration (and of course ignoring default mapping rule, if any). An explicit *option (order N)* at the end of quad map pattern will tweak the priority. By default, order will grow from 1000 for the first declaration in the statement to 1999 for the last, explicit configuration is especially useful to make order persistent to *alter storage* statements.

The *option (exclusive)* trick is ugly, low-level and prone to cause compilation errors after altering storage declarations. When misused, it is as bad as "red cut" in PROLOG, but one must use this trick to build scalable storages.

The *option (exclusive)* helps the SPARQL compiler to prepare better SQL queries, but sometimes it is "too exclusive". For instance, if a grouping quad map pattern specify only quad map value for graph and no other fields then making it exclusive prohibits the use of all declarations of the storage after that one. Sometimes it is better to notify compiler that quads made by the given quad map pattern are supposed to be different from all quads made by declarations listed after the given one.

Consider an application that exports users' personal data as graphs whose IRIs looks like *http://www.example.com/DAV/home/* username*/RDF/personal/* ; the application makes a query and a triple pattern is proven to be restrictive enough to filter out all quads that are not similar to quads generated by the given quad map pattern (say, the graph is constant *http://www.example.com/DAV/home/JohnSmith/RDF/personal/*). The application do not hope to find any quads that match the pattern but made by other applications, because graphs named like in the pattern are supposed to be solely for this single purpose; if, say, DB.DBA.RDF_QUAD occasionally contains some quads with graph equal to *http://www.example.com/DAV/home/JohnSmith/RDF/personal/* then they can be ignored.

Under this circumstances, the quad map pattern may have *option (soft exclusive)* . That grants a permission to the compiler to ignore rest of storage as soon as it is proven that the triple pattern can not access quads that does not match the pattern. So if that is proven then the pattern is exclusive and it makes the query faster; when unsure, the compiler work like there is no option at all.

Note

The *option (exclusive)* can be used as a security measure, *option (soft exclusive)* can not. Say, if an financial application exports its data as a single graph *http://www.example.com/front-office/cash/* using *exclusive* then the query that explicitly refers to that graph will never access any quads written by the attacker into DB.DBA.RDF_QUAD using same graph IRI. The use of *soft exclusive* gives no such protection. From the compiler's perspective, the *option (soft exclusive)* is a hint that may be ignored, not an unambiguous order.

There is one exception from the rules described above. This exception is for *virtrdf:DefaultQuadStorage* only. If a graph variable of a quad map pattern is not bound and no source graph specified by *FROM* clauses then quad maps for specific constant graphs are ignored. In other words, if a default quad storage contains quad maps for specific graphs then the query in that storage should explicitly specify the graph in order to use a map for graph. This rule will not work if the default quad map is removed from the *virtrdf:DefaultQuadStorage* . This rule relates to the default storage itself, not to the containing patterns; copying some or all patterns into other storage will not reproduce there this special effect.

So for example the query from below returns results when graph is specified i.e. when no graph is referenced, then run over physical store only is performed:

```
SQL>SPARQL
SELECT *
WHERE
  {
    <http://localhost:8990/Demo/categories/CategoryID/1#this>  ?p ?o
  };
p         o
VARCHAR  VARCHAR
_______________________________________________________________________________

0 Rows. -- 0 msec.

SQL>SPARQL
SELECT *
WHERE
  {
    GRAPH ?g
      {
        <http://localhost:8990/Demo/categories/CategoryID/1#this>  ?p ?o
      }
  };
```

```
g                               p                                                   o
VARCHAR                         VARCHAR                                             VARCHAR
_______________________________________________________________________________________________

http://localhost:8990/Demo#     http://www.w3.org/1999/02/22-rdf-syntax-ns#type     http://localhost:899
http://localhost:8990/Demo#     http://localhost:8990/schemas/Demo/categoryid       1
http://localhost:8990/Demo#     http://localhost:8990/schemas/Demo/categoryname     ...
...
```

16.5.6. Describing Source Relational Tables

Quad map patterns of an application usually share a common set of source tables and quad map values of one pattern usually share either a single table or very small number of joined tables. Join and filtering conditions are also usually repeated in different patterns. It is not necessary to type table descriptions multiple times, they are declare once in the beginning of storage declaration statement and shared between all quad map declarations inside the statement. Names of aliases can be used instead of table names in quad map values.

```
FROM DB.DBA.SYS_USERS as user WHERE (^{user.}^.U_IS_ROLE = 0)
FROM DB.DBA.SYS_USERS as group WHERE (^{group.}^.U_IS_ROLE = 1)
FROM DB.DBA.SYS_USERS as account
FROM user as active_user
  WHERE (^{active_user.}^.U_ACCOUNT_DISABLED = 0)
FROM DB.DBA.SYS_ROLE_GRANTS as grant
  WHERE (^{grant.}^.GI_SUPER = ^{account.}^.U_ID)
  WHERE (^{grant.}^.GI_SUB = ^{group.}^.U_ID)
  WHERE (^{grant.}^.GI_SUPER = ^{user.}^.U_ID)
```

This declares five distinct aliases for two distinct tables, and six filtering conditions. Every condition is an SQL expression with placeholders where a reference to the table should be printed. The SPARQL compiler will not try to parse texts of these expressions (except dummy search for placeholders), so any logical expressions are acceptable. When a quad map pattern declaration refers to some aliases, the *WHERE* clause of the generated SQL code will contain a conjunction of all distinct texts of "relevant" conditions. A condition is relevant if every alias inside the condition is used in some quad map value of the map pattern, either directly or via clause like *from user as active_user* . (*user* is a "*base alias* " for *active_user*).

Consider a group of four declarations.

```
graph <http://myhost/sys>
  {
    oplsioc:user_iri (active_user.U_ID)
        a oplsioc:active-user .
    oplsioc:membership_iri (grant.GI_SUPER, grant.GI_SUB).
        oplsioc:is_direct
            grant.GI_DIRECT ;
        oplsioc:member-e-mail
            active_user.U_E_MAIL
               where (^{active_user.}^.U_E_MAIL like 'mailto:%').
    ldap:account-ref (account.U_NAME)
        ldap:belongs-to
            ldap:account-ref (group.U_NAME) option (using grant).
  }
```

The first declaration will extend <*http://myhost/sys*> graph with one imaginary triples *{ user a oplsioc:active-user }* for every account record that is not a role and not disabled. The second declaration deals with membership records. A membership is a pair of a grantee ("super") and a granted role ("sub") stored as a row in *DB.DBA.SYS_ROLE_GRANTS*).

The second declaration states that every membership has *oplsioc:is_direct* property with value from *GI_DIRECT* column of that table (roles may be granted to other roles and users, so permissions are "direct" or "recursive").

The third declaration declares *oplsioc:member-e-mail* property of memberships. The value is a literal string from *DB.DBA.SYS_USERS.U_E_MAIL* , if the grantee is active (not disabled) and is not a role and its e-mail address starts with *'mailto:'* . The join between *DB.DBA.SYS_ROLE_GRANTS* and *DB.DBA.SYS_USERS* is made by equality *(GI_SUPER = U_ID)* because the alias *active_user* in the declaration "inherits" all conditions specified for *user* . In addition, the SPARQL compiler will add one more condition to check if the *U_E_MAIL* is not null because the NULL value is not a valid object and it knows that *U_E_MAIL* is not declared as *NOT NULL* .

The last declaration contains an *option* clause. As usual, this indicates that the basic functionality is good for many tasks but not for all. In this declaration, the *ldap:belongs-to* property establishes a relation between grantee (subject) and a granted role (object). Both subject and object IRIs are based on account name, *DB.DBA.SYS_USERS.U_NAME* , so the quad map pattern contains two references to different aliases of *DB.DBA.SYS_USERS* but no alias for *DB.DBA.SYS_ROLE_GRANTS* . Hence the declaration could produce a triple for every row of the Cartesian product of the *DB.DBA.SYS_USERS* . To fix the problem, *option (using alias-name)* tells the compiler to process the alias-name as if it's used in some quad map value of the pattern.

It is an error to use an alias only in *where* clause of the quad map pattern but neither in values or in *option (using alias-name)* . To detect more typos, an alias used in quad map values can not appear in *option (using alias-name)* clause.

16.5.7. Function-Based IRI Classes

Most of IRI classes can be declared by a sprintf format string, but sophisticated cases may require calculations, not only printing the string. *create IRI class using function* allows the application transform relational values to IRIs by any custom routines.

Let us extend the previous example about users and groups by a new class for grantees. Both users and groups are grantees and we have defined two IRI classes for them. Classes *oplsioc:user_iri* and *oplsioc:group_iri* work fine for quad maps of *U_ID* if and only if the value of *U_IS_ROLE* is accordingly restricted to FALSE or TRUE, otherwise one may occasionally generate, say, user IRI for a group. To create and parse IRIs that correspond to any U_IDs, two functions should be created:

```
create function DB.DBA.GRANTEE_URI (in id integer)
returns varchar
{
  declare isrole integer;
  isrole := coalesce ((SELECT top 1 U_IS_ROLE
      FROM DB.DBA.SYS_USERS WHERE U_ID = id ) );
  if (isrole is null)
    return NULL;
  else if (isrole)
    return sprintf ('http://%s/sys/group?id=%d', id);
  else
    return sprintf ('http://%s/sys/user?id=%d', id);
};

create function DB.DBA.GRANTEE_URI_INVERSE (in id_iri varchar)
returns integer
{
  declare parts any;
  parts := sprintf_inverse (id_iri,
      'http://myhost/sys/user?id=%d', 1 );
  if (parts is not null)
    {
      if (exists (SELECT top 1 1 FROM DB.DBA.SYS_USERS
          WHERE U_ID = parts[0] and not U_IS_ROLE ) )
        return parts[0];
    }
  parts := sprintf_inverse (id_iri,
      'http://myhost/sys/group?id=%d', 1 );
  if (parts is not null)
    {
      if (exists (SELECT top 1 1 FROM DB.DBA.SYS_USERS
          WHERE U_ID = parts[0] and U_IS_ROLE ) )
        return parts[0];
    }
  return NULL;
};
```

These functions may be more useful if the SPARQL web service endpoint is allowed to use them:

```
grant execute on DB.DBA.GRANTEE_URI to "SPARQL";
grant execute on DB.DBA.GRANTEE_URI_INVERSE to "SPARQL";
```

The next declaration creates an IRI class based on these two functions:

```
create iri class oplsioc:grantee_iri using
  function DB.DBA.GRANTEE_URI (in id integer)
    returns varchar,
  function DB.DBA.GRANTEE_URI_INVERSE (in id_iri varchar)
```

```
returns integer .
```

In common case, IRI class declaration contains an N-array function that composes IRIs and N inverse functions that gets an IRI as an argument and extracts the Nth SQL value. IRI composing function should silently return NULL on incorrect arguments instead of error signal. Inverse functions should return NULL if the argument has an incorrect type or value.

It is possible to specify only composing function without any of inverse functions. However *option (bijection)* can not be used in that case, obviously.

16.5.8. Connection Variables in IRI Classes

Writing function-based IRI class is overkill when the IRI can in principle be made by a `sprintf_iri` but the format should contain some context-specific data, such as host name used for the dynamic renaming of local IRIs . Format strings offer a special syntax for that cases. *%{varname}U* acts as *%U* but the function `sprintf` will take the value from client connection variable *varname* , not from list of arguments. Similarly, `sprintf_inverse` will not return fragment that match to *%{varname}U* in the vector of other fragments; instead it will get the value from connection environment and ensure that it matches the fragment of input; mismatch between printed and actual value of variable will means that the whole string do not match the format.

SPARQL optimizer knows about this formatting feature and sometimes it makes more deductions from occurrence of *%{varname}U* than from occurrence of plain *%U* , so this notation may be used in *option (returns ...)* when appropriate. Of course, the optimizer has no access to the actual value of connection variable because it may vary from run to run or may change between the compilation and the run, but the value is supposed to be persistent during any single query run so *%{myvariable}U* in one place is equal to *%{myvariable}U* in other.

Connection variables are set by `connection_set` and some of them have default values that are used if not overridden by application:

- *URIQADefaultHost*

  is for default host as it is specified in Virtuoso configuration file. Note, however, that it will be escaped when printed so if it contains colon and port number then the colon is escaped. In addition, there are special variables that match dynamic renaming of local IRIs more accurately.
- *WSHost*

  is for host and port as it is used by current client connection for dynamic renaming. The colon before port will be escaped.
- *WSHostName*

  is for host name only, without port, as it is used by current client connection for dynamic renaming.
- *WSHostPort*

  is for port part of host IRI. That is string, not integer. The only real use of the variable is in formats like

  http://%{WSHostName}U:%{WSHostPort}U/...

  .

It is inconvenient to write different format strings for different cases. Two most common policies are different host names for default HTTP port of a publicly available service and different non-default ports for one or more host names of an intranet installation; these two approaches are almost never used in a mix. So declaration of IRI classes may use shorthand *^{DynamicLocalFormat}^* in format strings that is expanded either to *http://%{WSHost}U* or to *http://%{WSHostName}U:%{WSHostPort}U/...* , depending on absence or presence of port number in the value of *DefaultHost* parameter of *URIQA* section of configuration file.

Note

*^{DynamicLocalFormat}^* is for IRI class declarations only and is not expanded in any other place, so it is useful sometimes to create an IRI class with empty argument list in order to get "almost constant" IRIs calculated without writing special procedures.

16.5.9. Lookup Optimization -- BIJECTION and RETURNS Options

There is one subtle problem with IRI class declarations. To get benefit from a relational index, SPARQL optimizer should compose equality between table column and some known SQL value, not between return value of IRI class and a known composed IRI. In addition, redundant calculations of IRIs takes time. To enable this optimization, an IRI class declaration should end with *option (bijection)* clause. For some simple format strings the compiler may recognize the bijection automatically but an explicit declaration is always a good idea.

Note:

See also: Wikipedia - Bijection . In mathematics, a bijection, or a bijective function is a function f from a set X to a set Y such that, for every y in Y, there is exactly one x in X such that f(x) = y.

Alternatively, f is bijective if it is a one-to-one correspondence between those sets; i.e., both one-to-one (injective) and onto (surjective).

The SPARQL compiler may produce big amounts of SQL code when the query contains equality of two calculated IRIs and these IRIs may come from many different IRI classes. It is possible to provide hints that will let the compiler check if two IRI classes form disjoint sets of possible IRI values. The more disjoint sets are found the less possible combinations remain so the resulting SQL query will contain fewer unions of joins. The SPARQL compiler can prove some properties of sprintf format strings. E.g., it can prove that set of all strings printed by "http://example.com/item%d" and the set of strings printed by "http://example.com/item%d/" are disjoint. It can prove some more complicated statements about unions and intersections of sets of strings. The IRI or literal class declaration may contain *option (returns ...)* clause that will specify one or more sprintf patterns that cover the set of generated values. Consider a better version of IRI class declaration listed above:

```
create iri class oplsioc:grantee_iri using
  function DB.DBA.GRANTEE_URI (in id integer)
    returns varchar,
  function DB.DBA.GRANTEE_URI_INVERSE (in id_iri varchar)
    returns integer
  option ( bijection,
    returns "http://myhost/sys/group?id=%d"
    union   "http://myhost/sys/user?id=%d" ) .
```

It is very important to keep IRI classes easily distinguishable by the text of IRI string and easy to parse.

- Format

  %U

  is better than

  %s

  , especially in the middle of IRI, because the

  %U

  fragment can not contain characters like "/" or "="; one may prove that

  /%U/

  and

  /abra%d/cadabra/

  are disjoint but

  /%s/

  and

  /abra%d/cadabra/

are not disjoint.

- It is better when the variable part like

  %U

  or

  %d

  is placed between characters that may not occur in the

  %U

  or

  %d

  output, i.e.

  %U

  is placed between "/", "&" or "=" and

  %d

  is placed between non-digits;

  order_line_%d

  is better than

  order-line-%d

  because minus may be part of

  %d

  output.
- End-of-line is treated as a special character, so placing

  %U

  or

  %d

  between "/" and end of line is as good as placing it between two "/".

In some cases *option (returns ...)* can be used for IRI classes that are declared using sprintf format, but actual data have more specific format. Consider a literal class declaration that is used to output strings and the application knows that all these strings are ISBN numbers:

```
create literal class example:isbn_ref "%s" (in isbn varchar not null)
  option ( bijection, returns "%u-%u-%u-%u" union "%u-%u-%u-X" )
```

Sometimes interoperability restrictions will force you to violate these rules but please try to follow them as often as possible.

16.5.10. Join Optimization -- Declaring IRI Subclasses

Additional problem appears when the equality is between two IRIs of two different IRI classes. Even if both of them are bijections, the compiler does not know if these IRI classes behave identically on the intersection of their domains. To let the

optimizer know this fact, one IRI class can be explicitly declared as a subclass of another:

```
make oplsioc:user_iri subclass of oplsioc:grantee_iri .
make oplsioc:group_iri subclass of oplsioc:grantee_iri .
```

The SPARQL compiler can not check the validity of a subclass declaration. The developer should carefully test functions to ensure that transformations are really subclasses, as well as to ensure that functions of an IRI class declarations are really inverse to each other.

When declaring that a table's primary key is converted into a IRI according to one IRI class, one usually declares that all foreign keys referring to this class also get converted into an IRI as per this same class, or subclass of same class.

Subclasses can be declared for literal classes as well as for IRI classes, but this case is rare. The reason is that most of literals are made by identity literal classes that are disjoint to each other even if values may be equal in SQL sense, such as *"2"* of type *xsd:integer* and *"2.0"* of type *xsd:double* .

16.5.11. RDF Metadata Maintenance and Recovery

This section refers to checking and backing up Linked Data View and storage declarations only. The checks and backup/restore do not affect physical quads, relational schema or tables or data therein. For general backup and restore, see server administration. To detect and fix automatically most popular sorts of RDF metadata corruption use `DB.DBA.RDF_AUDIT_METADATA` . It is also possible to backup RDF data by `DB.DBA.RDF_BACKUP_METADATA` and restore the saved state later by using `DB.DBA.RDF_RESTORE_METADATA` . It is convenient to make a backup before any modification of quad storages, quad map patterns or IRI classes, especially during debugging new Linked Data Views.

Note

In SQL, adding a new view can not break anything. This is because SQL lacks the ability of querying "everything" so data sources are always specified. This is not true for SPARQL, so please treat *any* metadata manipulation as potentially destructive operation. If an RDF storage is supposed to be used by more than one application then these applications should be tested together, not one after other, and they should be installed/upgraded on live database in the very same order as they were installed/upgraded on instrumental machine during testing. Always remember that these applications share RDF tables so they may interfere.

16.5.12. Split Linked Data View

Linked Data View can be created by two or more "sparql alter storage" statements. In each statement can be created one quad map that contains mappings for half or a third of all tables. Quad maps created should have distinct names but may mention same graph. The important fact is that if the Linked Data View in question is exclusive for a graph then only the last quad map should be exclusive but all previous should not have this option. This is because if a map is exclusive on a graph the rest of maps on that graph will be silently ignored.

The example below shows a sample part of the Virtuoso eCRM Views code, where the Linked Data View is split in two parts: with quad map virtrdf:ecrmDemo1 and with quad map virtrdf:ecrmDemo2:

```
SPARQL
prefix ecrm: <http://demo.openlinksw.com/schemas/ecrm#>
prefix oplsioc: <http://www.openlinksw.com/schemas/oplsioc#>
prefix sioc: <http://rdfs.org/sioc/ns#>
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix cal: <http://www.w3.org/2002/12/cal/ical#>
prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
prefix product: <http://www.swop-project.eu/ontologies/pmo/product.owl#>
prefix owl: <http://www.w3.org/2002/07/owl#>
drop quad map virtrdf:ecrmDemo1 .
;

SPARQL
prefix ecrm: <http://demo.openlinksw.com/schemas/ecrm#>
prefix oplsioc: <http://www.openlinksw.com/schemas/oplsioc#>
prefix sioc: <http://rdfs.org/sioc/ns#>
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix cal: <http://www.w3.org/2002/12/cal/ical#>
prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
prefix product: <http://www.swop-project.eu/ontologies/pmo/product.owl#>
prefix owl: <http://www.w3.org/2002/07/owl#>
```

```
drop quad map virtrdf:ecrmDemo2 .
;

...

SPARQL
prefix ecrm: <http://demo.openlinksw.com/schemas/ecrm#>
prefix oplsioc: <http://www.openlinksw.com/schemas/oplsioc#>
prefix sioc: <http://rdfs.org/sioc/ns#>
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix cal: <http://www.w3.org/2002/12/cal/ical#>
prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
prefix product: <http://www.swop-project.eu/ontologies/pmo/product.owl#>
prefix owl: <http://www.w3.org/2002/07/owl#>
alter quad storage virtrdf:DefaultQuadStorage
FROM eCRM.DBA.SFA_SALES_QUOTA_VIEW2 as sales_quotas
FROM eCRM.DBA.SFA_COMPANIES_VIEW2 as companies
FROM eCRM.DBA.SFA_COMPANIES as companies_table text literal companies_table.DESCRIPTION of (companies.DES
FROM eCRM.DBA.SFA_CONTACTS_VIEW2 as contacts
FROM eCRM.DBA.SFA_CONTACTS as contacts_table text literal contacts_table.NAME_FIRST of (contacts.NAME_FIR
FROM eCRM.DBA.SFA_EMPLOYMENTS_VIEW2 as employments
FROM eCRM.DBA.SFA_LEADS_VIEW2 as leads
FROM eCRM.DBA.SFA_LEADS as leads_table text literal leads_table.SUBJECT of (leads.SUBJECT)
FROM eCRM.DBA.SFA_OPPORTUNITIES_VIEW2 as opportunities
FROM eCRM.DBA.SFA_OPPORTUNITIES as opportunities_table text literal opportunities_table.OPPORTUNITY_NAME
FROM eCRM.DBA.SFA_ACTIVITIES as activities
FROM eCRM.DBA.SFA_MAIL_MESSAGES as messages
FROM eCRM.DBA.SFA_DOCUMENTS_VIEW2 as documents
FROM eCRM.DBA.SFA_INFLUENCERS_VIEW2 as influencers
FROM eCRM.DBA.SFA_TEAMS_VIEW2 as teams
FROM eCRM.DBA.SFA_NOTES_VIEW2 as notes
FROM eCRM.DBA.SFA_NOTES as notes_table text literal notes_table.DESCRIPTION of (notes.DESCRIPTION)
FROM eCRM.DBA.SFA_COMPETITORS_VIEW2 as competitors
FROM eCRM.DBA.SFA_ISSUES_VIEW2 as issues
FROM eCRM.DBA.SFA_CUSTOM_FIELD_DEFS_VIEW2 as custom_field_defs
FROM eCRM.DBA.SFA_CUSTOM_FIELDS_VIEW2 as custom_fields
FROM eCRM.DBA.SFA_CASES_VIEW2 as cases
FROM eCRM.DBA.SFA_CASES as cases_table text literal cases_table.SUMMARY of (cases.SUMMARY)
FROM eCRM.DBA.SFA_ORDERS_VIEW2 as orders
FROM eCRM.DBA.SFA_ORDERS as orders_table text literal orders_table.EMAIL of (orders.EMAIL)
FROM eCRM.DBA.SFA_ORDER_ITEMS_VIEW2 as order_items
FROM eCRM.DBA.PM_CATEGORIES_VIEW2 as categories
FROM eCRM.DBA.PM_PRODUCT_ATTRIBUTE_DEFS_VIEW2 as product_attribute_defs
FROM eCRM.DBA.PM_PRODUCTS_VIEW2 as products
FROM eCRM.DBA.PM_PRODUCTS as products_table text literal products_table.DESCRIPTION of (products.DESCRIPT
FROM eCRM.DBA.PM_PRODUCT_ATTRIBUTES_VIEW2 as product_attributes
FROM eCRM.DBA.PM_CATALOGS_VIEW2 as catalogs
FROM eCRM.DBA.PM_CATALOG_PRODUCTS_VIEW2 as catalog_products
FROM eCRM.DBA.XSYS_MODULES as modules
FROM eCRM.DBA.XSYS_REGISTRY as registries
FROM eCRM.DBA.XSYS_ORGANIZATIONS_DATA as organizations_data
FROM eCRM.DBA.XSYS_MESSAGES as xsysmessages
FROM eCRM.DBA.XSYS_COUNTRIES_VIEW2 as countries
FROM eCRM.DBA.XSYS_PROVINCES_VIEW2 as provinces
FROM eCRM.DBA.XSYS_TIMEZONES as timezones
FROM eCRM.DBA.XSYS_MIME_TYPES as mimetypes
FROM eCRM.DBA.XSYS_MIME_EXTENSIONS as mimeexts
FROM eCRM.DBA.XSYS_CNAMES as cnames
FROM eCRM.DBA.XSYS_QUOTAS as quotas
FROM eCRM.DBA.XSYS_ROLES as roles
FROM eCRM.DBA.XSYS_ACCOUNTS as accounts
FROM eCRM.DBA.XSYS_USERDATA as userdatas
FROM eCRM.DBA.XSYS_GROUPDATA as groupdatas
FROM eCRM.DBA.XSYS_MEMBERS as members
FROM eCRM.DBA.XSYS_SESSIONS_DATA as sessionsdatas
FROM eCRM.DBA.XSYS_SESSION_DATA as sessiondatas
FROM eCRM.DBA.XSYS_LIST_MEMBERS_DEFS as list_members_defs
FROM eCRM.DBA.XSYS_CLASSES as classes
FROM eCRM.DBA.XSYS_ORG_CLASSES as org_classes
FROM eCRM.DBA.XSYS_CLASS_METHODS as class_methods
FROM eCRM.DBA.XSYS_CLASS_VIEWS as class_views
FROM eCRM.DBA.XSYS_ROLE_PRIVILEGES as role_priveleges
FROM eCRM.DBA.XSYS_USER_PRIVILEGES as user_priveleges
```

```
FROM eCRM.DBA.XSYS_HISTORY as history
FROM eCRM.DBA.XSYS_USERS as xsys_users
FROM eCRM.DBA.AP_PROCESSES_VIEW2 as ap_processes
FROM eCRM.DBA.AP_RULES_VIEW2 as ap_rules
FROM eCRM.DBA.AP_QUEUE as ap_queues
WHERE (^{companies.}^.COUNTRY_NAME = ^{countries.}^.COUNTRY_NAME)
WHERE (^{contacts.}^.COUNTRY_NAME = ^{countries.}^.COUNTRY_NAME)
WHERE (^{leads.}^.COUNTRY_NAME = ^{countries.}^.COUNTRY_NAME)
WHERE (^{products.}^.COUNTRY_NAME = ^{countries.}^.COUNTRY_NAME)
WHERE (^{orders.}^.SHIP_COUNTRY_NAME = ^{countries.}^.COUNTRY_NAME)
WHERE (^{leads_table.}^.FREETEXT_ID = ^{leads.}^.FREETEXT_ID)
WHERE (^{contacts_table.}^.FREETEXT_ID = ^{contacts.}^.FREETEXT_ID)
WHERE (^{companies_table.}^.FREETEXT_ID = ^{companies.}^.FREETEXT_ID)
WHERE (^{opportunities_table.}^.FREETEXT_ID = ^{opportunities.}^.FREETEXT_ID)
WHERE (^{cases_table.}^.FREETEXT_ID = ^{cases.}^.FREETEXT_ID)
WHERE (^{notes_table.}^.FREETEXT_ID = ^{notes.}^.FREETEXT_ID)
WHERE (^{orders_table.}^.FREETEXT_ID = ^{orders.}^.FREETEXT_ID)
WHERE (^{products_table.}^.FREETEXT_ID = ^{products.}^.FREETEXT_ID)
{
        create virtrdf:ecrmDemo1 as graph iri ("http://^{URIQADefaultHost}^/ecrm") option (order 1501)
        {
            ecrm:Country (countries.COUNTRY_NAME)
                a ecrm:Country
                    as virtrdf:Country-Countrys2 ;
                a geo:SpatialThing
                    as virtrdf:Country-Countrys ;
                owl:sameAs ecrm:dbpedia_iri (countries.COUNTRY_NAME) ;
                ecrm:countryID countries.COUNTRY_ID
                        as virtrdf:Country-COUNTRY_ID ;
                ecrm:countryID3 countries.COUNTRY_ID3
                        as virtrdf:Country-COUNTRY_ID3 ;
                ecrm:isoCode countries.ISO_CODE
                        as virtrdf:Country-ISO_CODE ;
                ecrm:countryName countries.COUNTRY_NAME
                        as virtrdf:Country-COUNTRY_NAME .

            ecrm:Country (countries.COUNTRY_NAME)
                        ecrm:has_province
            ecrm:Province (provinces.COUNTRY_ID, provinces.PROVINCE_NAME) where
                        (^{provinces.}^.COUNTRY_ID = ^{countries.}^.COUNTRY_ID) as virtrdf:ecrmCountry-ha

...
  } .
} .
;
SPARQL
prefix ecrm: <http://demo.openlinksw.com/schemas/ecrm#>
prefix oplsioc: <http://www.openlinksw.com/schemas/oplsioc#>
prefix sioc: <http://rdfs.org/sioc/ns#>
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
prefix cal: <http://www.w3.org/2002/12/cal/ical#>
prefix product: <http://www.swop-project.eu/ontologies/pmo/product.owl#>
prefix owl: <http://www.w3.org/2002/07/owl#>
alter quad storage virtrdf:DefaultQuadStorage
FROM eCRM.DBA.SFA_SALES_QUOTA_VIEW2 as sales_quotas
FROM eCRM.DBA.SFA_COMPANIES_VIEW2 as companies
FROM eCRM.DBA.SFA_COMPANIES as companies_table text literal companies_table.DESCRIPTION of (companies.DES
FROM eCRM.DBA.SFA_CONTACTS_VIEW2 as contacts
FROM eCRM.DBA.SFA_CONTACTS as contacts_table text literal contacts_table.NAME_FIRST of (contacts.NAME_FIR
FROM eCRM.DBA.SFA_EMPLOYMENTS_VIEW2 as employments
FROM eCRM.DBA.SFA_LEADS_VIEW2 as leads
FROM eCRM.DBA.SFA_LEADS as leads_table text literal leads_table.SUBJECT of (leads.SUBJECT)
FROM eCRM.DBA.SFA_OPPORTUNITIES_VIEW2 as opportunities
FROM eCRM.DBA.SFA_OPPORTUNITIES as opportunities_table text literal opportunities_table.OPPORTUNITY_NAME
FROM eCRM.DBA.SFA_ACTIVITIES as activities
FROM eCRM.DBA.SFA_MAIL_MESSAGES as messages
FROM eCRM.DBA.SFA_DOCUMENTS_VIEW2 as documents
FROM eCRM.DBA.SFA_INFLUENCERS_VIEW2 as influencers
FROM eCRM.DBA.SFA_TEAMS_VIEW2 as teams
FROM eCRM.DBA.SFA_NOTES_VIEW2 as notes
FROM eCRM.DBA.SFA_NOTES as notes_table text literal notes_table.DESCRIPTION of (notes.DESCRIPTION)
FROM eCRM.DBA.SFA_COMPETITORS_VIEW2 as competitors
```

```
FROM eCRM.DBA.SFA_ISSUES_VIEW2 as issues
FROM eCRM.DBA.SFA_CUSTOM_FIELD_DEFS_VIEW2 as custom_field_defs
FROM eCRM.DBA.SFA_CUSTOM_FIELDS_VIEW2 as custom_fields
FROM eCRM.DBA.SFA_CASES_VIEW2 as cases
FROM eCRM.DBA.SFA_CASES as cases_table text literal cases_table.SUMMARY of (cases.SUMMARY)
FROM eCRM.DBA.SFA_ORDERS_VIEW2 as orders
FROM eCRM.DBA.SFA_ORDERS as orders_table text literal orders_table.EMAIL of (orders.EMAIL)
FROM eCRM.DBA.SFA_ORDER_ITEMS_VIEW2 as order_items
FROM eCRM.DBA.PM_CATEGORIES_VIEW2 as categories
FROM eCRM.DBA.PM_PRODUCT_ATTRIBUTE_DEFS_VIEW2 as product_attribute_defs
FROM eCRM.DBA.PM_PRODUCTS_VIEW2 as products
FROM eCRM.DBA.PM_PRODUCTS as products_table text literal products_table.DESCRIPTION of (products.DESCRIPT
FROM eCRM.DBA.PM_PRODUCT_ATTRIBUTES_VIEW2 as product_attributes
FROM eCRM.DBA.PM_CATALOGS_VIEW2 as catalogs
FROM eCRM.DBA.PM_CATALOG_PRODUCTS_VIEW2 as catalog_products
FROM eCRM.DBA.XSYS_MODULES as modules
FROM eCRM.DBA.XSYS_REGISTRY as registries
FROM eCRM.DBA.XSYS_ORGANIZATIONS_DATA as organizations_data
FROM eCRM.DBA.XSYS_MESSAGES as xsysmessages
FROM eCRM.DBA.XSYS_COUNTRIES_VIEW2 as countries
FROM eCRM.DBA.XSYS_PROVINCES_VIEW2 as provinces
FROM eCRM.DBA.XSYS_TIMEZONES as timezones
FROM eCRM.DBA.XSYS_MIME_TYPES as mimetypes
FROM eCRM.DBA.XSYS_MIME_EXTENSIONS as mimeexts
FROM eCRM.DBA.XSYS_CNAMES as cnames
FROM eCRM.DBA.XSYS_QUOTAS as quotas
FROM eCRM.DBA.XSYS_ROLES as roles
FROM eCRM.DBA.XSYS_ACCOUNTS as accounts
FROM eCRM.DBA.XSYS_USERDATA as userdatas
FROM eCRM.DBA.XSYS_GROUPDATA as groupdatas
FROM eCRM.DBA.XSYS_MEMBERS as members
FROM eCRM.DBA.XSYS_SESSIONS_DATA as sessionsdatas
FROM eCRM.DBA.XSYS_SESSION_DATA as sessiondatas
FROM eCRM.DBA.XSYS_LIST_MEMBERS_DEFS as list_members_defs
FROM eCRM.DBA.XSYS_CLASSES as classes
FROM eCRM.DBA.XSYS_ORG_CLASSES as org_classes
FROM eCRM.DBA.XSYS_CLASS_METHODS as class_methods
FROM eCRM.DBA.XSYS_CLASS_VIEWS as class_views
FROM eCRM.DBA.XSYS_ROLE_PRIVILEGES as role_priveleges
FROM eCRM.DBA.XSYS_USER_PRIVILEGES as user_priveleges
FROM eCRM.DBA.XSYS_HISTORY as history
FROM eCRM.DBA.XSYS_USERS as xsys_users
FROM eCRM.DBA.AP_PROCESSES_VIEW2 as ap_processes
FROM eCRM.DBA.AP_RULES_VIEW2 as ap_rules
FROM eCRM.DBA.AP_QUEUE as ap_queues
WHERE (^{companies.}^.COUNTRY_NAME = ^{countries.}^.COUNTRY_NAME)
WHERE (^{contacts.}^.COUNTRY_NAME = ^{countries.}^.COUNTRY_NAME)
WHERE (^{leads.}^.COUNTRY_NAME = ^{countries.}^.COUNTRY_NAME)
WHERE (^{products.}^.COUNTRY_NAME = ^{countries.}^.COUNTRY_NAME)
WHERE (^{orders.}^.SHIP_COUNTRY_NAME = ^{countries.}^.COUNTRY_NAME)
WHERE (^{leads_table.}^.FREETEXT_ID = ^{leads.}^.FREETEXT_ID)
WHERE (^{contacts_table.}^.FREETEXT_ID = ^{contacts.}^.FREETEXT_ID)
WHERE (^{companies_table.}^.FREETEXT_ID = ^{companies.}^.FREETEXT_ID)
WHERE (^{opportunities_table.}^.FREETEXT_ID = ^{opportunities.}^.FREETEXT_ID)
WHERE (^{cases_table.}^.FREETEXT_ID = ^{cases.}^.FREETEXT_ID)
WHERE (^{notes_table.}^.FREETEXT_ID = ^{notes.}^.FREETEXT_ID)
WHERE (^{orders_table.}^.FREETEXT_ID = ^{orders.}^.FREETEXT_ID)
WHERE (^{products_table.}^.FREETEXT_ID = ^{products.}^.FREETEXT_ID)
{
        create virtrdf:ecrmDemo2 as graph iri ("http://^{URIQADefaultHost}^/ecrm") option (exclusive, ord
        {
            ecrm:Order (orders.ORG_ID, orders.ORDER_ID)
                a ecrm:Order
                    as virtrdf:Order-Orders ;
                ecrm:has_ecrm_organization ecrm:OrganizationsData(orders.ORG_ID, organizations_data.DNS_Z
                        as virtrdf:Order-ORG_ID ;
                ecrm:owner ecrm:XSys_User(orders.ORG_ID, xsys_users.ACCOUNT_NAME, orders.OWNER_ID)
                        where (^{orders.}^.OWNER_ID = ^{xsys_users.}^.ACCOUNT_ID and ^{orders.}^.ORG_ID =
                        as virtrdf:Order-OWNER_ID ;
                ecrm:FREETEXT_ID orders.FREETEXT_ID
                        as virtrdf:Order-FREETEXT_ID ;
                ecrm:has_company ecrm:Company(orders.COMPANY_NAME, orders.COMPANY_ID, orders.ORG_ID)
                        as virtrdf:Order-COMPANY_ID ;
```

```
                    ecrm:companyName orders.COMPANY_NAME
                            as virtrdf:Order-COMPANY_NAME ;
                    ecrm:has_contact ecrm:Contact(contacts.NAME_FIRST, contacts.NAME_MIDDLE, contacts.NAME_LA
                            where (^{orders.}^.CONTACT_ID = ^{contacts.}^.CONTACT_ID and ^{orders.}^.ORG_ID =
                            as virtrdf:Order-CONTACT_ID ;
                    ecrm:contactName orders.CONTACT_NAME
                            as virtrdf:Order-CONTACT_NAME ;
                    ecrm:orderNo orders.ORDER_NO
                            as virtrdf:Order-ORDER_NO ;
                    ecrm:shipFirstName orders.SHIP_FNAME
                            as virtrdf:Order-SHIP_FNAME ;
                    ecrm:shipSecondName orders.SHIP_SNAME
                            as virtrdf:Order-SHIP_SNAME ;
                    ecrm:phoneNumber orders.PHONE_NUMBER
                            as virtrdf:Order-PHONE_NUMBER ;
                    ecrm:phoneExtension orders.PHONE_EXTENSION
                            as virtrdf:Order-PHONE_EXTENSION ;
                    ecrm:email orders.EMAIL
                            as virtrdf:Order-EMAIL ;
                    ecrm:shipCountry ecrm:Country(orders.SHIP_COUNTRY_NAME)
                            as virtrdf:Order-SHIP_COUNTRY_NAME ;
                    ecrm:shipCountryCode ecrm:Country (countries.COUNTRY_NAME) where  (^{countries.}^.COUNTRY
                            as virtrdf:Order-SHIP_COUNTRY_CODE ;
                    ecrm:shipProvince orders.SHIP_PROVINCE
                            as virtrdf:Order-SHIP_PROVINCE ;
                    ecrm:shipCity orders.SHIP_CITY
                            as virtrdf:Order-SHIP_CITY ;
                    ecrm:dbpedia_shipCity ecrm:dbpedia_iri (orders.SHIP_CITY)
                            as virtrdf:Order-SHIP_dbpedia_CITY ;
                    ecrm:shipPostalCode orders.SHIP_POSTAL_CODE
                            as virtrdf:Order-SHIP_POSTAL_CODE ;
                    ecrm:shipAddress1 orders.SHIP_ADDRESS1
                            as virtrdf:Order-SHIP_ADDRESS1 ;
                    ecrm:shipAddress2 orders.SHIP_ADDRESS2
                            as virtrdf:Order-SHIP_ADDRESS2 ;
                    ecrm:salesRep orders.SALESREP
                            as virtrdf:Order-SALESREP ;
                    ecrm:orderDate orders.ORDER_DATE
                            as virtrdf:Order-ORDER_DATE ;
                    ecrm:orderValue orders.ORDER_VALUE
                            as virtrdf:Order-ORDER_VALUE ;
                    ecrm:refund orders.REFUND
                            as virtrdf:Order-REFUND ;
                    ecrm:year orders.YEAR
                            as virtrdf:Order-YEAR ;
                    ecrm:month orders.MONTH
                            as virtrdf:Order-MONTH ;
                    ecrm:quarter orders.QUARTER
                            as virtrdf:Order-QUARTER ;
                    ecrm:financialYear orders.FINANCIAL_YEAR
                            as virtrdf:Order-FINANCIAL_YEAR ;
                    ecrm:CONTACT_REL_ID orders.CONTACT_REL_ID
                            as virtrdf:Order-CONTACT_REL_ID ;
                    ecrm:COMPANY_REL_ID orders.COMPANY_REL_ID
                            as virtrdf:Order-COMPANY_REL_ID .

...
            } .
} .
;
```

16.5.13. Linked Data Views and recursive FK relationships

Here is sample example of a script to include an additional table alias for a table:

```
alter quad storage virtrdf:DefaultQuadStorage
  :
  FROM isports_rdf.prs10_isports_rdf.VRef_Call      as Ref_Call_tbl
  FROM isports_rdf.prs10_isports_rdf.VRef_Call      as Ref_Call_tbl_1
  :
{
```

```
  :
  refcall:ref-call_iri (Ref_Call_tbl.Call_Num) a refcall:Ref-Call as
virtrdf:ref-call_pk ;
  :
  refcall:has_parent  refcall:ref-call_iri (Ref_Call_tbl_1.Call_Num)
where  ( ^{Ref_Call_tbl.}^.Parent    = ^{Ref_Call_tbl_1.}^.Call_Num )   as
virtrdf:Ref-Call_has_parent .
```

This demonstrates the way to self-join the table VRef_Call with itself. Like in SQL, are needed two different aliases for one table if you want to join it with itself.

16.6. Automated Generation of Linked Data Views over Relational Data Sources

16.6.1. Introduction

Virtuoso offers from Conductor UI an HTML based Wizard interface for dynamically generating & publishing RDF based Linked Data from ODBC or JDBC accessible relational data sources. Basically, a mechanism for building RDF based Linked Data views over relational data sources.

The proliferation of relational databases across enterprises and behind Web sites, makes them a vital data source for the burgeoning Linked Data Web. Thus, the process of publishing Linked Data from these sources needs to be as unobtrusive as possible. Naturally, a balance has to be struck between unobtrusive generation of Linked Data and traditional relational database management system (RDBMS) virtues such as:

- Scalability
- Security
- Analytical Expressivity of SQL
- Separation of Data Access and Data Storage via ODBC, JDBC, ADO.NET CLIs.

The following steps must be taken to publish RDF-based Linked Data:

1. Identifying ODBC or JDBC data sources that host the data you seek to publish (assuming the data isn't Virtuoso RDBMS hosted -- in which case, skip ahead to step #3).
2. Attach/Link TABLEs or VIEWs from the external data sources into Virtuoso via their Data Source Names (DSNs).
3. Identify the internal or external TABLEs or VIEWs that hold the data you wish to publish.
4. Configure Endpoints and Re-write Rules to disambiguate data object (resource) identity and description through HTTP-based content negotiation.
5. Expose the Data Source Ontology and associated Instance Data in Linked Data form through those Endpoints and Re-write Rules.

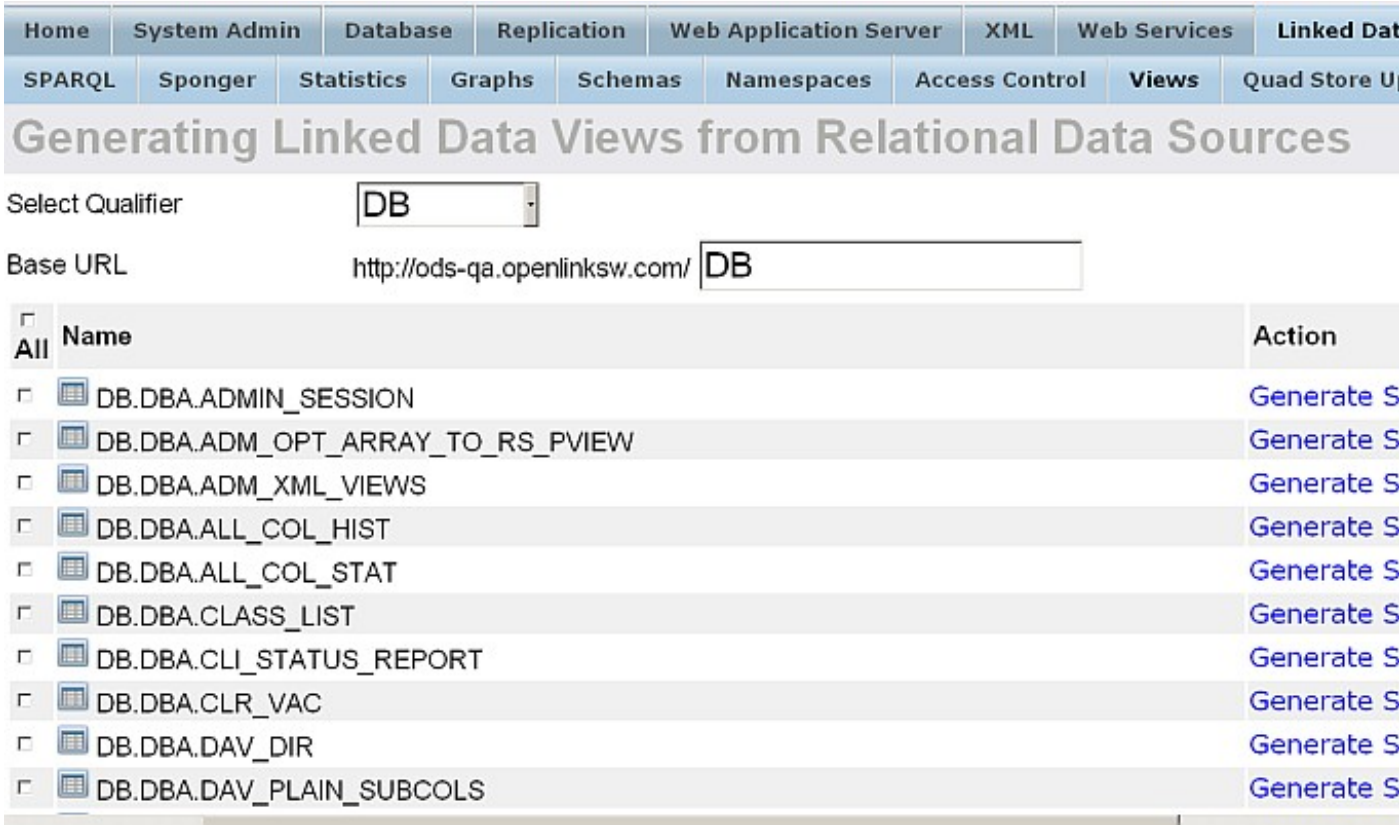
These steps may be largely automated (the "One-Click" Deployment below), or performed manually ("Using the Conductor's HTML-based Wizard" further down).

16.6.2. One Click Linked Data Generation & Deployment

The following steps provide a one-click guide for publishing ODBC- or JDBC-accessible RDBMS data in RDF Linked Data form, using the "Generate & Publish" Conductor feature.

1. Go to http://<cname>:port/conductor ;
2. Log in as user dba (or another user with DBA privileges);
3. Follow menu path Linked Data -> Views;

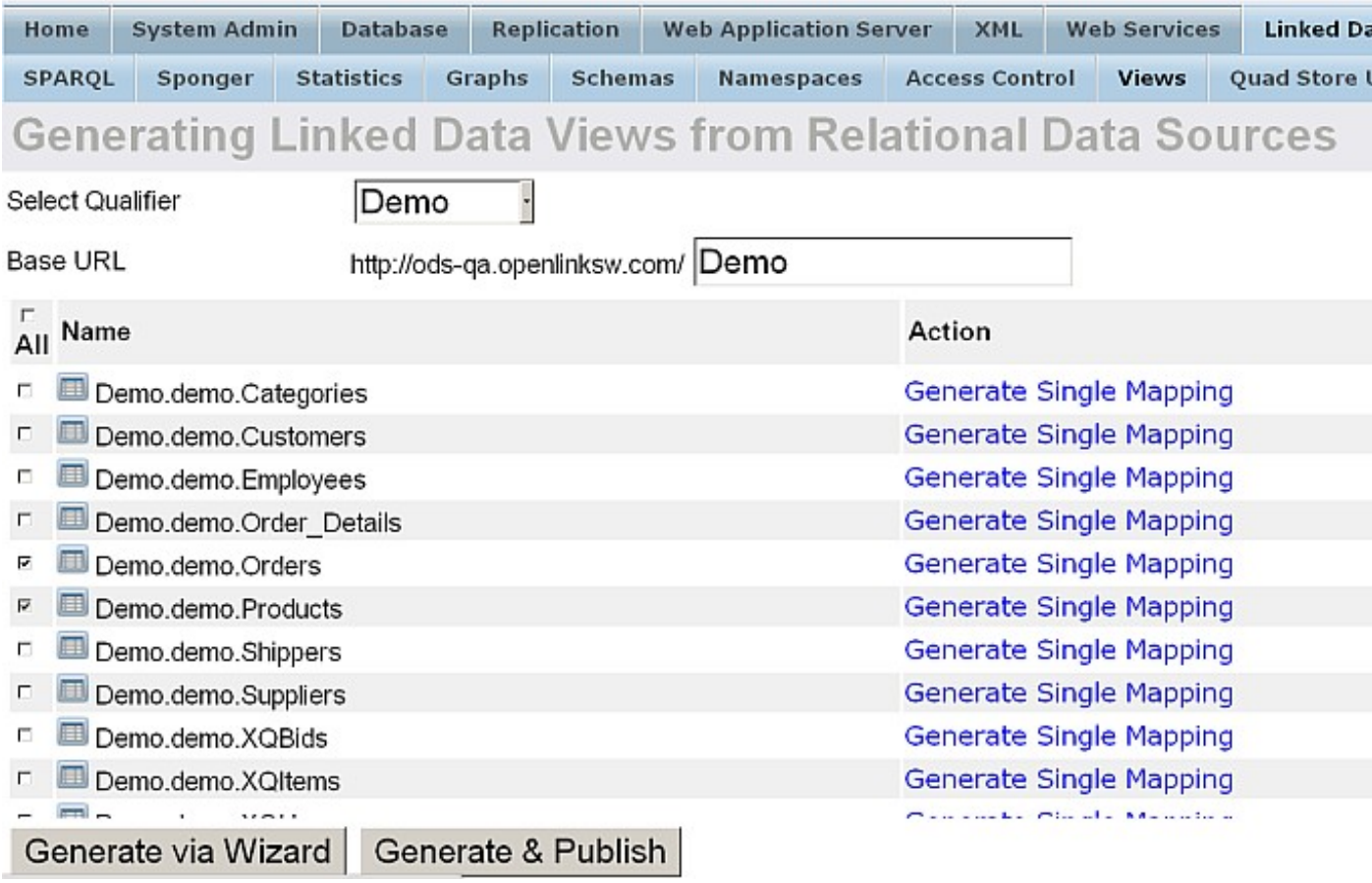
Figure 16.52. Linked Data Views



4. In the form presented, perform the following steps:

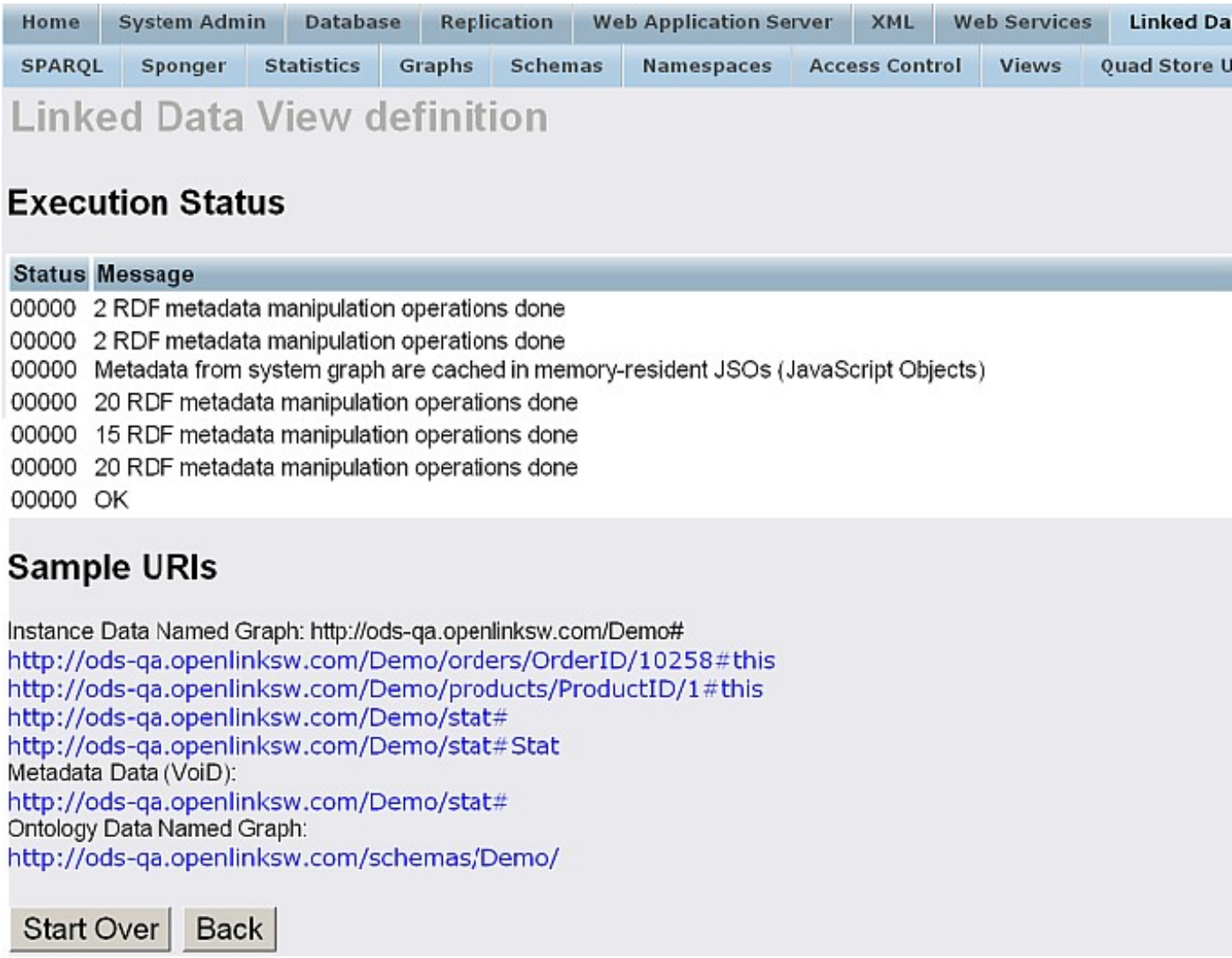
    a. Select the Database Name Qualifier (e.g., "Demo") that exposes the Tables / Views for this exercise
    b. Enter the Base URL to which your URL rewrite rules will be bound (e.g. http://<cname>:8890/Demo)
    c. Select specific Tables containing the data to be published (e.g. Demo.demo.Orders and Demo.demo.Products)
    d. Click the "Generate & Publish" button

Figure 16.53. Linked Data Views Generate and Publish



5. Virtuoso will perform the entire process of ontology generation, instance data generation, and linked data deployment (re-write rules generation and application).
6. Error messages will be presented if the Wizard encounters problems. If there are no error messages, your Linked Data View declarations and Linked Data publishing activities will have completed successfully.

Figure 16.54. Linked Data View declarations and Linked Data publishing activities



7. Optionally, you could also perform one of the following tasks:

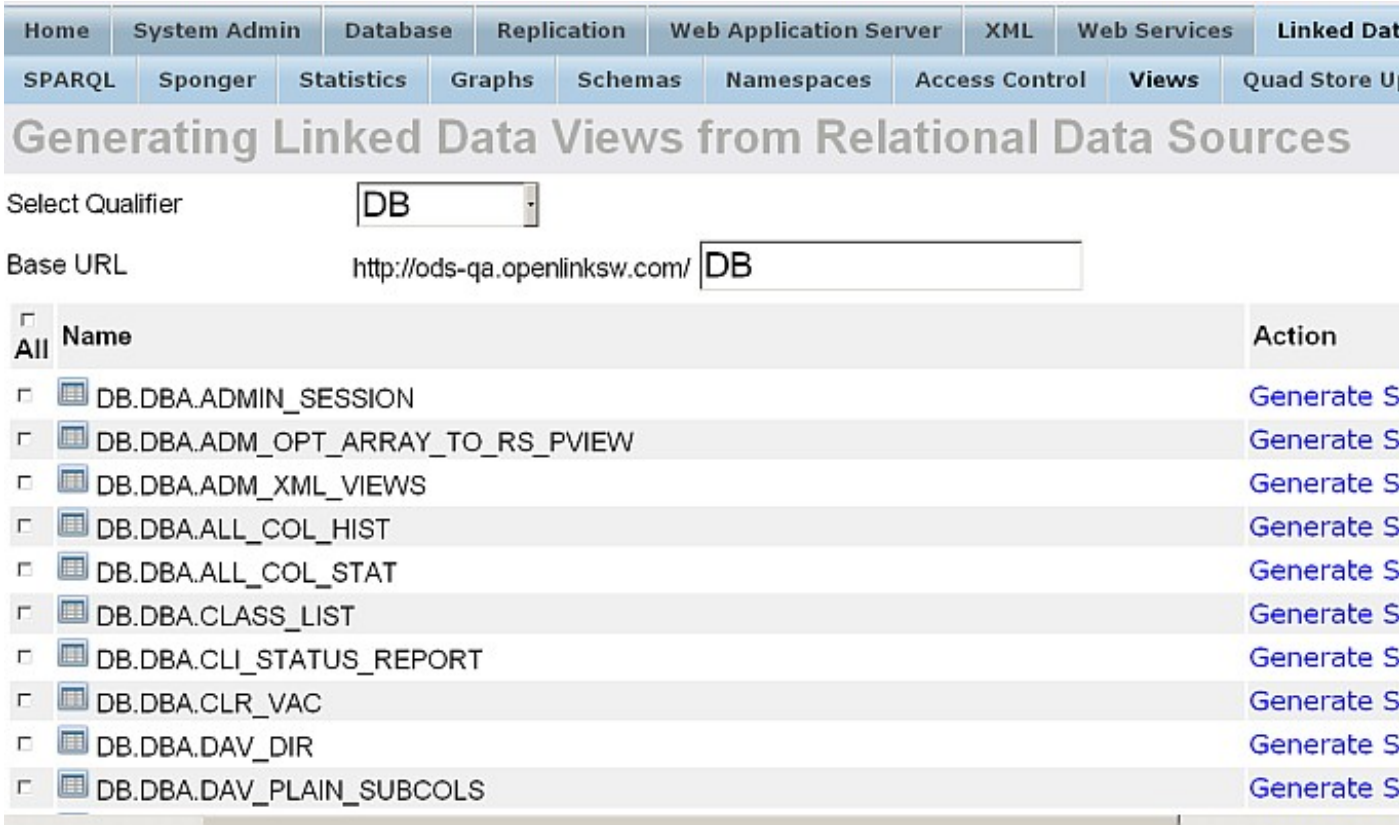
    a. Save Data Mappings: when clicked, offers to save the generated Definitions to local file system
    b. Save Ontology Mappings: when clicked, offers to save the generated Ontology to local file system
    c. Click on the "Cancel" should you want to return to the initial Linked Data View Generation form.

16.6.3. Manual Linked Data Generation & Deployment using the Conductor's HTML-based wizard

The following step-by guide will lead you through manually publishing ODBC- or JDBC-accessible RDBMS data in RDF Linked Data form, using the Conductor's HTML-based wizard:

1. Go to http://<cname>:port/conductor
2. Log in as user dba (or another user with DBA privileges)
3. Follow menu path Linked Data -> Views

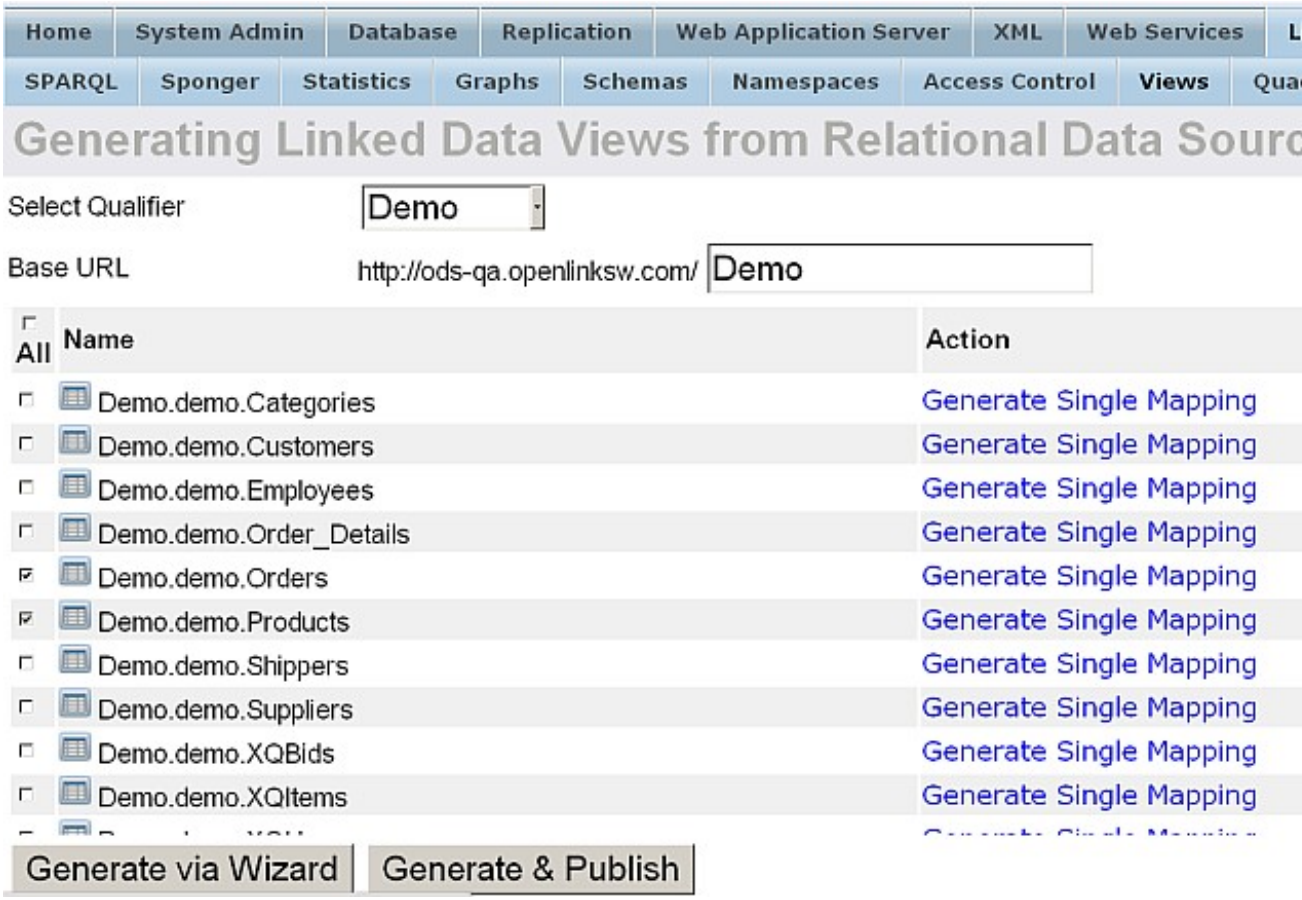
Figure 16.55. Linked Data Views



4. In the form presented, perform the following steps:

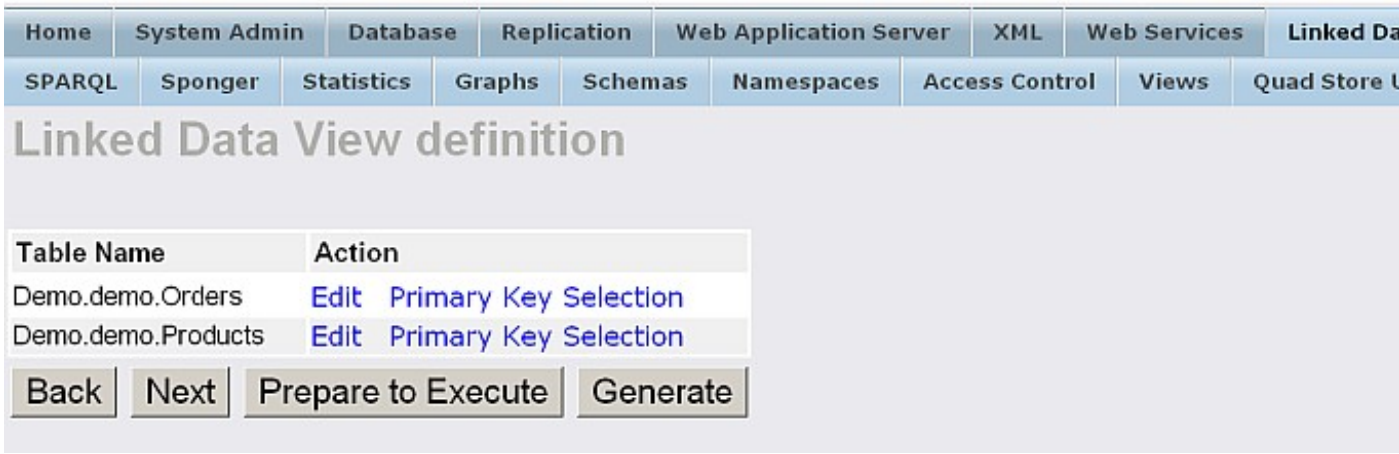
    a. Select the Database Name Qualifier (e.g., "Demo") that exposes the Tables / Views for this exercise
    b. Enter the Base URL to which your URL rewrite rules will be bound (e.g. http://<cname>:8890/Demo)
    c. Select specific Tables containing the data to be published (e.g., Demo.demo.Orders and Demo.demo.Products)
    d. Click the "Generate via Wizard" button

    Figure 16.56. Generate via Wizard



5. At this point, you are presented with the option to edit your column selection. Select the "Edit" link, for example, for table Demo.demo.Products.

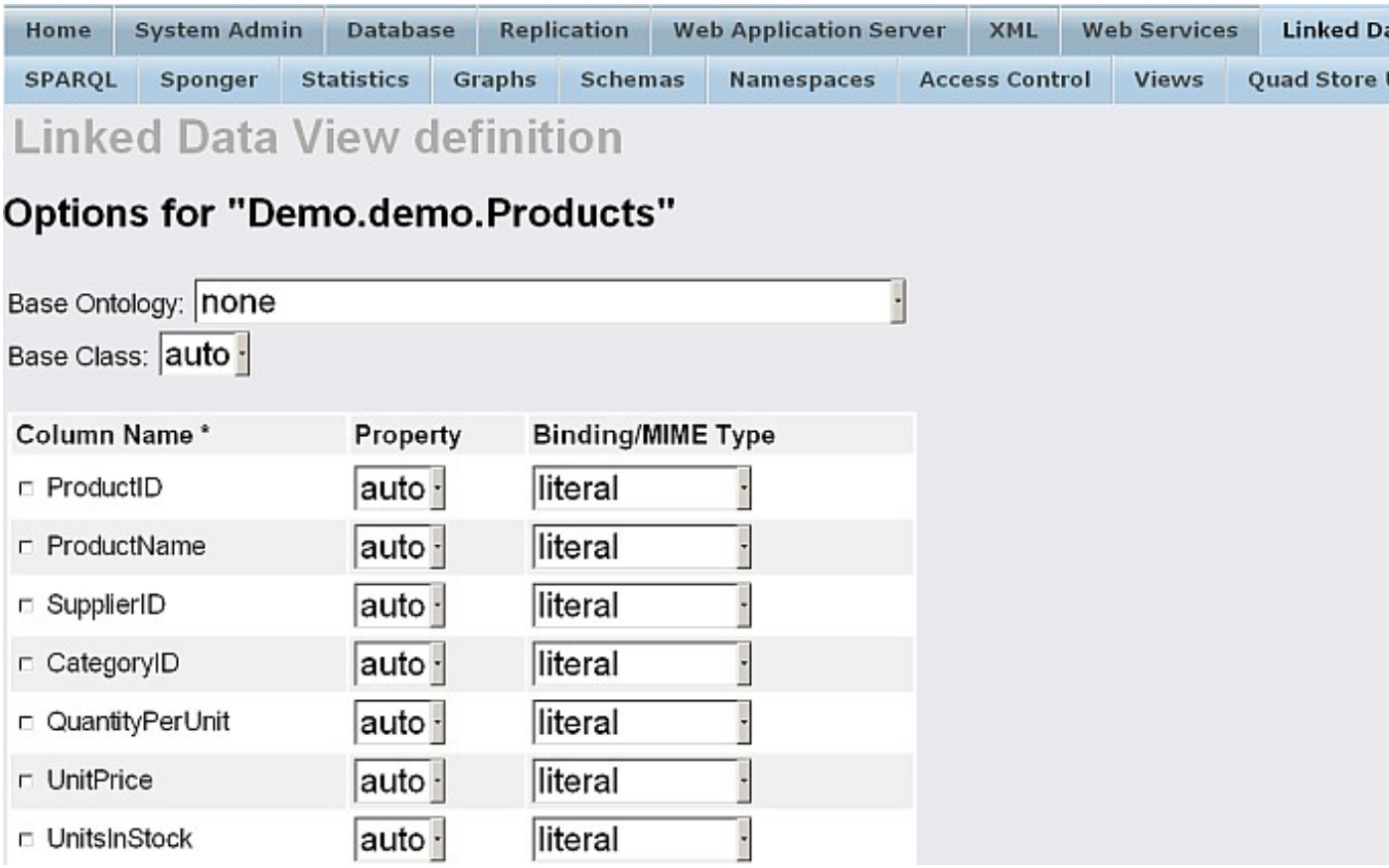
Figure 16.57. Column Selection



6. For images or other binary data in MIME formats to be revealed as anything other than generic "binary objects", you must map large varbinary types to the appropriate MIME types like image/gif. To do so, select the Edit link for Binding/MIME Type of the relevant table columns. You can:

    a. Leave the Binding/MIME Type literal; or
    b. Set to skip, such that the column will not be used in RDF generation; or
    c. Select the binary object value in order for the column to be referenced as binary.

Figure 16.58. Binding/MIME Types



7. After finishing with your changes click the Save button, or cancel the changes and go back by clicking the Cancel button.
8. Make sure you click the "Next" button.
9. At this point, the Linked Data View Definition form will let you Select Generation Targets options:

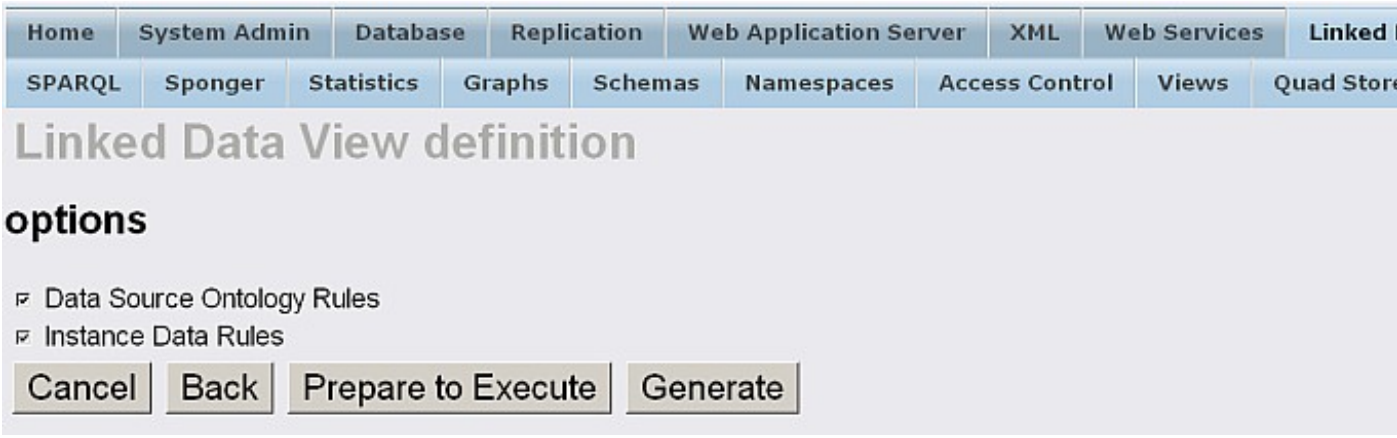
    a. Data Source Ontology Mappings
    b. Instance Data View Mappings
    c. VoID statistic

Figure 16.59. Generation Targets options

10. Make sure you click the "Next" button.
11. Based on your selections in the prior form, the Linked Data View Definition Deployment Options form will be offered:

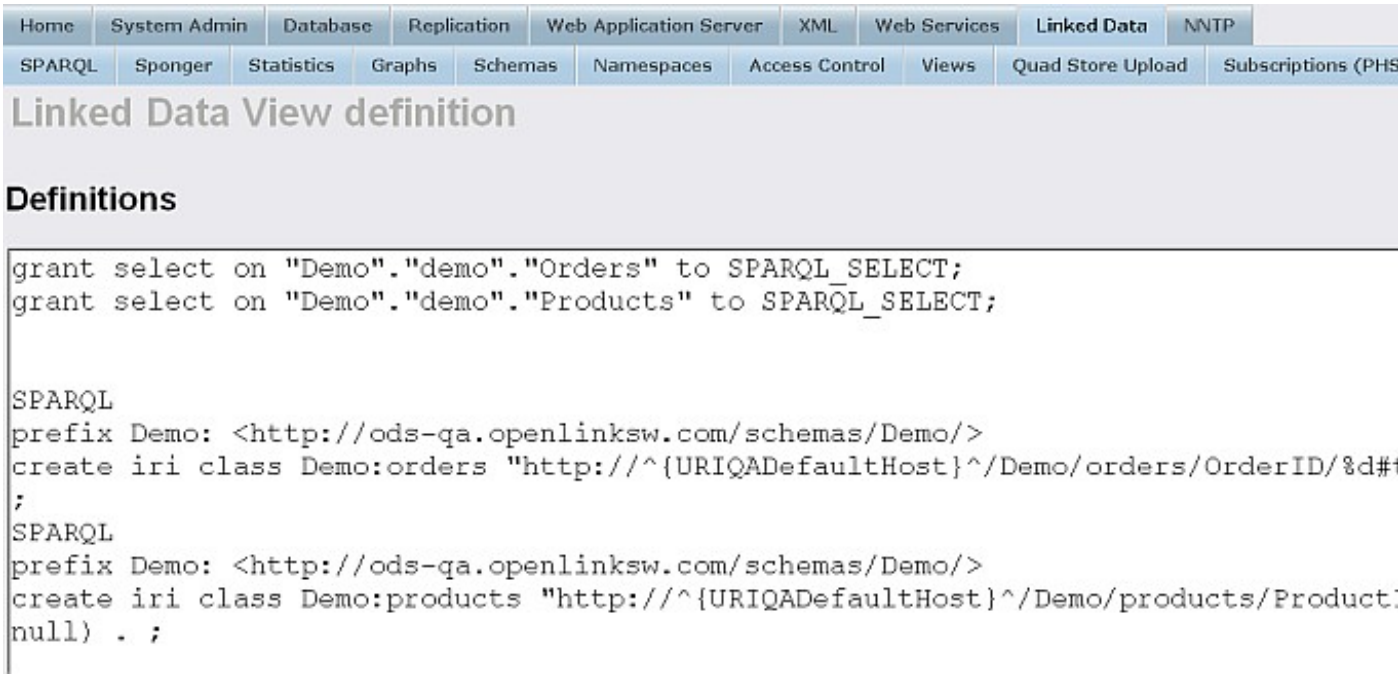
    a. Data Source Ontology Rules
    b. Instance Data Rules

Figure 16.60. Generation Targets options



12. Select the desired option(s) and click the "Prepare to Execute" button which unveils a generated Instance Data and/or Ontology form.

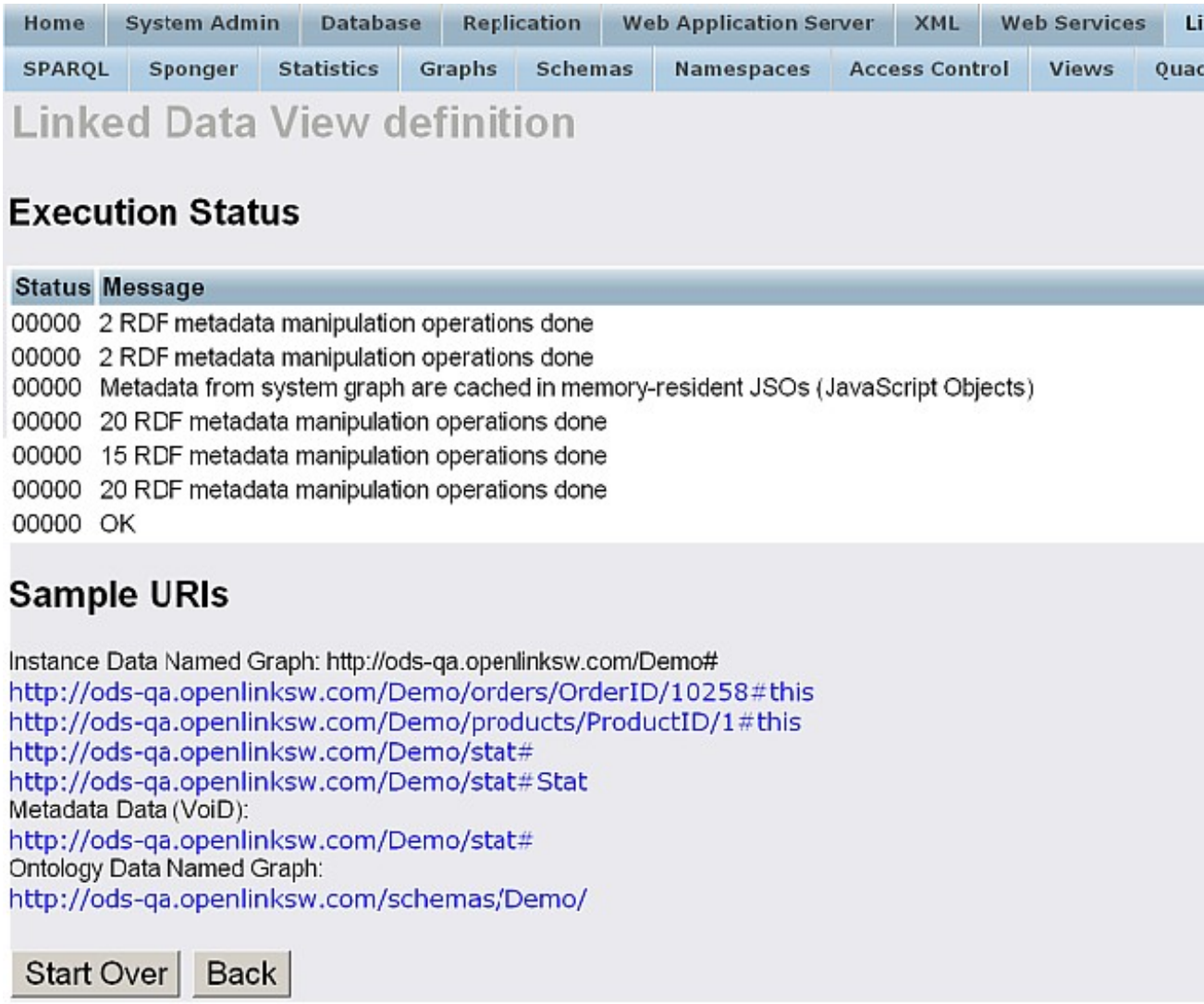
Figure 16.61. Instance Data and/or Ontology



13. Click the Execute button and Virtuoso will:

    a. Apply the generated declarations (instance data and ontology) to your Virtuoso instance
    b. Publish / Deploy declarations that expose the Wizard-generated Rewrite Rules and associated endpoints.

Figure 16.62. Publishing / Deployment declarations

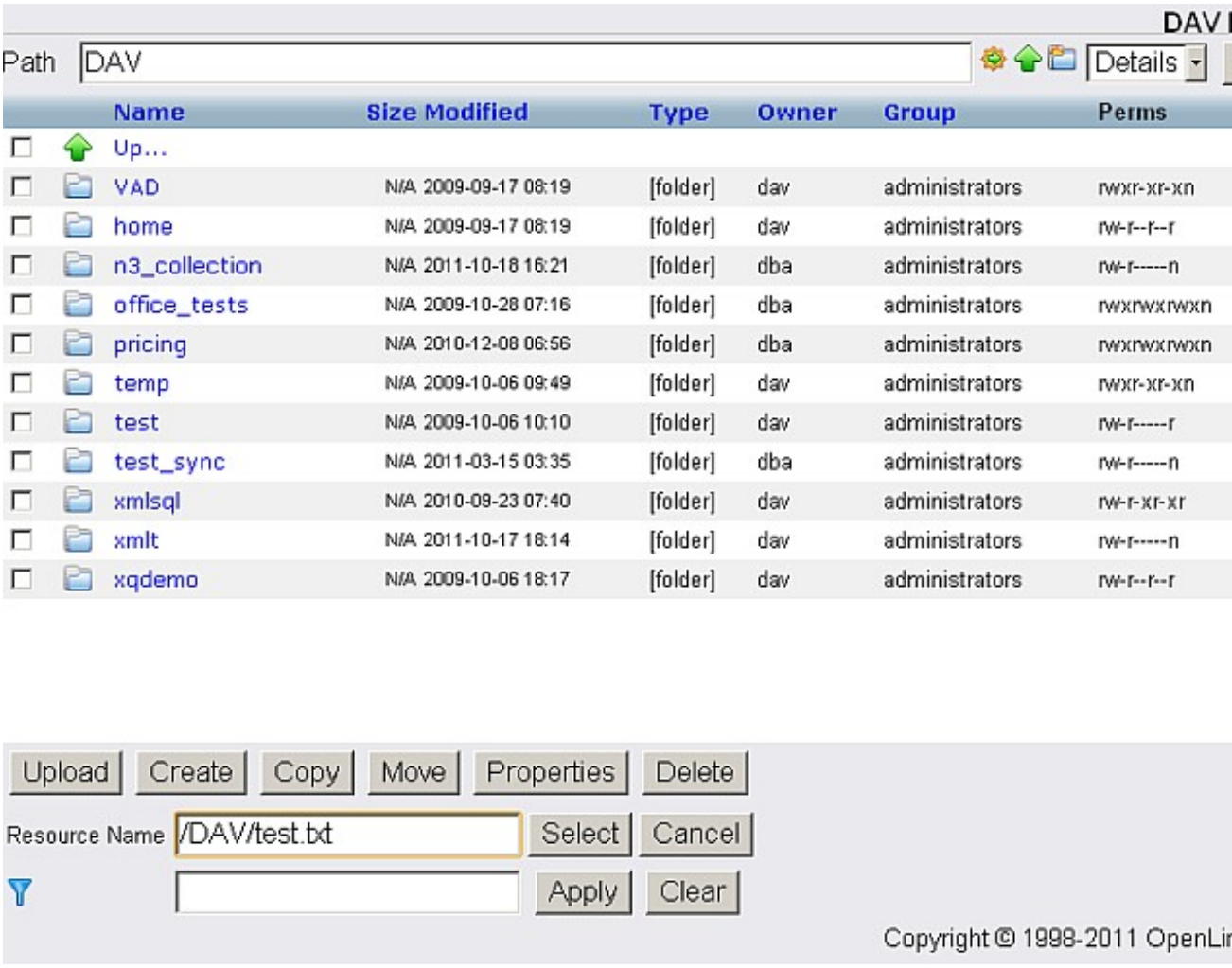


14. Optionally, you can also perform one of the following tasks:

    a. Save Data Mappings: when clicked, offers to save the generated Definitions to local file system
    b. Save Ontology Mappings: when clicked, offers to save the generated Ontology to local file system
    c. Export as WebDAV resource: exports the selected objects/items as a WebDAV resource:

        i. Click "Browse"
        ii. Enter a WebDAV resource and click the "Select" button.

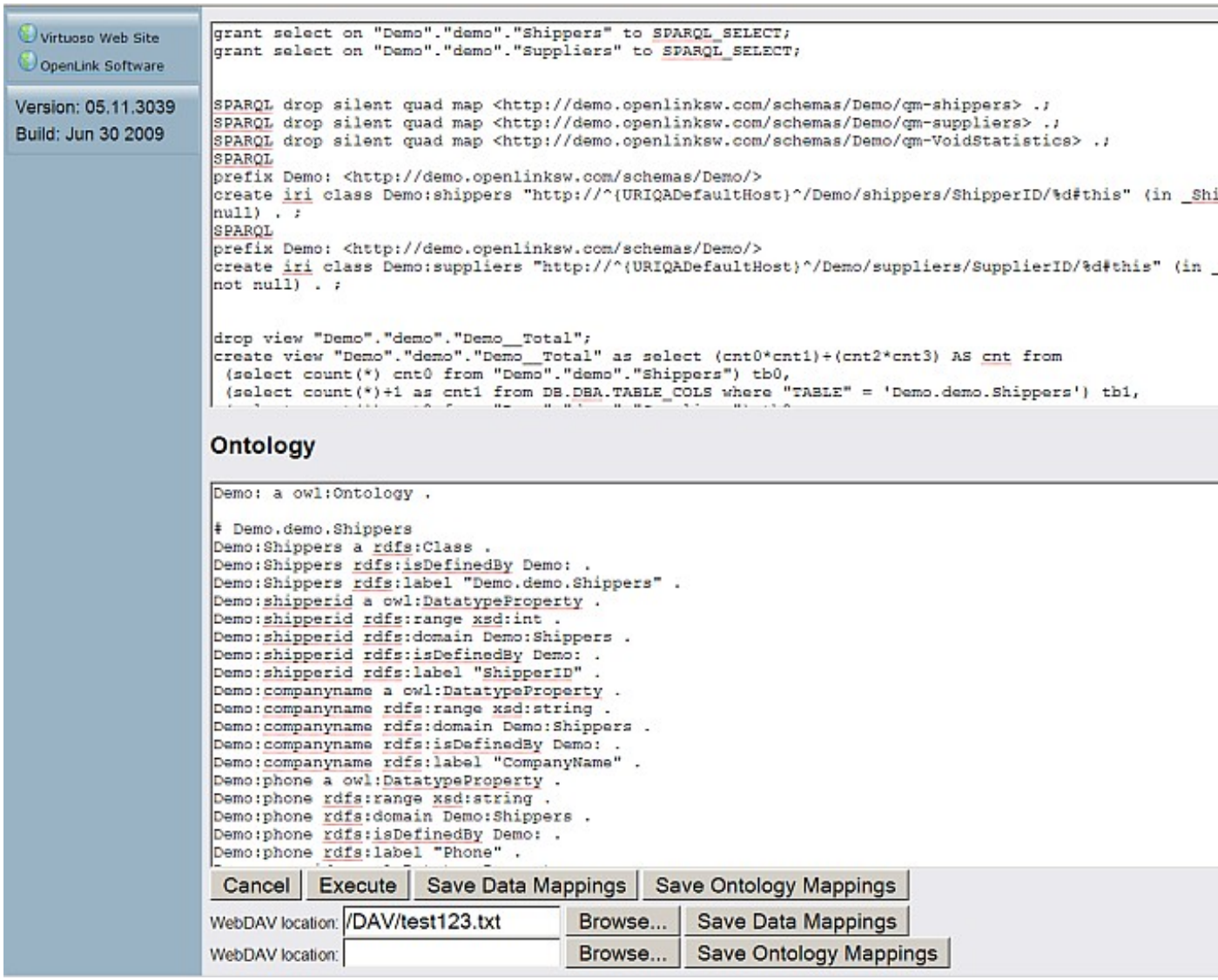
Figure 16.63. WebDAV resource



Note, the WebDAV resource path value will be shown in the WebDAV location field.

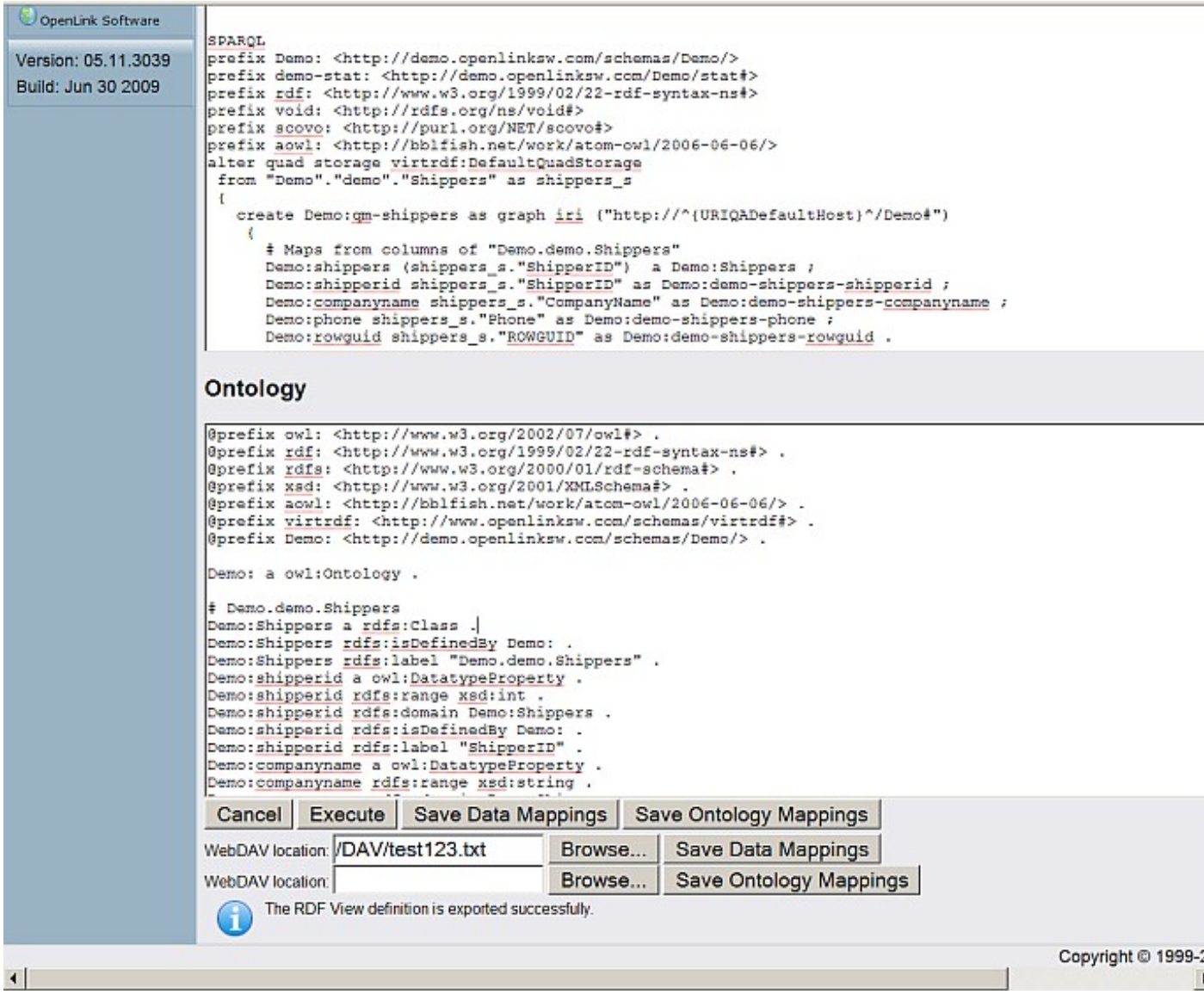
15. Then click the "Save Data Mappings" or "Save Ontology Mappings" button, to complete the option task of saving your generated (or edited) view declarations.

Figure 16.64. WebDAV resource



16. Error messages will be presented if the Wizard encounters problems. If there are no error messages, your Linked Data View declarations and Linked Data publishing activities will have completed successfully.

Figure 16.65. Linked Data View declarations and Linked Data publishing activities Finish



17. Click on Cancel to return to the initial Linked Data View Generation form.

16.6.4. Manual Linked Data Generation & Deployment using iSQL command-line

To generate data synchronization triggers and/or actual RDF quad store data from transient Linked Data views, one can use the Virtuoso built-in `RDF_VIEW_SYNC_TO_PHYSICAL()` function calling it from the iSQL command-line. See `more details and usage examples.`

16.7. Virtuoso R2RML Support

16.7.1. What is R2RML?

R2RML is a language for expressing customized mappings from relational databases to RDF data sets. Such mappings provide the ability to view existing relational data in the RDF data model, expressed in a structure and target vocabulary of the mapping author's choice.

R2RML mappings are themselves RDF graphs written in Turtle syntax.

16.7.2. Why use it?

As a W3C working draft, R2RML is becoming the generic standard adopted by most vendors of tools mapping relational data to RDF, enabling the interoperability of R2RML scripts, whether created with such tools or by hand.

16.7.3. How do I use it with Virtuoso?

Virtuoso has its own previously-developed proprietary equivalent of R2RML called Linked Data Views , which uses Virtuoso's Meta Schema Mapping Language to map relational data to RDF.

R2RML support is achieved by the inclusion of a simple translator which basically translates R2RML syntax to Virtuoso's own Linked Data Views syntax, which can then be executed to create the Linked Data Views themselves.

Install R2RML VAD package

First you will need to ensure you have the R2RML VAD package (rdb2rdf_dav.vad) installed.

Test with simple test script (basic.sql)

Having installed the R2RML VAD package, to test R2RML functionality, the easiest way is by executing a basic.sql script via the command line isql tool:

```
CREATE TABLE "R2RML"."TEST"."PRODUCT"(
  id integer primary key,
  name VARCHAR(100)
);

INSERT SOFT "R2RML"."TEST"."PRODUCT" VALUES (1, 'Virtuoso');

SPARQL CLEAR GRAPH <http://temp/product>;
SPARQL CLEAR GRAPH <http://example.com/>;

DB.DBA.TTLP ('
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix exa: <http://example.com/ns#> .
@prefix product: <http://example.com/product#> .

<http://example.com/ns#TriplesMap1>
    a rr:TriplesMap;

    rr:logicalTable
    [
      rr:tableSchema "R2RML";
      rr:tableOwner "TEST";
      rr:tableName  "PRODUCT"
    ];

    rr:subjectMap
    [
      rr:template "http://example.com/product/{id}";
      rr:class exa:product;
      rr:graph <http://example.com/>;
    ];

    rr:predicateObjectMap
    [
      rr:predicate product:id;
      rr:objectMap [ rr:column "id" ];
    ];

    rr:predicateObjectMap
    [
      rr:predicate product:name;
      rr:objectMap [ rr:column "name" ];
    ];
.
', 'http://temp/product', 'http://temp/product' )
;

--select DB.DBA.R2RML_TEST ('http://temp/product');

--DB.DBA.OVL_VALIDATE ('http://temp/product', 'http://www.w3.org/ns/r2rml#OVL');

-- Running the validation in order to find error in name of R2RML description graph
```

```
--DB.DBA.OVL_VALIDATE ('http://temp/product-nosuch', 'http://www.w3.org/ns/r2rml#OVL');

-- Running the validation in order to find error in name of R2RML metadata graph
--DB.DBA.OVL_VALIDATE ('http://temp/product', 'http://www.w3.org/ns/r2rml#OVL-nosuch');

--select DB.DBA.R2RML_EXECUTE ('http://temp/product');

exec ('sparql ' || DB.DBA.R2RML_MAKE_QM_FROM_G ('http://temp/product'));

--sparql select distinct ?g where { graph ?g { ?s a ?t }};

SPARQL
SELECT * FROM <http://example.com/>
WHERE {?s ?p ?o .};
```

1. First, copy basic.sql into:

   ```
   <VIRTUOSO_INSTALL>/bin/
   ```

2. Next, open Unix session or Windows Command Prompt and execute:

   ```
   cd <OPENLINK_INSTALL>/bin

   ./isql     (Unix)
   isql.exe   (Windows)

   OpenLink Interactive SQL (Virtuoso), version 0.9849b.
   Type HELP; for help and EXIT; to exit.
   SQL>
   ```

3. Then, within isql execute:

   ```
   SQL> load basic.sql;
   ```

4. Execution should finish with a simple SPARQL query that will return Linked Data for the test table created at the start of the script:

   ```
   s                              p                                                  o
   VARCHAR                        VARCHAR                                            VARCHAR
   _______________________________________________________________________________________________

   http://example.com/product/1   http://example.com/product#id                      1
   http://example.com/product/1   http://example.com/product#name                    Virtuoso
   http://example.com/product/1   http://www.w3.org/1999/02/22-rdf-syntax-ns#type    http://example.
   3 Rows. -- 0 msec.
   ```

Note : Subsequent executions of basic.sql will return an error since the test table will already exist. However, the remainder of the script will execute fine.

Examining basic.sql

1. We start by creating and populating the test table:

   ```
   CREATE TABLE "R2RML"."TEST"."PRODUCT"
     (
       id   INTEGER PRIMARY KEY ,
       name VARCHAR(100)
     );

   INSERT SOFT "R2RML"."TEST"."PRODUCT"
     VALUES
       (
         1, 'Virtuoso'
       );
   ```

2. Next we clear any graphs (temporary or permanent) that are to be used during this process:

   ```
   SPARQL CLEAR GRAPH <http://temp/product> ;
   SPARQL CLEAR GRAPH <http://example.com/> ;
   ```

3. Next we use the `DB.DBA.TTLP()` procedure to insert the R2RML into a temporary graph, <http://temp/product>:

   ```
   DB.DBA.TTLP
   ```

```
    (
      ' @prefix       rr:  <http://www.w3.org/ns/r2rml#>    .
        @prefix      exa:  <http://example.com/ns#>         .
        @prefix  product:  <http://example.com/product#>    .

        <http://example.com/ns#TriplesMap1>
            a                               rr:TriplesMap  ;

            rr:logicalTable
              [
                rr:tableSchema  "R2RML"    ;
                rr:tableOwner   "TEST"     ;
                rr:tableName    "PRODUCT"
              ];

            rr:subjectMap
              [
                rr:template  "http://example.com/product/{id}" ;
                rr:class     exa:product                       ;
                rr:graph     <http://example.com/>
              ];

            rr:predicateObjectMap
              [
                rr:predicate  product:id  ;
                rr:objectMap
                  [
                    rr:column  "id"
                  ];
              ];

            rr:predicateObjectMap
              [
                rr:predicate  product:name  ;
                rr:objectMap
                  [
                    rr:column "name"
                  ];
              ];
          .
      ',
      'http://temp/product',
      'http://temp/product'
    );
```

4. Next, there is a series of commented out lines that can be used for sanity checking:

```
--SELECT DB.DBA.R2RML_TEST ('http://temp/product');

--DB.DBA.OVL_VALIDATE ('http://temp/product', 'http://www.w3.org/ns/r2rml#OVL');

-- Running the validation in order to find error in name of R2RML description graph
--DB.DBA.OVL_VALIDATE ('http://temp/product-nosuch', 'http://www.w3.org/ns/r2rml#OVL');

-- Running the validation in order to find error in name of R2RML metadata graph
--DB.DBA.OVL_VALIDATE ('http://temp/product', 'http://www.w3.org/ns/r2rml#OVL-nosuch');

--SELECT DB.DBA.R2RML_EXECUTE ('http://temp/product');
```

5. Next, DB.DBA.R2RML_MAKE_QM_FROM_G() is used to perform the conversion from R2RML into Virtuoso's own Linked Data Views script. The output is then prepended with the keyword 'SPARQL' and a space, and executed using `exec()`:

```
EXEC ('SPARQL ' || DB.DBA.R2RML_MAKE_QM_FROM_G ('http://temp/product'));
```

Note : The final triples are placed in the graph defined in the R2RML script itself (<http://example.com/>)

Alternatively, the destination graph can be specified as an optional second parameter of DB.DBA.R2RML_MAKE_QM_FROM_G():

```
DB.DBA.R2RML_MAKE_QM_FROM_G
  (
    (
```

```
    IN g VARCHAR
  [, IN target_graph VARCHAR := NULL]
   )
)
```

6. Finally, a simple SPARQL statement is executed to prove data is returned:

```
SPARQL
  SELECT *
  FROM <http://example.com/>
  WHERE {?s ?p ?o .};
```

16.7.4. Known Limitations

rr:sqlQuery is not currently supported, due to limitations in the optimizer used for Virtuoso's native implementation of Linked Data Views.

16.7.5. Generating an R2RML based Linked Data View from iSQL command-line

Using Virtuoso you can programmatically generate Linked Data Views atop Relational Data Sources, using R2RML via the built-in function: *R2RML_GENERATE_LINKED_VIEW* function. In order to use this function, you need to have the rdb2rdf_dav.vad package installed.

```
R2RML_GENERATE_LINKED_VIEW
  (
    in source varchar,
    in destination_graph varchar,
    in graph_type int default 0,
    in clear_source_graph int default 1
  )
```

Here is detailed description of the funcion's parameter:

- *source*

  : The source R2RMLdocument URI. Acceptable schemes include: file:, dav:, http: and https:. These are also acceptable as source graph URI;
- *destination graph*

  : This is a default graph name (an IRI) applicable to either virtual or physical graph.
- *graph_type*

  : 0 - virtual; 1 - physical graph which sets the actual graph type;
- *clear_source_graph*

  : Determines if existing R2RML source graphs (those holding view declarations) are replaced as part of processing pipeline.

Note : The R2RML mapping script may have a triples like:

```
[] rr:graph <graph_name>
```

and in this case they take precedence and virtual graph would be defined as in the R2RML. If so, then if destination graph is specified as physical, all virtual graphs found in the R2RML would go in the destination_graph.

Usage Example

1. Ensure the R2RML VAD package rdb2rdf_dav.vad is installed.
2. To clear out existing mappings execute:

```
SQL> SELECT RDF_VIEW_DROP_STMT_BY_GRAPH ('http://example.com');
VARCHAR
_______________________________________________________________________________
```

```
SPARQL drop silent quad map <http://demo.openlinksw.com/r2rmldemo.n3> .;

1 Rows. -- 16 msec.

SQL> SPARQL DROP SILENT QUAD MAP <http://demo.openlinksw.com/r2rmldemo.n3> ;

STATE    MESSAGE
VARCHAR  VARCHAR
_______________________________________________________________________________

00000    Quad map <http://demo.openlinksw.com/r2rmldemo.n3> is no longer used in storage <http://w
>
00000    Quad map <http://demo.openlinksw.com/r2rmldemo.n3> is deleted
00000    Transaction committed, SPARQL compiler re-configured
00000    2 RDF metadata manipulation operations done

4 Rows. -- 406 msec.

SQL> SPARQL CLEAR <http://demo.openlinksw.com/r2rmldemo.n3>;

callret-0
VARCHAR
_______________________________________________________________________________

Clear <http://demo.openlinksw.com/r2rmldemo.n3> -- done

1 Rows. -- 15 msec.

SQL> DROP TABLE "R2RML"."TEST"."PRODUCT" ;

Done. -- 0 msec.

SQL> CREATE TABLE "R2RML"."TEST"."PRODUCT"
(
  "id" INTEGER,
  "name" VARCHAR(100),
  PRIMARY KEY ("id")
);
Done. -- 16 msec.
```

3. Insert sample data into a Table by executing:

```
SQL> INSERT SOFT "R2RML"."TEST"."PRODUCT" VALUES(1, 'Virtuoso');
Done. -- 0 msec.
```

4. Locate or create your R2RML mapping document, for example: .n3 file with the following content:

```
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix exa: <http://example.com/ns#> .
@prefix product: <http://example.com/product#> .

<http://example.com/ns#TriplesMap1>
    a rr:TriplesMap;

    rr:logicalTable
    [
      rr:tableSchema "R2RML";
      rr:tableOwner "TEST";
      rr:tableName  "PRODUCT"
    ];

    rr:subjectMap
    [
      rr:template "http://example.com/product/{id}";
      rr:class exa:product;
    ];

    rr:predicateObjectMap
    [
      rr:predicate product:id;
      rr:objectMap [ rr:column "id" ];
    ];

    rr:predicateObjectMap
    [
```

```
      rr:predicate product:name;
      rr:objectMap [ rr:column "name" ];
    ];
.
```

5. Generate a Linked Data View from the R2RML document that applies to the sample data (created earlier) by executing the statement:

```
SQL> DB.DBA.R2RML_GENERATE_LINKED_VIEW('http://demo.openlinksw.com/r2rmldemo.n3', 'http://example.
STATE    MESSAGE
VARCHAR  VARCHAR
_______________________________________________________________________________

00000    IRI class <r2rml:virt02-8513ca7e0ce41d2e38f0c750fd552139> has been defined (inherited fro
00000    Literal class <r2rml:virt02-daca9ceddea29d53dbbdb6bd0f3dee68> has been defined (inherited
00000    Quad storage <http://www.openlinksw.com/schemas/virtrdf#DefaultQuadStorage> is flagged as
00000    Quad map <http://demo.openlinksw.com/r2rmldemo.n3> has been created and added to the <htt
orage>
00000    Quad map <sys:qm-1be5dbd931459cf9e2df2338428f418d> has been created and added to the <htt
orage>
00000    Quad map <sys:qm-c5f81d7126efa3e7a93f7e903fd5fa93> has been created and added to the <htt
orage>
00000    Quad map <sys:qm-25c4599111b9f07fbd8fc60ce0b42eaf> has been created and added to the <htt
orage>
00000    Quad storage <http://www.openlinksw.com/schemas/virtrdf#DefaultQuadStorage> is unflagged
00000    Transaction committed, SPARQL compiler re-configured
00000    9 RDF metadata manipulation operations done

10 Rows. -- 1109 msec.
SQL>
```

6. Verify successful creation of the Linked Data View by executing the following SPARQL query via iSQL or Conductor interface:

```
SQL> SPARQL
SELECT *
FROM <http://example.com>
WHERE {?s ?p ?o} ;

s                              p                                                o
VARCHAR                        VARCHAR                                          VARCHAR
_______________________________________________________________________________

http://example.com/product/1   http://example.com/product#id                    1
http://example.com/product/1   http://example.com/product#name                  Virtuoso
http://example.com/product/1   http://www.w3.org/1999/02/22-rdf-syntax-ns#type  http://exampl

3 Rows. -- 15 msec.
```

16.7.6. Virtuoso Conductor R2RML Import Wizard

The Virtuoso Conductor can be used for importing existing R2RML scripts into Virtuoso and generate the necessary RDF Linked Data Views for Virtuoso hosting and deployment.

Usage Example

1. Ensure the R2RML rdb2rdf_dav.vad and latest Conductor conductor_dav.vad VAD packages are installed.
2. Create a test table with sample data:

```
SQL> CREATE TABLE "R2RML"."TEST"."PRODUCT"
(
  "id" INTEGER,
  "name" VARCHAR(100),
  PRIMARY KEY ("id")
);
Done. -- 16 msec.
SQL> INSERT SOFT "R2RML"."TEST"."PRODUCT" VALUES(1, 'Virtuoso');
Done. -- 0 msec.
SQL> INSERT SOFT "R2RML"."TEST"."PRODUCT" VALUES(2, 'UDA');
Done. -- 0 msec.
SQL>
```

3. Grant select privileges on the "R2RML"."TEST"."PRODUCT" table to the SPARQL user to enable execution via SPARQL endpoint:

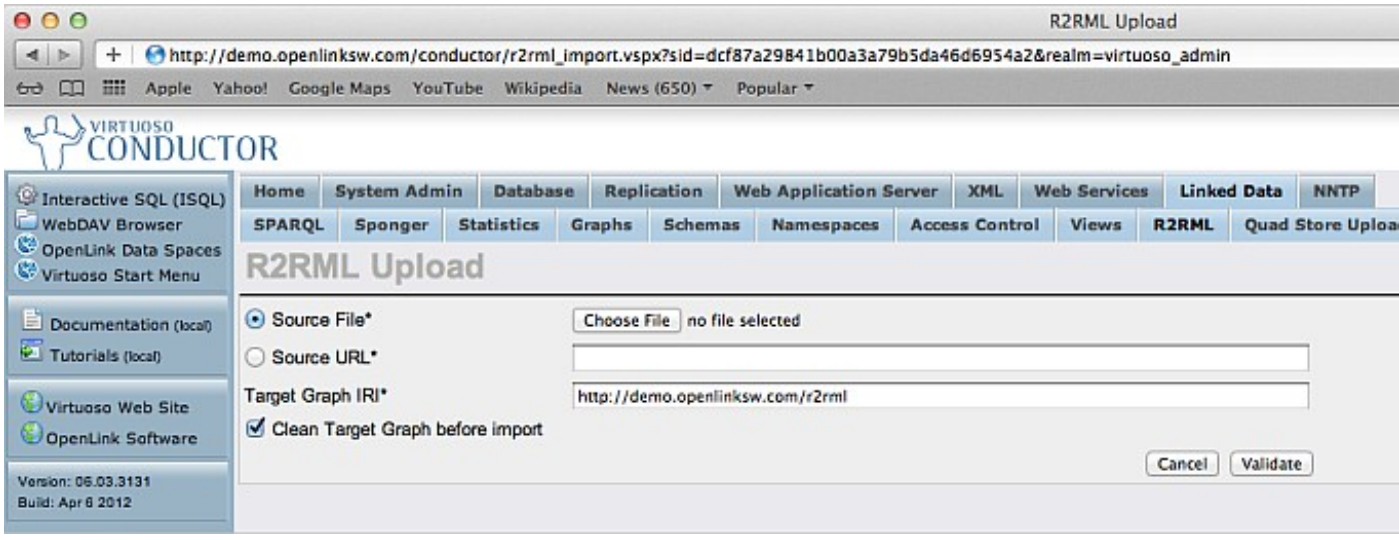
```
SQL> GRANT SELECT ON R2RML.TEST.PRODUCT  TO "SPARQL", "SPARQL_UPDATE"
Done. -- 1 msec.
```

4. Create the following R2RML mapping script for the "R2RML"."TEST"."PRODUCT" table:

```
$ cat demo.n3
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix exa: <http://example.com/ns#> .
@prefix product: <http://example.com/product#> .

<http://example.com/ns#TriplesMap1>
    a rr:TriplesMap;

    rr:logicalTable
    [
      rr:tableSchema "R2RML";
      rr:tableOwner "TEST";
      rr:tableName  "PRODUCT"
    ];

    rr:subjectMap
    [
      rr:template "http://example.com/product/{id}";
      rr:class exa:product;
    ];

    rr:predicateObjectMap
    [
      rr:predicate product:id;
      rr:objectMap [ rr:column "id" ];
    ];

    rr:predicateObjectMap
    [
      rr:predicate product:name;
      rr:objectMap [ rr:column "name" ];
    ];
.
$
```

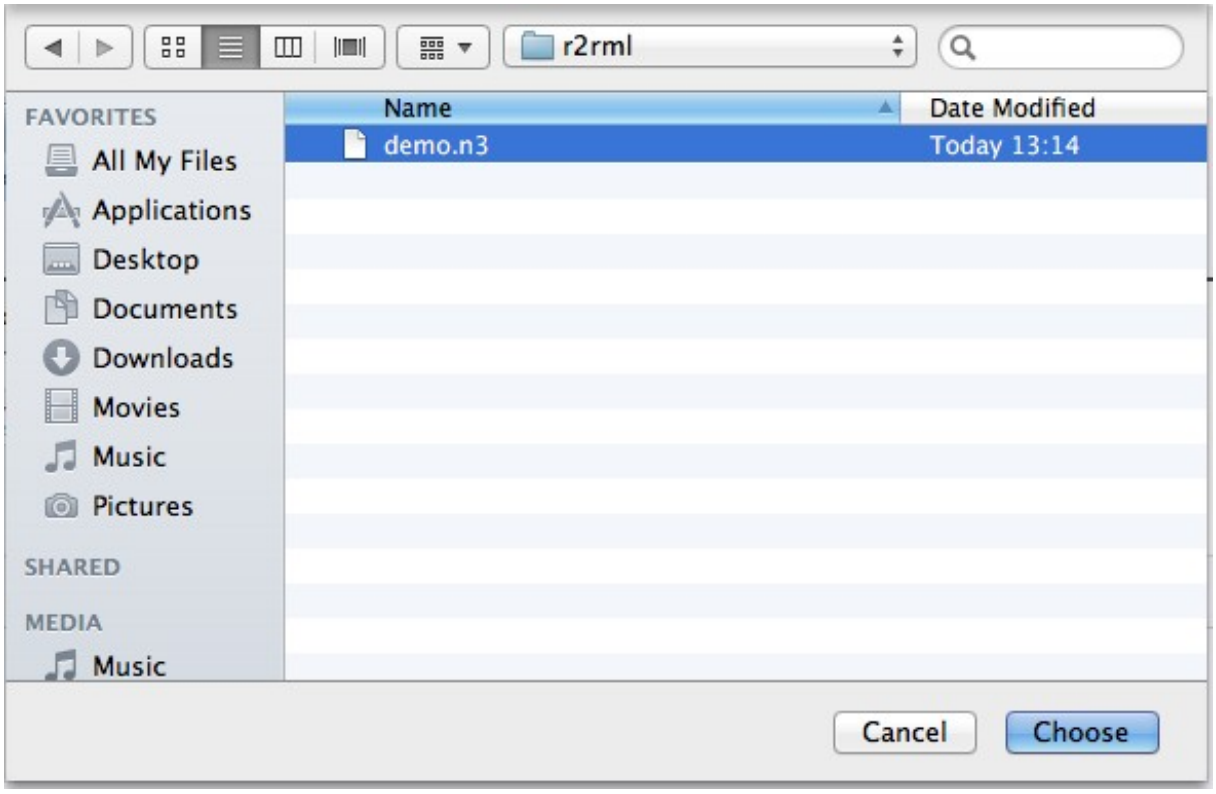
5. Got to the Linked Data -> R2RML tab of the Virtuoso Conductor:

Figure 16.66. Conductor R2RML Import Wizard



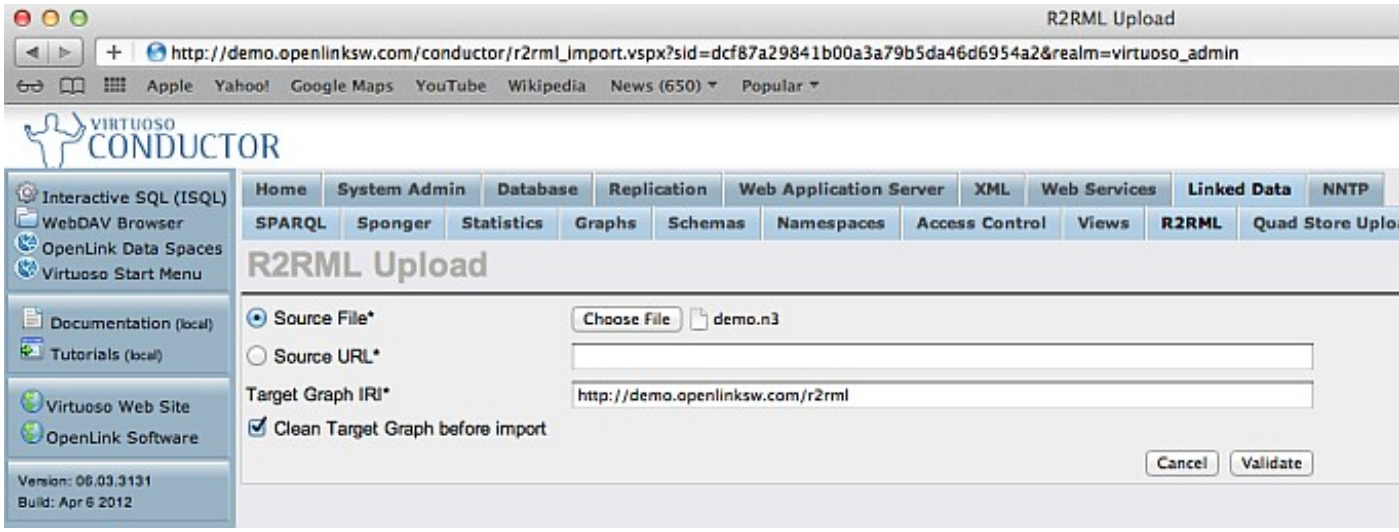
6. Select the Choose File button and select the R2RML file to load:

Figure 16.67. Conductor R2RML Import Wizard



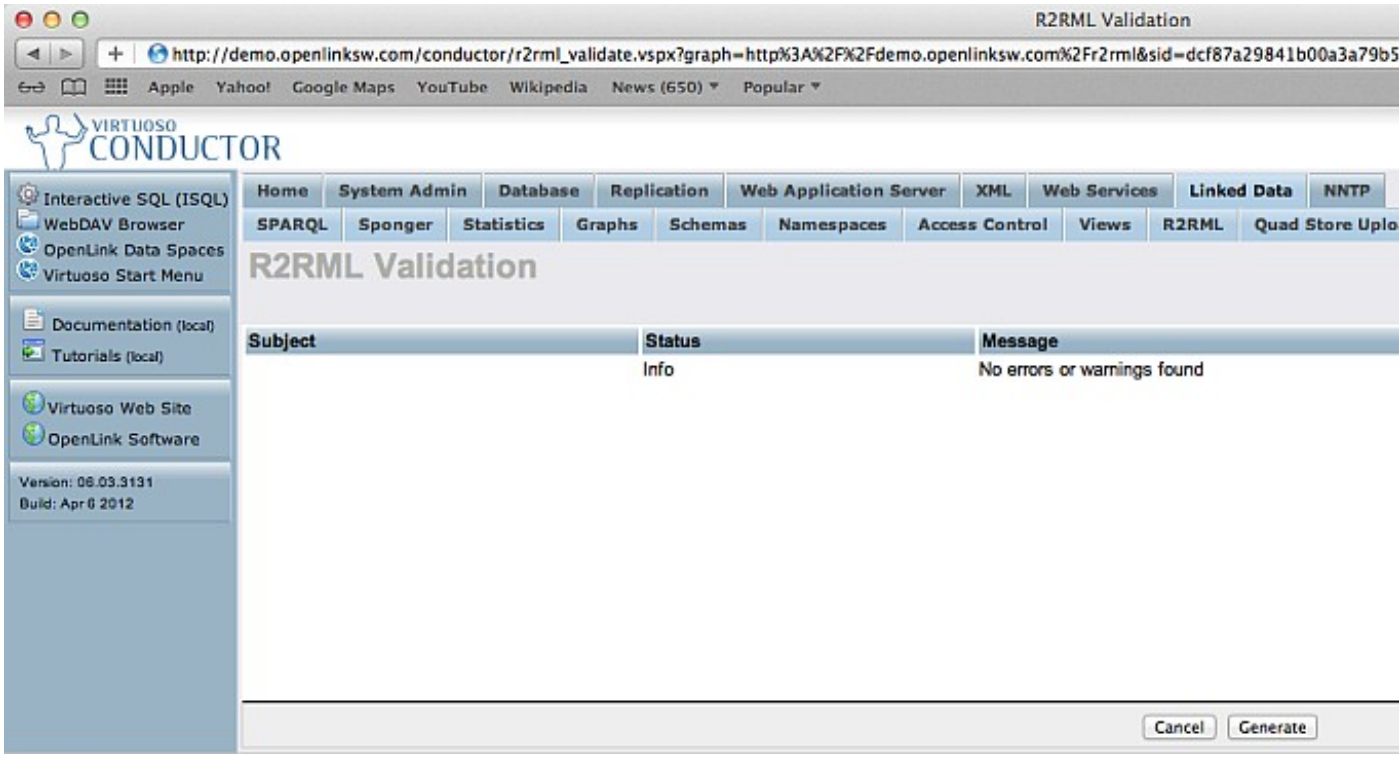
7. Select the Validate button to verify the R2RML mapping script:

Figure 16.68. Conductor R2RML Import Wizard



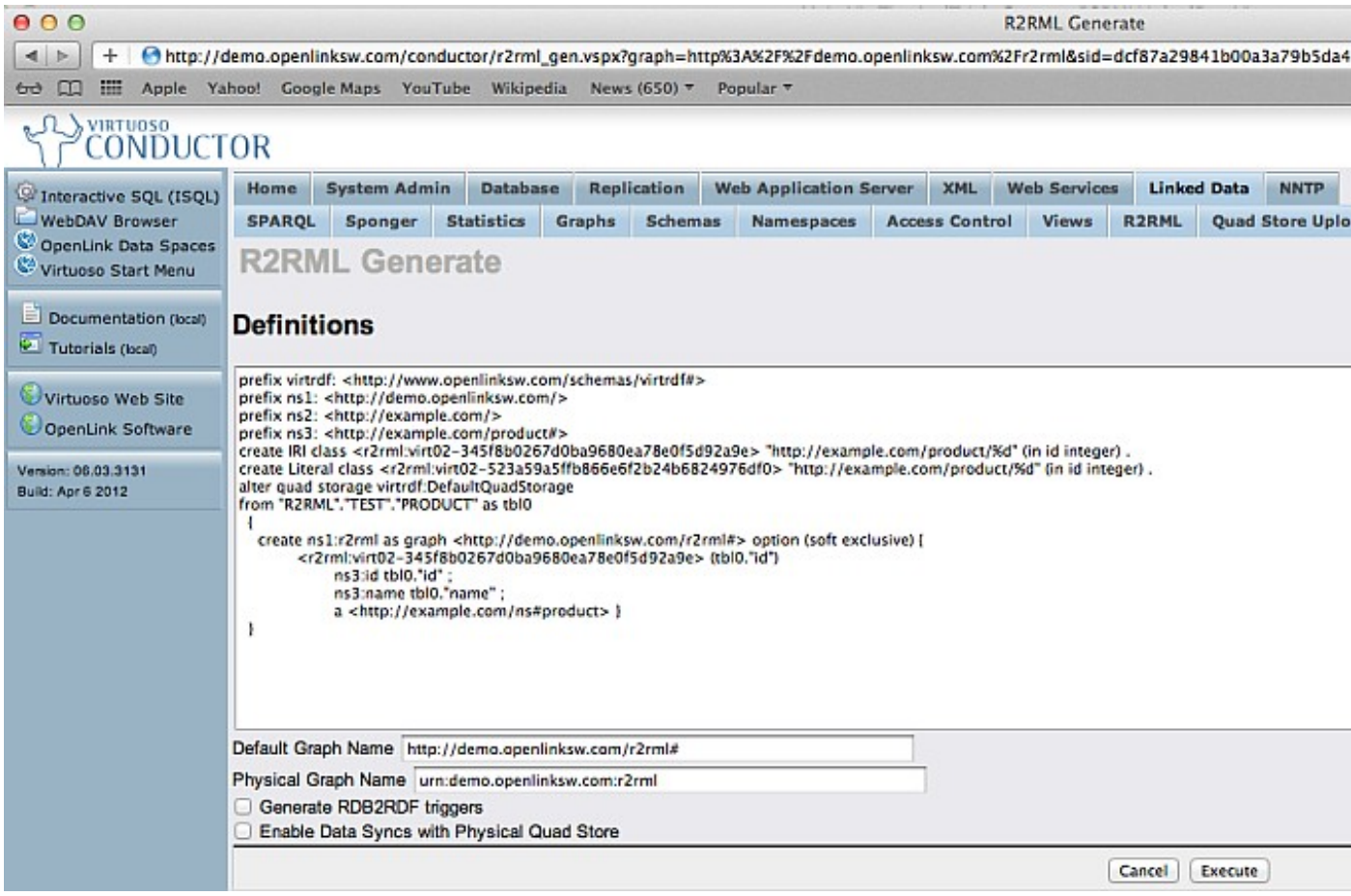
8. Select the Generate button to generate the RDF Linked Data Views mappings for the R2RML mapping script:

Figure 16.69. Conductor R2RML Import Wizard



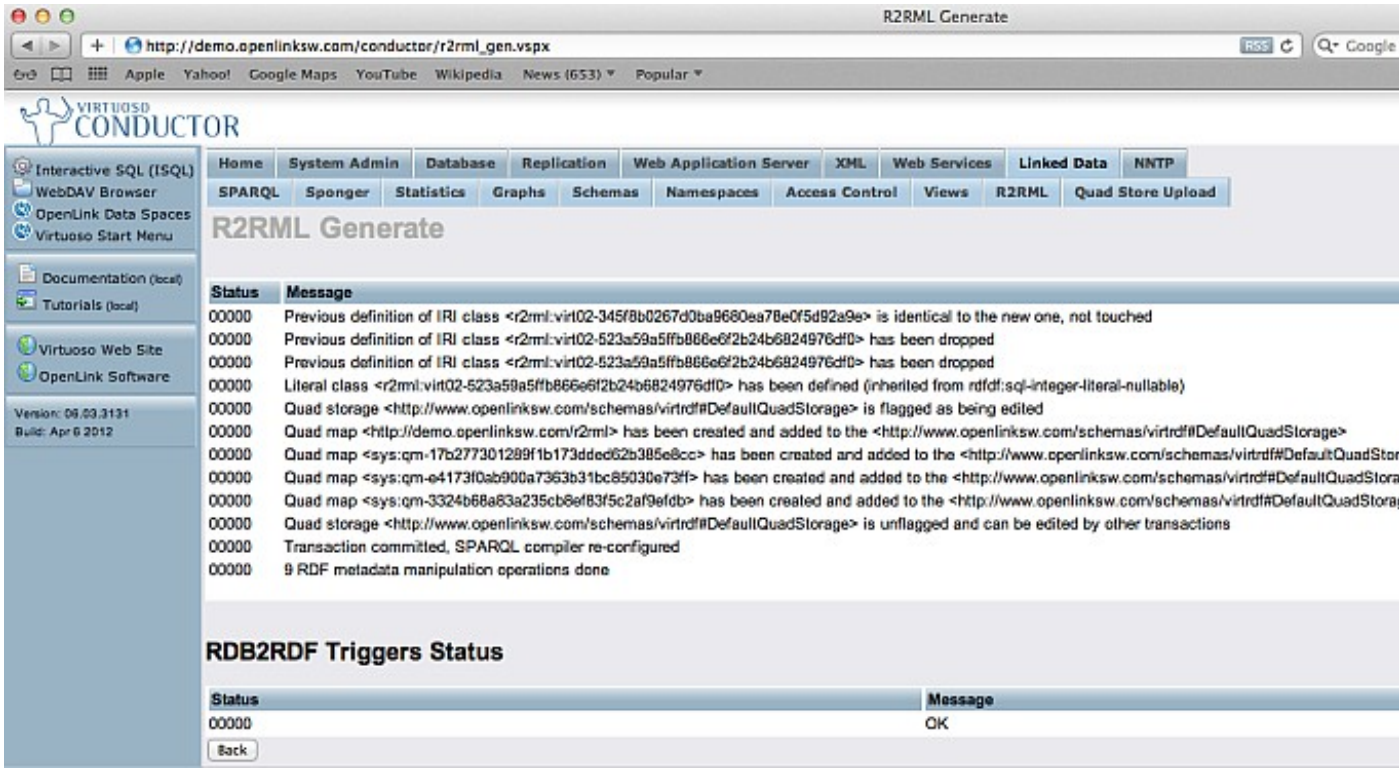
9. Select the Execute button to create the RDF Linked Data Views mapping the the Quad Store:

Figure 16.70. Conductor R2RML Import Wizard



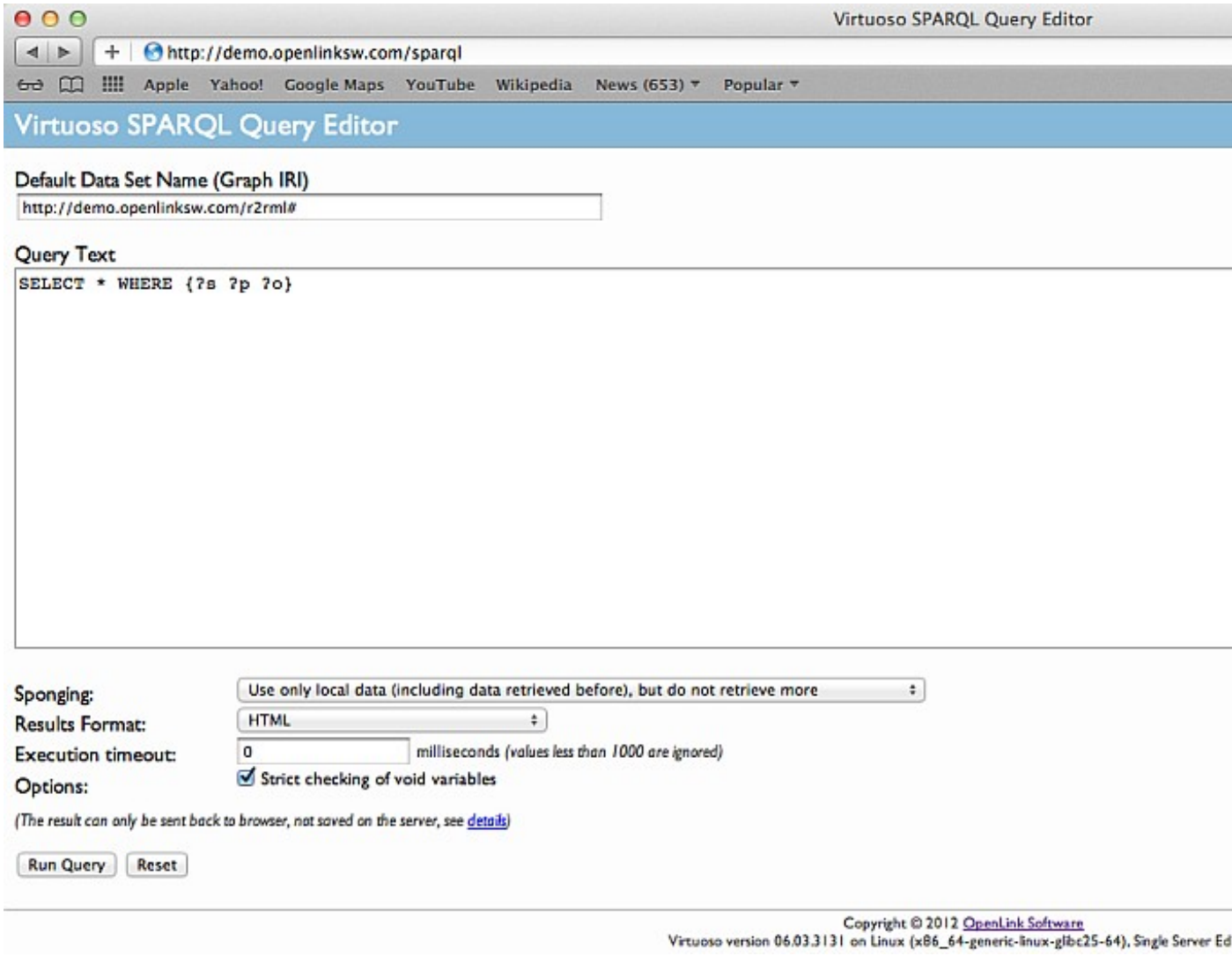
10. The RDF Linked Data View creation is complete and status is displayed:

Figure 16.71. Conductor R2RML Import Wizard



11. The Default Graph Name (transient) specified http://demo.openlinksw.com/r2rml# can now be used to run a SPARQL query against the created Linked Data View. If the Generate RDB2RDF triggers and Enable Data Syncs with Physical Quad Store check boxes are selected the Physical Graph Name (persistent) specified urn:demo.openlinksw.com/r2rml# can be used to run a SPARQL query against the materialized triples in the Quad Store.

Figure 16.72. Conductor R2RML Import Wizard



12. The results set for the Linked Data View graph are displayed:

Figure 16.73. Conductor R2RML Import Wizard

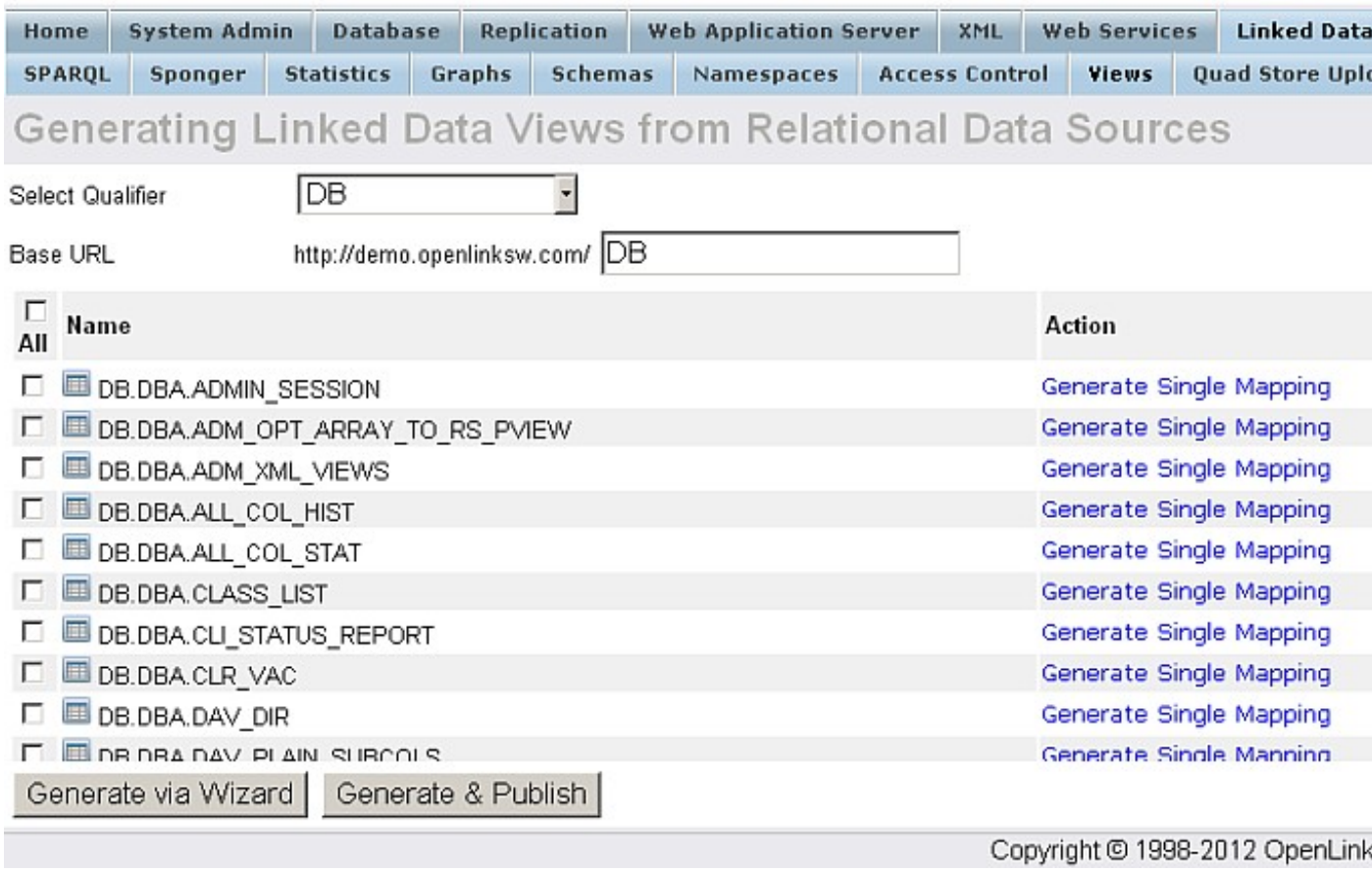
s	p	o
http://example.com/product/1	http://example.com/product#id	1
http://example.com/product/2	http://example.com/product#id	2
http://example.com/product/1	http://example.com/product#name	Virtuoso
http://example.com/product/2	http://example.com/product#name	UDA
http://example.com/product/1	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://example.com/ns#produ
http://example.com/product/2	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://example.com/ns#produ
http://example.com/product/1	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://example.com/ns#produ
http://example.com/product/1	http://example.com/product#id	1
http://example.com/product/1	http://example.com/product#name	Virtuoso
http://example.com/product/2	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://example.com/ns#produ
http://example.com/product/2	http://example.com/product#id	2
http://example.com/product/2	http://example.com/product#name	UDA

16.7.7. Generate Transient and/or Persistent Linked Data Views atop Remote Relational Data Sources Using Conductor

This section describes how you can generate R2RML Scripts from Linked Data Views, using the Virtuoso Conductor ODBC or JDBC accessible.

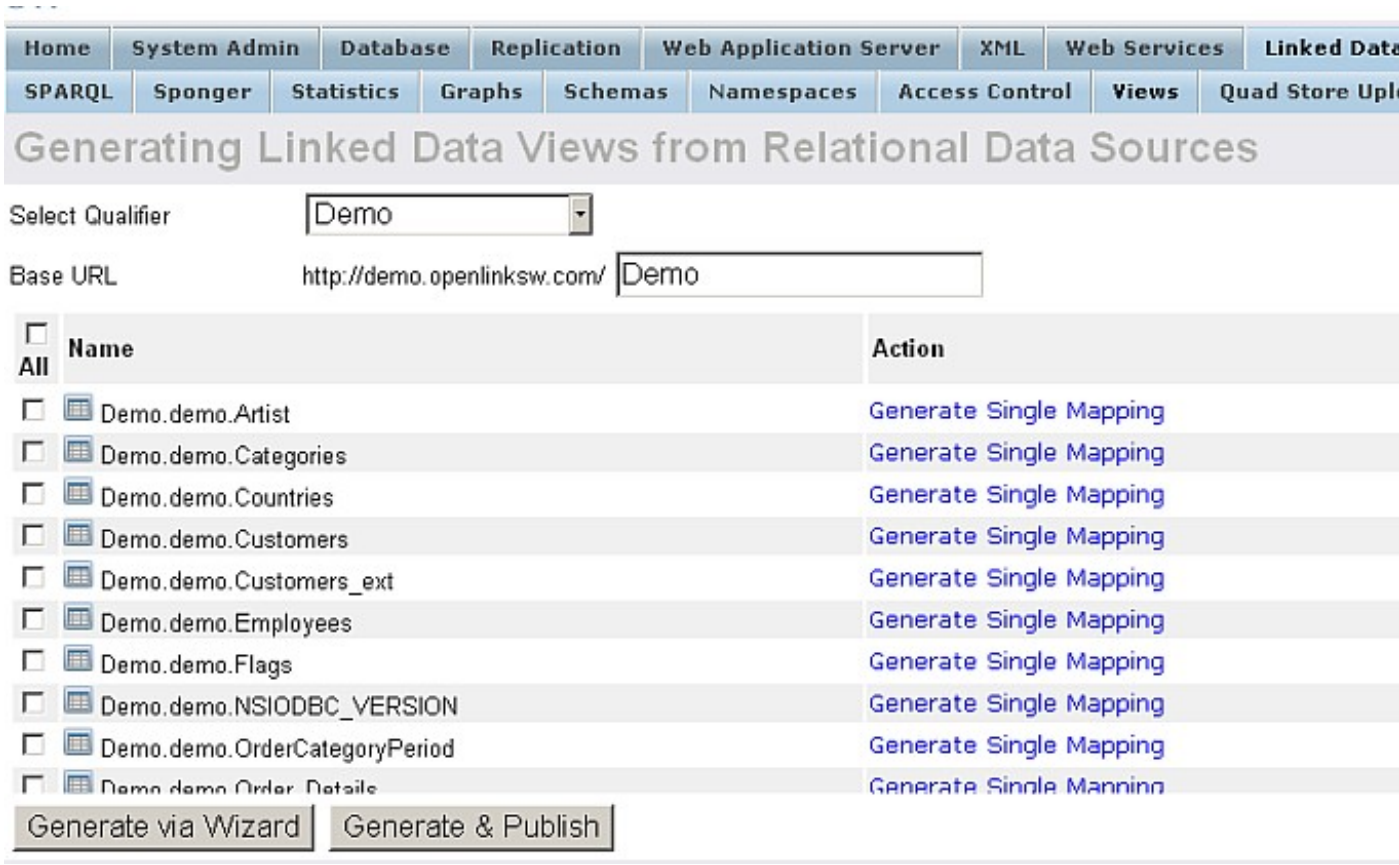
1. Ensure you have installed Conductor conductor_dav.vad VAD package with version 1.32.38 or higher.
2. Go to http://<cname>[:<port>]/conductor.
3. Enter dba credentials.
4. Go to Linked Data -> Views:

Figure 16.74. Generating Transient and/or Persistent Linked Data Views



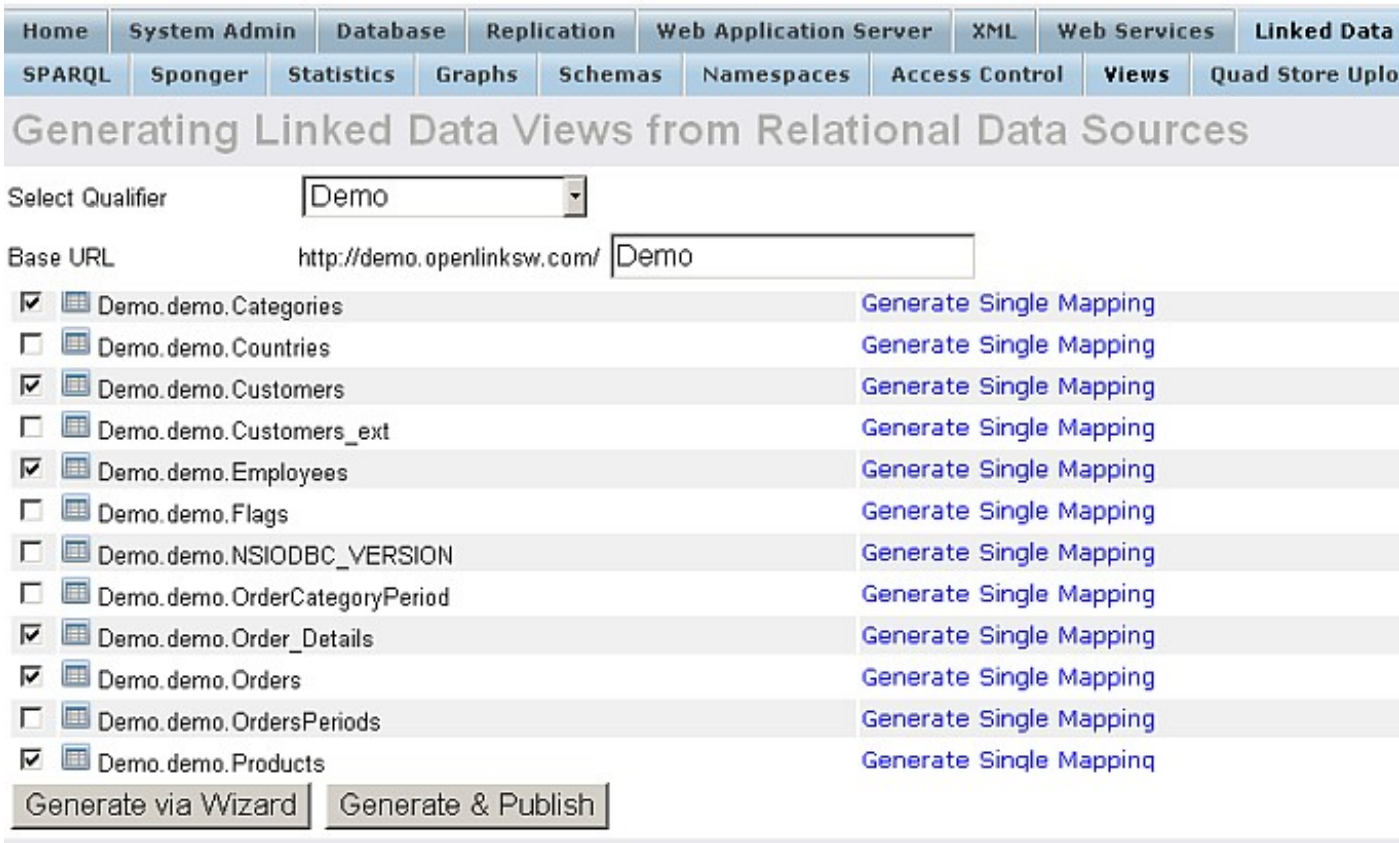
5. Select Qualifier Demo:

Figure 16.75. Generating Transient and/or Persistent Linked Data Views



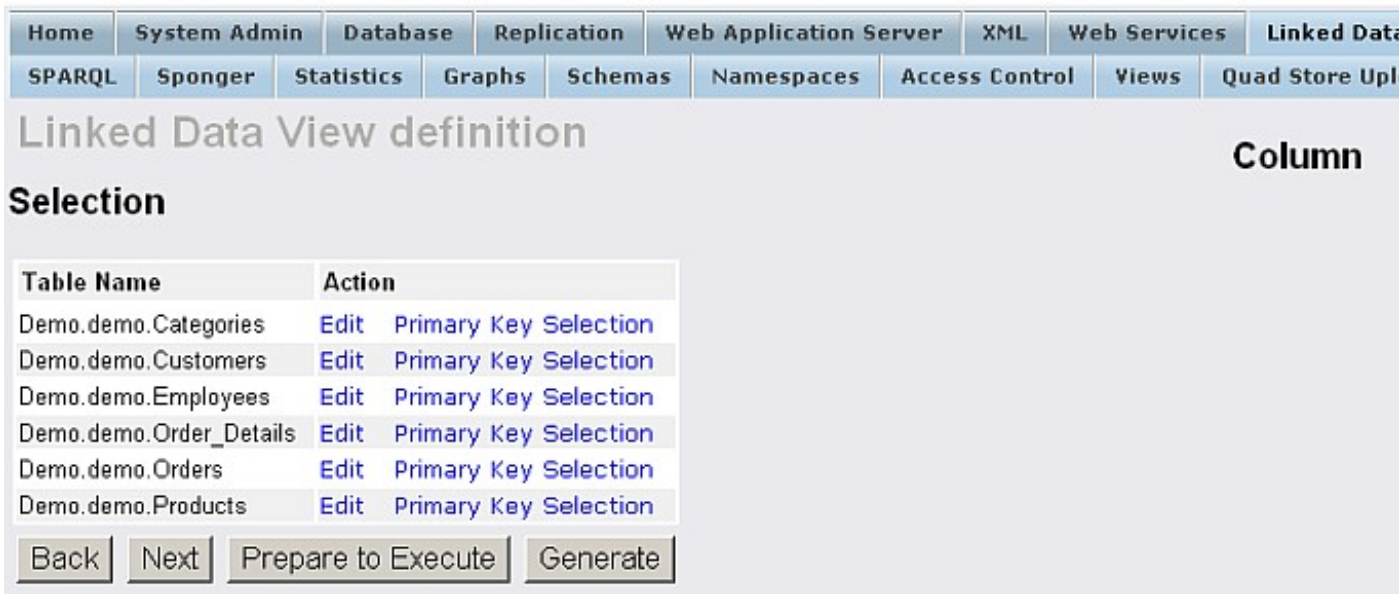
6. Select table(s) by hatching the check-box to the left of the table name; for example, select the following tables from the Northwind DB: Categories, Customers, Employees, Order_Details, Orders, Products .

Figure 16.76. Generating Transient and/or Persistent Linked Data Views



7. Click Generate via Wizard:

Figure 16.77. Generating Transient and/or Persistent Linked Data Views



8. Click Prepare to Execute.
9. The R2RML script for the selected table(s) will be generated and displayed in the R2RML Graph text-area:

Figure 16.78. Generating Transient and/or Persistent Linked Data Views

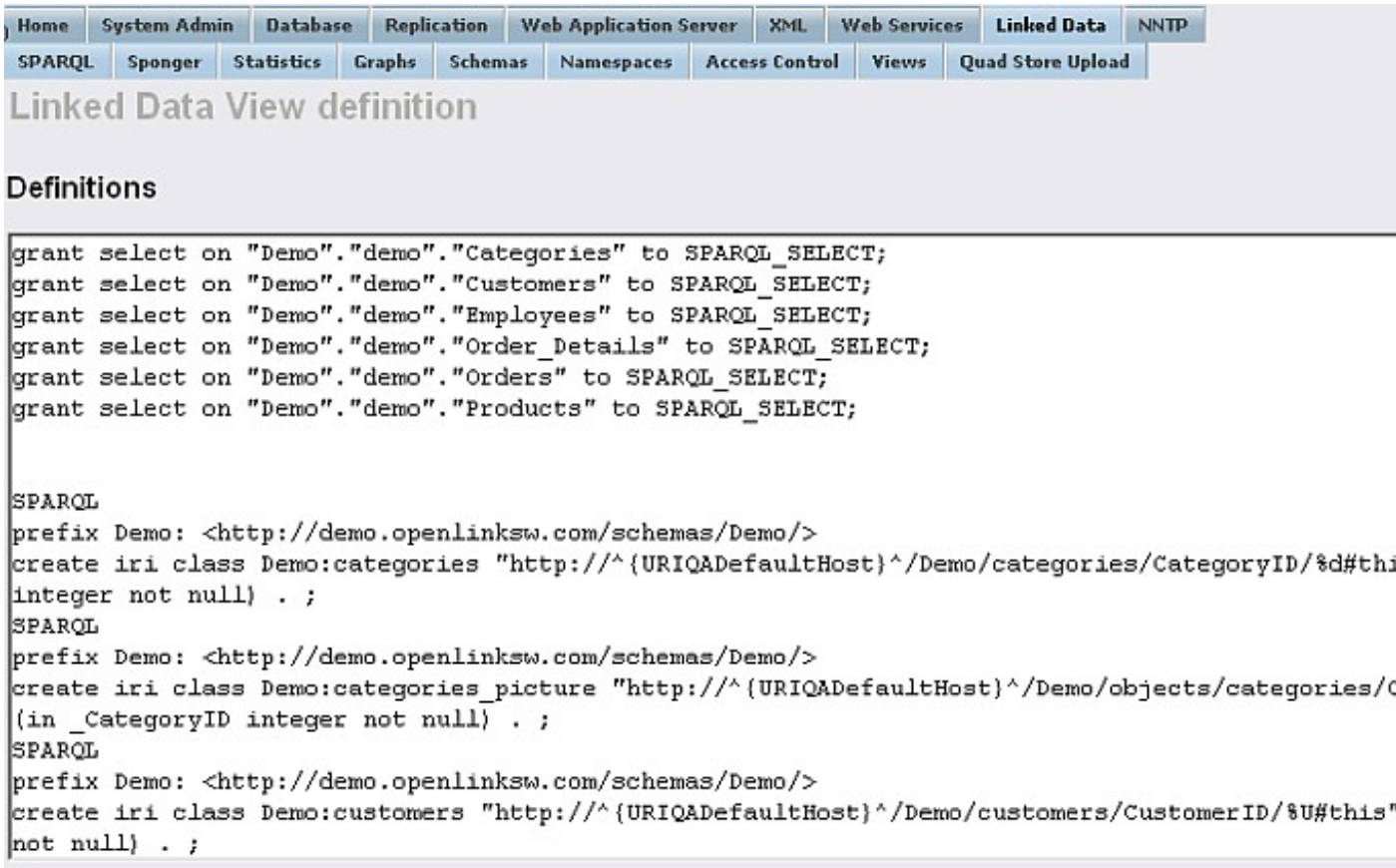


Figure 16.79. Generating Transient and/or Persistent Linked Data Views

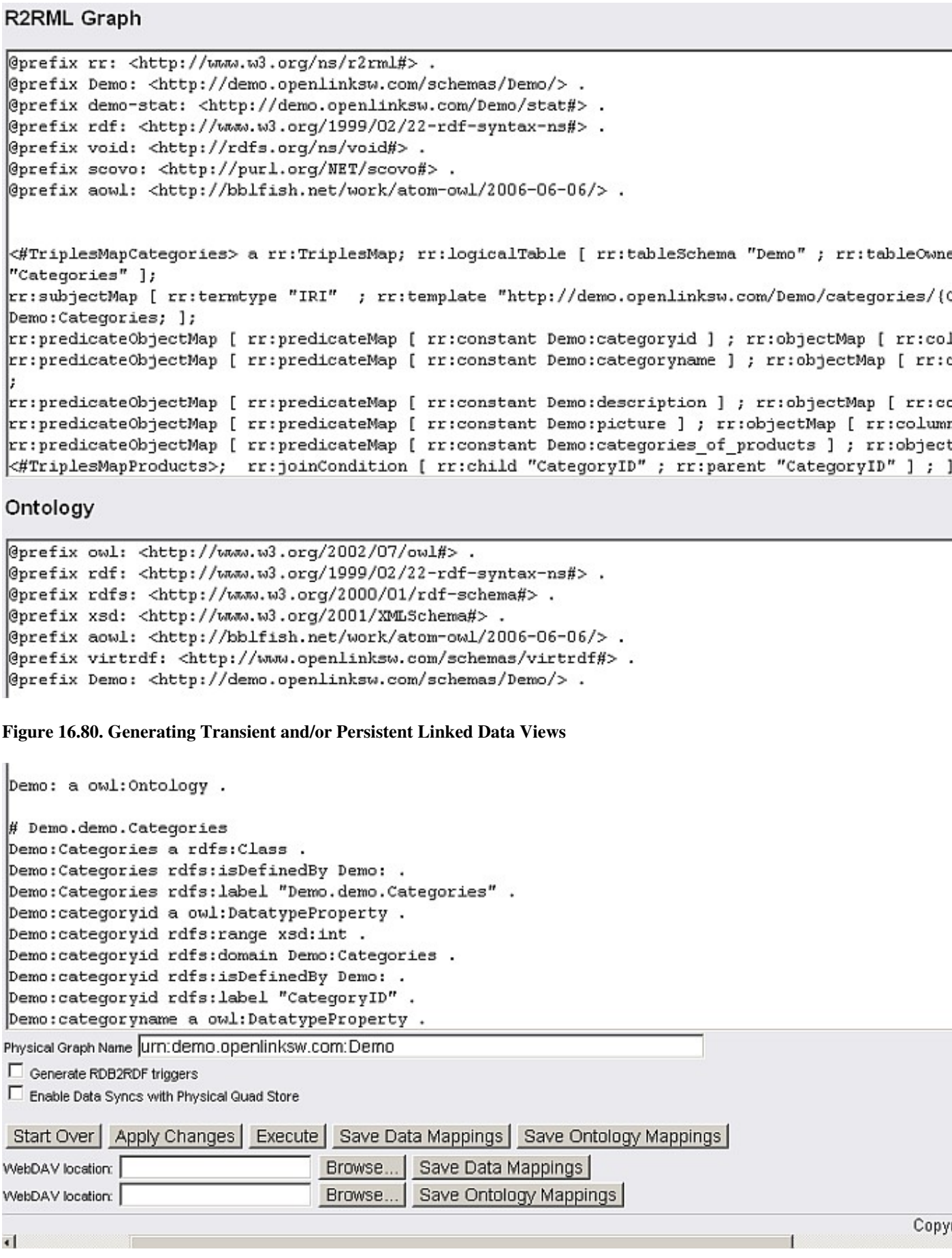


Figure 16.80. Generating Transient and/or Persistent Linked Data Views

10. As result the following R2RML script should be generated for the Northwind DB collection:

```
@prefix rr: <http://www.w3.org/ns/r2rml#> .
```

```
@prefix Demo: <http://demo.openlinksw.com/schemas/Demo/> .
@prefix demo-stat: <http://demo.openlinksw.com/Demo/stat#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix void: <http://rdfs.org/ns/void#> .
@prefix scovo: <http://purl.org/NET/scovo#> .
@prefix aowl: <http://bblfish.net/work/atom-owl/2006-06-06/> .

<#TriplesMapCategories> a rr:TriplesMap; rr:logicalTable [ rr:tableSchema "Demo" ; rr:tableOwner "
rr:subjectMap [ rr:termtype "IRI"  ; rr:template "http://demo.openlinksw.com/Demo/categories/{Cate
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:categoryid ] ; rr:objectMap [ rr:column
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:categoryname ] ; rr:objectMap [ rr:colu
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:description ] ; rr:objectMap [ rr:colum
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:picture ] ; rr:objectMap [ rr:column "P
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:categories_of_products ] ; rr:objectMap

<#TriplesMapCustomers> a rr:TriplesMap; rr:logicalTable [ rr:tableSchema "Demo" ; rr:tableOwner "d
rr:subjectMap [ rr:termtype "IRI"  ; rr:template "http://demo.openlinksw.com/Demo/customers/{Custo
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:customerid ] ; rr:objectMap [ rr:column
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:companyname ] ; rr:objectMap [ rr:colum
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:contactname ] ; rr:objectMap [ rr:colum
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:contacttitle ] ; rr:objectMap [ rr:colu
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:address ] ; rr:objectMap [ rr:column "A
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:city ] ; rr:objectMap [ rr:column "City
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:region ] ; rr:objectMap [ rr:column "Re
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:postalcode ] ; rr:objectMap [ rr:column
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:country ] ; rr:objectMap [ rr:column "C
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:countrycode ] ; rr:objectMap [ rr:colum
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:phone ] ; rr:objectMap [ rr:column "Pho
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:fax ] ; rr:objectMap [ rr:column "Fax"
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:customers_of_orders ] ; rr:objectMap [

<#TriplesMapEmployees> a rr:TriplesMap; rr:logicalTable [ rr:tableSchema "Demo" ; rr:tableOwner "d
rr:subjectMap [ rr:termtype "IRI"  ; rr:template "http://demo.openlinksw.com/Demo/employees/{Emplo
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:employeeid ] ; rr:objectMap [ rr:column
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:lastname ] ; rr:objectMap [ rr:column "
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:firstname ] ; rr:objectMap [ rr:column
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:title ] ; rr:objectMap [ rr:column "Tit
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:titleofcourtesy ] ; rr:objectMap [ rr:c
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:birthdate ] ; rr:objectMap [ rr:column
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:hiredate ] ; rr:objectMap [ rr:column "
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:address ] ; rr:objectMap [ rr:column "A
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:city ] ; rr:objectMap [ rr:column "City
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:region ] ; rr:objectMap [ rr:column "Re
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:postalcode ] ; rr:objectMap [ rr:column
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:country ] ; rr:objectMap [ rr:column "C
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:countrycode ] ; rr:objectMap [ rr:colum
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:homephone ] ; rr:objectMap [ rr:column
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:extension ] ; rr:objectMap [ rr:column
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:photo ] ; rr:objectMap [ rr:column "Pho
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:notes ] ; rr:objectMap [ rr:column "Not
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:reportsto ] ; rr:objectMap [ rr:column
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:employees_of_orders ] ; rr:objectMap [

<#TriplesMapOrder_Details> a rr:TriplesMap; rr:logicalTable [ rr:tableSchema "Demo" ; rr:tableOwne
rr:subjectMap [ rr:termtype "IRI"  ; rr:template "http://demo.openlinksw.com/Demo/order_details/{O
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:unitprice ] ; rr:objectMap [ rr:column
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:quantity ] ; rr:objectMap [ rr:column "
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:discount ] ; rr:objectMap [ rr:column "
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:order_details_has_orders ] ; rr:objectM
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:order_details_has_products ] ; rr:objec

<#TriplesMapOrders> a rr:TriplesMap; rr:logicalTable [ rr:tableSchema "Demo" ; rr:tableOwner "demo
rr:subjectMap [ rr:termtype "IRI"  ; rr:template "http://demo.openlinksw.com/Demo/orders/{OrderID}
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:orderid ] ; rr:objectMap [ rr:column "O
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:orderdate ] ; rr:objectMap [ rr:column
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:requireddate ] ; rr:objectMap [ rr:colu
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:shippeddate ] ; rr:objectMap [ rr:colum
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:freight ] ; rr:objectMap [ rr:column "F
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:shipname ] ; rr:objectMap [ rr:column "
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:shipaddress ] ; rr:objectMap [ rr:colum
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:shipcity ] ; rr:objectMap [ rr:column "
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:shipregion ] ; rr:objectMap [ rr:column
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:shippostalcode ] ; rr:objectMap [ rr:co
```

```
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:shipcountry ] ; rr:objectMap [ rr:colum
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:shipcountrycode ] ; rr:objectMap [ rr:c
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:rowguid ] ; rr:objectMap [ rr:column "R
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:orders_has_customers ] ; rr:objectMap [
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:orders_has_employees ] ; rr:objectMap [
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:orders_has_shippers ] ; rr:objectMap [
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:orders_of_order_details ] ; rr:objectMa

<#TriplesMapProducts> a rr:TriplesMap; rr:logicalTable [ rr:tableSchema "Demo" ; rr:tableOwner "de
rr:subjectMap [ rr:termtype "IRI"  ; rr:template "http://demo.openlinksw.com/Demo/products/{Produc
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:productid ] ; rr:objectMap [ rr:column
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:productname ] ; rr:objectMap [ rr:colum
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:quantityperunit ] ; rr:objectMap [ rr:c
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:unitprice ] ; rr:objectMap [ rr:column
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:unitsinstock ] ; rr:objectMap [ rr:colu
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:unitsonorder ] ; rr:objectMap [ rr:colu
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:reorderlevel ] ; rr:objectMap [ rr:colu
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:discontinued ] ; rr:objectMap [ rr:colu
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:products_has_categories ] ; rr:objectMa
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:products_has_suppliers ] ; rr:objectMap
rr:predicateObjectMap [ rr:predicateMap [ rr:constant Demo:products_of_order_details ] ; rr:object
```

16.8. Examples of Linked Data Views

16.8.1. Simple Mapping Example -- Northwind Linked Data View

Here is example of the basic Northwind Linked Data Views deployment. The sequence of operations is very common for adding SPARQL access to existing application.

There exist few important questions to answer. Who should have access to data behind Linked Data View? Should someone have access to other sorts of RDF data but not to the new View? What are applications that should be interoperable with the new RDF data source? Are there any applications that produce similar data but that data sould be kept apart from data made by view? How to ensure that deployment the view will not cause problems for other applications?

First of all, we decide whether the default web service endpoint should have access to the data in question. If it should then we have to grant SELECT privileges to the account "SPARQL" that is used for the default endpoint; if it should not but some custom edpoint should then grant to the owner account of that account. Granting access is less trivial that it is usual. On one hand, those who can make SQL SELECT statements on application's tables can also make SPARQL queries on Linked Data View over that tables, because it makes SQL inside. On the other hand, those who do not intend to query that data at all may get unexpected "permission denied" errors on queries that worked fine before adding an Linked Data View. If SPARQL compiler can not prove that the query can not access data from the view then it will generate SQL code that will access tables behind the view. In some cases permission problems should be resolved by creating Linked Data View in a separate RDF storage . In this example, data are public:

```
use DB;

GRANT SELECT ON "Demo"."demo"."Products" TO "SPARQL";
GRANT SELECT ON "Demo"."demo"."Suppliers" TO "SPARQL";
GRANT SELECT ON "Demo"."demo"."Shippers" TO "SPARQL";
GRANT SELECT ON "Demo"."demo"."Categories" TO "SPARQL";
GRANT SELECT ON "Demo"."demo"."Customers" TO "SPARQL";
GRANT SELECT ON "Demo"."demo"."Employees" TO "SPARQL";
GRANT SELECT ON "Demo"."demo"."Orders" TO "SPARQL";
GRANT SELECT ON "Demo"."demo"."Order_Details" TO "SPARQL";
GRANT SELECT ON "Demo"."demo"."Countries" TO "SPARQL";
GRANT SELECT ON "Demo"."demo"."Provinces" TO "SPARQL";
```

Interoperability is the next question. The example is not interoperable with anything so in can provide data of any form, a real application will probably use some ontology from external source. Sometimes data should be converted from internal application's representation to something different (such as metric to imperial or ATT country code to two-character country id); sometimes composed IRIs should follow special rules; function-based IRI classes may help in that cases. As this is the first example, only plain format-string-based IRI classes are used.

We should also ensure that data generated by the new view will not be accidentally mixed with other data of the database. For that purpose the example will use a unique graph name that includes both application name and host name. In addition, the script will drop declarations that might remain from a previous run of the same script. The script is executed many times during the

development so erasing previous version is worth writing. It will report an error if there's nothing to erase but it's better than unpredictable errors due to writing new declarations over existing ones.

Note

Making graph name unique for every host is not needed if the application is supposed to be "local" and nobody will access more than one installation of the application. If this is the case, use some fixed graph IRI, not necessarily starting with hostname at all; this is much more convenient for querying because you don't have to calculate the graph name in each query. With fixed graph in use, it is still possible to clone the Linked Data View to map to a unique graph as soon as the application become "public" and requires merging data from many installations.

```
SPARQL drop quad map graph iri("http://^{URIQADefaultHost}^/Northwind")
;
SPARQL drop quad map virtrdf:NorthwindDemo
;
```

The *{URIQADefaultHost}* macro is replaced with the value of *DefaultHost* parameter of [URIQA] section of configuration file. The IRI is written as *iri("http://^{URIQADefaultHost}^/Northwind")* , not as *<http://^{URIQADefaultHost}^/Northwind>* because macro of this sort works only inside SPARQL string values.

Now it's safe to create IRI classes needed for the view. If these classes are used only in the view we define then it is safe to create all of them in a single statement. If some of them are used across multiple declarations then errors may occur. it is impossible to redefine an IRI class that is in use; the compiler will try to avoid reporting errors by checking whether the new declaration is identical to the existing one and by trying garbage collection in hope that the IRI class is used only in garbage, but errors may occur anyway. Thus it is better to declare "shared" IRI classes by individual statements and group together only "private" IRI classes of a view. If a "class redefinition" error occurs in the middle of a group then "undefined class" errors may occur after because the processing of the group was interrupted before rest of group was not executed. When in trouble, try `DB.DBA.RDF_AUDIT_METADATA` procedure.

```
SPARQL
create iri class northwind:Category "http://^{URIQADefaultHost}^/Northwind/Category/%d#this" (in category
create iri class northwind:Shipper "http://^{URIQADefaultHost}^/Northwind/Shipper/%d#this" (in shipper_id
create iri class northwind:Supplier "http://^{URIQADefaultHost}^/Northwind/Supplier/%d#this" (in supplier
create iri class northwind:Product   "http://^{URIQADefaultHost}^/Northwind/Product/%d#this" (in product_
create iri class northwind:Customer "http://^{URIQADefaultHost}^/Northwind/Customer/%U#this" (in customer
create iri class northwind:Employee "http://^{URIQADefaultHost}^/Northwind/Employee/%U%U%d#this" (in empl
create iri class northwind:Order "http://^{URIQADefaultHost}^/Northwind/Order/%d#this" (in order_id integ
create iri class northwind:CustomerContact "http://^{URIQADefaultHost}^/Northwind/CustomerContact/%U#this
create iri class northwind:OrderLine "http://^{URIQADefaultHost}^/Northwind/OrderLine/%d/%d#this" (in ord
create iri class northwind:Province "http://^{URIQADefaultHost}^/Northwind/Province/%U/%U#this" (in count
create iri class northwind:Country "http://^{URIQADefaultHost}^/Northwind/Country/%U#this" (in country_na
create iri class northwind:Flag "http://^{URIQADefaultHost}^%U#this" (in flag_path varchar not null) .
create iri class northwind:dbpedia_iri "http://dbpedia.org/resource/%U" (in uname varchar not null) .
create iri class northwind:EmployeePhoto "http://^{URIQADefaultHost}^/DAV/VAD/demo/sql/EMP%d#this" (in em
create iri class northwind:CategoryPhoto "http://^{URIQADefaultHost}^/DAV/VAD/demo/sql/CAT%d#this" (in ca
;
```

One IRI class per subject type; format strings begin with same host but different directory names so this will let the compiler to guess the type of subject by the text of IRI. Most of declarations are bijections and may get *option (bijection)* hint but these format strings are so simple that the compiler may understand it by itself. (*northwind:Employee* is not a bijection because `sprintf_inverse` will be unable to split the tail of IRI string and find the boundary between first and last name.)

The final operation is extending the default quad storage with new tree of quad map patterns.

```
SPARQL
prefix northwind: <http://demo.openlinksw.com/schemas/northwind#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix sioc: <http://rdfs.org/sioc/ns#>
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix wgs: <http://www.w3.org/2003/01/geo/wgs84_pos#>

alter quad storage virtrdf:DefaultQuadStorage
from Demo.demo.Products as products
from Demo.demo.Suppliers as suppliers
from Demo.demo.Shippers as shippers
from Demo.demo.Categories as categories
from Demo.demo.Customers as customers
from Demo.demo.Employees as employees
```

```
from Demo.demo.Orders as orders
from Demo.demo.Order_Details as order_lines
from Demo.demo.Countries as countries
from Demo.demo.Provinces as provinces
where (^{suppliers.}^.Country = ^{countries.}^.Name)
where (^{customers.}^.Country = ^{countries.}^.Name)
where (^{employees.}^.Country = ^{countries.}^.Name)
where (^{orders.}^.ShipCountry = ^{countries.}^.Name)
{
        create virtrdf:NorthwindDemo as graph iri ("http://^{URIQADefaultHost}^/Northwind") option (exclu
        {
                northwind:CustomerContact (customers.CustomerID)
                        a foaf:Person
                                as virtrdf:CustomerContact-foaf_Person .

                northwind:CustomerContact (customers.CustomerID)
                        a northwind:CustomerContact
                                as virtrdf:CustomerContact-CustomerContact;
                        foaf:name customers.ContactName
                                as virtrdf:CustomerContact-contact_name ;
                        foaf:phone customers.Phone
                                as virtrdf:CustomerContact-foaf_phone ;
                        northwind:is_contact_at northwind:Customer (customers.CustomerID)
                                as virtrdf:CustomerContact-is_contact_at ;
                        northwind:country northwind:Country (customers.Country)
                                as virtrdf:CustomerContact-country ;
                        rdfs:isDefinedBy northwind:customercontact_iri (customers.CustomerID) ;
                        rdfs:isDefinedBy northwind:CustomerContact (customers.CustomerID) .

                northwind:Country (customers.Country)
                        northwind:is_country_of
                northwind:CustomerContact (customers.CustomerID) as virtrdf:CustomerContact-is_country_of

                northwind:Product (products.ProductID)
                        a northwind:Product
                                as virtrdf:Product-ProductID ;
                        northwind:has_category northwind:Category (products.CategoryID)
                                as virtrdf:Product-product_has_category ;
                        northwind:has_supplier northwind:Supplier (products.SupplierID)
                                as virtrdf:Product-product_has_supplier ;
                        northwind:productName products.ProductName
                                as virtrdf:Product-name_of_product ;
                        northwind:quantityPerUnit products.QuantityPerUnit
                                as virtrdf:Product-quantity_per_unit ;
                        northwind:unitPrice products.UnitPrice
                                as virtrdf:Product-unit_price ;
                        northwind:unitsInStock products.UnitsInStock
                                as virtrdf:Product-units_in_stock ;
                        northwind:unitsOnOrder products.UnitsOnOrder
                                as virtrdf:Product-units_on_order ;
                        northwind:reorderLevel products.ReorderLevel
                                as virtrdf:Product-reorder_level ;
                        northwind:discontinued products.Discontinued
                                as virtrdf:Product-discontinued ;
                        rdfs:isDefinedBy northwind:product_iri (products.ProductID) ;
                        rdfs:isDefinedBy northwind:Product (products.ProductID) .

                northwind:Category (products.CategoryID)
                        northwind:category_of northwind:Product (products.ProductID) as virtrdf:Product-c

                northwind:Supplier (products.SupplierID)
                        northwind:supplier_of northwind:Product (products.ProductID) as virtrdf:Product-s

                northwind:Supplier (suppliers.SupplierID)
                        a northwind:Supplier
                                as virtrdf:Supplier-SupplierID ;
                        northwind:companyName suppliers.CompanyName
                                as virtrdf:Supplier-company_name ;
                        northwind:contactName suppliers.ContactName
                                as virtrdf:Supplier-contact_name ;
                        northwind:contactTitle suppliers.ContactTitle
                                as virtrdf:Supplier-contact_title ;
                        northwind:address suppliers.Address
```

```
                as virtrdf:Supplier-address ;
        northwind:city suppliers.City
                as virtrdf:Supplier-city ;
        northwind:dbpedia_city northwind:dbpedia_iri(suppliers.City)
                as virtrdf:Supplier-dbpediacity ;
        northwind:region suppliers.Region
                as virtrdf:Supplier-region ;
        northwind:postalCode suppliers.PostalCode
                as virtrdf:Supplier-postal_code ;
        northwind:country northwind:Country(suppliers.Country)
                as virtrdf:Supplier-country ;
        northwind:phone suppliers.Phone
                as virtrdf:Supplier-phone ;
        northwind:fax suppliers.Fax
                as virtrdf:Supplier-fax ;
        northwind:homePage suppliers.HomePage
                as virtrdf:Supplier-home_page ;
        rdfs:isDefinedBy northwind:supplier_iri (suppliers.SupplierID) ;
        rdfs:isDefinedBy northwind:Supplier (suppliers.SupplierID) .

northwind:Country (suppliers.Country)
        northwind:is_country_of
northwind:Supplier (suppliers.SupplierID) as virtrdf:Supplier-is_country_of .

northwind:Category (categories.CategoryID)
        a northwind:Category
                as virtrdf:Category-CategoryID ;
        northwind:categoryName categories.CategoryName
                as virtrdf:Category-home_page ;
        northwind:description categories.Description
                as virtrdf:Category-description ;
        foaf:img northwind:CategoryPhoto(categories.CategoryID)
                as virtrdf:Category-categories.CategoryPhoto ;
        rdfs:isDefinedBy northwind:category_iri (categories.CategoryID) ;
        rdfs:isDefinedBy northwind:Category (categories.CategoryID) .

northwind:CategoryPhoto(categories.CategoryID)
        a northwind:CategoryPhoto
                as virtrdf:Category-categories.CategoryPhotoID ;
        rdfs:isDefinedBy northwind:categoryphoto_iri (categories.CategoryID) ;
        rdfs:isDefinedBy northwind:CategoryPhoto(categories.CategoryID) .

northwind:Shipper (shippers.ShipperID)
        a northwind:Shipper
                as virtrdf:Shipper-ShipperID ;
        northwind:companyName shippers.CompanyName
                as virtrdf:Shipper-company_name ;
        northwind:phone shippers.Phone
                as virtrdf:Shipper-phone ;
        rdfs:isDefinedBy northwind:shipper_iri (shippers.ShipperID) ;
        rdfs:isDefinedBy northwind:Shipper (shippers.ShipperID) .

northwind:Customer (customers.CustomerID)
        a  northwind:Customer
                as virtrdf:Customer-CustomerID2 ;
        a  foaf:Organization
                as virtrdf:Customer-CustomerID ;
        foaf:name customers.CompanyName
                as virtrdf:Customer-foaf_name ;
        northwind:companyName customers.CompanyName
                as virtrdf:Customer-company_name ;
        northwind:has_contact northwind:CustomerContact (customers.CustomerID)
                as virtrdf:Customer-contact ;
        northwind:country northwind:Country (customers.Country)
                as virtrdf:Customer-country ;
        northwind:contactName customers.ContactName
                as virtrdf:Customer-contact_name ;
        northwind:contactTitle customers.ContactTitle
                as virtrdf:Customer-contact_title ;
        northwind:address customers.Address
                as virtrdf:Customer-address ;
        northwind:city customers.City
                as virtrdf:Customer-city ;
```

```
        northwind:dbpedia_city northwind:dbpedia_iri(customers.City)
                as virtrdf:Customer-dbpediacity ;
        northwind:region customers.Region
                as virtrdf:Customer-region ;
        northwind:PostalCode customers.PostalCode
                as virtrdf:Customer-postal_code ;
        foaf:phone customers.Phone
                as virtrdf:Customer-foaf_phone ;
        northwind:phone customers.Phone
                as virtrdf:Customer-phone ;
        northwind:fax customers.Fax
                as virtrdf:Customer-fax ;
        rdfs:isDefinedBy northwind:customer_iri (customers.CustomerID) ;
        rdfs:isDefinedBy northwind:Customer (customers.CustomerID) .

northwind:Country (customers.Country)
        northwind:is_country_of
northwind:Customer (customers.CustomerID) as virtrdf:Customer-is_country_of .

northwind:Employee (employees.FirstName, employees.LastName, employees.EmployeeID)
        a northwind:Employee
                as virtrdf:Employee-EmployeeID2 ;
        a foaf:Person
                as virtrdf:Employee-EmployeeID ;
        foaf:surname employees.LastName
                as virtrdf:Employee-foaf_last_name ;
        northwind:lastName employees.LastName
                as virtrdf:Employee-last_name ;
        foaf:firstName employees.FirstName
                as virtrdf:Employee-foaf_first_name ;
        northwind:firstName employees.FirstName
                as virtrdf:Employee-first_name ;
        foaf:title employees.Title
                as virtrdf:Employee-title ;
        northwind:titleOfCourtesy employees.TitleOfCourtesy
                as virtrdf:Employee-title_of_courtesy ;
        foaf:birthday employees.BirthDate
                as virtrdf:Employee-foaf_birth_date ;
        northwind:birthday employees.BirthDate
                as virtrdf:Employee-birth_date ;
        northwind:hireDate employees.HireDate
                as virtrdf:Employee-hire_date ;
        northwind:address employees.Address
                as virtrdf:Employee-address ;
        northwind:city employees.City
                as virtrdf:Employee-city ;
        northwind:dbpedia_city northwind:dbpedia_iri(employees.City)
                as virtrdf:Employee-dbpediacity ;
        northwind:region employees.Region
                as virtrdf:Employee-region ;
        northwind:postalCode employees.PostalCode
                as virtrdf:Employee-postal_code ;
        northwind:country northwind:Country(employees.Country)
                as virtrdf:Employee-country ;
        foaf:phone employees.HomePhone
                as virtrdf:Employee-home_phone ;
        northwind:extension employees.Extension
                as virtrdf:Employee-extension ;
        northwind:notes employees.Notes
                as virtrdf:Employee-notes ;
        northwind:reportsTo northwind:Employee(employees.FirstName, employees.LastName, e
                as virtrdf:Employee-reports_to ;
        foaf:img northwind:EmployeePhoto(employees.EmployeeID)
                as virtrdf:Employee-employees.EmployeePhoto ;
        rdfs:isDefinedBy northwind:employee_iri (employees.EmployeeID) ;
        rdfs:isDefinedBy northwind:Employee (employees.FirstName, employees.LastName, emp

northwind:EmployeePhoto(employees.EmployeeID)
        a northwind:EmployeePhoto
                as virtrdf:Employee-employees.EmployeePhotoId ;
        rdfs:isDefinedBy northwind:employeephoto_iri (employees.EmployeeID) ;
        rdfs:isDefinedBy northwind:EmployeePhoto (employees.EmployeeID) .
```

```
northwind:Employee (employees.FirstName, employees.LastName, orders.EmployeeID)
        northwind:is_salesrep_of
northwind:Order (orders.OrderID) where (^{orders.}^.EmployeeID = ^{employees.}^.EmployeeI

northwind:Country (employees.Country)
        northwind:is_country_of
northwind:Employee (employees.FirstName, employees.LastName, employees.EmployeeID) as vir

northwind:Order (orders.OrderID)
        a northwind:Order
                as virtrdf:Order-Order ;
        northwind:has_customer northwind:Customer (orders.CustomerID)
                as virtrdf:Order-order_has_customer ;
        northwind:has_salesrep northwind:Employee (employees.FirstName, employees.LastNam
                as virtrdf:Customer-has_salesrep ;
        northwind:has_employee northwind:Employee (employees.FirstName, employees.LastNam
                as virtrdf:Order-order_has_employee ;
        northwind:orderDate orders.OrderDate
                as virtrdf:Order-order_date ;
        northwind:requiredDate orders.RequiredDate
                as virtrdf:Order-required_date ;
        northwind:shippedDate orders.ShippedDate
                as virtrdf:Order-shipped_date ;
        northwind:order_ship_via northwind:Shipper (orders.ShipVia)
                as virtrdf:Order-order_ship_via ;
        northwind:freight orders.Freight
                as virtrdf:Order-freight ;
        northwind:shipName orders.ShipName
                as virtrdf:Order-ship_name ;
        northwind:shipAddress orders.ShipAddress
                as virtrdf:Order-ship_address ;
        northwind:shipCity orders.ShipCity
                as virtrdf:Order-ship_city ;
        northwind:dbpedia_shipCity northwind:dbpedia_iri(orders.ShipCity)
                as virtrdf:Order-dbpediaship_city ;
        northwind:shipRegion orders.ShipRegion
                as virtrdf:Order-ship_region ;
        northwind:shipPostal_code orders.ShipPostalCode
                as virtrdf:Order-ship_postal_code ;
        northwind:shipCountry northwind:Country(orders.ShipCountry)
                as virtrdf:ship_country ;
        rdfs:isDefinedBy northwind:order_iri (orders.OrderID) ;
        rdfs:isDefinedBy northwind:Order (orders.OrderID) .

northwind:Country (orders.ShipCountry)
        northwind:is_ship_country_of
northwind:Order (orders.OrderID) as virtrdf:Order-is_country_of .

northwind:Customer (orders.CustomerID)
        northwind:has_order northwind:Order (orders.OrderID) as virtrdf:Order-has_order .

northwind:Shipper (orders.ShipVia)
        northwind:ship_order northwind:Order (orders.OrderID) as virtrdf:Order-ship_order

northwind:OrderLine (order_lines.OrderID, order_lines.ProductID)
        a northwind:OrderLine
                as virtrdf:OrderLine-OrderLines ;
        northwind:has_order_id northwind:Order (order_lines.OrderID)
                as virtrdf:order_lines_has_order_id ;
        northwind:has_product_id northwind:Product (order_lines.ProductID)
                as virtrdf:order_lines_has_product_id ;
        northwind:unitPrice order_lines.UnitPrice
                as virtrdf:OrderLine-unit_price ;
        northwind:quantity order_lines.Quantity
                as virtrdf:OrderLine-quantity ;
        northwind:discount order_lines.Discount
                as virtrdf:OrderLine-discount ;
        rdfs:isDefinedBy northwind:orderline_iri (order_lines.OrderID, order_lines.Produc
        rdfs:isDefinedBy northwind:OrderLine (order_lines.OrderID, order_lines.ProductID)

northwind:Order (orders.OrderID)
        northwind:is_order_of
northwind:OrderLine (order_lines.OrderID, order_lines.ProductID) where (^{orders.}^.Order
```

```
                northwind:Product (products.ProductID)
                        northwind:is_product_of
                northwind:OrderLine (order_lines.OrderID, order_lines.ProductID) where (^{products.}^.Pro

                northwind:Country (countries.Name)
                        a northwind:Country
                                as virtrdf:Country-Type2 ;
                        a wgs:SpatialThing
                                as virtrdf:Country-Type ;
                        owl:sameAs northwind:dbpedia_iri (countries.Name) ;
                        northwind:name countries.Name
                                as virtrdf:Country-Name ;
                        northwind:code countries.Code
                                as virtrdf:Country-Code ;
                        northwind:smallFlagDAVResourceName countries.SmallFlagDAVResourceName
                                as virtrdf:Country-SmallFlagDAVResourceName ;
                        northwind:largeFlagDAVResourceName countries.LargeFlagDAVResourceName
                                as virtrdf:Country-LargeFlagDAVResourceName ;
                        northwind:smallFlagDAVResourceURI northwind:Flag(countries.SmallFlagDAVResourceUR
                                as virtrdf:Country-SmallFlagDAVResourceURI ;
                        northwind:largeFlagDAVResourceURI northwind:Flag(countries.LargeFlagDAVResourceUR
                                as virtrdf:Country-LargeFlagDAVResourceURI ;
                        wgs:lat countries.Lat
                                as virtrdf:Country-Lat ;
                        wgs:long countries.Lng
                                as virtrdf:Country-Lng ;
                        rdfs:isDefinedBy northwind:country_iri (countries.Name) ;
                        rdfs:isDefinedBy northwind:Country (countries.Name) .

                northwind:Country (countries.Name)
                        northwind:has_province
                northwind:Province (provinces.CountryCode, provinces.Province) where (^{provinces.}^.Coun

                northwind:Province (provinces.CountryCode, provinces.Province)
                        a northwind:Province
                                as virtrdf:Province-Provinces ;
                        northwind:has_country_code provinces.CountryCode
                                as virtrdf:has_country_code ;
                        northwind:provinceName provinces.Province
                                as virtrdf:Province-ProvinceName ;
                        rdfs:isDefinedBy northwind:province_iri (provinces.CountryCode, provinces.Provinc
                        rdfs:isDefinedBy northwind:Province (provinces.CountryCode, provinces.Province) .

                northwind:Province (provinces.CountryCode, provinces.Province)
                        northwind:is_province_of
                northwind:Country (countries.Name) where  (^{countries.}^.Code = ^{provinces.}^.CountryCo
        }.
}.
;
```

The created Linked Data View is sufficient for querying relational data via SPARQL but not for accessing data by dereferencing IRIs of subjects. Making IRIs dereferenceable requires configuring HTTP server; that is explained in second part of the example .

16.8.2. BSBM to RDF

This section includes example mapping SQL Data to RDF for BSBM.

Each BSBM installation includes a demonstration database that includes a simple management system schema and instance data. This sample database provides an easy route for introducing BSBM users and developers to the virtues of RDF based Linked Data Views built atop existing BSBM relational databases.

```
use DB;

CREATE TABLE DB.DBA.ProductFeature (
  nr integer primary key,
  label varchar(100) not null,
  comment varchar(1500) not null,
  publisher integer not null,
  publishDate date not null
)
```

```
;

grant SELECT on DB.DBA.ProductFeature to public
;

CREATE TABLE DB.DBA.ProductType (
  nr integer primary key,
  label varchar(100) not null,
  comment varchar(1500) not null,
  parent integer,
  publisher integer not null,
  publishDate date not null
)
;

grant SELECT on DB.DBA.ProductType to public
;

CREATE TABLE DB.DBA.Producer (
  nr integer primary key,
  label varchar(100) not null,
  comment varchar(1500) not null,
  homepage varchar(100) not null,
  country char(2) not null,
  publisher integer not null,
  publishDate date not null
)
;

grant SELECT on DB.DBA.Producer to public
;
create index producer_homepage on DB.DBA.Producer (homepage)
;

CREATE TABLE DB.DBA.Product (
  nr integer primary key,
  label varchar(100) not null,
  comment varchar not null,
  producer integer not null,
  propertyNum1 integer,
  propertyNum2 integer,
  propertyNum3 integer,
  propertyNum4 integer,
  propertyNum5 integer,
  propertyNum6 integer,
  propertyTex1 varchar(200),
  propertyTex2 varchar(200),
  propertyTex3 varchar(200),
  propertyTex4 varchar(200),
  propertyTex5 varchar(200),
  propertyTex6 varchar(200),
  publisher integer not null,
  publishDate date not null
)
;

grant SELECT on DB.DBA.Product to public
;

create index product_lbl on DB.DBA.Product (label)
;
create unique index product_producer_nr on DB.DBA.Product (producer, nr)
;
create index product_pn1 on DB.DBA.Product (propertyNum1)
;
create index product_pn2 on DB.DBA.Product (propertyNum2)
;
create index product_pn3 on DB.DBA.Product (propertyNum3)
;

create text index on DB.DBA.Product (label) with key nr
;
```

```
CREATE TABLE DB.DBA.ProductTypeProduct (
  product integer not null,
  productType integer not null,
  PRIMARY KEY (product, productType)
)
;

grant SELECT on DB.DBA.ProductTypeProduct to public
;

create index ptype_inv on DB.DBA.ProductTypeProduct (productType, product)
;

CREATE TABLE DB.DBA.ProductFeatureProduct (
  product integer not null,
  productFeature integer not null,
  PRIMARY KEY (product, productFeature)
)
;

grant SELECT on DB.DBA.ProductFeatureProduct to public
;

create index pfeature_inv on DB.DBA.ProductFeatureProduct (productFeature, product)
;

CREATE TABLE DB.DBA.Vendor (
  nr integer primary key,
  label varchar(100) not null,
  comment varchar not null,
  homepage varchar(100) not null,
  country char(2) not null,
  publisher integer not null,
  publishDate date not null
)
;

grant SELECT on DB.DBA.Vendor to public
;

create index vendor_country on DB.DBA.Vendor (country)
;
create index vendor_homepage on DB.DBA.Vendor (homepage)
;

CREATE TABLE DB.DBA.Offer (
  nr integer primary key,
  product integer not null,
  producer integer,
  vendor integer not null,
  price double precision not null,
  validFrom date not null,
  validTo date not null,
  deliveryDays integer not null,
  offerWebpage varchar(100) not null,
  publisher integer not null,
  publishDate date not null
)
;

grant SELECT on DB.DBA.Offer to public
;

create index offer_product on DB.DBA.Offer (product, deliveryDays)
;
create unique index offer_producer_product on DB.DBA.Offer (producer, product, nr)
;
create index offer_validto on DB.DBA.Offer (validTo)
;
create index offer_vendor_product on DB.DBA.Offer (vendor, product)
;
create index offer_webpage on DB.DBA.Offer (offerWebpage)
;
```

```
CREATE TABLE DB.DBA.Person (
  nr integer primary key,
  name varchar(30) not null,
  mbox_sha1sum char(40) not null,
  country char(2) not null,
  publisher integer not null,
  publishDate date not null
)
;

grant SELECT on DB.DBA.Person to public
;

CREATE TABLE DB.DBA.Review (
  nr integer primary key,
  product integer not null,
  producer integer,
  person integer not null,
  reviewDate date not null,
  title varchar(200) not null,
  text long varchar not null,
  textlang char(2) not null,
  rating1 integer,
  rating2 integer,
  rating3 integer,
  rating4 integer,
  publisher integer not null,
  publishDate date not null
)
;

grant SELECT on DB.DBA.Review to public
;

create unique index review_product on DB.DBA.Review (product, producer, nr)
;

create unique index review_producer_product on DB.DBA.Review (producer, product, nr)
;

create bitmap index review_textlang on DB.DBA.Review (textlang)
;

DB.DBA.XML_SET_NS_DECL ('foaf', 'http://xmlns.com/foaf/0.1/', 2)
;
DB.DBA.XML_SET_NS_DECL ('dc', 'http://purl.org/dc/elements/1.1/', 2)
;
DB.DBA.XML_SET_NS_DECL ('xsd', 'http://www.w3.org/2001/XMLSchema-datatypes/', 2)
;
DB.DBA.XML_SET_NS_DECL ('rev', 'http://purl.org/stuff/rev#', 2)
;
DB.DBA.XML_SET_NS_DECL ('bsbm', 'http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/', 2)
;
DB.DBA.XML_SET_NS_DECL ('bsbm-inst', 'http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/', 2)
;

SPARQL drop quad map bsbm:SingleGraphView
;

SPARQL create iri class bsbm:ProductFeature-iri "http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances
;

SPARQL create iri class bsbm:ProductType-iri "http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/Pr
;

SPARQL create iri class bsbm:Producer-iri "http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataF
;

SPARQL create iri class bsbm:Product-iri "http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFr
;

SPARQL create iri class bsbm:Vendor-iri "http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFro
```

```
;

SPARQL create iri class bsbm:Offer-iri "http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFrom
;

SPARQL create iri class bsbm:StdInst-iri "http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/Standa
;

SPARQL create iri class bsbm:Person-iri "http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFro
;

SPARQL create iri class bsbm:Review-iri "http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFro
;

SPARQL create iri class bsbm:ISO3166-country-iri "http://downlode.org/rdf/iso-3166/countries#%s" (in code
;

SPARQL create iri class bsbm:homepage-iri "%s" (in homepage varchar not null) option (returns "http://%s"
;

SPARQL create iri class bsbm:RatingSite-iri "http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dat
;

SPARQL
alter quad storage virtrdf:DefaultQuadStorage
FROM DB.DBA.ProductFeature as pfeature
FROM DB.DBA.ProductType as ptype
FROM DB.DBA.Producer as producer
FROM DB.DBA.Product as product text literal product.label
FROM DB.DBA.ProductTypeProduct as ptypeproduct
FROM DB.DBA.ProductFeatureProduct as pfeatureproduct
FROM DB.DBA.Vendor as vendor
FROM DB.DBA.Offer as offer
FROM DB.DBA.Person as person
FROM DB.DBA.Review as review
where (^{product.}^.nr = ^{ptypeproduct.}^.product)
where (^{product.}^.nr = ^{pfeatureproduct.}^.product)
  {
    create bsbm:SingleGraphView as graph <BSBM> option (exclusive)
      {
        bsbm:Product-iri (product.producer, product.nr)
          a bsbm:Product ;
          rdfs:label product.label ;
          rdfs:comment product.comment ;
          bsbm:producer bsbm:Producer-iri (product.producer, product.producer) ;
          bsbm:productPropertyTextual1 product.propertyTex1 ;
          bsbm:productPropertyTextual2 product.propertyTex2 ;
          bsbm:productPropertyTextual3 product.propertyTex3 ;
          bsbm:productPropertyTextual4 product.propertyTex4 ;
          bsbm:productPropertyTextual5 product.propertyTex5 ;
          bsbm:productPropertyTextual6 product.propertyTex6 ;
          bsbm:productPropertyNumeric1 product.propertyNum1 ;
          bsbm:productPropertyNumeric2 product.propertyNum2 ;
          bsbm:productPropertyNumeric3 product.propertyNum3 ;
          bsbm:productPropertyNumeric4 product.propertyNum4 ;
          bsbm:productPropertyNumeric5 product.propertyNum5 ;
          bsbm:productPropertyNumeric6 product.propertyNum6 ;
          rdf:type bsbm:ProductType-iri (ptypeproduct.productType) ;
          bsbm:productFeature bsbm:ProductFeature-iri (pfeatureproduct.productFeature) ;
          dc:publisher bsbm:Producer-iri (product.publisher, product.publisher) ;
          dc:date product.publishDate .

        bsbm:ProductType-iri (ptype.nr)
          a bsbm:ProductType ;
          rdfs:label ptype.label ;
          rdfs:comment ptype.comment ;
          rdfs:subClassOf bsbm:ProductType-iri (ptype.parent) ;
          dc:publisher bsbm:StdInst-iri (ptype.publisher) ;
          dc:date ptype.publishDate .

        bsbm:ProductFeature-iri (pfeature.nr)
          a bsbm:ProductFeature ;
          rdfs:label pfeature.label ;
```

```
        rdfs:comment pfeature.comment ;
        dc:publisher bsbm:StdInst-iri (pfeature.publisher) ;
        dc:date pfeature.publishDate .

      bsbm:Producer-iri (producer.nr, producer.nr)
        a bsbm:Producer ;
        rdfs:label producer.label ;
        rdfs:comment producer.comment ;
        foaf:homepage bsbm:homepage-iri (producer.homepage) ;
        bsbm:country bsbm:ISO3166-country-iri (producer.country) ;
        dc:publisher bsbm:Producer-iri (producer.nr, producer.nr) ;
        dc:date producer.publishDate .

      bsbm:Vendor-iri (vendor.nr, vendor.nr)
        a bsbm:Vendor ;
        rdfs:label vendor.label ;
        rdfs:comment vendor.comment ;
        foaf:homepage bsbm:homepage-iri (vendor.homepage) ;
        bsbm:country bsbm:ISO3166-country-iri (vendor.country) ;
        dc:publisher bsbm:Vendor-iri (vendor.publisher, vendor.publisher) ;
        dc:date vendor.publishDate .

      bsbm:Offer-iri (offer.vendor, offer.nr)
        a bsbm:Offer ;
        bsbm:product bsbm:Product-iri (offer.producer, offer.product) ;
        bsbm:vendor bsbm:Vendor-iri (offer.vendor, offer.vendor) ;
        bsbm:vendor bsbm:Vendor-iri (offer.vendor, offer.vendor) ;
        bsbm:price offer.price ;
        bsbm:validFrom offer.validFrom ;
        bsbm:validTo offer.validTo ;
        bsbm:deliveryDays offer.deliveryDays ;
        bsbm:offerWebpage bsbm:homepage-iri (offer.offerWebpage) ;
        dc:publisher bsbm:Vendor-iri (offer.publisher, offer.publisher) ;
        dc:date offer.publishDate .

      bsbm:Person-iri (person.publisher, person.nr)
        a foaf:Person ;
        foaf:name person.name ;
        foaf:mbox_sha1sum person.mbox_sha1sum ;
        bsbm:country bsbm:ISO3166-country-iri (person.country) ;
        dc:publisher bsbm:RatingSite-iri (person.publisher, person.publisher) ;
        dc:date person.publishDate .

      bsbm:Review-iri (review.publisher, review.nr)
        a rev:Review ;
        bsbm:reviewFor bsbm:Product-iri (review.producer, review.product) ;
        bsbm:producer bsbm:Producer-iri (review.producer, review.producer) ;
        rev:reviewer bsbm:Person-iri (review.publisher, review.person) ;
        bsbm:reviewDate review.reviewDate ;
        dc:title review.title ;
        rev:text review.text lang review.textlang ;
        bsbm:rating1 review.rating1 ;
        bsbm:rating2 review.rating2 ;
        bsbm:rating3 review.rating3 ;
        bsbm:rating4 review.rating4 ;
        dc:publisher bsbm:RatingSite-iri (review.publisher, review.publisher) ;
        dc:date review.publishDate .
    }
  }
;
```

16.8.3. TPCH to RDF

```
use DB;

GRANT SELECT ON TPCH.DBA.PARTSUPP  TO "SPARQL";
GRANT SELECT ON TPCH.DBA.SUPPLIER  TO "SPARQL";
GRANT SELECT ON TPCH.DBA.CUSTOMER  TO "SPARQL";
GRANT SELECT ON TPCH.DBA.HISTORY   TO "SPARQL";
GRANT SELECT ON TPCH.DBA.PART      TO "SPARQL";
GRANT SELECT ON TPCH.DBA.LINEITEM  TO "SPARQL";
GRANT SELECT ON TPCH.DBA.ORDERS    TO "SPARQL";
GRANT SELECT ON TPCH.DBA.NATION    TO "SPARQL";
```

```
GRANT SELECT ON TPCH.DBA.REGION    TO "SPARQL";

SPARQL
drop quad map virtrdf:TPCH
;

SPARQL
prefix tpch: <http://www.openlinksw.com/schemas/tpch#>
prefix sioc: <http://rdfs.org/sioc/ns#>
prefix foaf: <http://xmlns.com/foaf/0.1/>
create iri class tpch:customer "http://^{URIQADefaultHost}^/tpch/customer/%U%d#this" (in custname varchar
create iri class tpch:lineitem "http://^{URIQADefaultHost}^/tpch/lineitem/%d/%d#this" (in l_orderkey inte
create iri class tpch:nation "http://^{URIQADefaultHost}^/tpch/nation/%U%d#this" (in name varchar, in l_n
create iri class tpch:order "http://^{URIQADefaultHost}^/tpch/order/%d#this" (in o_orderkey integer not n
create iri class tpch:part "http://^{URIQADefaultHost}^/tpch/part/%U%d#this" (in p_partname varchar, in p
create iri class tpch:partsupp "http://^{URIQADefaultHost}^/tpch/partsupp/%d/%d#this" (in ps_partkey inte
create iri class tpch:region "http://^{URIQADefaultHost}^/tpch/region/%U%d#this" (in name varchar, in r_r
create iri class tpch:supplier "http://^{URIQADefaultHost}^/tpch/supplier/%U%d#this" (in name varchar, in
;

SPARQL
prefix tpch: <http://www.openlinksw.com/schemas/tpch#>
prefix sioc: <http://rdfs.org/sioc/ns#>
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix wgs: <http://www.w3.org/2003/01/geo/wgs84_pos#>
alter quad storage virtrdf:DefaultQuadStorage
FROM TPCH.DBA.LINEITEM as lineitems
FROM TPCH.DBA.CUSTOMER as customers
FROM TPCH.DBA.NATION as nations
FROM TPCH.DBA.ORDERS as orders
FROM TPCH.DBA.PART as parts
FROM TPCH.DBA.PARTSUPP as partsupps
FROM TPCH.DBA.REGION as regions
FROM TPCH.DBA.SUPPLIER as suppliers
where (^{suppliers.}^.S_NATIONKEY = ^{nations.}^.N_NATIONKEY)
where (^{customers.}^.C_NATIONKEY = ^{nations.}^.N_NATIONKEY)
{
    create virtrdf:TPCH as graph iri ("http://^{URIQADefaultHost}^/tpch") option (exclusive)
    {
# Customers
        tpch:customer (customers.C_NAME, customers.C_CUSTKEY)
            a  tpch:customer
                as virtrdf:customer-tpch-type ;
            a  foaf:Organization
                as virtrdf:customer-foaf-type ;
            tpch:custkey customers.C_CUSTKEY
                as virtrdf:customer-c_custkey ;
            foaf:name customers.C_NAME
                as virtrdf:customer-foaf_name ;
            tpch:companyName customers.C_NAME
                as virtrdf:customer-c_name ;
            tpch:has_nation tpch:nation (nations.N_NAME, customers.C_NATIONKEY)
                as virtrdf:customer-c_nationkey ;
            tpch:address customers.C_ADDRESS
                as virtrdf:customer-c_address ;
            foaf:phone customers.C_PHONE
                as virtrdf:customer-foaf_phone ;
            tpch:phone customers.C_PHONE
                as virtrdf:customer-phone ;
            tpch:acctbal customers.C_ACCTBAL
                as virtrdf:customer-acctbal ;
            tpch:mktsegment customers.C_MKTSEGMENT
                as virtrdf:customer-c_mktsegment ;
            tpch:comment customers.C_COMMENT
                as virtrdf:customer-c_comment .

# Nations
        tpch:nation (nations.N_NAME, customers.C_NATIONKEY)
            tpch:nation_of tpch:customer (customers.C_NAME, customers.C_CUSTKEY)
            as virtrdf:customer-nation_of .

        tpch:lineitem (lineitems.L_ORDERKEY, lineitems.L_LINENUMBER)
            a tpch:lineitem
```

```
                as virtrdf:lineitem-lineitems ;
            tpch:has_order tpch:order (lineitems.L_ORDERKEY)
                as virtrdf:lineitem-l_orderkey ;
            tpch:has_part tpch:part (parts.P_NAME, lineitems.L_PARTKEY)
                where (^{parts.}^.P_PARTKEY = ^{lineitems.}^.L_PARTKEY)
                as virtrdf:lineitem-l_partkey ;
            tpch:has_supplier tpch:supplier (suppliers.S_NAME, lineitems.L_SUPPKEY)
                where (^{suppliers.}^.S_SUPPKEY = ^{lineitems.}^.L_SUPPKEY)
                as virtrdf:lineitem-l_suppkey ;
            tpch:linenumber lineitems.L_LINENUMBER
                as virtrdf:lineitem-l_linenumber ;
            tpch:linequantity lineitems.L_QUANTITY
                as virtrdf:lineitem-l_linequantity ;
            tpch:lineextendedprice lineitems.L_EXTENDEDPRICE
                as virtrdf:lineitem-l_lineextendedprice ;
            tpch:linediscount lineitems.L_DISCOUNT
                as virtrdf:lineitem-l_linediscount ;
            tpch:linetax lineitems.L_TAX
                as virtrdf:lineitem-l_linetax ;
            tpch:returnflag lineitems.L_RETURNFLAG
                as virtrdf:lineitem-l_returnflag ;
            tpch:linestatus lineitems.L_LINESTATUS
                as virtrdf:lineitem-l_linestatus ;
            tpch:shipdate lineitems.L_SHIPDATE
                as virtrdf:lineitem-l_shipdate ;
            tpch:commitdate lineitems.L_COMMITDATE
                as virtrdf:lineitem-l_commitdate ;
            tpch:receiptdate lineitems.L_RECEIPTDATE
                as virtrdf:lineitem-l_receiptdate ;
            tpch:shipinstruct lineitems.L_SHIPINSTRUCT
                as virtrdf:lineitem-l_shipinstruct ;
            tpch:shipmode lineitems.L_SHIPMODE
                as virtrdf:lineitem-l_shipmode ;
            tpch:comment lineitems.L_COMMENT
                as virtrdf:lineitem-l_comment .

        tpch:part (parts.P_NAME, lineitems.L_PARTKEY)
            tpch:part_of tpch:lineitem (lineitems.L_ORDERKEY, lineitems.L_LINENUMBER)
            where (^{parts.}^.P_PARTKEY = ^{lineitems.}^.L_PARTKEY)
            as virtrdf:lineitem-part_of .

        tpch:order (lineitems.L_ORDERKEY)
            tpch:order_of tpch:lineitem (lineitems.L_ORDERKEY, lineitems.L_LINENUMBER) as virtrdf:lineite

        tpch:supplier (suppliers.S_NAME, lineitems.L_SUPPKEY)
            tpch:supplier_of tpch:lineitem (lineitems.L_ORDERKEY, lineitems.L_LINENUMBER)
            where (^{suppliers.}^.S_SUPPKEY = ^{lineitems.}^.L_SUPPKEY)
            as virtrdf:lineitem-supplier_of .

# Nation
        tpch:nation (nations.N_NAME, nations.N_NATIONKEY)
            a tpch:nation
                as virtrdf:nation-nations ;
            tpch:name nations.N_NAME
                as virtrdf:nation-n_name ;
            tpch:has_region tpch:region (regions.R_NAME, nations.N_REGIONKEY)
                where (^{regions.}^.R_REGIONKEY = ^{nations.}^.N_REGIONKEY)
                as virtrdf:nation-n_regionkey ;
            tpch:comment nations.N_COMMENT
                as virtrdf:nation-n_comment .

        tpch:region (regions.R_NAME, nations.N_REGIONKEY)
            tpch:region_of tpch:nation (nations.N_NAME, nations.N_NATIONKEY)
            where (^{regions.}^.R_REGIONKEY = ^{nations.}^.N_REGIONKEY)
            as virtrdf:nation-region_of .

# Order
        tpch:order (orders.O_ORDERKEY)
            a tpch:order
                as virtrdf:order-orders ;
            tpch:orderkey orders.O_ORDERKEY
                as virtrdf:order-o_orderkey ;
            tpch:has_customer tpch:customer (customers.C_NAME, orders.O_CUSTKEY)
```

```
                where (^{orders.}^.O_CUSTKEY = ^{customers.}^.C_CUSTKEY)
                as virtrdf:order-o_custkey ;
            tpch:orderstatus orders.O_ORDERSTATUS
                as virtrdf:order-o_orderstatus ;
            tpch:ordertotalprice orders.O_TOTALPRICE
                as virtrdf:order-o_totalprice ;
            tpch:orderdate orders.O_ORDERDATE
                as virtrdf:order-o_orderdate ;
            tpch:orderpriority orders.O_ORDERPRIORITY
                as virtrdf:order-o_orderpriority ;
            tpch:clerk orders.O_CLERK
                as virtrdf:order-o_clerk ;
            tpch:shippriority orders.O_SHIPPRIORITY
                as virtrdf:order-o_shippriority ;
            tpch:comment orders.O_COMMENT
                as virtrdf:order-o_comment .

        tpch:customer (customers.C_CUSTKEY, orders.O_CUSTKEY)
            tpch:customer_of tpch:order (orders.O_ORDERKEY)
            where (^{orders.}^.O_CUSTKEY = ^{customers.}^.C_CUSTKEY)
            as virtrdf:order-customer_of .

# Part
        tpch:part (parts.P_NAME, parts.P_PARTKEY)
            a tpch:part
                as virtrdf:part-parts ;
            tpch:partkey parts.P_PARTKEY
                as virtrdf:part-p_partkey ;
            tpch:name parts.P_NAME
                as virtrdf:part-p_name ;
            tpch:mfgr parts.P_MFGR
                as virtrdf:part-p_mfgr ;
            tpch:brand parts.P_BRAND
                as virtrdf:part-p_brand ;
            tpch:type parts.P_TYPE
                as virtrdf:part-p_type ;
            tpch:size parts.P_SIZE
                as virtrdf:part-p_size ;
            tpch:container parts.P_CONTAINER
                as virtrdf:part-p_container ;
            tpch:comment parts.P_COMMENT
                as virtrdf:part-p_comment .

# Partsupp
        tpch:partsupp (partsupps.PS_PARTKEY, partsupps.PS_SUPPKEY)
            a tpch:partsupp
                as virtrdf:partsupp-partsupps ;
            tpch:has_part tpch:part (parts.P_NAME, partsupps.PS_PARTKEY)
                where (^{parts.}^.P_PARTKEY = ^{partsupps.}^.PS_PARTKEY)
                as virtrdf:partsupp-ps_partkey ;
            tpch:has_supplier tpch:supplier (suppliers.S_NAME, partsupps.PS_SUPPKEY)
                where (^{suppliers.}^.S_SUPPKEY = ^{partsupps.}^.PS_SUPPKEY)
                as virtrdf:partsupp-ps_suppkey ;
            tpch:availqty partsupps.PS_AVAILQTY
                as virtrdf:partsupp-ps_availqty ;
            tpch:supplycost partsupps.PS_SUPPLYCOST
                as virtrdf:partsupp-ps_supplycost ;
            tpch:comment partsupps.PS_COMMENT
                as virtrdf:partsupp-ps_comment .

        tpch:part (parts.P_NAME, partsupps.PS_PARTKEY)
            tpch:part_of tpch:partsupp (partsupps.PS_PARTKEY, partsupps.PS_SUPPKEY)
            where (^{parts.}^.P_PARTKEY = ^{partsupps.}^.PS_PARTKEY)
            as virtrdf:partsupp-part_of .

        tpch:supplier (suppliers.S_NAME, partsupps.PS_SUPPKEY)
            tpch:supplier_of tpch:partsupp (partsupps.PS_PARTKEY, partsupps.PS_SUPPKEY)
            where (^{suppliers.}^.S_SUPPKEY = ^{partsupps.}^.PS_SUPPKEY)
            as virtrdf:partsupp-supplier_of .

# Region
        tpch:region (regions.R_NAME, regions.R_REGIONKEY)
            a tpch:region
```

```
                as virtrdf:region-regions ;
            tpch:name regions.R_NAME
                as virtrdf:region-r_name ;
            tpch:comment regions.R_COMMENT
                as virtrdf:region-r_comment .

    # Supplier
        tpch:supplier (suppliers.S_NAME, suppliers.S_SUPPKEY)
            a tpch:supplier
                as virtrdf:supplier-suppliers ;
            tpch:name suppliers.S_NAME
                as virtrdf:supplier-s_name ;
            tpch:address suppliers.S_ADDRESS
                as virtrdf:supplier-s_address ;
            tpch:has_nation tpch:nation (nations.N_NAME, suppliers.S_NATIONKEY)
                where (^{nations.}^.N_NATIONKEY = ^{suppliers.}^.S_NATIONKEY)
                as virtrdf:supplier-s_nationkey ;
            foaf:phone suppliers.S_PHONE
                as virtrdf:supplier-foaf_phone ;
            tpch:phone suppliers.S_PHONE
                as virtrdf:supplier-s_phone ;
            tpch:acctbal suppliers.S_ACCTBAL
                as virtrdf:supplier-s_acctbal ;
            tpch:comment suppliers.S_COMMENT
                as virtrdf:supplier-s_comment .

        tpch:nation (nations.N_NAME, suppliers.S_NATIONKEY)
            tpch:nation_of tpch:supplier (suppliers.S_NAME, suppliers.S_SUPPKEY)
            where (^{nations.}^.N_NATIONKEY = ^{suppliers.}^.S_NATIONKEY)
            as virtrdf:supplier-nation_of .
    } .
} .
;

DELETE FROM db.dba.url_rewrite_rule_list WHERE urrl_list like 'tpch_rule%';
DELETE FROM db.dba.url_rewrite_rule WHERE urr_rule like 'tpch_rule%';

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'tpch_rule2',
    1,
    '([^#]*)',
    vector('path'),
    1,
    '/sparql?query=CONSTRUCT+{+%%3Chttp%%3A//^{URIQADefaultHost}^%U%%23this%%3E+%%3Fp+%%3Fo+}+FROM+%%3Cht
    vector('path', 'path', '*accept*'),
    null,
    '(text/rdf.n3)|(application/rdf.xml)',
    0,
    null
    );

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'tpch_rule1',
    1,
    '([^#]*)',
    vector('path'),
    1,
    '/about/html/http/^{URIQADefaultHost}^%s%%23this',
    vector('path'),
    null,
    '(text/html)|(\\*/\\*)',
    0,
    303
    );

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'tpch_rule3',
    1,
    '(/[^#]*)/\x24',
    vector('path'),
    1,
    '%s',
    vector('path'),
```

```
    null,
    null,
    0,
    null
    );

create procedure DB.DBA.REMOVE_TPCH_RDF_DET()
{
  declare colid int;
  colid := DAV_SEARCH_ID('/DAV/home/demo/tpch', 'C');
  if (colid < 0)
    return;
  update WS.WS.SYS_DAV_COL set COL_DET=null where COL_ID = colid;
}
;

DB.DBA.REMOVE_TPCH_RDF_DET();

drop procedure DB.DBA.REMOVE_TPCH_RDF_DET;

create procedure DB.DBA.TPCH_MAKE_RDF_DET()
{
    declare uriqa_str varchar;
    uriqa_str := cfg_item_value(virtuoso_ini_path(), 'URIQA','DefaultHost');
    uriqa_str := 'http://' || uriqa_str || '/tpch';
    DB.DBA."RDFData_MAKE_DET_COL" ('/DAV/home/demo/tpch/RDFData/', uriqa_str, NULL);
    VHOST_REMOVE (lpath=>'/tpch/data/rdf');
    DB.DBA.VHOST_DEFINE (lpath=>'/tpch/data/rdf', ppath=>'/DAV/home/demo/tpch/RDFData/All/', is_dav=>1, v
}
;

DB.DBA.TPCH_MAKE_RDF_DET();

drop procedure DB.DBA.TPCH_MAKE_RDF_DET;

create procedure DB.DBA.TPCH_DET_REF (in par varchar, in fmt varchar, in val varchar)
{
  declare res, iri any;
  declare uriqa_str varchar;
  uriqa_str := cfg_item_value(virtuoso_ini_path(), 'URIQA','DefaultHost');
  uriqa_str := 'http://' || uriqa_str || '/tpch';
  iri := uriqa_str || val;
  res := sprintf ('iid (%d).rdf', iri_id_num (iri_to_id (iri)));
  return sprintf (fmt, res);
}
;

DB.DBA.URLREWRITE_CREATE_REGEX_RULE ('tpch_rdf', 1,
    '/tpch/(.*)', vector('path'), 1,
    '/tpch/data/rdf/%U', vector('path'),
    'DB.DBA.TPCH_DET_REF',
    'application/rdf.xml',
    2,
    303);

DB.DBA.URLREWRITE_CREATE_RULELIST (
    'tpch_rule_list1',
    1,
    vector (
                'tpch_rule1',
                'tpch_rule2',
                'tpch_rule3',
                'tpch_rdf'
          ));

DB.DBA.VHOST_REMOVE (lpath=>'/tpch');
DB.DBA.VHOST_DEFINE (lpath=>'/tpch', ppath=>'/DAV/home/demo/tpch/', vsp_user=>'dba', is_dav=>1,
          is_brws=>0, opts=>vector ('url_rewrite', 'tpch_rule_list1'));

DB.DBA.VHOST_REMOVE (lpath=>'/tpch/linkeddata');
DB.DBA.VHOST_DEFINE (lpath=>'/tpch/linkeddata', ppath=>'/DAV/home/demo/tpch/', vsp_user=>'dba', is_dav=>1
          is_brws=>1);
```

16.8.4. TPCD to RDF

Please load ~\binsrc\dav\DET_RDFData.sql before loadding this script (tpc-d has no vad to do this automatic)

```
use DB;

create procedure DB.DBA.exec_no_error (in expr varchar) {
  declare state, message, meta, result any;
  exec(expr, state, message, vector(), 0, meta, result);
}
;

DB.DBA.exec_no_error('GRANT \"SPARQL_UPDATE\" TO \"SPARQL\"')
;
GRANT SELECT ON tpcd.DBA.partsupp  TO "SPARQL";
GRANT SELECT ON tpcd.DBA.supplier  TO "SPARQL";
GRANT SELECT ON tpcd.DBA.customer  TO "SPARQL";
GRANT SELECT ON tpcd.DBA.history   TO "SPARQL";
GRANT SELECT ON tpcd.DBA.part      TO "SPARQL";
GRANT SELECT ON tpcd.DBA.lineitem  TO "SPARQL";
GRANT SELECT ON tpcd.DBA.orders    TO "SPARQL";
GRANT SELECT ON tpcd.DBA.nation    TO "SPARQL";
GRANT SELECT ON tpcd.DBA.region    TO "SPARQL";

--SPARQL
--prefix tpcd: <http://demo.openlinksw.com/schemas/tpcd#>
--prefix sioc: <http://rdfs.org/sioc/ns#>
--prefix foaf: <http://xmlns.com/foaf/0.1/>
--prefix wgs: <http://www.w3.org/2003/01/geo/wgs84_pos#>
--drop quad map graph iri("http://^{URIQADefaultHost}^/tpcd") .
--;

SPARQL
prefix tpcd: <http://demo.openlinksw.com/schemas/tpcd#>
prefix sioc: <http://rdfs.org/sioc/ns#>
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix wgs: <http://www.w3.org/2003/01/geo/wgs84_pos#>
drop silent quad map virtrdf:TpcdDemo .
;

SPARQL
prefix tpcd: <http://demo.openlinksw.com/schemas/tpcd#>
prefix sioc: <http://rdfs.org/sioc/ns#>
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix wgs: <http://www.w3.org/2003/01/geo/wgs84_pos#>
create iri class tpcd:customer "http://^{URIQADefaultHost}^/tpcd/customer/%d#this" (in c_custkey integer
create iri class tpcd:lineitem "http://^{URIQADefaultHost}^/tpcd/lineitem/%d/%d#this" (in l_orderkey inte
create iri class tpcd:nation "http://^{URIQADefaultHost}^/tpcd/nation/%d#this" (in l_nationkey integer no
create iri class tpcd:order "http://^{URIQADefaultHost}^/tpcd/order/%d#this" (in o_orderkey integer not n
create iri class tpcd:part "http://^{URIQADefaultHost}^/tpcd/part/%d#this" (in p_partkey integer not null
create iri class tpcd:partsupp "http://^{URIQADefaultHost}^/tpcd/partsupp/%d/%d#this" (in ps_partkey inte
create iri class tpcd:region "http://^{URIQADefaultHost}^/tpcd/region/%d#this" (in r_regionkey integer no
create iri class tpcd:supplier "http://^{URIQADefaultHost}^/tpcd/supplier/%d#this" (in s_supplierkey inte
;

SPARQL
prefix tpcd: <http://demo.openlinksw.com/schemas/tpcd#>
prefix sioc: <http://rdfs.org/sioc/ns#>
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix wgs: <http://www.w3.org/2003/01/geo/wgs84_pos#>
alter quad storage virtrdf:DefaultQuadStorage
from tpcd.DBA.lineitem as lineitems
from tpcd.DBA.customer as customers
from tpcd.DBA.nation as nations
from tpcd.DBA.orders as orders
from tpcd.DBA.part as parts
from tpcd.DBA.partsupp as partsupps
from tpcd.DBA.region as regions
from tpcd.DBA.supplier as suppliers
where (^{suppliers.}^.s_nationkey = ^{nations.}^.n_nationkey)
where (^{customers.}^.c_nationkey = ^{nations.}^.n_nationkey)
{
```

```
    create virtrdf:TpcdDemo as graph iri ("http://^{URIQADefaultHost}^/tpcd") option (exclusive)
    {
# Customers
        tpcd:customer (customers.c_custkey)
            a  tpcd:customer
                as virtrdf:tpcdcustomer-type ;
            a  foaf:Organization
                as virtrdf:tpcdcustomer-foaf-type ;
            tpcd:custkey customers.c_custkey
                as virtrdf:tpcdcustomer-c_custkey ;
            foaf:name customers.c_name
                as virtrdf:tpcdcustomer-foaf_name ;
            tpcd:companyName customers.c_name
                as virtrdf:tpcdcustomer-c_name ;
            tpcd:has_nation tpcd:nation (customers.c_nationkey)
                as virtrdf:tpcdcustomer-c_nationkey ;
            tpcd:address customers.c_address
                as virtrdf:tpcdcustomer-c_address ;
            foaf:phone customers.c_phone
                as virtrdf:tpcdcustomer-foaf_phone ;
            tpcd:phone customers.c_phone
                as virtrdf:tpcdcustomer-phone ;
            tpcd:acctbal customers.c_acctbal
                as virtrdf:tpcdcustomer-acctbal ;
            tpcd:mktsegment customers.c_mktsegment
                as virtrdf:tpcdcustomer-c_mktsegment ;
            tpcd:comment customers.c_comment
                as virtrdf:tpcdcustomer-c_comment .

# Nations
        tpcd:nation (customers.c_nationkey)
            tpcd:nation_of tpcd:customer (customers.c_custkey) as virtrdf:tpcdcustomer-nation_of .

# Lineitems
        tpcd:lineitem (lineitems.l_orderkey, lineitems.l_linenumber)
            a tpcd:lineitem
                as virtrdf:tpcdlineitem-lineitems ;
            tpcd:has_order tpcd:order (lineitems.l_orderkey)
                as virtrdf:tpcdlineitem-l_orderkey ;
            tpcd:has_part tpcd:part (lineitems.l_partkey)
                as virtrdf:tpcdlineitem-l_partkey ;
            tpcd:has_supplier tpcd:supplier (lineitems.l_suppkey)
                as virtrdf:tpcdlineitem-l_suppkey ;
            tpcd:linenumber lineitems.l_linenumber
                as virtrdf:tpcdlineitem-l_linenumber ;
            tpcd:linequantity lineitems.l_quantity
                as virtrdf:tpcdlineitem-l_linequantity ;
            tpcd:lineextendedprice lineitems.l_extendedprice
                as virtrdf:tpcdlineitem-l_lineextendedprice ;
            tpcd:linediscount lineitems.l_discount
                as virtrdf:tpcdlineitem-l_linediscount ;
            tpcd:linetax lineitems.l_tax
                as virtrdf:tpcdlineitem-l_linetax ;
            tpcd:returnflag lineitems.l_returnflag
                as virtrdf:tpcdlineitem-l_returnflag ;
            tpcd:linestatus lineitems.l_linestatus
                as virtrdf:tpcdlineitem-l_linestatus ;
            tpcd:shipdate lineitems.l_shipdate
                as virtrdf:tpcdlineitem-l_shipdate ;
            tpcd:commitdate lineitems.l_commitdate
                as virtrdf:tpcdlineitem-l_commitdate ;
            tpcd:receiptdate lineitems.l_receiptdate
                as virtrdf:tpcdlineitem-l_receiptdate ;
            tpcd:shipinstruct lineitems.l_shipinstruct
                as virtrdf:tpcdlineitem-l_shipinstruct ;
            tpcd:shipmode lineitems.l_shipmode
                as virtrdf:tpcdlineitem-l_shipmode ;
            tpcd:comment lineitems.l_comment
                as virtrdf:tpcdlineitem-l_comment .

        tpcd:part (lineitems.l_partkey)
            tpcd:part_of tpcd:lineitem (lineitems.l_orderkey, lineitems.l_linenumber) as virtrdf:tpcdline
```

```
        tpcd:order (lineitems.l_orderkey)
            tpcd:order_of tpcd:lineitem (lineitems.l_orderkey, lineitems.l_linenumber) as virtrdf:tpcdlin

        tpcd:supplier (lineitems.l_suppkey)
            tpcd:supplier_of tpcd:lineitem (lineitems.l_orderkey, lineitems.l_linenumber) as virtrdf:tpcd

# Nation
        tpcd:nation (nations.n_nationkey)
            a tpcd:nation
                as virtrdf:tpcdnation-nations ;
            tpcd:name nations.n_name
                as virtrdf:tpcdnation-n_name ;
            tpcd:has_region tpcd:region (nations.n_regionkey)
                as virtrdf:tpcdnation-n_regionkey ;
            tpcd:comment nations.n_comment
                as virtrdf:tpcdnation-n_comment .

        tpcd:region (nations.n_regionkey)
            tpcd:region_of tpcd:nation (nations.n_nationkey) as virtrdf:tpcdnation-region_of .

# Order
        tpcd:order (orders.o_orderkey)
            a tpcd:order
                as virtrdf:tpcdorder-orders ;
            tpcd:orderkey orders.o_orderkey
                as virtrdf:tpcdorder-o_orderkey ;
            tpcd:has_customer tpcd:customer (orders.o_custkey)
                as virtrdf:tpcdorder-o_custkey ;
            tpcd:orderstatus orders.o_orderstatus
                as virtrdf:tpcdorder-o_orderstatus ;
            tpcd:ordertotalprice orders.o_totalprice
                as virtrdf:tpcdorder-o_totalprice ;
            tpcd:orderdate orders.o_orderdate
                as virtrdf:tpcdorder-o_orderdate ;
            tpcd:orderpriority orders.o_orderpriority
                as virtrdf:tpcdorder-o_orderpriority ;
            tpcd:clerk orders.o_clerk
                as virtrdf:tpcdorder-o_clerk ;
            tpcd:shippriority orders.o_shippriority
                as virtrdf:tpcdorder-o_shippriority ;
            tpcd:comment orders.o_comment
                as virtrdf:tpcdorder-o_comment .

        tpcd:customer (orders.o_custkey)
            tpcd:customer_of tpcd:order (orders.o_orderkey) as virtrdf:tpcdorder-customer_of .

# Part
        tpcd:part (parts.p_partkey)
            a tpcd:part
                as virtrdf:tpcdpart-parts ;
            tpcd:partkey parts.p_partkey
                as virtrdf:tpcdpart-p_partkey ;
            tpcd:name parts.p_name
                as virtrdf:tpcdpart-p_name ;
            tpcd:mfgr parts.p_mfgr
                as virtrdf:tpcdpart-p_mfgr ;
            tpcd:brand parts.p_brand
                as virtrdf:tpcdpart-p_brand ;
            tpcd:type parts.p_type
                as virtrdf:tpcdpart-p_type ;
            tpcd:size parts.p_size
                as virtrdf:tpcdpart-p_size ;
            tpcd:container parts.p_container
                as virtrdf:tpcdpart-p_container ;
            tpcd:comment parts.p_comment
                as virtrdf:tpcdpart-p_comment .

# Partsupp
        tpcd:partsupp (partsupps.ps_partkey, partsupps.ps_suppkey)
            a tpcd:partsupp
                as virtrdf:tpcdpartsupp-partsupps ;
            tpcd:has_part tpcd:part (partsupps.ps_partkey)
                as virtrdf:tpcdpartsupp-ps_partkey ;
```

```
            tpcd:has_supplier tpcd:supplier (partsupps.ps_suppkey)
                as virtrdf:tpcdpartsupp-ps_suppkey ;
            tpcd:availqty partsupps.ps_availqty
                as virtrdf:tpcdpartsupp-ps_availqty ;
            tpcd:supplycost partsupps.ps_supplycost
                as virtrdf:tpcdpartsupp-ps_supplycost ;
            tpcd:comment partsupps.ps_comment
                as virtrdf:tpcdpartsupp-ps_comment .

        tpcd:part (partsupps.ps_partkey)
            tpcd:part_of tpcd:partsupp (partsupps.ps_partkey, partsupps.ps_suppkey) as virtrdf:tpcdpartsu

        tpcd:supplier (partsupps.ps_suppkey)
            tpcd:supplier_of tpcd:partsupp (partsupps.ps_partkey, partsupps.ps_suppkey) as virtrdf:tpcdpa

# Region
        tpcd:region (regions.r_regionkey)
            a tpcd:region
                as virtrdf:tpcdregion-regions ;
            tpcd:name regions.r_name
                as virtrdf:tpcdregion-r_name ;
            tpcd:comment regions.r_comment
                as virtrdf:tpcdregion-r_comment .

# Supplier
        tpcd:supplier (suppliers.s_suppkey)
            a tpcd:supplier
                as virtrdf:tpcdsupplier-suppliers ;
            tpcd:name suppliers.s_name
                as virtrdf:tpcdsupplier-s_name ;
            tpcd:address suppliers.s_address
                as virtrdf:tpcdsupplier-s_address ;
            tpcd:has_nation tpcd:nation (suppliers.s_nationkey)
                as virtrdf:tpcdsupplier-s_nationkey ;
            foaf:phone suppliers.s_phone
                as virtrdf:tpcdsupplier-foaf_phone ;
            tpcd:phone suppliers.s_phone
                as virtrdf:tpcdsupplier-s_phone ;
            tpcd:acctbal suppliers.s_acctbal
                as virtrdf:tpcdsupplier-s_acctbal ;
            tpcd:comment suppliers.s_comment
                as virtrdf:tpcdsupplier-s_comment .

        tpcd:nation (suppliers.s_nationkey)
            tpcd:nation_of tpcd:supplier (suppliers.s_suppkey) as virtrdf:tpcdsupplier-nation_of .
    }
}
;

create procedure tcpd_rdf_doc (in path varchar)
{
  declare r any;
  r := regexp_match ('[^/]*\x24', path);
  return r||'#this';
};

create procedure tcpd_html_doc (in path varchar)
{
  declare r any;
  r := regexp_match ('[^/]*#', path);
  return subseq (r, 0, length (r)-1);
};

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'tcpd_rule2',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/sparql?query=CONSTRUCT+{+%%3Chttp%%3A//^{URIQADefaultHost}^%U%%23this%%3E+%%3Fp+%%3Fo+}+FROM+%%3Cht
    vector('path', 'path', '*accept*'),
    null,
    '(text/rdf.n3)|(application/rdf.xml)',
```

```
    0,
    null
    );

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'tcpd_rule1',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/rdfbrowser/index.html?uri=http%%3A//^{URIQADefaultHost}^%U%%23this',
    vector('path'),
    null,
    '(text/html)|(\\*/\\*)',
    0,
    303
    );

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'tcpd_rule3',
    1,
    '(/[^#]*)/\x24',
    vector('path'),
    1,
    '%s',
    vector('path'),
    null,
    null,
    0,
    null
    );

DB.DBA."RDFData_MAKE_DET_COL" ('/DAV/home/tpcd/RDFData/', 'http://^{URIQADefaultHost}^/tpcd', NULL);
VHOST_REMOVE (lpath=>'/tpcd/data/rdf');
DB.DBA.VHOST_DEFINE (lpath=>'/tpcd/data/rdf', ppath=>'/DAV/home/tpcd/RDFData/All/', is_dav=>1, vsp_user=>

-- procedure to convert path to DET resource name
create procedure DB.DBA.TPCD_DET_REF (in par varchar, in fmt varchar, in val varchar)
{
  declare res, iri any;
  iri := 'http://^{URIQADefaultHost}^/tpcd' || val;
  res := sprintf ('iid (%d).rdf', iri_id_num (iri_to_id (iri)));
  return sprintf (fmt, res);
}
;

DB.DBA.URLREWRITE_CREATE_REGEX_RULE ('tpcd_rdf', 1,
    '/tpcd/(.*)', vector('path'), 1,
    '/tpcd/data/rdf/%U', vector('path'),
    'DB.DBA.TPCD_DET_REF',
    'application/rdf.xml',
    2,
    303);

DB.DBA.URLREWRITE_CREATE_RULELIST (
    'tpcd_rule_list1',
    1,
    vector (
                'tcpd_rule1',
                'tcpd_rule2',
                'tcpd_rule3',
                'tpcd_rdf'
          ));

VHOST_REMOVE (lpath=>'/tpcd');
DB.DBA.VHOST_DEFINE (lpath=>'/tpcd', ppath=>'/DAV/home/', vsp_user=>'dba', is_dav=>1, def_page=>'sfront.v
    is_brws=>0, opts=>vector ('url_rewrite', 'tpcd_rule_list1'));

create procedure DB.DBA.LOAD_TPCD_ONTOLOGY_FROM_DAV()
{
        declare content, urihost varchar;
        select cast (RES_CONTENT as varchar) into content from WS.WS.SYS_DAV_RES where RES_FULL_PATH = '/
        DB.DBA.RDF_LOAD_RDFXML (content, 'http://demo.openlinksw.com/schemas/tpcd#', 'http://demo.openlin
```

```
        urihost := cfg_item_value(virtuoso_ini_path(), 'URIQA','DefaultHost');
        if (urihost = 'demo.openlinksw.com')
        {
                DB.DBA.VHOST_REMOVE (lpath=>'/schemas/tpcd');
                DB.DBA.VHOST_DEFINE (lpath=>'/schemas/tpcd', ppath=>'/DAV/VAD/tpcd/tpcd.owl', vsp_user=>'
        }
}
;
DB.DBA.LOAD_TPCD_ONTOLOGY_FROM_DAV()
;
drop procedure DB.DBA.LOAD_TPCD_ONTOLOGY_FROM_DAV
;

XML_SET_NS_DECL ('virt-tpcd', 'http://demo.openlinksw.com/schemas/tpcd#', 2)
;
```

16.8.5. Thalia to RDF

```
use DB
;

SPARQL drop quad map virtrdf:ThaliaDemo
;

create procedure DB.DBA.SPARQL_THALIA_RUN (in txt varchar)
{
  declare REPORT, stat, msg, sqltext varchar;
  declare metas, rowset any;
  result_names (REPORT);
  sqltext := string_output_string (sparql_to_sql_text (txt));
  stat := '00000';
  msg := '';
  rowset := null;
  exec (sqltext, stat, msg, vector (), 1000, metas, rowset);
}
;

create procedure DB.DBA.exec_no_error(in expr varchar)
{
        declare state, message, meta, result any;
        exec(expr, state, message, vector(), 0, meta, result);
}
;

DB.DBA.exec_no_error('drop View thalia.Demo.asu_v');
DB.DBA.exec_no_error('create View thalia.Demo.asu_v as select left(Title,3) code,* from thalia.Demo.asu')
DB.DBA.exec_no_error('drop View thalia.Demo.gatech_v');
DB.DBA.exec_no_error('create View thalia.Demo.gatech_v as select *, Room||\' \'||Building Place from thal
DB.DBA.SPARQL_THALIA_RUN('drop quad map graph iri("http://^{URIQADefaultHost}^/Thalia") .
')
;

GRANT SELECT ON thalia.Demo.asu TO "SPARQL"
;
GRANT SELECT ON thalia.Demo.asu_v TO "SPARQL"
;
GRANT SELECT ON thalia.Demo.brown TO "SPARQL"
;
GRANT SELECT ON thalia.Demo.cmu TO "SPARQL"
;
GRANT SELECT ON thalia.Demo.gatech TO "SPARQL"
;
GRANT SELECT ON thalia.Demo.gatech_v TO "SPARQL"
;
GRANT SELECT ON thalia.Demo.toronto TO "SPARQL"
;
GRANT SELECT ON thalia.Demo.ucsd TO "SPARQL"
;
GRANT SELECT ON thalia.Demo.umd TO "SPARQL"
;
```

```
DB.DBA.SPARQL_THALIA_RUN('drop quad map graph iri("http://^{URIQADefaultHost}^/thalia") .
');

DB.DBA.SPARQL_THALIA_RUN('drop quad map virtrdf:ThaliaDemo .
');

DB.DBA.SPARQL_THALIA_RUN('
prefix sioc: <http://rdfs.org/sioc/ns#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix dc: <http://purl.org/dc/elements/1.1/>
prefix time: <http://www.w3.org/2006/time#>
prefix event: <http://purl.org/NET/c4dm/event.owl#>
prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix skos: <http://www.w3.org/2004/02/skos/core#>
prefix th: <http://demo.openlinksw.com/schemas/thalia#>

create iri class th:Asu "http://^{URIQADefaultHost}^/thalia/asu/course/%U#this" (in code varchar not null
create iri class th:Brown "http://^{URIQADefaultHost}^/thalia/brown/course/%U#this" (in Code varchar not
create iri class th:BrownInstructor "http://^{URIQADefaultHost}^/thalia/brown/instructor/%U#this" (in Cod
create iri class th:BrownLecture "http://^{URIQADefaultHost}^/thalia/brown/lecture/%U#this" (in Code varc
create iri class th:BrownPlace "http://^{URIQADefaultHost}^/thalia/brown/place/%U#this" (in Code varchar

create iri class th:Cmu "http://^{URIQADefaultHost}^/thalia/cmu/course/%U/%U#this" (in Code varchar not n
create iri class th:CmuInstructor "http://^{URIQADefaultHost}^/thalia/cmu/instructor/%U/%U#this" (in Code
create iri class th:CmuLecture "http://^{URIQADefaultHost}^/thalia/cmu/lecture/%U/%U#this" (in Code varch
create iri class th:CmuPlace "http://^{URIQADefaultHost}^/thalia/cmu/place/%U/%U#this" (in Code varchar n
create iri class th:CmuEventTime "http://^{URIQADefaultHost}^/thalia/cmu/eventtime/%U/%U#this" (in Code v
create iri class th:CmuDatetime "http://^{URIQADefaultHost}^/thalia/cmu/datetime/%U/%U#this" (in Code var

create iri class th:Gatech "http://^{URIQADefaultHost}^/thalia/gatech/course/%U/%d/%U#this" (in Departmen
create iri class th:GatechInstructor "http://^{URIQADefaultHost}^/thalia/gatech/instructor/%U/%d/%U#this"
create iri class th:GatechLecture "http://^{URIQADefaultHost}^/thalia/gatech/lecture/%U/%d/%U#this" (in D
create iri class th:GatechEventTime "http://^{URIQADefaultHost}^/thalia/gatech/eventtime/%U/%d/%U#this" (
create iri class th:GatechDatetime "http://^{URIQADefaultHost}^/thalia/gatech/datetime/%U/%d/%U#this" (in
create iri class th:GatechPlace "http://^{URIQADefaultHost}^/thalia/gatech/place/%U/%d/%U#this" (in Depar

create iri class th:Toronto "http://^{URIQADefaultHost}^/thalia/toronto/course/%U#this" (in No_ varchar)
create iri class th:TorontoInstructor "http://^{URIQADefaultHost}^/thalia/toronto/instructor/%U#this" (in
create iri class th:TorontoLecture "http://^{URIQADefaultHost}^/thalia/toronto/lecture/%U#this" (in No_ v
create iri class th:TorontoPlace "http://^{URIQADefaultHost}^/thalia/toronto/place/%U#this" (in No_ varch

create iri class th:Ucsd "http://^{URIQADefaultHost}^/thalia/ucsd/course/%U#this" (in Number varchar) .
create iri class th:UcsdInstructor1 "http://^{URIQADefaultHost}^/thalia/ucsd/instructor1/%U#this" (in Num
create iri class th:UcsdInstructor2 "http://^{URIQADefaultHost}^/thalia/ucsd/instructor2/%U#this" (in Num
create iri class th:UcsdInstructor3 "http://^{URIQADefaultHost}^/thalia/ucsd/instructor3/%U#this" (in Num

create iri class th:Umd "http://^{URIQADefaultHost}^/thalia/umd/course/%U#this" (in Code varchar) .
create iri class th:UmdLecture "http://^{URIQADefaultHost}^/thalia/umd/lecture/%U#this" (in Code varchar)
create iri class th:UmdEventTime "http://^{URIQADefaultHost}^/thalia/umd/eventtime/%U#this" (in Code varc
create iri class th:UmdDatetime "http://^{URIQADefaultHost}^/thalia/umd/datetime/%U#this" (in Code varcha
')
;

DB.DBA.SPARQL_THALIA_RUN('prefix sioc: <http://rdfs.org/sioc/ns#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix dc: <http://purl.org/dc/elements/1.1/>
prefix time: <http://www.w3.org/2006/time#>
prefix event: <http://purl.org/NET/c4dm/event.owl#>
prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix skos: <http://www.w3.org/2004/02/skos/core#>
prefix th: <http://demo.openlinksw.com/schemas/thalia#>
alter quad storage virtrdf:DefaultQuadStorage
from thalia.demo.asu_v as asus
from thalia.demo.brown as browns
from thalia.demo.cmu as cmus
```

```
from thalia.demo.gatech_v as gatechs
from thalia.demo.toronto as torontos
from thalia.demo.ucsd as ucsds
from thalia.demo.umd as umds
{
        create virtrdf:ThaliaDemo as graph iri ("http://^{URIQADefaultHost}^/thalia") option (exclusive)
        {
                th:Asu (asus.code)
                    a th:Course
                        as virtrdf:Asu-Course ;
                    dc:title asus.Title
                        as virtrdf:Asu-Title ;
                    dc:description asus.Description
                        as virtrdf:Asu-Description ;
                    rdfs:seeAlso asus.MoreInfoURL
                        as virtrdf:Asu-MoreInfoURL ;
                        th:forUniversity "http://purl.org/thalia/university/asu/university/asu"
                            as virtrdf:Asu-University ;
                        skos:subject "http://purl.org/subject/thalia/ComputerScience"
                            as virtrdf:Asu-Subject
                            .

                th:Brown (browns.Code)
                        a th:Course
                                as virtrdf:Brown-Course ;
                        dc:title browns.Title
                            as virtrdf:Brown-Title ;
                        th:hasInstructor th:BrownInstructor (browns.Code)
                            as virtrdf:Brown-hasInstructor ;
                        th:hasLecture th:BrownLecture(browns.Code)
                            as virtrdf:Brown-hasLecture ;
                        th:forUniversity "http://purl.org/thalia/university/brown"
                            as virtrdf:Brown-University ;
                        skos:subject "http://purl.org/subject/thalia/ComputerScience"
                            as virtrdf:Brown-Subject
                            .
                th:BrownInstructor (browns.Code)
                        a th:Instructor
                                as virtrdf:Brown-Instructor ;
                        dc:homepage browns.Instructor
                            as virtrdf:Brown-Instructor-Homepage
                            .
                th:BrownLecture (browns.Code)
                        a event:Event
                                as virtrdf:Brown-Lecture ;
                        event:place th:BrownPlace(browns.Code)
                            as virtrdf:Brown-hasPlace
                            .
                th:BrownPlace (browns.Code)
                        a geo:Point
                                as virtrdf:Brown-Place;
                        dc:title browns.Room
                            as virtrdf:Brown-Room
                            .

                th:Cmu (cmus.Code, cmus.Sec)
                    a th:Course
                        as virtrdf:Cmu-Course ;
                    dc:title cmus.CourseTitle
                        as virtrdf:Cmu-CourseTitle ;
                        th:hasInstructor th:CmuInstructor (cmus.Code, cmus.Sec)
                            as virtrdf:Cmu-hasInstructor ;
                        th:hasLecture th:CmuLecture(cmus.Code, cmus.Sec)
                            as virtrdf:Cmu-hasLecture ;
                        th:hasUnits cmus.Units
                            as virtrdf:Cmu-hasUnits ;
                        th:forUniversity "http://purl.org/thalia/university/cmu"
                            as virtrdf:Cmu-University ;
                        skos:subject "http://purl.org/subject/thalia/ComputerScience"
                            as virtrdf:Cmu-Subject
                        .
        th:CmuInstructor (cmus.Code, cmus.Sec)
                        a th:Instructor
```

```
                        as virtrdf:Cmu-Instructor ;
            foaf:name cmus.Lecturer
                as virtrdf:Cmu-Lecturer
                    .
th:CmuLecture (cmus.Code, cmus.Sec)
                a event:Event
                        as virtrdf:Cmu-Lecture ;
                event:time th:CmuEventTime(cmus.Code, cmus.Sec)
                    as virtrdf:Cmu-hasEventTime ;
                event:place th:CmuPlace(cmus.Code, cmus.Sec)
                    as virtrdf:Cmu-hasPlace
                    .
th:CmuPlace (cmus.Code, cmus.Sec)
                a geo:Point
                        as virtrdf:Cmu-Place;
                dc:title cmus.Room
                    as virtrdf:Cmu-Room
                    .
th:CmuEventTime (cmus.Code, cmus.Sec)
                a time:Interval
                        as virtrdf:Cmu-EventTime;
                time:inDateTime th:CmuDatetime(cmus.Code, cmus.Sec)
                    as virtrdf:Cmu-inDateTime
                    .
th:CmuDatetime (cmus.Code, cmus.Sec)
                a time:DateTimeDescription
                        as virtrdf:Cmu-Datetime;
            time:dayOfWeek cmus.Day_
                as virtrdf:Cmu-Day ;
            time:hour cmus.Time_
                as virtrdf:Cmu-Time
                    .

        th:Gatech (gatechs.Department, gatechs.Code, gatechs.Section)
            a th:Course
                as virtrdf:Gatech-Course ;
            dc:title gatechs.Title
                as virtrdf:Gatech-Title ;
                th:hasInstructor th:GatechInstructor(gatechs.Department, gatechs.Code, gatechs.Se
                    as virtrdf:Gatech-hasInstructor ;
            dc:description gatechs.Description
                as virtrdf:Gatech-Description ;
                th:hasLecture th:GatechLecture(gatechs.Department, gatechs.Code, gatechs.Section)
                    as virtrdf:Gatech-hasLecture ;
                th:forUniversity "http://purl.org/thalia/university/gatech"
                    as virtrdf:Gatech-University ;
                skos:subject "http://purl.org/subject/thalia/ComputerScience"
                    as virtrdf:Gatech-Subject
                    .
th:GatechInstructor (gatechs.Department, gatechs.Code, gatechs.Section)
                a th:Instructor
                        as virtrdf:Gatech-Instructor ;
            foaf:name gatechs.Instructor
                as virtrdf:Gatech-InstructorName
        .
th:GatechLecture (gatechs.Department, gatechs.Code, gatechs.Section)
                a event:Event
                        as virtrdf:Gatech-Lecture ;
                event:time th:GatechEventTime(gatechs.Department, gatechs.Code, gatechs.Section)
                    as virtrdf:Gatech-hasEventTime ;
                event:place th:GatechPlace(gatechs.Department, gatechs.Code, gatechs.Section)
                    as virtrdf:Gatech-hasPlace
        .
th:GatechEventTime (gatechs.Department, gatechs.Code, gatechs.Section)
                a time:Interval
                        as virtrdf:Gatech-EventTime ;
                time:inDateTime th:GatechDatetime(gatechs.Department, gatechs.Code, gatechs.Secti
                    as virtrdf:Gatech-inDateTime
        .
th:GatechDatetime (gatechs.Department, gatechs.Code, gatechs.Section)
                a time:DateTimeDescription
                        as virtrdf:Gatech-Datetime ;
            time:dayOfWeek gatechs.Days
```

```
                as virtrdf:Gatech-Days ;
            time:hour gatechs.Time_
                as virtrdf:Gatech-Time_
        .
th:GatechPlace (gatechs.Department, gatechs.Code, gatechs.Section)
                a geo:Point
                        as virtrdf:Gatech-Place ;
                dc:title gatechs.Place
                    as virtrdf:Gatech-RoomBuilding
        .

        th:Toronto (torontos.No_)
                a th:Course
                        as virtrdf:Toronto-Course ;
                dc:title torontos.title
                    as virtrdf:Toronto-Title ;
                dc:description torontos.text_
                    as virtrdf:Toronto-Description ;
                th:hasInstructor th:TorontoInstructor(torontos.No_)
                    as virtrdf:Toronto-hasInstructor ;
                th:hasLecture th:TorontoLecture(torontos.No_)
                    as virtrdf:Toronto-hasLecture ;
                rdfs:seeAlso torontos.coursewebsite
                    as virtrdf:Toronto-CourseWebSite ;
                th:hasPrerequisite torontos.prereq
                    as virtrdf:Toronto-prereq ;
                th:text torontos.text_
                    as virtrdf:Toronto-text;
                th:forUniversity "http://purl.org/thalia/university/toronto"
                    as virtrdf:Toronto-University ;
                skos:subject "http://purl.org/subject/thalia/ComputerScience"
                    as virtrdf:Toronto-Subject
                    .
th:TorontoInstructor (torontos.No_)
                a th:Instructor
                        as virtrdf:Toronto-Instructor ;
                foaf:name torontos.instructorName
                    as virtrdf:Toronto-InstructorName ;
                foaf:mbox torontos.instructorEmail
                    as virtrdf:Toronto-InstructorEmail
        .
th:TorontoLecture (torontos.No_)
                a event:Event
                        as virtrdf:Toronto-Lecture ;
                event:place th:TorontoPlace(torontos.No_)
                    as virtrdf:Toronto-hasPlace
        .
th:TorontoPlace (torontos.No_)
                a geo:Point
                        as virtrdf:Toronto-Place ;
                dc:title torontos.location
                    as virtrdf:Toronto-Location
        .

        th:Ucsd (ucsds.Number)
                a th:Course
                        as virtrdf:Ucsd-Course ;
                dc:title ucsds.Title
                    as virtrdf:Ucsd-Title ;
                th:hasInstructor1 th:UcsdInstructor1 (ucsds.Number)
                    as virtrdf:Ucsd-hasInstructor1 ;
                th:hasInstructor2 th:UcsdInstructor2 (ucsds.Number)
                    as virtrdf:Ucsd-hasInstructor2 ;
                th:hasInstructor3 th:UcsdInstructor3 (ucsds.Number)
                    as virtrdf:Ucsd-hasInstructor3 ;
                th:forUniversity "http://purl.org/thalia/university/ucsd"
                    as virtrdf:Ucsd-University ;
                skos:subject "http://purl.org/subject/thalia/ComputerScience"
                    as virtrdf:Ucsd-Subject
                    .
        th:UcsdInstructor1 (ucsds.Number)
                a th:Instructor
                        as virtrdf:Ucsd-Instructor1 ;
```

```
                        foaf:name ucsds.Fall2003
                            as virtrdf:Ucsd-Instructor-Fall2003
                            .
                    th:UcsdInstructor2 (ucsds.Number)
                            a th:Instructor
                                    as virtrdf:Ucsd-Instructor2 ;
                            foaf:name ucsds.Winter2004
                            as virtrdf:Ucsd-Instructor-Winter2004
                            .
                    th:UcsdInstructor3 (ucsds.Number)
                            a th:Instructor
                                    as virtrdf:Ucsd-Instructor3 ;
                            foaf:name ucsds.Spring2004
                            as virtrdf:Ucsd-Instructor-Spring2004
                            .

                    th:Umd (umds.Code)
                        a th:Course
                            as virtrdf:Umd-Course ;
                        dc:title umds.CourseName
                            as virtrdf:Umd-Title ;
                            th:hasSection th:SectionTitle
                            as virtrdf:Umd-hasSection ;
                            th:hasLecture th:UmdLecture(umds.Code)
                            as virtrdf:Umd-hasLecture ;
                            th:forUniversity "http://purl.org/thalia/university/umd"
                            as virtrdf:Umd-University ;
                            skos:subject "http://purl.org/subject/thalia/ComputerScience"
                            as virtrdf:Umd-Subject
                            .
            th:UmdLecture (umds.Code)
                            a event:Event
                                    as virtrdf:Umd-Lecture ;
                            event:time th:UmdEventTime(umds.Code)
                            as virtrdf:Umd-hasEventTime
                    .
            th:UmdEventTime (umds.Code)
                            a time:Interval
                                    as virtrdf:Umd-EventTime ;
                            time:inDateTime th:UmdDatetime(umds.Code)
                            as virtrdf:Umd-inDateTime
                    .
            th:UmdDatetime (umds.Code)
                            a time:DateTimeDescription
                                    as virtrdf:Umd-Datetime ;
                        time:hour umds.SectionTime
                            as virtrdf:Umd-SectionTime
                    .
            }
}
')
;

delete from db.dba.url_rewrite_rule_list where urrl_list like 'tut_th_%';
delete from db.dba.url_rewrite_rule where urr_rule like 'tut_th_%';

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'tut_th_rule1',
    1,
    '([^#]*)',
    vector('path'),
    1,
    '/about/html/http/^{URIQADefaultHost}^%s%%01this',
    vector('path'),
    null,
    '(text/html)|(\\*/\\*)',
    0,
    303
    );

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'tut_th_rule2',
    1,
```

```
    '([^#]*)',
    vector('path'),
    1,
    '/sparql?query=CONSTRUCT+{+%%3Chttp%%3A//^{URIQADefaultHost}^%U%%23this%%3E+%%3Fp+%%3Fo+}+FROM+%%3Cht
    vector('path', 'path', '*accept*'),
    null,
    '(text/rdf.n3)|(application/rdf.xml)',
    0,
    null
    );

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'tut_th_rule3',
    1,
    '(/[^#]*)/\x24',
    vector('path'),
    1,
    '%U',
    vector('path'),
    null,
    null,
    0,
    null
    );

create procedure DB.DBA.REMOVE_THALIA_RDF_DET()
{
  declare colid int;
  colid := DAV_SEARCH_ID('/DAV/Thalia', 'C');
  if (colid < 0)
    return;
  update WS.WS.SYS_DAV_COL set COL_DET=null where COL_ID = colid;
}
;

DB.DBA.REMOVE_THALIA_RDF_DET();

drop procedure DB.DBA.REMOVE_THALIA_RDF_DET;

create procedure DB.DBA.THALIA_MAKE_RDF_DET()
{
    declare uriqa_str varchar;
    uriqa_str := cfg_item_value(virtuoso_ini_path(), 'URIQA','DefaultHost');
    uriqa_str := 'http://' || uriqa_str || '/thalia';
    DB.DBA."RDFData_MAKE_DET_COL" ('/DAV/Thalia/RDFData/', uriqa_str, NULL);
    VHOST_REMOVE (lpath=>'/thalia/data/rdf');
    DB.DBA.VHOST_DEFINE (lpath=>'/thalia/data/rdf', ppath=>'/DAV/Thalia/RDFData/All/', is_dav=>1, vsp_use
}
;

DB.DBA.THALIA_MAKE_RDF_DET();

drop procedure DB.DBA.THALIA_MAKE_RDF_DET;

-- procedure to convert path to DET resource name
create procedure DB.DBA.THALIA_DET_REF (in par varchar, in fmt varchar, in val varchar)
{
  declare res, iri any;
  declare uriqa_str varchar;
  uriqa_str := cfg_item_value(virtuoso_ini_path(), 'URIQA','DefaultHost');
  uriqa_str := 'http://' || uriqa_str || '/thalia';
  iri := uriqa_str || replace(val, '/', '_');
  res := sprintf ('iid (%d).rdf', iri_id_num (iri_to_id (iri)));
  return sprintf (fmt, res);
}
;

DB.DBA.URLREWRITE_CREATE_REGEX_RULE ('tut_th_rdf', 1,
    '/thalia/(.*)', vector('path'), 1,
    '/thalia/data/rdf/%U', vector('path'),
    'DB.DBA.THALIA_DET_REF',
    'application/rdf.xml',
    2,
```

```
    303);

DB.DBA.URLREWRITE_CREATE_RULELIST (
    'tut_th_rule_list1',
    1,
    vector (
                'tut_th_rule1',
                'tut_th_rule2',
                'tut_th_rule3',
                'tut_th_rdf'
          ));

DB.DBA.VHOST_REMOVE (lpath=>'/thalia');
DB.DBA.VHOST_DEFINE (lpath=>'/thalia', ppath=>'/DAV/Thalia/', vsp_user=>'dba', is_dav=>1,
           is_brws=>0, opts=>vector ('url_rewrite', 'tut_th_rule_list1'));

create procedure DB.DBA.LOAD_THALIA_ONTOLOGY_FROM_DAV()
{
        declare content, urihost varchar;
        select cast (RES_CONTENT as varchar) into content from WS.WS.SYS_DAV_RES where RES_FULL_PATH = '/
        DB.DBA.RDF_LOAD_RDFXML (content, 'http://demo.openlinksw.com/schemas/thalia#', 'http://demo.openl
        urihost := cfg_item_value(virtuoso_ini_path(), 'URIQA','DefaultHost');
        if (urihost = 'demo.openlinksw.com')
        {
                DB.DBA.VHOST_REMOVE (lpath=>'/schemas/thalia');
                DB.DBA.VHOST_DEFINE (lpath=>'/schemas/thalia', ppath=>'/DAV/Thalia/thalia.owl', vsp_user=
        }
}
;
DB.DBA.LOAD_THALIA_ONTOLOGY_FROM_DAV()
;
drop procedure DB.DBA.LOAD_THALIA_ONTOLOGY_FROM_DAV
;

DB.DBA.XML_SET_NS_DECL ('thalia', 'http://demo.openlinksw.com/schemas/thalia#', 2)
;

    * Demo : Thalia test queries

#service:/sparql
#should-sponge:soft
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX event: <http://purl.org/NET/c4dm/event.owl#>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX th: <http://purl.org/ontology/thalia/1.0/>

SELECT distinct ?room
FROM <http://demo.openlinksw.com/thalia>
WHERE
  {
    ?course a th:Course;
              dc:title ?title;
              th:hasLecture ?lecture.
    ?lecture event:place [dc:title ?room].
    FILTER regex(?title, "Software Engineering")
}

#service:/sparql
#should-sponge:soft
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX event: <http://purl.org/NET/c4dm/event.owl#>
```

```
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX th: <http://purl.org/ontology/thalia/1.0/>

SELECT ?day, ?hour ?course
FROM <http://demo.openlinksw.com/thalia>
WHERE
  {
    ?course a th:Course;
    th:hasLecture [event:time ?time];
    dc:title ?title.
    ?time time:inDateTime [time:dayOfWeek ?day];
    time:inDateTime [time:hour ?hour].
    FILTER regex(?title, "Computer Networks")
  }

#service:/sparql
#should-sponge:soft

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX event: <http://purl.org/NET/c4dm/event.owl#>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX th: <http://purl.org/ontology/thalia/1.0/>

SELECT distinct ?course
FROM <http://demo.openlinksw.com/thalia>
WHERE
  {
    ?course a th:Course;
              dc:title ?Title;
              th:hasLecture ?lecture.
    ?lecture event:time [time:inDateTime ?dateTime].
    ?dateTime time:hour ?hour.
    FILTER regex(?Title, "Database System")
    FILTER regex(?hour, "1:30 - 2:50")
  }

#service:/sparql
#should-sponge:soft
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX event: <http://purl.org/NET/c4dm/event.owl#>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX th: <http://purl.org/ontology/thalia/1.0/>

SELECT distinct ?course ?instructor ?name
FROM <http://demo.openlinksw.com/thalia>
WHERE
  {
    ?course a th:Course;
              th:hasInstructor ?instructor.
    ?instructor foaf:name ?name.

  }

#service:/sparql
#should-sponge:soft
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX event: <http://purl.org/NET/c4dm/event.owl#>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX th: <http://purl.org/ontology/thalia/1.0/>

SELECT distinct ?instructor
FROM <http://demo.openlinksw.com/thalia>
WHERE
  {
    ?course a th:Course;
              th:hasInstructor ?instructor;
              dc:title  ?title.
    FILTER regex(?title, "Database")
  }

#service:/sparql
#should-sponge:soft
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX event: <http://purl.org/NET/c4dm/event.owl#>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX th: <http://purl.org/ontology/thalia/1.0/>

SELECT distinct ?instructor
FROM <http://demo.openlinksw.com/thalia>
WHERE
  {
    ?course a th:Course;
              dc:title ?title;
              th:hasInstructor ?instructor.
    FILTER regex(?title, "Software")
  }

#service:/sparql
#should-sponge:soft
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX event: <http://purl.org/NET/c4dm/event.owl#>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX th: <http://purl.org/ontology/thalia/1.0/>

SELECT DISTINCT ?course
FROM <http://demo.openlinksw.com/thalia>
WHERE
  {
    ?course a th:Course;
              dc:title ?title;
              th:forUniversity 'http://purl.org/thalia/university/umd'.
    FILTER regex(?title, "Data Structures")
  }

#service:/sparql
#should-sponge:soft
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
```

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX event: <http://purl.org/NET/c4dm/event.owl#>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX th: <http://purl.org/ontology/thalia/1.0/>

SELECT distinct ?course
FROM <http://demo.openlinksw.com/thalia>
WHERE
  {
    ?course a th:Course;
              dc:title ?Title;
              th:hasUnits ?credits.
    FILTER (xsd:integer(?credits) > 10)
    FILTER regex(?Title, "Database")
  }

#service:/sparql
#should-sponge:soft
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX event: <http://purl.org/NET/c4dm/event.owl#>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX th: <http://purl.org/ontology/thalia/1.0/>

SELECT distinct ?course
FORM <http://demo.openlinksw.com/thalia>
WHERE
  {
    ?course a th:Course;
              dc:title ?title;
              th:forUniversity 'http://purl.org/thalia/university/umd'.
    FILTER regex(?title, "Database")
  }

#service:/sparql
#should-sponge:soft
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX event: <http://purl.org/NET/c4dm/event.owl#>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX th: <http://purl.org/ontology/thalia/1.0/>

SELECT distinct ?course
FROM <http://demo.openlinksw.com/thalia>
WHERE
  {
    ?course a th:Course;
              dc:title ?Title;
              th:hasUnits ?credits.
    FILTER (xsd:integer(?credits) > 10)
    FILTER regex(?Title, "Database")
  }

#service:/sparql
#should-sponge:soft
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX event: <http://purl.org/NET/c4dm/event.owl#>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX th: <http://purl.org/ontology/thalia/1.0/>

SELECT distinct ?text_
FROM <http://demo.openlinksw.com/thalia>
WHERE
  {
    ?course a th:Course;
             dc:title ?title;
             th:text ?text_.
    FILTER regex(?title, "Verification")
  }

#service:/sparql
#should-sponge:soft
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX event: <http://purl.org/NET/c4dm/event.owl#>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX : <http://purl.org/ontology/thalia/1.0/>

SELECT distinct ?course
FROM <http://demo.openlinksw.com/thalia>
WHERE
  {
    ?course a th:Course;
              dc:description ?description;
              th:forUniversity 'http://purl.org/thalia/university/gatech'.
    FILTER regex(?description, "JR")
  }
```

16.8.6. Musicbrainz to RDF

The following code creates the Musicbrainz Linked Data Views Deployment and Demo Scripts:

```
create text index on ZITGIST.MO.artist ("name") with key id;
create text index on ZITGIST.MO.artistalias ("name") with key id;
create text index on ZITGIST.MO.album ("name") with key id;
create text index on ZITGIST.MO.track ("name") with key id;
vt_batch_update (fix_identifier_case ('ZITGIST.MO.artist'), 'ON', NULL);
vt_batch_update (fix_identifier_case ('ZITGIST.MO.artistalias'), 'ON', NULL);
vt_batch_update (fix_identifier_case ('ZITGIST.MO.album'), 'ON', NULL);
vt_batch_update (fix_identifier_case ('ZITGIST.MO.track'), 'ON', NULL);
VT_INC_INDEX_DB_MO_artist ();
VT_INC_INDEX_DB_MO_artistalias ();
VT_INC_INDEX_DB_MO_album ();
VT_INC_INDEX_DB_MO_track ();
```

Note: Making sure that the graphs and views are deleting to clean Virtuoso from the old definitions

```
SPARQL
drop quad storage virtrdf:MBZROOT.
;

SPARQL
prefix mbz: <http://musibrainz.org/schemas/mbz#>
drop literal class mbz:duration
;
```

```
SPARQL
prefix mbz: <http://musibrainz.org/schemas/mbz#>
drop literal class mbz:created.
drop literal class mbz:official_iri.
drop literal class mbz:bootleg_iri.
drop literal class mbz:promotion_iri.
drop literal class mbz:album_iri.
drop literal class mbz:single_iri.
drop literal class mbz:ep_iri.
drop literal class mbz:compilation_iri.
drop literal class mbz:soundtrack_iri.
drop literal class mbz:spokenword_iri.
drop literal class mbz:interview_iri.
drop literal class mbz:audiobook_iri.
drop literal class mbz:live_iri.
drop literal class mbz:remix_iri.
;
```

The following SPARQL query will fix an issue Virtuoso has with its JSO system. Perform this query for now, the issue should be fixed in a future release

```
SPARQL define input:storage ""
DELETE FROM GRAPH (iri(bif:JSO_SYS_GRAPH NIL)) { ?s virtrdf:version ?o }
WHERE { graph `iri(bif:JSO_SYS_GRAPH NIL)` {?s virtrdf:version ?o}};
SPARQL_RELOAD_QM_GRAPH();
```

Creation of IRIs classes.

```
SPARQL

prefix mbz: <http://musibrainz.org/schemas/mbz#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix dc: <http://purl.org/dc/elements/1.1/>
prefix bio: <http://vocab.org/bio/0.1/#>
prefix rel: <http://vocab.org/relationship/#>
prefix mo: <http://purl.org/ontology/mo/>
prefix timeline: <http://purl.org/NET/c4dm/timeline.owl#>
prefix event: <http://purl.org/NET/c4dm/event.owl#>
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix sim: <http://purl.org/ontology/sim/>

create iri class mbz:artist_iri  "http://zitgist.com/music/artist/%U" (in gid varchar not null) option (b
create iri class mbz:artist_birth_event_iri  "http://zitgist.com/music/artist/birth/%U" (in gid varchar n
create iri class mbz:artist_death_event_iri  "http://zitgist.com/music/artist/death/%U" (in gid varchar n
create iri class mbz:sim_link_iri  "http://zitgist.com/music/artist/simlink/%U" (in gid varchar not null)

#create iri class mbz:band_iri  "http://zitgist.com/music/band/%U" (in gid varchar not null) option (bije
#create iri class mbz:band_birth_event_iri  "http://zitgist.com/music/band/birth/%U" (in gid varchar not
#create iri class mbz:band_death_event_iri  "http://zitgist.com/music/band/death/%U" (in gid varchar not

create iri class mbz:record_iri  "http://zitgist.com/music/record/%U" (in gid varchar not null) option (b
create iri class mbz:performance_iri  "http://zitgist.com/music/performance/%U" (in gid varchar not null)
create iri class mbz:composition_iri  "http://zitgist.com/music/composition/%U" (in gid varchar not null)
create iri class mbz:musicalwork_iri  "http://zitgist.com/music/musicalwork/%U" (in gid varchar not null)
create iri class mbz:sound_iri  "http://zitgist.com/music/sound/%U" (in gid varchar not null) option (bij
create iri class mbz:recording_iri  "http://zitgist.com/music/recording/%U" (in gid varchar not null) opt
create iri class mbz:signal_iri  "http://zitgist.com/music/signal/%U" (in gid varchar not null) option (b

create iri class mbz:track_iri  "http://zitgist.com/music/track/%U" (in gid varchar not null) option (bij

create iri class mbz:image_iri  "http://ec1.images-amazon.com/images/P/%U.01.MZZZZZZZ.jpg" (in image varc

create iri class mbz:amazon_asin_iri  "http://amazon.com/exec/obidos/ASIN/%U/searchcom07-20" (in gid varc

create literal class mbz:created using
    function ZITGIST.MO.RECORD_CREATION_DATE (in datestring varchar) returns varchar,
    function ZITGIST.MO.RECORD_CREATION_DATE_INVERSE (in datestring varchar) returns varchar .

create iri class mbz:official_iri using
    function ZITGIST.MO.RECORD_ATTRIBUTE_OFFICIAL (in attributes varchar) returns varchar
    option (returns 'http://purl.org/ontology/mo/official') .
```

```
create iri class mbz:promotion_iri using
    function ZITGIST.MO.RECORD_ATTRIBUTE_PROMOTION (in attributes varchar) returns varchar
    option (returns 'http://purl.org/ontology/mo/promotion') .

create iri class mbz:bootleg_iri using
    function ZITGIST.MO.RECORD_ATTRIBUTE_BOOTLEG (in attributes varchar) returns varchar
    option (returns 'http://purl.org/ontology/mo/bootleg') .

create iri class mbz:album_iri using
    function ZITGIST.MO.RECORD_ATTRIBUTE_ALBUM (in attributes varchar) returns varchar
    option (returns 'http://purl.org/ontology/mo/album') .

create iri class mbz:single_iri using
    function ZITGIST.MO.RECORD_ATTRIBUTE_SINGLE (in attributes varchar) returns varchar
    option (returns 'http://purl.org/ontology/mo/single') .

create iri class mbz:ep_iri using
    function ZITGIST.MO.RECORD_ATTRIBUTE_EP (in attributes varchar) returns varchar
    option (returns 'http://purl.org/ontology/mo/ep') .

create iri class mbz:compilation_iri using
    function ZITGIST.MO.RECORD_ATTRIBUTE_COMPILATION (in attributes varchar) returns varchar
    option (returns 'http://purl.org/ontology/mo/compilation') .

create iri class mbz:soundtrack_iri using
    function ZITGIST.MO.RECORD_ATTRIBUTE_SOUNDTRACK (in attributes varchar) returns varchar
    option (returns 'http://purl.org/ontology/mo/soundtrack') .

create iri class mbz:spokenword_iri using
    function ZITGIST.MO.RECORD_ATTRIBUTE_SPOKENWORD (in attributes varchar) returns varchar
    option (returns 'http://purl.org/ontology/mo/spokenword') .

create iri class mbz:interview_iri using
    function ZITGIST.MO.RECORD_ATTRIBUTE_INTERVIEW (in attributes varchar) returns varchar
    option (returns 'http://purl.org/ontology/mo/interview') .

create iri class mbz:audiobook_iri using
    function ZITGIST.MO.RECORD_ATTRIBUTE_AUDIOBOOK (in attributes varchar) returns varchar
    option (returns 'http://purl.org/ontology/mo/audiobook') .

create iri class mbz:live_iri using
    function ZITGIST.MO.RECORD_ATTRIBUTE_LIVE (in attributes varchar) returns varchar
    option (returns 'http://purl.org/ontology/mo/live') .

create iri class mbz:remix_iri using
    function ZITGIST.MO.RECORD_ATTRIBUTE_REMIX (in attributes varchar) returns varchar
    option (returns 'http://purl.org/ontology/mo/remix') .

create iri class mbz:duration_iri  "http://zitgist.com/music/track/duration/%U" (in gid varchar not null)

create literal class mbz:duration using
    function ZITGIST.MO.TRACK_DURATION (in duration integer) returns varchar ,
    function ZITGIST.MO.TRACK_DURATION_INVERSE (in durationXSD varchar) returns integer .

create iri class mbz:geoname_country_iri  "http://www.geonames.org/countries/#%U" (in country varchar not

create iri class mbz:url_iri  "%s" (in url varchar not null) .
create iri class mbz:mbz_release_url_iri  "http://musicbrainz.org/release/%s.html" (in mbz_gid varchar no
create iri class mbz:mbz_track_url_iri  "http://musicbrainz.org/track/%s.html" (in mbz_gid varchar not nu
create iri class mbz:mbz_artist_url_iri  "http://musicbrainz.org/artist/%s.html" (in mbz_gid varchar not
;
```

List of functions used to compute some IRI classes:

Note:These functions have been developed to handle some weird user cases of the Musicbrainz data model (like the Attribute column of the album table, etc).

```
create function ZITGIST.MO.TRACK_DURATION_INVERSE(in durationXSD varchar)
{
    return null;
};
```

```
create function ZITGIST.MO.TRACK_DURATION(in duration integer)
{
    declare minutes, seconds, milliseconds integer;

    minutes := ((duration / 1000) / 60);

    if(minutes >= 1)
    {
        minutes := cast(minutes as integer);
    }
    else
    {
        minutes := 0;
    }

    seconds := (duration / 1000) - (minutes * 60);

    if(seconds >= 1)
    {
        seconds := cast(seconds as integer);
    }

    milliseconds := duration - (seconds * 1000) - (minutes * 60000);

    return sprintf('PT%dM%dS', minutes, seconds);
}
;

create function ZITGIST.MO.RECORD_CREATION_DATE(in datestring varchar)
{
    return sprintf('%sT00:00:00Z', datestring);
};

create function ZITGIST.MO.RECORD_CREATION_DATE_INVERSE(in datestring varchar)
{
    declare pos integer;
    pos := locate('T00:00:00Z', datestring) - 1;
    return substring(datestring, 1, pos);
};

create function ZITGIST.MO.RECORD_ATTRIBUTE(in attribute integer, in attributes varchar)
{
    declare attributes_array any;

    attributes_array := split_and_decode(ltrim(rtrim(attributes, '}'), '{'), 0, '\0\0,');

    foreach(int attr in attributes_array) do
    {
        attr := cast(attr as integer);
        if(attr = attribute)
        {
            if(attr = 100) return 'http://purl.org/ontology/mo/official';
            if(attr = 101) return 'http://purl.org/ontology/mo/promotion';
            if(attr = 102) return 'http://purl.org/ontology/mo/bootleg';
            if(attr = 1)   return 'http://purl.org/ontology/mo/album';
            if(attr = 2)   return 'http://purl.org/ontology/mo/single';
            if(attr = 3)   return 'http://purl.org/ontology/mo/ep';
            if(attr = 4)   return 'http://purl.org/ontology/mo/compilation';
            if(attr = 5)   return 'http://purl.org/ontology/mo/soundtrack';
            if(attr = 6)   return 'http://purl.org/ontology/mo/spokenword';
            if(attr = 7)   return 'http://purl.org/ontology/mo/interview';
            if(attr = 8)   return 'http://purl.org/ontology/mo/audiobook';
            if(attr = 9)   return 'http://purl.org/ontology/mo/live';
            if(attr = 10)  return 'http://purl.org/ontology/mo/remix';
        }
    }
    return null;
}
;

create function ZITGIST.MO.RECORD_ATTRIBUTE_OFFICIAL(in attributes varchar)
{    return ZITGIST.MO.RECORD_ATTRIBUTE(100, attributes); }
;
```

```
create function ZITGIST.MO.RECORD_ATTRIBUTE_PROMOTION(in attributes varchar)
{    return ZITGIST.MO.RECORD_ATTRIBUTE(101, attributes);}
;
create function ZITGIST.MO.RECORD_ATTRIBUTE_BOOTLEG(in attributes varchar)
{    return ZITGIST.MO.RECORD_ATTRIBUTE(102, attributes);}
;
create function ZITGIST.MO.RECORD_ATTRIBUTE_ALBUM(in attributes varchar)
{    return ZITGIST.MO.RECORD_ATTRIBUTE(1, attributes);}
;
create function ZITGIST.MO.RECORD_ATTRIBUTE_SINGLE(in attributes varchar)
{    return ZITGIST.MO.RECORD_ATTRIBUTE(2, attributes);}
;
create function ZITGIST.MO.RECORD_ATTRIBUTE_EP(in attributes varchar)
{    return ZITGIST.MO.RECORD_ATTRIBUTE(3, attributes);}
;
create function ZITGIST.MO.RECORD_ATTRIBUTE_COMPILATION(in attributes varchar)
{    return ZITGIST.MO.RECORD_ATTRIBUTE(4, attributes);}
;
create function ZITGIST.MO.RECORD_ATTRIBUTE_SOUNDTRACK(in attributes varchar)
{    return ZITGIST.MO.RECORD_ATTRIBUTE(5, attributes);}
;
create function ZITGIST.MO.RECORD_ATTRIBUTE_SPOKENWORD(in attributes varchar)
{    return ZITGIST.MO.RECORD_ATTRIBUTE(6, attributes);}
;
create function ZITGIST.MO.RECORD_ATTRIBUTE_INTERVIEW(in attributes varchar)
{    return ZITGIST.MO.RECORD_ATTRIBUTE(7, attributes);}
;
create function ZITGIST.MO.RECORD_ATTRIBUTE_AUDIOBOOK(in attributes varchar)
{    return ZITGIST.MO.RECORD_ATTRIBUTE(8, attributes);}
;
create function ZITGIST.MO.RECORD_ATTRIBUTE_LIVE(in attributes varchar)
{    return ZITGIST.MO.RECORD_ATTRIBUTE(9, attributes);}
;
create function ZITGIST.MO.RECORD_ATTRIBUTE_REMIX(in attributes varchar)
{    return ZITGIST.MO.RECORD_ATTRIBUTE(10, attributes);}
;
```

Definition of the quad map patterns

This what creates the RDF triples from the musicbrainz relational database schema.

```
SPARQL
prefix mbz: <http://musibrainz.org/schemas/mbz#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix dc: <http://purl.org/dc/elements/1.1/>
prefix dcterms: <http://purl.org/dc/terms/>
prefix bio: <http://vocab.org/bio/0.1/#>
prefix rel: <http://vocab.org/relationship/#>
prefix mo: <http://purl.org/ontology/mo/>
prefix timeline: <http://purl.org/NET/c4dm/timeline.owl#>
prefix event: <http://purl.org/NET/c4dm/event.owl#>
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix sim: <http://purl.org/ontology/sim/>

create quad storage virtrdf:MBZROOT

#
# Definition of the source tables from the mbz relational database and their joints.
#########

FROM ZITGIST.MO.track as track text literal name
FROM ZITGIST.MO.artist as track_artist
FROM ZITGIST.MO.puid as track_puid
FROM ZITGIST.MO.track as track_track
FROM ZITGIST.MO.url as track_url

FROM ZITGIST.MO.artist as track_artist_creator where (^{track.}^.artist = ^{track_artist_creator.}^.id)

FROM ZITGIST.MO.albumjoin as track_albumjoin where (^{track.}^.id = ^{track_albumjoin.}^.track)

FROM ZITGIST.MO.l_artist_track as l_artist_track2 where (^{track.}^.id = ^{l_artist_track2.}^.link1)
                                                        where (^{track_artist.}^.id = ^{l_artist_track2.}^.
```

```
FROM ZITGIST.MO.puidjoin as puidjoin where (^{track.}^.id = ^{puidjoin.}^.track)
                                        where (^{puidjoin.}^.puid = ^{track_puid.}^.id)

FROM ZITGIST.MO.l_track_track as l_track_track where (^{track.}^.id = ^{l_track_track.}^.link0)
                                                 where (^{track_track.}^.id = ^{l_track_track.}^.link1)

FROM ZITGIST.MO.l_track_url as l_track_url where (^{track.}^.id = ^{l_track_url.}^.link0)
                                            where (^{track_url.}^.id = ^{l_track_url.}^.link1)

FROM ZITGIST.MO.album as album text literal name
FROM ZITGIST.MO.artist as album_artist
FROM ZITGIST.MO.album as album_album
FROM ZITGIST.MO.url as album_url
FROM ZITGIST.MO.country as album_release_country
FROM ZITGIST.MO.track as album_albumjoin_track

FROM ZITGIST.MO.artist as album_artist_creator where (^{album.}^.artist = ^{album_artist_creator.}^.id)

FROM ZITGIST.MO.album_amazon_asin as album_amazon_asin where (^{album.}^.id = ^{album_amazon_asin.}^.albu

FROM ZITGIST.MO.albumjoin as album_albumjoin where (^{album.}^.id = ^{album_albumjoin.}^.album)
                                              where (^{album_albumjoin.}^.track = ^{album_albumjoin_tr

FROM ZITGIST.MO.l_album_artist as l_album_artist2 where (^{album.}^.id = ^{l_album_artist2.}^.link0)
                                                   where (^{album_artist.}^.id = ^{l_album_artist2.}^.

FROM ZITGIST.MO.l_album_album as l_album_album where (^{album.}^.id = ^{l_album_album.}^.link0)
                                                 where (^{album_album.}^.id = ^{l_album_album.}^.link1)

FROM ZITGIST.MO.l_album_url as l_album_url where (^{album.}^.id = ^{l_album_url.}^.link0)
                                            where (^{album_url.}^.id = ^{l_album_url.}^.link1)

FROM ZITGIST.MO.release as album_release where (^{album.}^.id = ^{album_release.}^.album)
                                         where (^{album_release.}^.country = ^{album_release_country.

FROM ZITGIST.MO.artist as sim_band
FROM ZITGIST.MO.artist as sim_artist
FROM ZITGIST.MO.url as band_url
FROM ZITGIST.MO.artist as band_member
FROM ZITGIST.MO.album as band_album
FROM ZITGIST.MO.track as band_track
FROM ZITGIST.MO.artist as band text literal name where (^{band.}^.type = 2)
#FROM ZITGIST.MO.artist as artist text literal name where (^{artist.}^.type <> 2)
FROM ZITGIST.MO.artist as artist text literal name where (__or (neq(^{artist.}^.type, 2), isnull (^{artis
FROM ZITGIST.MO.artist as artist_untyped text literal name where (^{artist_untyped.}^.type <> 2)
                                                           where (^{artist.}^.gid = ^{artist_untyped.}^.gid)

FROM ZITGIST.MO.album as band_album_creatorOf where (^{band_album_creatorOf.}^.artist = ^{band.}^.id)
FROM ZITGIST.MO.track as band_track_creatorOf where (^{band_track_creatorOf.}^.artist = ^{band.}^.id)

FROM ZITGIST.MO.artistalias as bandalias text literal name where (^{band.}^.id = ^{bandalias.}^."ref")

FROM ZITGIST.MO.l_artist_artist as band_l_artist_artist where (^{band_member.}^.id = ^{band_l_artist_arti
                                                                where (^{band.}^.id = ^{band_l_artist_artist.
                                                                where (^{band_l_artist_artist.}^.link_type =

FROM ZITGIST.MO.artist_relation as band_relation
where (^{artist.}^.id = ^{band_relation.}^.artist)
where (^{band.}^.id = ^{band_relation.}^.artist)
where (^{sim_band.}^.id = ^{band_relation.}^."ref")

FROM ZITGIST.MO.artist_relation as artist_relation
where (^{artist.}^.id = ^{artist_relation.}^.artist)
where (^{band.}^.id = ^{artist_relation.}^.artist)
where (^{sim_artist.}^.id = ^{artist_relation.}^."ref")

FROM ZITGIST.MO.l_artist_url as l_artist_url3 where (^{band.}^.id = ^{l_artist_url3.}^.link0)
                                                 where (^{band_url.}^.id = ^{l_artist_url3.}^.link1)

FROM ZITGIST.MO.l_album_artist as l_album_artist3 where (^{band.}^.id = ^{l_album_artist3.}^.link1)
                                                   where (^{band_album.}^.id = ^{l_album_artist3.}^.li
```

```
FROM ZITGIST.MO.l_artist_track as l_artist_track3 where (^{band.}^.id = ^{l_artist_track3.}^.link0)
                                                    where (^{band_track.}^.id = ^{l_artist_track3.}^.li

FROM ZITGIST.MO.url as artist_url
FROM ZITGIST.MO.artist as artist_artist
FROM ZITGIST.MO.track as artist_track
FROM ZITGIST.MO.album as artist_album

FROM ZITGIST.MO.album as artist_album_creatorOf where (^{artist_album_creatorOf.}^.artist = ^{artist.}^.i
FROM ZITGIST.MO.track as artist_track_creatorOf where (^{artist_track_creatorOf.}^.artist = ^{artist.}^.i

FROM ZITGIST.MO.artistalias as artistalias text literal name where (^{artist.}^.id = ^{artistalias.}^."re
FROM ZITGIST.MO.l_artist_url as l_artist_url where (^{artist.}^.id = ^{l_artist_url.}^.link0)
                                                 where (^{artist_url.}^.id = ^{l_artist_url.}^.link1)

FROM ZITGIST.MO.l_artist_artist as l_artist_artist where (^{artist.}^.id = ^{l_artist_artist.}^.link0)
                                                    where (^{artist_artist.}^.id = ^{l_artist_artist.}

FROM ZITGIST.MO.l_artist_track as l_artist_track where (^{artist.}^.id = ^{l_artist_track.}^.link0)
                                                   where (^{artist_track.}^.id = ^{l_artist_track.}^.li
FROM ZITGIST.MO.l_album_artist as l_album_artist where (^{artist.}^.id = ^{l_album_artist.}^.link1)
                                                   where (^{artist_album.}^.id = ^{l_album_artist.}^.li

{
  create virtrdf:MBZ as graph iri ("http://musicbrainz.org/") option (exclusive)
    {

        # Track Composition Event
        mbz:composition_iri (track.gid)
            a mo:Composition as mbz:track_is_composition;
            dc:title track.name as mbz:title_of_track;
            mo:composer mbz:artist_iri (track_artist_creator.gid) as mbz:creator_composer_of_track;
            mo:composer mbz:artist_iri (track_artist.gid) where (^{l_artist_track2.}^.link_type = 14) opt
            mo:producesWork mbz:musicalwork_iri (track.gid) as mbz:track_producesWork.

        # Track Musical Work
        mbz:musicalwork_iri (track.gid)
            a mo:MusicalWork as mbz:track_is_mw;
            dc:title track.name as mbz:name_of_mw;

            mo:productOfComposition mbz:composition_iri(track.gid) as mbz:mw_is_productOfComposition_of;
            mo:usedInPerformance mbz:performance_iri(track.gid) as mbz:mw_usedInPerformance.

        # Track Performance Event
        mbz:performance_iri (track.gid)
            a mo:Performance;
            dc:title track.name;
            mo:performer mbz:artist_iri (track_artist_creator.gid);
            mo:performer mbz:artist_iri (track_artist.gid) where (^{l_artist_track2.}^.link_type = 2) opt
            mo:conductor mbz:artist_iri (track_artist.gid) where (^{l_artist_track2.}^.link_type = 9) opt

            mo:usesWork mbz:musicalwork_iri (track.gid);
            mo:producesSound mbz:sound_iri (track.gid);

            mo:recordedAs mbz:signal_iri(track.gid).

        # Track Sound
        mbz:sound_iri (track.gid)
            a mo:Sound;
            dc:title track.name;

            mo:productOfPerformance mbz:performance_iri (track.gid);
            mo:usedInRecording mbz:recording_iri (track.gid).

        # Track Recording Event
        mbz:recording_iri (track.gid)
            a mo:Recording;
            dc:title track.name;

            mo:recordsSound mbz:sound_iri (track.gid);
            mo:producesSignal mbz:signal_iri (track.gid).

        # Track Signal (Musical Expression)
```

```
mbz:signal_iri (track.gid)
    a mo:Signal;
    dc:title track.name;

    mo:remixer mbz:artist_iri (track_artist.gid) where (^{l_artist_track2.}^.link_type = 11) opti
    mo:sampler mbz:artist_iri (track_artist.gid) where (^{l_artist_track2.}^.link_type = 12) opti
    mo:djmixed mbz:artist_iri (track_artist.gid) where (^{l_artist_track2.}^.link_type = 40) opti

    mo:djmix_of mbz:track_iri (track_track.gid) where (^{l_track_track.}^.link_type = 13) option
    mo:remix_of mbz:track_iri (track_track.gid) where (^{l_track_track.}^.link_type = 6) option (
    mo:remix_of mbz:track_iri (track_track.gid) where (^{l_track_track.}^.link_type = 11) option
    mo:mashup_of mbz:track_iri (track_track.gid) where (^{l_track_track.}^.link_type = 8) option
    mo:mashup_of mbz:track_iri (track_track.gid) where (^{l_track_track.}^.link_type = 4) option
    mo:remaster_of mbz:track_iri (track_track.gid) where (^{l_track_track.}^.link_type = 3) optio
    mo:compilation_of mbz:track_iri (track_track.gid) where (^{l_track_track.}^.link_type = 10) o
    mo:compilation_of mbz:track_iri (track_track.gid) where (^{l_track_track.}^.link_type = 12) o
    mo:medley_of mbz:record_iri (track_track.gid) where (^{l_track_track.}^.link_type = 14) optio

    mo:published_as mbz:track_iri (track.gid);
    mo:signalTime mbz:duration_iri(track.gid);
    mo:puid track_puid.puid option (using puidjoin).

# Track duration
mbz:duration_iri(track.gid)
    a timeline:Interval;
    timeline:durationXSD mbz:duration(track.length).

mbz:track_iri(track.gid)
    a mo:Track;
    dc:title track.name;

    mo:trackNum track_albumjoin.sequence;

    dc:creator mbz:artist_iri (track_artist_creator.gid);
    dc:creator mbz:artist_iri (track_artist.gid) where (^{l_artist_track2.}^.link_type = 14) opti
                mo:compiler mbz:artist_iri (track_artist.gid) where (^{l_artist_track2.}^.link_ty
    mo:producer mbz:artist_iri (track_artist.gid) where (^{l_artist_track2.}^.link_type = 18) opt
    mo:publisher mbz:artist_iri (track_artist.gid) where (^{l_artist_track2.}^.link_type = 35) op
    mo:engineer mbz:artist_iri (track_artist.gid) where (^{l_artist_track2.}^.link_type = 19) opt

    mo:licence mbz:url_iri(track_url.url) where (^{l_track_url.}^.link_type = 21) option (using l
    mo:paiddownload mbz:url_iri(track_url.url) where (^{l_track_url.}^.link_type = 16) option (us
    mo:freedownload mbz:url_iri(track_url.url) where (^{l_track_url.}^.link_type = 17) option (us
    mo:olga mbz:url_iri(track_url.url) where (^{l_track_url.}^.link_type = 19) option (using l_tr

    mo:musicbrainz mbz:mbz_track_url_iri(track.gid);

    mo:duration track.length.

# Record Composition Event
mbz:composition_iri (album.gid)
    a mo:Composition;
    dc:title album.name;

    mo:composer mbz:artist_iri (album_artist_creator.gid);
    mo:composer mbz:artist_iri (album_artist.gid) where (^{l_album_artist2.}^.link_type = 14) opt

    mo:producesWork mbz:musicalwork_iri (album.gid).

# Record Musical Work
mbz:musicalwork_iri (album.gid)
    a mo:MusicalWork;
    dc:title album.name;

    mo:productOfComposition mbz:composition_iri(album.gid);
    mo:usedInPerformance mbz:performance_iri(album.gid).

# Record Performance Event
mbz:performance_iri (album.gid)
    a mo:Performance;
    dc:title album.name;
    mo:performer mbz:artist_iri (album_artist_creator.gid);
    mo:performer mbz:artist_iri (album_artist.gid) where (^{l_album_artist2.}^.link_type = 2) opt
```

```
    mo:conductor mbz:artist_iri (album_artist.gid) where (^{l_album_artist2.}^.link_type = 9) opt

    mo:usesWork mbz:musicalwork_iri (album.gid);
    mo:producesSound mbz:sound_iri (album.gid);

    mo:recordedAs mbz:record_iri(album.gid).

# Record Sound
mbz:sound_iri (album.gid)
    a mo:Sound;
    dc:title album.name;

    mo:productOfPerformance mbz:performance_iri (album.gid);
    mo:usedInRecording mbz:recording_iri (album.gid).

# Record Recording Event
mbz:recording_iri (album.gid)
    a mo:Recording;
    dc:title album.name;

    mo:recordsSound mbz:sound_iri (album.gid);
    mo:producesSignal mbz:signal_iri (album.gid).

# Record Signal (Musical Expression)
mbz:signal_iri (album.gid)
    a mo:Signal;
    dc:title album.name;

    mo:djmix_of mbz:record_iri (album_album.gid) where (^{l_album_album.}^.link_type = 9) option
    mo:remix_of mbz:record_iri (album_album.gid) where (^{l_album_album.}^.link_type = 7) option
    mo:remix_of mbz:record_iri (album_album.gid) where (^{l_album_album.}^.link_type = 4) option
    mo:mashup_of mbz:record_iri (album_album.gid) where (^{l_album_album.}^.link_type = 5) option
    mo:remaster_of mbz:record_iri (album_album.gid) where (^{l_album_album.}^.link_type = 3) opti
    mo:tribute_to mbz:artist_iri (album_artist.gid) where (^{l_album_artist2.}^.link_type = 44) o

    mo:remixer mbz:artist_iri (album_artist.gid) where (^{l_album_artist2.}^.link_type = 11) opti
    mo:djmixed mbz:artist_iri (album_artist.gid) where (^{l_album_artist2.}^.link_type = 38) opti
    mo:sampler mbz:artist_iri (album_artist.gid) where (^{l_album_artist2.}^.link_type = 12) opti

    mo:published_as mbz:record_iri (album.gid).

# Record (Musical Manifestation)
mbz:record_iri (album.gid)
    a mo:Record;
    dc:title album.name;

    dc:date mbz:created(album_release.releasedate);
    mo:image mbz:image_iri(album_amazon_asin.asin);

    #Empty for now.
    mo:compilation_of mbz:record_iri (album_album.gid) where (^{l_album_album.}^.link_type = 8) o
    mo:releaseStatus mbz:official_iri(album.attributes);
    mo:releaseStatus mbz:promotion_iri(album.attributes);
    mo:releaseStatus mbz:bootleg_iri(album.attributes);

    mo:releaseType mbz:album_iri(album.attributes);
    mo:releaseType mbz:single_iri(album.attributes);
    mo:releaseType mbz:ep_iri(album.attributes);
    mo:releaseType mbz:compilation_iri(album.attributes);
    mo:releaseType mbz:soundtrack_iri(album.attributes);
    mo:releaseType mbz:spokenword_iri(album.attributes);
    mo:releaseType mbz:interview_iri(album.attributes);
    mo:releaseType mbz:audiobook_iri(album.attributes);
    mo:releaseType mbz:live_iri(album.attributes);
    mo:releaseType mbz:remix_iri(album.attributes);

    dc:creator mbz:artist_iri (album_artist_creator.gid);
    dc:creator mbz:artist_iri (album_artist.gid) where (^{l_album_artist2.}^.link_type = 14) opti

    mo:compiler mbz:artist_iri (album_artist.gid) where (^{l_album_artist2.}^.link_type = 41) opt
    mo:producer mbz:artist_iri (album_artist.gid) where (^{l_album_artist2.}^.link_type = 18) opt
    mo:publisher mbz:artist_iri (album_artist.gid) where (^{l_album_artist2.}^.link_type = 35) op
    mo:engineer mbz:artist_iri (album_artist.gid) where (^{l_album_artist2.}^.link_type = 19) opt
```

```
            mo:musicbrainz mbz:mbz_release_url_iri(album.gid);

            mo:musicmoz mbz:url_iri(album_url.url) where (^{l_album_url.}^.link_type = 25) option (using
            mo:discogs mbz:url_iri(album_url.url) where (^{l_album_url.}^.link_type = 24) option (using l
            mo:wikipedia mbz:url_iri(album_url.url) where (^{l_album_url.}^.link_type = 23) option (using
            mo:discography mbz:url_iri(album_url.url) where (^{l_album_url.}^.link_type = 1) option (usin
            mo:freedownload mbz:url_iri(album_url.url) where (^{l_album_url.}^.link_type = 21) option (us
            mo:discography mbz:url_iri(album_url.url) where (^{l_album_url.}^.link_type = 16) option (usi
            mo:mailorder mbz:url_iri(album_url.url) where (^{l_album_url.}^.link_type = 19) option (using
            mo:imdb mbz:url_iri(album_url.url) where (^{l_album_url.}^.link_type = 27) option (using l_al
            mo:paiddownload mbz:url_iri(album_url.url) where (^{l_album_url.}^.link_type = 20) option (us
            mo:licence mbz:url_iri(album_url.url) where (^{l_album_url.}^.link_type = 32) option (using l
            mo:review mbz:url_iri(album_url.url) where (^{l_album_url.}^.link_type = 17) option (using l_

            mo:amazon_asin mbz:amazon_asin_iri(album_amazon_asin.asin);

            mo:has_track mbz:track_iri (album_albumjoin_track.gid) option (using album_albumjoin).

        # Music Group (Band)
#        mbz:band_iri(band.gid)
         mbz:artist_iri(band.gid)
            a mo:MusicArtist;
            a mo:MusicGroup;
            a foaf:Group;
            foaf:name band.name;
            foaf:nick bandalias.name;

#           bio:event mbz:band_birth_event_iri(band.gid);
#           bio:event mbz:band_death_event_iri(band.gid);
            bio:event mbz:artist_birth_event_iri(band.gid);
            bio:event mbz:artist_death_event_iri(band.gid);

#           mo:similar_to mbz:band_iri(sim_band.gid) option (using band_relation);
            mo:similar_to mbz:artist_iri(sim_band.gid) option (using band_relation);
            mo:similar_to mbz:artist_iri(sim_artist.gid) option (using artist_relation);
#            sim:link mbz:sim_link_iri(sim_band.gid)  option (using band_relation);
#            sim:link mbz:sim_link_iri(sim_artist.gid)  option (using artist_relation);

            foaf:member mbz:artist_iri(band_member.gid) option (using band_l_artist_artist);

            # l_artist_url
            mo:myspace mbz:url_iri(band_url.url) where (^{l_artist_url3.}^.link_type = 19) option (using
            mo:musicmoz mbz:url_iri(band_url.url) where (^{l_artist_url3.}^.link_type = 12) option (using
            mo:discogs mbz:url_iri(band_url.url) where (^{l_artist_url3.}^.link_type = 11) option (using
            mo:wikipedia mbz:url_iri(band_url.url) where (^{l_artist_url3.}^.link_type = 10) option (usin
            mo:discography mbz:url_iri(band_url.url) where (^{l_artist_url3.}^.link_type = 1) option (usi
            mo:freedownload mbz:url_iri(band_url.url) where (^{l_artist_url3.}^.link_type = 8) option (us
            mo:fanpage mbz:url_iri(band_url.url) where (^{l_artist_url3.}^.link_type = 3) option (using l
            mo:biography mbz:url_iri(band_url.url) where (^{l_artist_url3.}^.link_type = 4) option (using
            mo:discography mbz:url_iri(band_url.url) where (^{l_artist_url3.}^.link_type = 5) option (usi
            mo:mailorder mbz:url_iri(band_url.url) where (^{l_artist_url3.}^.link_type = 15) option (usin
            mo:imdb mbz:url_iri(band_url.url) where (^{l_artist_url3.}^.link_type = 17) option (using l_a
            mo:paiddownload mbz:url_iri(band_url.url) where (^{l_artist_url3.}^.link_type = 7) option (us
            foaf:depiction mbz:url_iri(band_url.url) where (^{l_artist_url3.}^.link_type = 14) option (us
            foaf:homepage mbz:url_iri(band_url.url) where (^{l_artist_url3.}^.link_type = 2) option (usin

            mo:musicbrainz mbz:mbz_artist_url_iri(band.gid);

            # l_album_artist
            mo:composed mbz:composition_iri (band_album.gid) where (^{l_album_artist3.}^.link_type = 14)
            mo:performed mbz:performance_iri (band_album.gid) where (^{l_album_artist3.}^.link_type = 14)
            mo:performed mbz:performance_iri (band_album.gid) where (^{l_album_artist3.}^.link_type = 2)
            mo:conducted mbz:performance_iri (band_album.gid) where (^{l_album_artist3.}^.link_type = 9)
            mo:compiled mbz:record_iri (band_album.gid) where (^{l_album_artist3.}^.link_type = 41) optio
            mo:djmixed mbz:record_iri (band_album.gid) where (^{l_album_artist3.}^.link_type = 38) option
            mo:remixed mbz:record_iri (band_album.gid) where (^{l_album_artist3.}^.link_type = 11) option
            mo:sampled mbz:record_iri (band_album.gid) where (^{l_album_artist3.}^.link_type = 12) option
            mo:produced mbz:record_iri (band_album.gid) where (^{l_album_artist3.}^.link_type = 18) optio
            mo:published mbz:record_iri (band_album.gid) where (^{l_album_artist3.}^.link_type = 35) opti
            mo:engineered mbz:record_iri (band_album.gid) where (^{l_album_artist3.}^.link_type = 19) opt

#    #       mo:creatorOfRecord mbz:record_iri(band_album_creatorOf.gid);
```

```
            foaf:made mbz:record_iri(band_album_creatorOf.gid);

            # l_artist_track
            mo:composed mbz:composition_iri (band_track.gid) where (^{l_artist_track3.}^.link_type = 14)
            mo:performed mbz:performance_iri (band_track.gid) where (^{l_artist_track3.}^.link_type = 14)
            mo:performed mbz:performance_iri (band_track.gid) where (^{l_artist_track3.}^.link_type = 2)
            mo:conducted mbz:performance_iri (band_track.gid) where (^{l_artist_track3.}^.link_type = 9)
            mo:compiled mbz:record_iri (band_track.gid) where (^{l_artist_track3.}^.link_type = 39) optio
            mo:djmixed mbz:track_iri (band_track.gid) where (^{l_artist_track3.}^.link_type = 40) option
            mo:remixed mbz:track_iri (band_track.gid) where (^{l_artist_track3.}^.link_type = 11) option
            mo:sampled mbz:track_iri (band_track.gid) where (^{l_artist_track3.}^.link_type = 12) option
            mo:produced mbz:track_iri (band_track.gid) where (^{l_artist_track3.}^.link_type = 18) option
            mo:published mbz:track_iri (band_track.gid) where (^{l_artist_track3.}^.link_type = 35) optio
            mo:engineered mbz:track_iri (band_track.gid) where (^{l_artist_track3.}^.link_type = 19) opti

#     #       mo:creatorOfTrack mbz:track_iri(band_track_creatorOf.gid).

        # Music Group (Band)'s Birth Event
#       mbz:band_birth_event_iri(band.gid)
        mbz:artist_birth_event_iri(band.gid)
            a bio:Birth;
            bio:date band.begindate.

        # Music Group (Band)'s Death Event
#       mbz:band_death_event_iri(band.gid)
        mbz:artist_death_event_iri(band.gid)
            a bio:Death;
            bio:date band.enddate.

        # Similarity link
        #mbz:sim_link_iri(sim_band.gid)
        #    sim:relation mo:similar_to;
        #    sim:level band_relation.weight;
        #    sim:to sim_band.gid.

        # Music Artist
        mbz:artist_iri (artist.gid)

            # artist
            a mo:MusicArtist;
            a mo:SoloMusicArtist where (^{artist_untyped.}^.gid is not null) option (using artist_untyped
            a foaf:Person where (^{artist_untyped.}^.gid is not null) option (using artist_untyped);
            foaf:name artist.name;
            foaf:nick artistalias.name;
            bio:event mbz:artist_birth_event_iri(artist.gid);
            bio:event mbz:artist_death_event_iri(artist.gid);

            mo:member_of mbz:artist_iri(artist_artist.gid) where (^{l_artist_artist.}^.link_type = 2) opt

            # l_artist_artist
            rel:siblingOf mbz:artist_iri(artist_artist.gid) where (^{l_artist_artist.}^.link_type = 7) op
            rel:friendOf mbz:artist_iri(artist_artist.gid) where (^{l_artist_artist.}^.link_type = 5) opt
            rel:parentOf mbz:artist_iri(artist_artist.gid) where (^{l_artist_artist.}^.link_type = 6) opt
            rel:collaborated_with mbz:artist_iri(artist_artist.gid) where (^{l_artist_artist.}^.link_type
            rel:engagedTo mbz:artist_iri(artist_artist.gid) where (^{l_artist_artist.}^.link_type = 9) op
            rel:spouseOf mbz:artist_iri(artist_artist.gid) where (^{l_artist_artist.}^.link_type = 8) opt
            mo:supporting_musician mbz:artist_iri(artist_artist.gid) where (^{l_artist_artist.}^.link_typ
            mo:supporting_musician mbz:artist_iri(artist_artist.gid) where (^{l_artist_artist.}^.link_typ
            mo:supporting_musician mbz:artist_iri(artist_artist.gid) where (^{l_artist_artist.}^.link_typ

            mo:similar_to mbz:artist_iri(sim_artist.gid) option (using artist_relation);
#           mo:similar_to mbz:band_iri(sim_band.gid) option (using band_relation);
            mo:similar_to mbz:artist_iri(sim_band.gid) option (using band_relation);

#            sim:link mbz:sim_link_iri(sim_band.gid)  option (using band_relation);
#            sim:link mbz:sim_link_iri(sim_artist.gid)  option (using artist_relation);

            # l_artist_url
            mo:myspace mbz:url_iri(artist_url.url) where (^{l_artist_url.}^.link_type = 19) option (using
            mo:musicmoz mbz:url_iri(artist_url.url) where (^{l_artist_url.}^.link_type = 12) option (usin
            mo:discogs mbz:url_iri(artist_url.url) where (^{l_artist_url.}^.link_type = 11) option (using
            mo:wikipedia mbz:url_iri(artist_url.url) where (^{l_artist_url.}^.link_type = 10) option (usi
            mo:discography mbz:url_iri(artist_url.url) where (^{l_artist_url.}^.link_type = 1) option (us
```

```
            mo:freedownload mbz:url_iri(artist_url.url) where (^{l_artist_url.}^.link_type = 8) option (u
            mo:fanpage mbz:url_iri(artist_url.url) where (^{l_artist_url.}^.link_type = 3) option (using
            mo:biography mbz:url_iri(artist_url.url) where (^{l_artist_url.}^.link_type = 4) option (usin
            mo:discography mbz:url_iri(artist_url.url) where (^{l_artist_url.}^.link_type = 5) option (us
            mo:mailorder mbz:url_iri(artist_url.url) where (^{l_artist_url.}^.link_type = 15) option (usi
            mo:imdb mbz:url_iri(artist_url.url) where (^{l_artist_url.}^.link_type = 17) option (using l_
            mo:paiddownload mbz:url_iri(artist_url.url) where (^{l_artist_url.}^.link_type = 7) option (u
            foaf:depiction mbz:url_iri(artist_url.url) where (^{l_artist_url.}^.link_type = 14) option (u
            foaf:homepage mbz:url_iri(artist_url.url) where (^{l_artist_url.}^.link_type = 2) option (usi

            mo:musicbrainz mbz:mbz_artist_url_iri(artist.gid);

            # l_album_artist
            mo:composed mbz:composition_iri (artist_album.gid) where (^{l_album_artist.}^.link_type = 14)
            mo:performed mbz:performance_iri (artist_album.gid) where (^{l_album_artist.}^.link_type = 14
            mo:performed mbz:performance_iri (artist_album.gid) where (^{l_album_artist.}^.link_type = 2)
            mo:conducted mbz:performance_iri (artist_album.gid) where (^{l_album_artist.}^.link_type = 9)
            mo:compiled mbz:record_iri (artist_album.gid) where (^{l_album_artist.}^.link_type = 41) opti
            mo:djmixed mbz:record_iri (artist_album.gid) where (^{l_album_artist.}^.link_type = 38) optio
            mo:remixed mbz:record_iri (artist_album.gid) where (^{l_album_artist.}^.link_type = 11) optio
            mo:sampled mbz:record_iri (artist_album.gid) where (^{l_album_artist.}^.link_type = 12) optio
            mo:produced mbz:record_iri (artist_album.gid) where (^{l_album_artist.}^.link_type = 18) opti
            mo:published mbz:record_iri (artist_album.gid) where (^{l_album_artist.}^.link_type = 35) opt
            mo:engineered mbz:record_iri (artist_album.gid) where (^{l_album_artist.}^.link_type = 19) op

        #       mo:creatorOfRecord mbz:record_iri(artist_album_creatorOf.gid);
            foaf:made mbz:record_iri(artist_album_creatorOf.gid);

            # l_artist_track
            mo:composed mbz:composition_iri (artist_track.gid) where (^{l_artist_track.}^.link_type = 14)
            mo:performed mbz:performance_iri (artist_track.gid) where (^{l_artist_track.}^.link_type = 14
            mo:performed mbz:performance_iri (artist_track.gid) where (^{l_artist_track.}^.link_type = 2)
            mo:conducted mbz:performance_iri (artist_track.gid) where (^{l_artist_track.}^.link_type = 9)
            mo:compiled mbz:track_iri (artist_track.gid) where (^{l_artist_track.}^.link_type = 39) optio
            mo:djmixed mbz:track_iri (artist_track.gid) where (^{l_artist_track.}^.link_type = 40) option
            mo:remixed mbz:track_iri (artist_track.gid) where (^{l_artist_track.}^.link_type = 11) option
            mo:sampled mbz:track_iri (artist_track.gid) where (^{l_artist_track.}^.link_type = 12) option
            mo:produced mbz:track_iri (artist_track.gid) where (^{l_artist_track.}^.link_type = 18) optio
            mo:published mbz:track_iri (artist_track.gid) where (^{l_artist_track.}^.link_type = 35) opti
            mo:engineered mbz:track_iri (artist_track.gid) where (^{l_artist_track.}^.link_type = 19) opt

        #       mo:creatorOfTrack mbz:track_iri(artist_track_creatorOf.gid).

          # Music Artist''s Birth Event
          mbz:artist_birth_event_iri(artist.gid)
              a bio:Birth;
              bio:date artist.begindate.

          # Music Artist''s Death Event
          mbz:artist_death_event_iri(artist.gid)
              a bio:Death;
              bio:date artist.enddate.

          # Similarity link
          #mbz:sim_link_iri(sim_artist.gid)
          #    sim:relation mo:similar_to;
          #    sim:level artist_relation.weight;
          #    sim:to sim_artist.gid.

          }
      }
;

grant execute on ZITGIST.MO.RECORD_CREATION_DATE to "SPARQL";
grant execute on ZITGIST.MO.RECORD_CREATION_DATE_INVERSE to "SPARQL";
grant execute on ZITGIST.MO.RECORD_ATTRIBUTE_OFFICIAL to "SPARQL";
grant execute on ZITGIST.MO.RECORD_ATTRIBUTE_PROMOTION to "SPARQL";
grant execute on ZITGIST.MO.RECORD_ATTRIBUTE_BOOTLEG to "SPARQL";
grant execute on ZITGIST.MO.RECORD_ATTRIBUTE_ALBUM to "SPARQL";
grant execute on ZITGIST.MO.RECORD_ATTRIBUTE_SINGLE to "SPARQL";
grant execute on ZITGIST.MO.RECORD_ATTRIBUTE_EP to "SPARQL";
grant execute on ZITGIST.MO.RECORD_ATTRIBUTE_COMPILATION to "SPARQL";
grant execute on ZITGIST.MO.RECORD_ATTRIBUTE_SOUNDTRACK to "SPARQL";
```

```
grant execute on ZITGIST.MO.RECORD_ATTRIBUTE_SPOKENWORD to "SPARQL";
grant execute on ZITGIST.MO.RECORD_ATTRIBUTE_INTERVIEW to "SPARQL";
grant execute on ZITGIST.MO.RECORD_ATTRIBUTE_AUDIOBOOK to "SPARQL";
grant execute on ZITGIST.MO.RECORD_ATTRIBUTE_LIVE to "SPARQL";
grant execute on ZITGIST.MO.RECORD_ATTRIBUTE_REMIX to "SPARQL";
grant execute on ZITGIST.MO.TRACK_DURATION to "SPARQL";
grant execute on ZITGIST.MO.TRACK_DURATION_INVERSE to "SPARQL";
grant execute on ZITGIST.MO.album_amazon_asin to "SPARQL";
grant execute on ZITGIST.MO.album_name_WORDS to "SPARQL";
grant execute on ZITGIST.MO.albumjoin to "SPARQL";
grant execute on ZITGIST.MO.albummeta to "SPARQL";
grant execute on ZITGIST.MO.artist to "SPARQL";
grant execute on ZITGIST.MO.artist_name_WORDS to "SPARQL";
grant execute on ZITGIST.MO.artist_relation to "SPARQL";
grant execute on ZITGIST.MO.artistalias to "SPARQL";
grant execute on ZITGIST.MO.artistalias_name_WORDS to "SPARQL";
grant execute on ZITGIST.MO.country to "SPARQL";
grant execute on ZITGIST.MO.l_album_album to "SPARQL";
grant execute on ZITGIST.MO.l_album_artist to "SPARQL";
grant execute on ZITGIST.MO.l_album_url to "SPARQL";
grant execute on ZITGIST.MO.l_artist_artist to "SPARQL";
grant execute on ZITGIST.MO.l_artist_track to "SPARQL";
grant execute on ZITGIST.MO.l_artist_url to "SPARQL";
grant execute on ZITGIST.MO.l_track_track to "SPARQL";
grant execute on ZITGIST.MO.l_track_url to "SPARQL";
grant execute on ZITGIST.MO."language" to "SPARQL";
grant execute on ZITGIST.MO.puid to "SPARQL";
grant execute on ZITGIST.MO.puidjoin to "SPARQL";
grant execute on ZITGIST.MO.release to "SPARQL";
grant execute on ZITGIST.MO.track to "SPARQL";
grant execute on ZITGIST.MO.track_name_WORDS to "SPARQL";
grant execute on ZITGIST.MO.url to "SPARQL";

DB.DBA.XML_SET_NS_DECL ('mbz', 'http://musibrainz.org/schemas/mbz#', 2);
```

16.8.7. Virtuoso ODS to RDF

Linked Data View for ODS (the consolidated Graph)

```
SPARQL drop quad map virtrdf:ODS_DS . ;

SPARQL prefix ods: <http://www.openlinksw.com/virtuoso/ods/>
       create iri class ods:graph "http://^{URIQADefaultHost}^/dataspace/%U" (in uname varchar not null)
                        option (returns "http://^{URIQADefaultHost}^/dataspace/%U") .
       create iri class ods:user "http://^{URIQADefaultHost}^/dataspace/%U#user" (in uname varchar not nu
                        option (returns "http://^{URIQADefaultHost}^/dataspace/%U#user") .
       create iri class ods:user_group "http://^{URIQADefaultHost}^/dataspace/%U#group" (in uname varchar
                        option (returns "http://^{URIQADefaultHost}^/dataspace/%U#group") .
       create iri class ods:person "http://^{URIQADefaultHost}^/dataspace/%U#this" (in uname varchar not
                        option (returns "http://^{URIQADefaultHost}^/dataspace/%U#this") .
       create iri class ods:mbox "mailto:%s" (in email varchar not null)
                        option (returns "mailto:%s") .
       create iri class ods:phone "tel:%s" (in tel varchar not null)
                        option (returns "tel:%s") .
       create iri class ods:geo_point "http://^{URIQADefaultHost}^/dataspace/%U#geo" (in uname varchar no
                        option (returns "http://^{URIQADefaultHost}^/dataspace/%U#geo") .
       create iri class ods:forum "http://^{URIQADefaultHost}^/dataspace/%U/%U/%U"
            ( in uname varchar not null, in forum_type varchar not null, in forum_name varchar not null)
                        option (returns "http://^{URIQADefaultHost}^/dataspace/%U/%U/%U") .
       create iri class ods:proxy "http://^{URIQADefaultHost}^/proxy/%U" (in url varchar not null)
                        option (returns  "http://^{URIQADefaultHost}^/proxy/%U") .
       create iri class ods:site "http://^{URIQADefaultHost}^/dataspace/%U#site" (in uname varchar not nu
                        option (returns "http://^{URIQADefaultHost}^/dataspace/%U#site") .
       create iri class ods:role "http://^{URIQADefaultHost}^/dataspace/%U/%U/%U#%U"
            (in uname varchar not null, in tp varchar not null, in inst varchar not null, in role_name va
                        option (returns  "http://^{URIQADefaultHost}^/dataspace/%U/%U/%U#%U" ) .
        # Blog
        create iri class ods:blog_forum "http://^{URIQADefaultHost}^/dataspace/%U/weblog/%U"
                (in uname varchar not null, in forum_name varchar not null)
               option (returns "http://^{URIQADefaultHost}^/dataspace/%U/weblog/%U") .
        create iri class ods:blog_post "http://^{URIQADefaultHost}^/dataspace/%U/weblog/%U/%U"
                (in uname varchar not null, in forum_name varchar not null, in postid varchar not null)
```

```
        option (returns "http://^{URIQADefaultHost}^/dataspace/%U/weblog/%U/%U" ) .
create iri class ods:blog_comment "http://^{URIQADefaultHost}^/dataspace/%U/weblog/%U/%U/%d"
        (in uname varchar not null, in forum_name varchar not null, in postid varchar not null, i
        option (returns  "http://^{URIQADefaultHost}^/dataspace/%U/weblog/%U/%U/%d" ) .
create iri class ods:tag "http://^{URIQADefaultHost}^/dataspace/%U/concept#%U"
        (in uname varchar not null, in tag varchar not null)
        option (returns  "http://^{URIQADefaultHost}^/dataspace/%U/concept#%U") .
create iri class ods:blog_post_text "http://^{URIQADefaultHost}^/dataspace/%U/weblog-text/%U/%U"
        (in uname varchar not null, in forum_name varchar not null, in postid varchar not null)
        option (returns "http://^{URIQADefaultHost}^/dataspace/%U/weblog-text/%U/%U" ) .
#Feeds
create iri class ods:feed "http://^{URIQADefaultHost}^/dataspace/feed/%d" (in feed_id integer not
        option (returns "http://^{URIQADefaultHost}^/dataspace/feed/%d" ) .
create iri class ods:feed_item "http://^{URIQADefaultHost}^/dataspace/feed/%d/%d" (in feed_id int
        option (returns  "http://^{URIQADefaultHost}^/dataspace/feed/%d/%d" ) .
create iri class ods:feed_item_text "http://^{URIQADefaultHost}^/dataspace/feed/%d/%d/text" (in f
        option (returns  "http://^{URIQADefaultHost}^/dataspace/feed/%d/%d/text" ) .
create iri class ods:feed_mgr "http://^{URIQADefaultHost}^/dataspace/%U/feeds/%U" (in uname varch
        option (returns "http://^{URIQADefaultHost}^/dataspace/%U/feeds/%U" ) .
create iri class ods:feed_comment "http://^{URIQADefaultHost}^/dataspace/%U/feeds/%U/%d/%d"
        (in uname varchar not null, in inst_name varchar not null, in item_id integer not null, i
        option (returns "http://^{URIQADefaultHost}^/dataspace/%U/feeds/%U/%d/%d" ) .
#Bookmark
create iri class ods:bmk_post "http://^{URIQADefaultHost}^/dataspace/%U/bookmark/%U/%d"
        (in uname varchar not null, in inst_name varchar not null, in bmk_id integer not null)
        option (returns "http://^{URIQADefaultHost}^/dataspace/%U/bookmark/%U/%d") .
create iri class ods:bmk_post_text "http://^{URIQADefaultHost}^/dataspace/%U/bookmark/%U/%d/text"
        (in uname varchar not null, in inst_name varchar not null, in bmk_id integer not null)
        option (returns "http://^{URIQADefaultHost}^/dataspace/%U/bookmark/%U/%d/text" ) .
create iri class ods:bmk_forum "http://^{URIQADefaultHost}^/dataspace/%U/bookmark/%U"
        ( in uname varchar not null, in forum_name varchar not null)
        option (returns "http://^{URIQADefaultHost}^/dataspace/%U/bookmark/%U") .
#Photo
create iri class ods:photo_forum "http://^{URIQADefaultHost}^/dataspace/%U/photos/%U"
        (in uname varchar not null, in inst_name varchar not null)
        option (returns  "http://^{URIQADefaultHost}^/dataspace/%U/photos/%U") .
create iri class ods:photo_post "http://^{URIQADefaultHost}^%s"
        (in path varchar not null) option (returns "http://^{URIQADefaultHost}^/DAV/%s") .
create iri class ods:photo_post_text "http://^{URIQADefaultHost}^%s/text"
        (in path varchar not null) option (returns "http://^{URIQADefaultHost}^/DAV/%s/text") .
create iri class ods:photo_comment "http://^{URIQADefaultHost}^%s:comment_%d"
        (in path varchar not null, in comment_id int not null)
        option (returns "http://^{URIQADefaultHost}^/DAV/%s:comment_%d") .
# Community
create iri class ods:community_forum "http://^{URIQADefaultHost}^/dataspace/%U/community/%U"
        (in uname varchar not null, in forum_name varchar not null)
        option (returns  "http://^{URIQADefaultHost}^/dataspace/%U/community/%U") .
# Briefcase
create iri class ods:odrive_forum "http://^{URIQADefaultHost}^/dataspace/%U/briefcase/%U"
        (in uname varchar not null, in inst_name varchar not null)
        option (returns "http://^{URIQADefaultHost}^/dataspace/%U/briefcase/%U" ) .
create iri class ods:odrive_post "http://^{URIQADefaultHost}^%s"
        (in path varchar not null) option (returns "http://^{URIQADefaultHost}^/DAV/%s") .
create iri class ods:odrive_post_text "http://^{URIQADefaultHost}^%s/text"
        (in path varchar not null) option (returns "http://^{URIQADefaultHost}^/DAV/%s/text") .
# Wiki
create iri class ods:wiki_post "http://^{URIQADefaultHost}^/dataspace/%U/wiki/%U/%U"
        (in uname varchar not null, in inst_name varchar not null, in topic_id varchar not null)
        option (returns "http://^{URIQADefaultHost}^/dataspace/%U/wiki/%U/%U") .
create iri class ods:wiki_post_text "http://^{URIQADefaultHost}^/dataspace/%U/wiki/%U/%U/text"
        (in uname varchar not null, in inst_name varchar not null, in topic_id varchar not null)
        option (returns "http://^{URIQADefaultHost}^/dataspace/%U/wiki/%U/%U/text" ) .
create iri class ods:wiki_forum "http://^{URIQADefaultHost}^/dataspace/%U/wiki/%U"
        ( in uname varchar not null, in forum_name varchar not null)
        option (returns  "http://^{URIQADefaultHost}^/dataspace/%U/wiki/%U" ) .
#Calendar
create iri class ods:calendar_event "http://^{URIQADefaultHost}^/dataspace/%U/calendar/%U/%d"
        (in uname varchar not null, in inst_name varchar not null, in calendar_id integer not nul
        option (returns "http://^{URIQADefaultHost}^/dataspace/%U/calendar/%U/%d" ) .
create iri class ods:calendar_event_text "http://^{URIQADefaultHost}^/dataspace/%U/calendar/%U/%d
        (in uname varchar not null, in inst_name varchar not null, in calendar_id integer not nul
        option (returns "http://^{URIQADefaultHost}^/dataspace/%U/calendar/%U/%d/text" ) .
create iri class ods:calendar_forum "http://^{URIQADefaultHost}^/dataspace/%U/calendar/%U"
```

```
                ( in uname varchar not null, in forum_name varchar not null)
                option (returns  "http://^{URIQADefaultHost}^/dataspace/%U/calendar/%U") .
          # NNTPF
          create iri class ods:nntp_forum "http://^{URIQADefaultHost}^/dataspace/discussion/%U"
                ( in forum_name varchar not null)
                option (returns "http://^{URIQADefaultHost}^/dataspace/discussion/%U").
          create iri class ods:nntp_post "http://^{URIQADefaultHost}^/dataspace/discussion/%U/%U"
                ( in group_name varchar not null, in message_id varchar not null)
                option (returns  "http://^{URIQADefaultHost}^/dataspace/discussion/%U/%U" ) .
          create iri class ods:nntp_post_text "http://^{URIQADefaultHost}^/dataspace/discussion/%U/%U/text"
                ( in group_name varchar not null, in message_id varchar not null)
                option (returns  "http://^{URIQADefaultHost}^/dataspace/discussion/%U/%U/text") .
          create iri class ods:nntp_role "http://^{URIQADefaultHost}^/dataspace/discussion/%U#reader"
                (in forum_name varchar not null)
                option (returns "http://^{URIQADefaultHost}^/dataspace/discussion/%U#reader") .
        ;

SPARQL
    prefix sioc: <http://rdfs.org/sioc/ns#>
    prefix sioct: <http://rdfs.org/sioc/types#>
    prefix atom: <http://atomowl.org/ontologies/atomrdf#>
    prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    prefix foaf: <http://xmlns.com/foaf/0.1/>
    prefix dc: <http://purl.org/dc/elements/1.1/>
    prefix dct: <http://purl.org/dc/terms/>
    prefix skos: <http://www.w3.org/2004/02/skos/core#>
    prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
    prefix bm: <http://www.w3.org/2002/01/bookmark#>
    prefix exif: <http://www.w3.org/2003/12/exif/ns/>
    prefix ann: <http://www.w3.org/2000/10/annotation-ns#>
    prefix wikiont: <http://sw.deri.org/2005/04/wikipedia/wikiont.owl#>
    prefix calendar: <http://www.w3.org/2002/12/cal#>
    prefix ods: <http://www.openlinksw.com/virtuoso/ods/>

    alter quad storage virtrdf:DefaultQuadStorage
      FROM DB.DBA.SIOC_USERS as users
      FROM DB.DBA.SIOC_ODS_FORUMS as forums
      FROM DB.DBA.SIOC_ROLES as roles
      FROM DB.DBA.SIOC_ROLE_GRANTS as grants
      FROM DB.DBA.SIOC_KNOWS as knows
      FROM DB.DBA.ODS_FOAF_PERSON as person
      where (^{person.}^.U_NAME = ^{users.}^.U_NAME)
      where (^{forums.}^.U_NAME = ^{users.}^.U_NAME)
      where (^{knows.}^.TO_NAME = ^{users.}^.U_NAME)
      where (^{knows.}^.FROM_NAME = ^{users.}^.U_NAME)
      where (^{grants.}^.U_NAME = ^{users.}^.U_NAME)
      where (^{roles.}^.U_NAME = ^{users.}^.U_NAME)

      FROM DB.DBA.ODS_BLOG_POSTS as blog_posts
      where (^{blog_posts.}^.B_OWNER = ^{users.}^.U_NAME)
      FROM DB.DBA.ODS_BLOG_POST_LINKS as blog_links
      where (^{blog_links.}^.B_OWNER = ^{users.}^.U_NAME)
      FROM DB.DBA.ODS_BLOG_POST_ATTS as blog_atts
      where (^{blog_atts.}^.B_OWNER = ^{users.}^.U_NAME)
      FROM DB.DBA.ODS_BLOG_POST_TAGS as blog_tags
      where (^{blog_tags.}^.U_NAME = ^{users.}^.U_NAME)
      FROM DB.DBA.ODS_BLOG_COMMENTS as blog_comms
      where (^{blog_comms.}^.U_NAME = ^{users.}^.U_NAME)

      FROM DB.DBA.ODS_BMK_POSTS as bmk_posts
      where (^{bmk_posts.}^.U_NAME = ^{users.}^.U_NAME)
      FROM DB.DBA.ODS_BMK_TAGS as bmk_tags
      where (^{bmk_tags.}^.U_NAME = ^{users.}^.U_NAME)

      FROM DB.DBA.ODS_ODRIVE_POSTS as odrv_posts
      where (^{odrv_posts.}^.U_MEMBER = ^{users.}^.U_NAME)
      FROM DB.DBA.ODS_ODRIVE_TAGS as odrv_tags
      where (^{odrv_tags.}^.U_OWNER = ^{users.}^.U_NAME)

      FROM DB.DBA.ODS_FEED_FEED_DOMAIN as feed_domain
      where (^{feed_domain.}^.U_NAME = ^{users.}^.U_NAME)
      FROM DB.DBA.ODS_FEED_POSTS as feed_posts
      where (^{feed_posts.}^.EFI_FEED_ID = ^{feed_domain.}^.EF_ID)
```

```
  FROM DB.DBA.ODS_FEED_COMMENTS as feed_comments
  where (^{feed_comments.}^.U_NAME = ^{users.}^.U_NAME)
  FROM DB.DBA.ODS_FEED_TAGS as feed_tags
  where (^{feed_tags.}^.U_NAME = ^{users.}^.U_NAME)
  FROM DB.DBA.ODS_FEED_LINKS as feed_links
  where (^{feed_links.}^.EFI_FEED_ID = ^{feed_domain.}^.EF_ID)
  FROM DB.DBA.ODS_FEED_ATTS as feed_atts
  where (^{feed_atts.}^.EFI_FEED_ID = ^{feed_domain.}^.EF_ID)

  FROM DB.DBA.ODS_PHOTO_POSTS as photo_posts
  where (^{photo_posts.}^.U_MEMBER = ^{users.}^.U_NAME)
  FROM DB.DBA.ODS_PHOTO_COMMENTS as photo_comments
  where (^{photo_comments.}^.U_MEMBER = ^{users.}^.U_NAME)
  FROM DB.DBA.ODS_PHOTO_TAGS as photo_tags
  where (^{photo_tags.}^.U_MEMBER = ^{users.}^.U_NAME)

  FROM DB.DBA.ODS_WIKI_POSTS as wiki_posts
  where (^{wiki_posts.}^.U_NAME = ^{users.}^.U_NAME)

  FROM DB.DBA.ODS_COMMUNITIES as community
  where (^{community.}^.C_OWNER = ^{users.}^.U_NAME)

  FROM DB.DBA.ODS_NNTP_GROUPS as nntp_groups
  FROM DB.DBA.ODS_NNTP_POSTS as nntp_posts
  FROM DB.DBA.ODS_NNTP_USERS as nntp_users
  where (^{nntp_users.}^.U_NAME = ^{users.}^.U_NAME)
  FROM DB.DBA.ODS_NNTP_LINKS as nntp_links

{
    create virtrdf:ODS_DS as graph ods:graph (users.U_NAME) option (exclusive)
      {
        ods:user (users.U_NAME) a sioc:User ;
        sioc:id users.U_NAME ;
        sioc:name users.U_FULL_NAME ;
        sioc:email ods:mbox (users.E_MAIL) ;
        sioc:email_sha1 users.E_MAIL_SHA1 ;
        sioc:account_of ods:person (users.U_NAME) .

        ods:person (person.U_NAME) a foaf:Person ;
        foaf:nick person.U_NAME ;
        foaf:name person.U_FULL_NAME ;
        foaf:mbox ods:mbox (person.E_MAIL) ;
        foaf:mbox_sha1sum person.E_MAIL_SHA1 ;
        foaf:holdsAccount ods:user (person.U_NAME) ;
        foaf:firstName person.FIRST_NAME ;
        foaf:family_name person.LAST_NAME ;
        foaf:gender person.GENDER ;
        foaf:icqChatID person.ICQ ;
        foaf:msnChatID person.MSN ;
        foaf:aimChatID person.AIM ;
        foaf:yahooChatID person.YAHOO ;
        foaf:birthday person.BIRTHDAY ;
        foaf:organization person.ORG ;
        foaf:phone ods:phone (person.PHONE) ;
        foaf:based_near ods:geo_point (person.U_NAME)
        .

        ods:geo_point (person.U_NAME) a geo:Point ;
             geo:lat person.LAT ;
             geo:lng person.LNG .

        ods:person (knows.FROM_NAME) foaf:knows ods:person (knows.TO_NAME) .
        ods:person (knows.TO_NAME) foaf:knows ods:person (knows.FROM_NAME) .

        ods:user_group (grants.G_NAME) a sioc:Usergroup ;
            sioc:id grants.G_NAME ;
            sioc:has_member ods:user (grants.U_NAME) .
        ods:user (grants.U_NAME)  sioc:member_of ods:user_group (grants.G_NAME) .

        ods:role (roles.U_NAME, roles.APP_TYPE, roles.WAM_INST, roles.WMT_NAME)
          sioc:has_scope ods:forum (roles.U_NAME, roles.APP_TYPE, roles.WAM_INST) ;
          sioc:function_of ods:user (roles.U_NAME) .
```

```
ods:forum (roles.U_NAME, roles.APP_TYPE, roles.WAM_INST)
    sioc:scope_of ods:role (roles.U_NAME, roles.APP_TYPE, roles.WAM_INST, roles.WMT_NAME) .
ods:user (roles.U_NAME)
    sioc:has_function ods:role (roles.U_NAME, roles.APP_TYPE, roles.WAM_INST, roles.WMT_NAME)

ods:forum (forums.U_NAME, forums.APP_TYPE, forums.WAM_INST) a sioc:Container ;
sioc:id forums.WAM_INST ;
sioc:type forums.APP_TYPE ;
sioc:description forums.WAI_DESCRIPTION ;
sioc:link ods:proxy (forums.LINK) ;
sioc:has_space ods:site (forums.U_NAME) .

# Weblog
ods:blog_post (blog_posts.B_OWNER, blog_posts.B_INST, blog_posts.B_POST_ID) a sioct:BlogPost
sioc:link ods:proxy (blog_posts.B_LINK) ;
sioc:has_creator ods:user (blog_posts.B_CREATOR) ;
foaf:maker ods:person (blog_posts.B_CREATOR) ;
sioc:has_container ods:blog_forum (blog_posts.B_OWNER, blog_posts.B_INST) ;
dc:title blog_posts.B_TITLE ;
dct:created blog_posts.B_CREATED ;
dct:modified blog_posts.B_MODIFIED ;
sioc:content blog_posts.B_CONTENT .

ods:blog_forum (blog_posts.B_OWNER, blog_posts.B_INST)
sioc:container_of
ods:blog_post (blog_posts.B_OWNER, blog_posts.B_INST, blog_posts.B_POST_ID) .

ods:user (blog_posts.B_CREATOR)
sioc:creator_of
ods:blog_post (blog_posts.B_OWNER, blog_posts.B_INST, blog_posts.B_POST_ID) .

ods:blog_post (blog_links.B_OWNER, blog_links.B_INST, blog_links.B_POST_ID)
sioc:links_to
ods:proxy (blog_links.PL_LINK) .
# end Weblog

# Bookmark
ods:bmk_post (bmk_posts.U_NAME, bmk_posts.WAI_NAME, bmk_posts.BD_BOOKMARK_ID)
a bm:Bookmark ;
dc:title bmk_posts.BD_NAME;
dct:created bmk_posts.BD_CREATED ;
dct:modified bmk_posts.BD_LAST_UPDATE ;
dc:date bmk_posts.BD_LAST_UPDATE ;
ann:created bmk_posts.BD_CREATED ;
dc:creator bmk_posts.U_NAME ;
bm:recalls ods:proxy (bmk_posts.B_URI) ;
sioc:link ods:proxy (bmk_posts.B_URI) ;
sioc:content bmk_posts.BD_DESCRIPTION ;
sioc:has_creator ods:user (bmk_posts.U_NAME) ;
foaf:maker ods:person (bmk_posts.U_NAME) ;
sioc:has_container ods:bmk_forum (bmk_posts.U_NAME, bmk_posts.WAI_NAME) .

ods:bmk_forum (bmk_posts.U_NAME, bmk_posts.WAI_NAME)
sioc:container_of
ods:bmk_post (bmk_posts.U_NAME, bmk_posts.WAI_NAME, bmk_posts.BD_BOOKMARK_ID) .

ods:user (bmk_posts.U_NAME)
sioc:creator_of
ods:bmk_post (bmk_posts.U_NAME, bmk_posts.WAI_NAME, bmk_posts.BD_BOOKMARK_ID) .

ods:bmk_post (bmk_tags.U_NAME, bmk_tags.WAM_INST, bmk_tags.ITEM_ID)
sioc:topic
ods:tag (bmk_tags.U_NAME, bmk_tags.BD_TAG) .

ods:tag (bmk_tags.U_NAME, bmk_tags.BD_TAG) a skos:Concept ;
skos:prefLabel bmk_tags.BD_TAG ;
skos:isSubjectOf ods:bmk_post (bmk_tags.U_NAME, bmk_tags.WAM_INST, bmk_tags.ITEM_ID) .
# end Bookmark

# Briefcase
ods:odrive_post (odrv_posts.RES_FULL_PATH) a foaf:Document ;
dc:title odrv_posts.RES_NAME ;
dct:created odrv_posts.RES_CREATED ;
```

```
dct:modified odrv_posts.RES_MODIFIED ;
sioc:content odrv_posts.RES_DESCRIPTION ;
sioc:has_creator ods:user (odrv_posts.U_OWNER) ;
foaf:maker ods:person (odrv_posts.U_OWNER) ;
sioc:has_container ods:odrive_forum (odrv_posts.U_MEMBER, odrv_posts.WAI_NAME) .

ods:odrive_forum (odrv_posts.U_MEMBER, odrv_posts.WAI_NAME)
sioc:container_of
ods:odrive_post (odrv_posts.RES_FULL_PATH) .

ods:user (odrv_posts.U_OWNER)
sioc:creator_of
ods:odrive_post (odrv_posts.RES_FULL_PATH) .

ods:odrive_post (odrv_tags.RES_FULL_PATH)
sioc:topic
ods:tag (odrv_tags.U_OWNER, odrv_tags.TAG) .

ods:tag (odrv_tags.U_OWNER, odrv_tags.TAG) a skos:Concept ;
skos:prefLabel odrv_tags.TAG ;
skos:isSubjectOf ods:odrive_post (odrv_tags.RES_FULL_PATH) .
# end Briefcase

# Feeds
ods:feed (feed_domain.EF_ID) a atom:Feed ;
sioc:link ods:proxy (feed_domain.EF_URI) ;
atom:link ods:proxy (feed_domain.EF_URI) ;
atom:title feed_domain.EF_TITLE ;
sioc:has_parent ods:feed_mgr (feed_domain.U_NAME, feed_domain.WAI_NAME) .

ods:feed_mgr (feed_domain.U_NAME, feed_domain.WAI_NAME)
sioc:parent_of ods:feed (feed_domain.EF_ID) .

ods:feed_item (feed_tags.EFI_FEED_ID, feed_tags.EFID_ITEM_ID)
sioc:topic
ods:tag (feed_tags.U_NAME, feed_tags.EFID_TAG) .

ods:tag (feed_tags.U_NAME, feed_tags.EFID_TAG) a skos:Concept ;
skos:prefLabel feed_tags.EFID_TAG ;
skos:isSubjectOf ods:feed_item (feed_tags.EFI_FEED_ID, feed_tags.EFID_ITEM_ID) .

ods:feed_comment (feed_comments.U_NAME, feed_comments.WAI_NAME, feed_comments.EFIC_ITEM_ID, f
a sioct:Comment ;
dc:title feed_comments.EFIC_TITLE ;
sioc:content feed_comments.EFIC_COMMENT ;
dct:modified feed_comments.LAST_UPDATE ;
dct:created feed_comments.LAST_UPDATE ;
sioc:link ods:proxy (feed_comments.LINK) ;
sioc:has_container ods:feed (feed_comments.EFI_FEED_ID) ;
sioc:reply_of ods:feed_item (feed_comments.EFI_FEED_ID, feed_comments.EFIC_ITEM_ID) ;
foaf:maker ods:proxy (feed_comments.EFIC_U_URL) .

ods:proxy (feed_comments.EFIC_U_URL) a foaf:Person ;
foaf:name feed_comments.EFIC_U_NAME;
foaf:mbox ods:mbox (feed_comments.EFIC_U_MAIL) .

ods:feed (feed_comments.EFI_FEED_ID)
sioc:container_of
ods:feed_comment (feed_comments.U_NAME, feed_comments.WAI_NAME, feed_comments.EFIC_ITEM_ID, f

ods:feed_item (feed_comments.EFI_FEED_ID, feed_comments.EFIC_ITEM_ID)
sioc:has_reply
ods:feed_comment (feed_comments.U_NAME, feed_comments.WAI_NAME, feed_comments.EFIC_ITEM_ID, f

ods:feed_item (feed_links.EFI_FEED_ID, feed_links.EFI_ID)
sioc:links_to
ods:proxy (feed_links.EFIL_LINK) .

ods:feed_item (feed_atts.EFI_FEED_ID, feed_atts.EFI_ID)
sioc:attachment
ods:proxy (feed_atts.EFIE_URL) .

ods:feed_item (feed_posts.EFI_FEED_ID, feed_posts.EFI_ID) a atom:Entry ;
```

```
sioc:has_container ods:feed (feed_posts.EFI_FEED_ID) ;
dc:title feed_posts.EFI_TITLE ;
dct:created feed_posts.PUBLISH_DATE ;
dct:modified feed_posts.PUBLISH_DATE ;
sioc:link ods:proxy (feed_posts.EFI_LINK) ;
sioc:content feed_posts.EFI_DESCRIPTION ;
atom:title feed_posts.EFI_TITLE ;
atom:source ods:feed (feed_posts.EFI_FEED_ID) ;
atom:published feed_posts.PUBLISH_DATE ;
atom:updated feed_posts.PUBLISH_DATE ;
atom:content ods:feed_item_text (feed_posts.EFI_FEED_ID, feed_posts.EFI_ID) .

ods:feed (feed_posts.EFI_FEED_ID) sioc:container_of ods:feed_item (feed_posts.EFI_FEED_ID, fe

ods:feed_item_text (feed_posts.EFI_FEED_ID, feed_posts.EFI_ID) a atom:Content ;
atom:type "text/xhtml" ;
atom:lang "en-US" ;
atom:body feed_posts.EFI_DESCRIPTION .

ods:feed (feed_posts.EFI_FEED_ID)
atom:contains
ods:feed_item (feed_posts.EFI_FEED_ID, feed_posts.EFI_ID) .
# end Feeds

# Photo
ods:photo_post (photo_posts.RES_FULL_PATH) a exif:IFD ;
dc:title photo_posts.RES_NAME ;
dct:created photo_posts.RES_CREATED ;
dct:modified photo_posts.RES_MODIFIED ;
sioc:content photo_posts.RES_DESCRIPTION ;
sioc:has_creator ods:user (photo_posts.U_OWNER) ;
foaf:maker ods:person (photo_posts.U_OWNER) ;
sioc:link ods:proxy (photo_posts.RES_LINK) ;
sioc:has_container ods:photo_forum (photo_posts.U_MEMBER, photo_posts.WAI_NAME) .

ods:photo_forum (photo_posts.U_MEMBER, photo_posts.WAI_NAME)
sioc:container_of
ods:photo_post (photo_posts.RES_FULL_PATH) .

ods:user (photo_posts.U_OWNER)
sioc:creator_of
ods:photo_post (photo_posts.RES_FULL_PATH) .

ods:photo_post (photo_tags.RES_FULL_PATH)
sioc:topic
ods:tag (photo_tags.U_MEMBER, photo_tags.RES_TAG) .

ods:tag (photo_tags.U_MEMBER, photo_tags.RES_TAG) a skos:Concept ;
skos:prefLabel photo_tags.RES_TAG ;
skos:isSubjectOf ods:photo_post (photo_tags.RES_FULL_PATH) .

ods:photo_comment (photo_comments.RES_FULL_PATH, photo_comments.COMMENT_ID) a sioct:Comment ;
sioc:reply_of ods:photo_post (photo_comments.RES_FULL_PATH) ;
sioc:has_container ods:photo_forum (photo_comments.U_MEMBER, photo_comments.WAI_NAME) ;
dc:title photo_comments.RES_NAME ;
dct:created photo_comments.CREATE_DATE ;
dct:modified photo_comments.MODIFY_DATE ;
sioc:content photo_comments.TEXT ;
foaf:maker ods:person (photo_comments.U_MAKER) .

ods:photo_post (photo_comments.RES_FULL_PATH)
sioc:has_reply
ods:photo_comment (photo_comments.RES_FULL_PATH, photo_comments.COMMENT_ID) .
# end Photo

# Polls
# end Polls

# Mail
# end Mail

# Wiki
ods:wiki_post (wiki_posts.U_NAME, wiki_posts.CLUSTERNAME, wiki_posts.LOCALNAME) a wikiont:Art
```

```
                dc:title wiki_posts.LOCALNAME ;
                dct:created wiki_posts.RES_CREATED ;
                dct:modified wiki_posts.RES_MODIFIED ;
                sioc:content wiki_posts.RES_CONTENT ;
                sioc:has_creator ods:user (wiki_posts.U_NAME) ;
                foaf:maker ods:person (wiki_posts.U_NAME) ;
                sioc:has_container ods:wiki_forum (wiki_posts.U_NAME, wiki_posts.CLUSTERNAME) .

                ods:wiki_forum (wiki_posts.U_NAME, wiki_posts.CLUSTERNAME)
                sioc:container_of
                ods:wiki_post (wiki_posts.U_NAME, wiki_posts.CLUSTERNAME, wiki_posts.LOCALNAME) .

                ods:user (wiki_posts.U_NAME)
                sioc:creator_of
                ods:wiki_post (wiki_posts.U_NAME, wiki_posts.CLUSTERNAME, wiki_posts.LOCALNAME) .

                # end Wiki

                # Community
                ods:community_forum (community.C_OWNER, community.CM_COMMUNITY_ID) a sioc:Community ;
                sioc:has_part ods:forum (community.A_OWNER, community.A_TYPE, community.CM_MEMBER_APP) .

                ods:forum (community.A_OWNER, community.A_TYPE, community.CM_MEMBER_APP)
                sioc:part_of
                ods:community_forum (community.C_OWNER, community.CM_COMMUNITY_ID) .
                # end Community

                # NNTP
                ods:nntp_forum (nntp_groups.NG_NAME) a sioct:MessageBoard ;
                sioc:id nntp_groups.NG_NAME ;
                sioc:description nntp_groups.NG_DESC .

                ods:nntp_post (nntp_posts.NG_NAME, nntp_posts.NM_ID) a sioct:BoardPost ;
                sioc:content nntp_posts.NM_BODY ;
                dc:title nntp_posts.FTHR_SUBJ ;
                dct:created  nntp_posts.REC_DATE ;
                dct:modified nntp_posts.REC_DATE ;
                foaf:maker ods:proxy (nntp_posts.MAKER) ;
                sioc:reply_of ods:nntp_post (nntp_posts.NG_NAME, nntp_posts.FTHR_REFER) ;
                sioc:has_container ods:nntp_forum (nntp_posts.NG_NAME) .

                ods:nntp_post (nntp_posts.NG_NAME, nntp_posts.FTHR_REFER)
                sioc:has_reply
                ods:nntp_post (nntp_posts.NG_NAME, nntp_posts.NM_ID) .

                ods:nntp_forum (nntp_posts.NG_NAME)
                sioc:container_of
                ods:nntp_post (nntp_posts.NG_NAME, nntp_posts.NM_ID) .

                ods:nntp_role (nntp_groups.NG_NAME)
                sioc:has_scope
                ods:nntp_forum (nntp_groups.NG_NAME) .

                ods:nntp_forum (nntp_groups.NG_NAME)
                sioc:scope_of
                ods:nntp_role (nntp_groups.NG_NAME) .

                ods:user (nntp_users.U_NAME)
                sioc:has_function
                ods:nntp_role (nntp_users.NG_NAME) .

                ods:nntp_role (nntp_users.NG_NAME)
                sioc:function_of
                ods:user (nntp_users.U_NAME) .

                ods:nntp_post (nntp_links.NG_NAME, nntp_links.NML_MSG_ID)
                sioc:links_to
                ods:proxy (nntp_links.NML_URL) .
                # end NNTP

            } .
        } .
    ;
```

```
create procedure DB.DBA.URL_REW_ODS_ACCEPT ()
{
  declare accept, ret any;
  accept := http_request_header (http_request_header (), 'Accept');
  if (not isstring (accept))
    return null;
  ret := null;
  if (regexp_match ('(application|text)/rdf.(xml|n3|turtle|ttl)', accept) is not null)
    {
      if (regexp_match ('application/rdf.xml', accept) is not null)
        {
          ret := 'rdf';
        }
      else if (regexp_match ('text/rdf.n3', accept) is not null)
        {
          ret := 'n3';
        }
      else if (regexp_match ('application/rdf.turtle', accept) is not null or
            regexp_match ('application/rdf.ttl', accept) is not null)
        {
          ret := 'n3';
        }
    }
  return ret;
};

create procedure  DB.DBA.URL_REW_ODS_SPQ (in graph varchar, in iri varchar, in acc varchar)
{
  declare q, ret any;
  iri := replace (iri, '''', '%27');
  iri := replace (iri, '<', '%3C');
  iri := replace (iri, '>', '%3E');
  q := sprintf ('define input:inference <%s> DESCRIBE <%s> FROM <%s>', graph, iri, graph);
  ret := sprintf ('/sparql?query=%U&format=%U', q, acc);
  return ret;
};

create procedure DB.DBA.URL_REW_ODS_USER (in par varchar, in fmt varchar, in val varchar)
{
--  dbg_obj_print (current_proc_name ());
  declare acc, ret any;
  declare q, iri, graph any;

  acc := DB.DBA.URL_REW_ODS_ACCEPT ();
  if (acc is not null)
    {
      graph := sioc..get_graph ();
      iri := sprintf ('%s/%U', graph, val);
      if (val like 'person/%')
        {
          val := substring (val, 8, length (val));
          ret := sprintf ('/ods/foaf.vsp?uname=%U&fmt=%U', val, acc);
        }
      else
        ret := DB.DBA.URL_REW_ODS_SPQ (graph, iri, acc);
    }
  else
    {
      http_header (http_header_get ()||sprintf ('X-XRDS-Location: %s\r\n',
            DB.DBA.wa_link (1, '/dataspace/'||val||'/yadis.xrds')));

      if (val like 'person/%')
        val := substring (val, 8, length (val));
      ret := sprintf ('/ods/uhome.vspx?page=1&ufname=%s', val);
    }
  return ret;
};

create procedure DB.DBA.URL_REW_ODS_USER_GEM (in par varchar, in fmt varchar, in val varchar)
{
--  dbg_obj_print (current_proc_name ());
```

```
  declare acc, ret any;
  declare q, iri, graph, path, is_person any;

  path := http_path ();
  if (path like '%.rdf')
    acc := 'rdf';
  else if (path like '%.n3')
    acc := 'n3';
  else if (path like '%.ttl')
    acc := 'n3';
  else if (path like '%/yadis.xrds')
    acc := 'yadis';
  else
    acc := 'rdf';

  if (acc <> 'yadis')
    {
      is_person := matches_like (path, '%/about.%');
      graph := sioc..get_graph ();
      if (is_person)
        {
          --iri := sprintf ('%s/person/%U', graph, val);
          ret := sprintf ('/ods/foaf.vsp?uname=%U&fmt=%U', val, acc);
        }
      else
        {
          iri := sprintf ('%s/%U', graph, val);
          ret := DB.DBA.URL_REW_ODS_SPQ (graph, iri, acc);
        }
    }
  else
    {
      ret := sprintf ('/ods/yadis.vsp?uname=%U', val);
    }
  return ret;
};

create procedure DB.DBA.URL_REW_ODS_GEM (in par varchar, in fmt varchar, in val varchar)
{
--  dbg_obj_print (current_proc_name ());
  declare acc, ret any;
  declare q, iri, graph, path, pos any;

  path := http_path ();
  if (path like '%.rdf')
    acc := 'rdf';
  else if (path like '%.n3')
    acc := 'n3';
  else if (path like '%.ttl')
    acc := 'n3';
  else
    acc := 'rdf';
  graph := sioc..get_graph ();
  pos := strrchr (path, '/');
  path := subseq (path, 0, pos);

  if (val = 'person')
    {
      pos := strrchr (path, '/');
      val := subseq (path, pos+1, length (path));
      ret := sprintf ('/ods/foaf.vsp?uname=%U&fmt=%U', val, acc);
    }
  else
    {
      iri := sprintf ('http://%s%s', sioc..get_cname (), path);
      ret := DB.DBA.URL_REW_ODS_SPQ (graph, iri, acc);
    }
  return ret;
};

create procedure DB.DBA.URL_REW_ODS_APP (in par varchar, in fmt varchar, in val varchar)
{
--  dbg_obj_print (current_proc_name (), val);
```

```
  if (par = 'app')
    return sprintf (fmt, wa_app_to_type (val));
  return sprintf (fmt, val);
};

create procedure DB.DBA.URL_REW_ODS_BLOG (in par varchar, in fmt varchar, in val varchar)
{
--  dbg_obj_print (par, fmt, val);
--  dbg_obj_print (current_proc_name (), val);
  declare acc, ret any;

  acc := DB.DBA.URL_REW_ODS_ACCEPT ();
  if (acc is not null)
    {
      if (par = 'inst')
        {
          declare q, iri, graph any;
          graph := sioc..get_graph ();
          iri := 'http://' || sioc..get_cname () || http_path ();
          ret := DB.DBA.URL_REW_ODS_SPQ (graph, iri, acc);
          return ret;
        }
      else
        return '';
    }
  else if (par = 'inst')
    {
      declare url any;
      val := split_and_decode (val)[0];
      url := (SELECT WAM_HOME_PAGE FROM WA_MEMBER WHERE WAM_INST = val AND WAM_MEMBER_TYPE = 1);
      if (url is not null)
        val := url;
      return sprintf (fmt, val);
    }
  else if (par = 'id' and val <> '')
    {
      if (atoi (val) = 0 and val <> '0')
        fmt := '%s';
      else
        fmt := '?id=%s';
      return sprintf (fmt, val);
    }
};

create procedure DB.DBA.URL_REW_ODS_NNTP (in par varchar, in fmt varchar, in val varchar)
{
--  dbg_obj_print (par, fmt, val);
--  dbg_obj_print (current_proc_name (), val);
  declare acc, ret any;

  acc := DB.DBA.URL_REW_ODS_ACCEPT ();
  if (acc is not null)
    {
       declare q, iri, graph any;
       graph := sioc..get_graph ();
       iri := 'http://' || sioc..get_cname () || http_path ();
--       dbg_obj_print (iri);
       ret := DB.DBA.URL_REW_ODS_SPQ (graph, iri, acc);
       return ret;
    }
  else if (par = 'grp')
    {
      declare gid int;
      val := split_and_decode (val)[0];
      gid := (SELECT NG_GROUP FROM DB.DBA.NEWS_GROUPS WHERE NG_NAME = val);
      ret := sprintf ('/nntpf/nntpf_nthread_view.vspx?group=%d', gid);
      return ret;
    }
  else if (par = 'post')
    {
      ret := sprintf ('/nntpf/nntpf_disp_article.vspx?id=%U', encode_base64 (val));
      return ret;
    }
```

```
}
;

create procedure DB.DBA.URL_REW_ODS_XD (in par varchar, in fmt varchar, in val varchar)
{
--  dbg_obj_print (par, fmt, val);
--  dbg_obj_print (current_proc_name (), val);
  declare acc, ret any;

  acc := DB.DBA.URL_REW_ODS_ACCEPT ();
  if (acc is not null)
    {
       declare q, iri, graph any;
       graph := sioc..get_graph ();
       iri := 'http://' || sioc..get_cname () || http_path ();
--       dbg_obj_print (iri);
       ret := DB.DBA.URL_REW_ODS_SPQ (graph, iri, acc);
       return ret;
    }
  else if (par = 'inst')
    {
      val := split_and_decode (val)[0];
      ret := (SELECT WAM_HOME_PAGE FROM WA_MEMBER WHERE WAM_INST = val and WAM_MEMBER_TYPE = 1);
      return ret;
    }
}
;

create procedure DB.DBA.URL_REW_ODS_WIKI (in par varchar, in fmt varchar, in val varchar)
{
--  dbg_obj_print (par, fmt, val);
--  dbg_obj_print (current_proc_name (), val);
  declare acc, ret any;

  acc := DB.DBA.URL_REW_ODS_ACCEPT ();
  if (acc is not null)
    {
       declare q, iri, graph any;
       graph := sioc..get_graph ();
       iri := 'http://' || sioc..get_cname () || http_path ();
--       dbg_obj_print (iri);
       ret := DB.DBA.URL_REW_ODS_SPQ (graph, iri, acc);
       return ret;
    }
  else if (par = 'inst')
    {
      declare _inst DB.DBA.web_app;
      _inst := (SELECT WAI_INST FROM WA_INSTANCE WHERE WAI_NAME = val);
      ret := _inst.wa_post_url (null, null, val, val);
--      dbg_obj_print ('ret', ret);
      return ret;
    }
  else if (par = 'post')
    {
      return '/'||val;
    }
}
;

create procedure DB.DBA.URL_REW_ODS_PHOTO (in par varchar, in fmt varchar, in val varchar)
{
--  dbg_obj_print (par, fmt, val);
--  dbg_obj_print (current_proc_name (), val);
  declare acc, ret any;

  acc := DB.DBA.URL_REW_ODS_ACCEPT ();
  if (acc is not null)
    {
       declare q, iri, graph any;
       graph := sioc..get_graph ();
       iri := 'http://' || sioc..get_cname () || http_path ();
--       dbg_obj_print (iri);
       ret := DB.DBA.URL_REW_ODS_SPQ (graph, iri, acc);
```

```
        return ret;
      }
    else if (par = 'inst')
      {
        val := split_and_decode (val)[0];
        ret := (SELECT WAM_HOME_PAGE FROM WA_MEMBER WHERE WAM_INST = val AND WAM_MEMBER_TYPE = 1);
        return ret;
      }
    else if (par = 'post')
      {
        declare id int;
        declare col, nam varchar;
        declare exit handler for not found
          {
            signal ('22023', sprintf ('The resource %d doesn''t exists', id));
          };
        id := atoi(ltrim(val, '/'));
        SELECT RES_FULL_PATH INTO nam FROM WS.WS.SYS_DAV_RES WHERE RES_ID = id;
        return nam;
      }
  }
  ;

  create procedure DB.DBA.URL_REW_ODS_ADDRESSBOOK (in par varchar, in fmt varchar, in val varchar)
  {
    declare acc, ret any;

    acc := DB.DBA.URL_REW_ODS_ACCEPT ();
    if (acc is not null)
      {
        if (par = 'instance')
          {
            declare q, iri, graph any;
            graph := sioc..get_graph ();
            iri := 'http://' || sioc..get_cname () || http_path ();
            ret := DB.DBA.URL_REW_ODS_SPQ (graph, iri, acc);
            return ret;
                }
        else
                return '';
      }
    else if (par = 'instance')
      {
        declare id, url any;
        val := split_and_decode (val)[0];
        id := AB.WA.domain_id (val);
        if (id is not null) {
          url := AB.WA.ab_url (id);
          if (url is not null)
            val := url;
        }
        return sprintf (fmt, val);
      }
    else if (par = 'params')
      {
        if (atoi (val) = 0 and val <> '0')
          fmt := '%s';
        else
          fmt := '?id=%s';
        return sprintf (fmt, val);
      }
  }
  ;

  create procedure DB.DBA.URL_REW_ODS_BOOKMARK (in par varchar, in fmt varchar, in val varchar)
  {
    declare acc, ret any;

    acc := DB.DBA.URL_REW_ODS_ACCEPT ();
    if (acc is not null)
      {
        if (par = 'instance')
          {
```

```
          declare q, iri, graph any;
          graph := sioc..get_graph ();
          iri := 'http://' || sioc..get_cname () || http_path ();
          ret := DB.DBA.URL_REW_ODS_SPQ (graph, iri, acc);
          return ret;
              }
      else
              return '';
    }
  else if (par = 'instance')
    {
      declare id, url any;
      val := split_and_decode (val)[0];
      id := BMK.WA.domain_id (val);
      if (id is not null) {
        url := BMK.WA.bookmark_url (id);
        if (url is not null)
          val := url;
      }
      return sprintf (fmt, val);
    }
  else if (par = 'params')
    {
      if (atoi (val) = 0 and val <> '0')
        fmt := '%s';
      else
        fmt := '?id=%s';
      return sprintf (fmt, val);
    }
}
;

create procedure DB.DBA.URL_REW_ODS_BRIEFCASE (in par varchar, in fmt varchar, in val varchar)
{
  declare acc, ret any;

  acc := DB.DBA.URL_REW_ODS_ACCEPT ();
  if (acc is not null)
    {
      if (par = 'instance')
        {
          declare q, iri, graph any;
          graph := sioc..get_graph ();
          iri := 'http://' || sioc..get_cname () || http_path ();
          ret := DB.DBA.URL_REW_ODS_SPQ (graph, iri, acc);
          return ret;
              }
      else
              return '';
    }
  else if (par = 'instance')
    {
      declare id, url any;
      val := split_and_decode (val)[0];
      id := ODRIVE.WA.domain_id (val);
      if (id is not null) {
        url := ODRIVE.WA.odrive_url (id);
        if (url is not null)
          val := url;
      }
      return sprintf (fmt, val);
    }
  else if (par = 'params')
    {
      if (atoi (val) = 0 and val <> '0')
        fmt := '%s';
      else
        fmt := '?id=%s';
      return sprintf (fmt, val);
    }
}
;
```

```
create procedure DB.DBA.URL_REW_ODS_CALENDAR (in par varchar, in fmt varchar, in val varchar)
{
  declare acc, ret any;

  acc := DB.DBA.URL_REW_ODS_ACCEPT ();
  if (acc is not null)
    {
      if (par = 'instance')
        {
          declare q, iri, graph any;
          graph := sioc..get_graph ();
          iri := 'http://' || sioc..get_cname () || http_path ();
          ret := DB.DBA.URL_REW_ODS_SPQ (graph, iri, acc);
          return ret;
              }
      else
              return '';
    }
  else if (par = 'instance')
    {
      declare id, url any;
      val := split_and_decode (val)[0];
      id := CAL.WA.domain_id (val);
      if (id is not null) {
        url := CAL.WA.calendar_url (id);
        if (url is not null)
          val := url;
      }
      return sprintf (fmt, val);
    }
  else if (par = 'params')
    {
      if (atoi (val) = 0 and val <> '0')
        fmt := '%s';
      else
        fmt := '?id=%s';
      return sprintf (fmt, val);
    }
}
;

create procedure DB.DBA.URL_REW_ODS_FEEDS (in par varchar, in fmt varchar, in val varchar)
{
  declare acc, ret any;

  acc := DB.DBA.URL_REW_ODS_ACCEPT ();
  if (acc is not null)
    {
      if (par = 'instance')
        {
          declare q, iri, graph any;
          graph := sioc..get_graph ();
          iri := 'http://' || sioc..get_cname () || http_path ();
          ret := DB.DBA.URL_REW_ODS_SPQ (graph, iri, acc);
          return ret;
              }
      else
              return '';
    }
  else if (par = 'instance')
    {
      declare id, url any;
      val := split_and_decode (val)[0];
      id := ENEWS.WA.domain_id (val);
      if (id is not null) {
        url := ENEWS.WA.enews_url (id) || 'news.vspx';
        if (url is not null)
          val := url;
      }
      return sprintf (fmt, val);
    }
  else if (par = 'params')
    {
```

```
      if (atoi (val) = 0 and val <> '0')
        fmt := '%s';
      else
        fmt := '?id=%s';
      return sprintf (fmt, val);
    }
}
;

create procedure DB.DBA.URL_REW_ODS_POLLS (in par varchar, in fmt varchar, in val varchar)
{
  declare acc, ret any;

  acc := DB.DBA.URL_REW_ODS_ACCEPT ();
  if (acc is not null)
    {
      if (par = 'instance')
        {
          declare q, iri, graph any;
          graph := sioc..get_graph ();
          iri := 'http://' || sioc..get_cname () || http_path ();
          ret := DB.DBA.URL_REW_ODS_SPQ (graph, iri, acc);
          return ret;
              }
      else
              return '';
    }
  else if (par = 'instance')
    {
      declare id, url any;
      val := split_and_decode (val)[0];
      id := POLLS.WA.domain_id (val);
      if (id is not null) {
        url := POLLS.WA.polls_url (id);
        if (url is not null)
          val := url;
      }
      return sprintf (fmt, val);
    }
  else if (par = 'params')
    {
      if (atoi (val) = 0 and val <> '0')
        fmt := '%s';
      else
        fmt := '?id=%s';
      return sprintf (fmt, val);
    }
}
;

create procedure DB.DBA.URL_REW_ODS_FOAF_EXT (in par varchar, in fmt varchar, in val varchar)
{
  if (par = '*accept*')
    {
      declare ext any;
      ext := 'rdf';
      if (val = 'text/rdf+n3')
        ext := 'n3';
      return sprintf (fmt, ext);
    }
  else
    return sprintf (fmt, val);
}
;

create procedure ur_ods_rdf_doc (in path varchar)
{
  declare r any;
  r := regexp_match ('[^/]*\x24', path);
  return r||'#this';
};

create procedure ur_ods_html_doc (in path varchar)
```

```
{
  declare pos, r any;
  if (path like '%/foaf.%')
    {
      pos := strrchr (path, '/');
    }
  else if (path like '%#%')
    {
      pos := strrchr (path, '#');
    }
  if (pos > 0)
    r := subseq (path, 0, pos);
  else
    r := '/';
  return r;
};
-- ODS Rules

-- http://cname/dataspace/uname
-- http://cname/dataspace/person/uname

DB.DBA.URLREWRITE_CREATE_REGEX_RULE ('ods_rule1', 1,
    '/dataspace/((person/)?[^/#]*)', vector('ufname'), 1,
    '%s', vector('ufname'),
    'DB.DBA.URL_REW_ODS_USER');

-- http://cname/dataspace/uname with Accept will do 303 to the /sparql
DB.DBA.URLREWRITE_CREATE_REGEX_RULE ('ods_rule2', 1,
    '/dataspace/([^/]*)', vector('ufname'), 1,
    '/sparql?query=define+input%%3Ainference+%%3Chttp%%3A//^{URIQADefaultHost}^/dataspace%%3E+DESCRIBE+%%
    null,
    '(application|text)/rdf.(xml|n3|turtle|ttl)',
    0,
    303);

-- http://cname/dataspace/uname/app_type
DB.DBA.URLREWRITE_CREATE_REGEX_RULE ('ods_rule3', 1,
    '/dataspace/((?!person)[^/]*)/([^\\./]*)', vector('ufname', 'app'), 2,
    '/ods/app_inst.vspx?app=%s&ufname=%s&l=1', vector('app', 'ufname'),
    'DB.DBA.URL_REW_ODS_APP');

-- http://cname/dataspace/uname/file.ext
DB.DBA.URLREWRITE_CREATE_REGEX_RULE ('ods_rule4', 1,
    '/dataspace/([^/]*)/(sioc|about|yadis)\\.(rdf|n3|ttl|xrds)', vector('ufname', 'file', 'fmt'), 3,
    '%s', vector('ufname'),
    'DB.DBA.URL_REW_ODS_USER_GEM');

-- Rules for FOAF profile

-- http://cname/dataspace/person/uname with Accept, do 303 to http://cname/dataspace/person/uname/foaf.ex
DB.DBA.URLREWRITE_CREATE_REGEX_RULE ('ods_rule5', 1,
    '/dataspace/person/([^/#]*)/?', vector('ufname'), 1,
    '/dataspace/person/%U/foaf.%s', vector('ufname', '*accept*'),
    'DB.DBA.URL_REW_ODS_FOAF_EXT',
    '(application|text)/rdf.(xml|n3|turtle|ttl)',
    2,
    303);

DB.DBA.URLREWRITE_CREATE_REGEX_RULE ('ods_rule6', 1,
    '/dataspace/person/([^/]*)/page/([^/]*)/?', vector('ufname', 'page'), 1,
    '/dataspace/person/%U/foaf.%s?page=%s', vector('ufname', '*accept*', 'page'),
    'DB.DBA.URL_REW_ODS_FOAF_EXT',
    '(application|text)/rdf.(xml|n3|turtle|ttl)',
    2,
    303);

-- http://cname/dataspace/person/uname/foaf.ext
DB.DBA.URLREWRITE_CREATE_REGEX_RULE ('ods_rule7', 1,
    '/dataspace/person/([^/]*)/foaf.(rdf|n3|ttl)', vector('ufname', 'fmt'), 1,
    '/ods/foaf.vsp?uname=%U&fmt=%U', vector('ufname', 'fmt'),
    null,
    null,
    2,
```

```
    null);

-- App Instance Gem

DB.DBA.URLREWRITE_CREATE_REGEX_RULE ('ods_post_gem_rule', 1,
    '/dataspace/([^/]*)/([^/]*)/([^/]*/)?([^/]*/)?(sioc|about)\\.(rdf|n3|ttl)', vector('ufname', 'app', '
    '%s', vector('ufname'),
    'DB.DBA.URL_REW_ODS_GEM');

-- Weblog Rules

DB.DBA.URLREWRITE_CREATE_REGEX_RULE ('ods_blog_rule1', 1,
    '/dataspace/([^/]*)/weblog/([^/]*)', vector('ufname', 'inst'), 2,
    '%s', vector('inst'),
    'DB.DBA.URL_REW_ODS_BLOG');

DB.DBA.URLREWRITE_CREATE_REGEX_RULE ('ods_blog_rule2', 1,
    '/dataspace/([^/]*)/weblog/([^/]*)/([^/]*)', vector('ufname', 'inst', 'id'), 3,
    '%s%s', vector('inst', 'id'),
    'DB.DBA.URL_REW_ODS_BLOG');

-- Discussion rules

DB.DBA.URLREWRITE_CREATE_REGEX_RULE ('ods_nntp_rule1', 1,
    '/dataspace/discussion/([^/]*)', vector('grp'), 1,
    '%s', vector('grp'),
    'DB.DBA.URL_REW_ODS_NNTP');

DB.DBA.URLREWRITE_CREATE_REGEX_RULE ('ods_nntp_rule2', 1,
    '/dataspace/discussion/([^/]*)/((?!sioc)(?!about)[^/]*)', vector('grp', 'post'), 2,
    '%s', vector('post'),
    'DB.DBA.URL_REW_ODS_NNTP');

-- Community

DB.DBA.URLREWRITE_CREATE_REGEX_RULE ('ods_xd_rule1', 1,
    '/dataspace/([^/]*)/community/([^/]*)', vector('ufname', 'inst'), 2,
    '%s', vector('inst'),
    'DB.DBA.URL_REW_ODS_XD');

-- Wiki

DB.DBA.URLREWRITE_CREATE_REGEX_RULE ('ods_wiki_rule1', 1,
    '/dataspace/([^/]*)/wiki/([^/]*)', vector('ufname', 'inst'), 2,
    '%s', vector('inst'),
    'DB.DBA.URL_REW_ODS_WIKI');

DB.DBA.URLREWRITE_CREATE_REGEX_RULE ('ods_wiki_rule2', 1,
    '/dataspace/([^/]*)/wiki/([^/]*)/([^/]*)', vector('ufname', 'inst', 'post'), 2,
    '%s%s', vector('inst', 'post'),
    'DB.DBA.URL_REW_ODS_WIKI');

-- Gallery

DB.DBA.URLREWRITE_CREATE_REGEX_RULE ('ods_photo_rule1', 1,
    '/dataspace/([^/]*)/photos/([^/]*)', vector('ufname', 'inst'), 2,
    '%s', vector('inst'),
    'DB.DBA.URL_REW_ODS_PHOTO');

DB.DBA.URLREWRITE_CREATE_REGEX_RULE ('ods_photo_rule2', 1,
    '/dataspace/([^/]*)/photos/([^/]*)/([^/]*)', vector('ufname', 'inst', 'post'), 2,
    '%s', vector('post'),
    'DB.DBA.URL_REW_ODS_PHOTO');

-- AddressBook

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'ods_addressbook_rule1',
    1,
    '/dataspace/([^/]*)/addressbook/([^/]*)',
    vector('uname', 'instance'),
    2,
    '%s', vector('instance'),
```

```
    'DB.DBA.URL_REW_ODS_ADDRESSBOOK');

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'ods_addressbook_rule2',
    1,
    '/dataspace/([^/]*)/addressbook/([^/]*)/(.*)',
    vector('uname', 'instance', 'params'),
    3,
    '%s%s',
    vector('instance', 'params'),
    'DB.DBA.URL_REW_ODS_ADDRESSBOOK');

-- Bookmark

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'ods_bookmark_rule1',
    1,
    '/dataspace/([^/]*)/bookmark/([^/]*)',
    vector('uname', 'instance'),
    2,
    '%s', vector('instance'),
    'DB.DBA.URL_REW_ODS_BOOKMARK');

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'ods_bookmark_rule2',
    1,
    '/dataspace/([^/]*)/bookmark/([^/]*)/(.*)',
    vector('uname', 'instance', 'params'),
    3,
    '%s%s',
    vector('instance', 'params'),
    'DB.DBA.URL_REW_ODS_BOOKMARK');

-- Briefcase

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'ods_briefcase_rule1',
    1,
    '/dataspace/([^/]*)/briefcase/([^/]*)',
    vector('uname', 'instance'),
    2,
    '%s', vector('instance'),
    'DB.DBA.URL_REW_ODS_BRIEFCASE');

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'ods_briefcase_rule2',
    1,
    '/dataspace/([^/]*)/briefcase/([^/]*)/(.*)',
    vector('uname', 'instance', 'params'),
    3,
    '%s%s',
    vector('instance', 'params'),
    'DB.DBA.URL_REW_ODS_BRIEFCASE');

-- Calendar

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'ods_calendar_rule1',
    1,
    '/dataspace/([^/]*)/calendar/([^/]*)',
    vector('uname', 'instance'),
    2,
    '%s', vector('instance'),
    'DB.DBA.URL_REW_ODS_CALENDAR');

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'ods_calendar_rule2',
    1,
    '/dataspace/([^/]*)/calendar/([^/]*)/(.*)',
    vector('uname', 'instance', 'params'),
    3,
    '%s%s',
    vector('instance', 'params'),
```

```
    'DB.DBA.URL_REW_ODS_CALENDAR');

-- Feeds

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'ods_feeds_rule1',
    1,
    '/dataspace/([^/]*)/feeds/([^/]*)',
    vector('uname', 'instance'),
    2,
    '%s', vector('instance'),
    'DB.DBA.URL_REW_ODS_FEEDS');

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'ods_feeds_rule2',
    1,
    '/dataspace/([^/]*)/feeds/([^/]*)/(.*)',
    vector('uname', 'instance', 'params'),
    3,
    '%s%s',
    vector('instance', 'params'),
    'DB.DBA.URL_REW_ODS_FEEDS');

-- Polls

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'ods_polls_rule1',
    1,
    '/dataspace/([^/]*)/polls/([^/]*)',
    vector('uname', 'instance'),
    2,
    '%s', vector('instance'),
    'DB.DBA.URL_REW_ODS_POLLS');

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'ods_polls_rule2',
    1,
    '/dataspace/([^/]*)/polls/([^/]*)/(.*)',
    vector('uname', 'instance', 'params'),
    3,
    '%s%s',
    vector('instance', 'params'),
    'DB.DBA.URL_REW_ODS_POLLS');

-- ODS Base rules
DB.DBA.URLREWRITE_CREATE_RULELIST ('ods_base_rule_list1', 1,
        vector(
                'ods_rule1', 'ods_rule2', 'ods_rule3', 'ods_rule4'
              ));

DB.DBA.URLREWRITE_CREATE_RULELIST ('ods_foaf_rule_list1', 1,
        vector(
                'ods_rule5', 'ods_rule6', 'ods_rule7'
              ));

DB.DBA.URLREWRITE_CREATE_RULELIST ('ods_gems_rule_list1', 1,
        vector(
                'ods_post_gem_rule'
              ));

-- ODS Blog rules
DB.DBA.URLREWRITE_CREATE_RULELIST ('ods_blog_rule_list1', 1,
        vector(
                'ods_blog_rule1', 'ods_blog_rule2'
              ));

-- ODS Discussion rules
DB.DBA.URLREWRITE_CREATE_RULELIST ('ods_nntp_rule_list1', 1,
        vector(
                'ods_nntp_rule1', 'ods_nntp_rule2'
              ));

-- ODS Community rules
```

```
DB.DBA.URLREWRITE_CREATE_RULELIST ('ods_xd_rule_list1', 1,
        vector(
                'ods_xd_rule1'
              ));

-- ODS Wiki rules
DB.DBA.URLREWRITE_CREATE_RULELIST ('ods_wiki_rule_list1', 1,
        vector(
                'ods_wiki_rule1', 'ods_wiki_rule2'
              ));

-- ODS Gallery rules
DB.DBA.URLREWRITE_CREATE_RULELIST ('ods_photo_rule_list1', 1,
        vector(
                'ods_photo_rule1', 'ods_photo_rule2'
              ));

-- ODS AddressBook rules
DB.DBA.URLREWRITE_CREATE_RULELIST (
    'ods_addressbook_rule_list1',
    1,
    vector (
                'ods_addressbook_rule1',
          'ods_addressbook_rule2'
         ));

-- ODS Bookmark rules
DB.DBA.URLREWRITE_CREATE_RULELIST (
    'ods_bookmark_rule_list1',
    1,
    vector (
                'ods_bookmark_rule1',
          'ods_bookmark_rule2'
        ));

-- ODS Briefcase rules
DB.DBA.URLREWRITE_CREATE_RULELIST (
    'ods_briefcase_rule_list1',
    1,
    vector (
                'ods_briefcase_rule1',
          'ods_briefcase_rule2'
        ));

-- ODS Calendar rules
DB.DBA.URLREWRITE_CREATE_RULELIST (
    'ods_calendar_rule_list1',
    1,
    vector (
                'ods_calendar_rule1',
          'ods_calendar_rule2'
        ));

-- ODS Feeds rules
DB.DBA.URLREWRITE_CREATE_RULELIST (
    'ods_feeds_rule_list1',
    1,
    vector (
                'ods_feeds_rule1',
          'ods_feeds_rule2'
        ));

-- ODS Polls rules
DB.DBA.URLREWRITE_CREATE_RULELIST (
    'ods_polls_rule_list1',
    1,
    vector (
                'ods_polls_rule1',
          'ods_polls_rule2'
        ));

-- All ODS Rules
DB.DBA.URLREWRITE_CREATE_RULELIST ('ods_rule_list1', 1,
```

```
        vector(
          'ods_base_rule_list1',
          'ods_foaf_rule_list1',
          'ods_blog_rule_list1',
          'ods_nntp_rule_list1',
          'ods_xd_rule_list1',
          'ods_wiki_rule_list1',
          'ods_photo_rule_list1',
          'ods_addressbook_rule_list1',
          'ods_bookmark_rule_list1',
          'ods_briefcase_rule_list1',
          'ods_calendar_rule_list1',
          'ods_feeds_rule_list1',
          'ods_polls_rule_list1',
          'ods_gems_rule_list1'
              ));

DB.DBA.XML_SET_NS_DECL ('ods', 'http://www.openlinksw.com/virtuoso/ods/', 2);
```

16.8.8. Sybase using demonstration 'pubs2' database

```
-- Setup script for Linked Data Views of Sybase 15 PUBS2 Sample Database --

DB..vd_remote_data_source ('syb15ma-pubs2', '', '<uid>','<pwd>');

ATTACH TABLE  "pubs2.dbo.au_pix"  PRIMARY KEY ("au_id")                  AS "pubs2"."syb"."au_pix"  FROM
ATTACH TABLE  "pubs2.dbo.authors"  PRIMARY KEY ("au_id")                  AS "pubs2"."syb"."authors"  FRO
ATTACH TABLE  "pubs2.dbo.discounts"  PRIMARY KEY ("stor_id")                  AS "pubs2"."syb"."discounts
ATTACH TABLE  "pubs2.dbo.publishers"  PRIMARY KEY ("pub_id")                  AS "pubs2"."syb"."publisher
ATTACH TABLE  "pubs2.dbo.roysched"  PRIMARY KEY ("title_id")                  AS "pubs2"."syb"."roysched"
ATTACH TABLE  "pubs2.dbo.sales"  PRIMARY KEY ("stor_id", "ord_num")                  AS "pubs2"."syb"."sa
ATTACH TABLE  "pubs2.dbo.salesdetail"  PRIMARY KEY ("stor_id", "ord_num", "title_id")                  A
ATTACH TABLE  "pubs2.dbo.stores"  PRIMARY KEY ("stor_id")                  AS "pubs2"."syb"."stores"  FRO
ATTACH TABLE  "pubs2.dbo.titleauthor"  PRIMARY KEY ("au_id", "title_id")                  AS "pubs2"."syb
ATTACH TABLE  "pubs2.dbo.titles"  PRIMARY KEY ("title_id", "pub_id")                  AS "pubs2"."syb"."t

COMMIT WORK;

GRANT SELECT ON pubs2.syb.au_pix TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON pubs2.syb.authors TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON pubs2.syb.discounts TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON pubs2.syb.publishers TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON pubs2.syb.roysched TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON pubs2.syb.sales TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON pubs2.syb.salesdetail TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON pubs2.syb.stores TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON pubs2.syb.titleauthor TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON pubs2.syb.titles TO "SPARQL", "SPARQL_UPDATE";

-----------------------------------------------------------------

-------- Create rdfs:Class definitions ----------------------------

ttlp (
'
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

@prefix syb: <http://example.com/schemas/sybasepubs2/> .

syb:titles a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/sybasepubs2> ;
        rdfs:label "titles" ;
        rdfs:comment "Sybase Pubs2 titles table" .

syb:title_id a rdf:Property ;
        rdfs:domain syb:titles ;
        rdfs:range xsd:string ;
        rdfs:label "title id" .

syb:title a rdf:Property ;
        rdfs:domain syb:titles ;
```

```
        rdfs:range xsd:string ;
        rdfs:label "title" .

syb:type a rdf:Property ;
        rdfs:domain syb:titles ;
        rdfs:range xsd:string ;
        rdfs:label "type" .

syb:pub_id a rdf:Property ;
        rdfs:domain syb:titles ;
        rdfs:range syb:publishers ;
        rdfs:label "pub_id" .

syb:advance a rdf:Property ;
        rdfs:domain syb:titles ;
        rdfs:range xsd:decimal ;
        rdfs:label "advance" .

syb:price a rdf:Property ;
        rdfs:domain syb:titles ;
        rdfs:range xsd:decimal ;
        rdfs:label "price" .

syb:total_sales a rdf:Property ;
        rdfs:domain syb:titles ;
        rdfs:range xsd:integer ;
        rdfs:label "total_sales" .

syb:notes a rdf:Property ;
        rdfs:domain syb:titles ;
        rdfs:range xsd:string ;
        rdfs:label "notes" .

syb:contract a rdf:Property ;
        rdfs:domain syb:titles ;
        rdfs:range xsd:integer ;
        rdfs:label "contract" .

syb:pubdate a rdf:Property ;
        rdfs:domain syb:titles ;
        rdfs:range xsd:dateTime ;
        rdfs:label "publish date" .

syb:authors a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/sybasepubs2> ;
        rdfs:label "authors" ;
        rdfs:comment "Sybase Pubs2 authors table" .

syb:au_id a rdf:Property ;
        rdfs:domain syb:authors ;
        rdfs:range xsd:string ;
        rdfs:label "author id" .

syb:au_lname a rdf:Property ;
        rdfs:domain syb:authors ;
        rdfs:range xsd:string ;
        rdfs:label "author last name" .

syb:au_fname a rdf:Property ;
        rdfs:domain syb:authors ;
        rdfs:range xsd:string ;
        rdfs:label "author first name" .

syb:phone a rdf:Property ;
        rdfs:domain syb:authors ;
        rdfs:range xsd:string ;
        rdfs:label "phone number" .

syb:address a rdf:Property ;
        rdfs:domain syb:authors ;
        rdfs:range xsd:string ;
        rdfs:label "address" .
```

```
syb:city a rdf:Property ;
        rdfs:domain syb:authors ;
        rdfs:range xsd:string ;
        rdfs:label "city" .

syb:state a rdf:Property ;
        rdfs:domain syb:authors ;
        rdfs:range xsd:string ;
        rdfs:label "state" .

syb:country a rdf:Property ;
        rdfs:domain syb:authors ;
        rdfs:range xsd:string ;
        rdfs:label "country" .

syb:postalcode a rdf:Property ;
        rdfs:domain syb:authors ;
        rdfs:range xsd:string ;
        rdfs:label "postalcode" .

syb:stores a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/sybasepubs2> ;
        rdfs:label "stores" ;
        rdfs:comment "Sybase Pubs2 stores table" .

syb:stor_id a rdf:Property ;
        rdfs:domain syb:stores ;
        rdfs:range xsd:string ;
        rdfs:label "store id" .

syb:stor_name a rdf:Property ;
        rdfs:domain syb:stores ;
        rdfs:range xsd:string ;
        rdfs:label "store name" .

syb:stor_address a rdf:Property ;
        rdfs:domain syb:stores ;
        rdfs:range xsd:string ;
        rdfs:label "store address" .

syb:city a rdf:Property ;
        rdfs:domain syb:stores ;
        rdfs:range xsd:string ;
        rdfs:label "city" .

syb:state a rdf:Property ;
        rdfs:domain syb:stores ;
        rdfs:range xsd:string ;
        rdfs:label "state" .

syb:country a rdf:Property ;
        rdfs:domain syb:stores ;
        rdfs:range xsd:string ;
        rdfs:label "country" .

syb:postalcode a rdf:Property ;
        rdfs:domain syb:stores ;
        rdfs:range xsd:string ;
        rdfs:label "postal code" .

syb:payterms a rdf:Property ;
        rdfs:domain syb:stores ;
        rdfs:range xsd:string ;
        rdfs:label "payment terms" .

syb:au_pix a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/sybasepubs2> ;
        rdfs:label "authors pictures" ;
        rdfs:comment "Sybase Pubs2 au_pix table" .

syb:au_id a rdf:Property ;
        rdfs:domain syb:au_pix ;
        rdfs:range syb:authors ;
```

```
        rdfs:label "author id" .

syb:format_type a rdf:Property ;
        rdfs:domain syb:au_pix ;
        rdfs:range xsd:string ;
        rdfs:label "format type" .

syb:bytesize a rdf:Property ;
        rdfs:domain syb:au_pix ;
        rdfs:range xsd:integer ;
        rdfs:label "byte size" .

syb:pixwidth_hor a rdf:Property ;
        rdfs:domain syb:au_pix ;
        rdfs:range xsd:string ;
        rdfs:label "picture horizontal width" .

syb:pixwidth_vert a rdf:Property ;
        rdfs:domain syb:au_pix ;
        rdfs:range xsd:string ;
        rdfs:label "picture vertical width" .

syb:discounts a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/sybasepubs2> ;
        rdfs:label "discounts" ;
        rdfs:comment "Sybase Pubs2 discount table" .

syb:discounttype a rdf:Property ;
        rdfs:domain syb:discounts ;
        rdfs:range xsd:string ;
        rdfs:label "discounttype" .

syb:stor_id a rdf:Property ;
        rdfs:domain syb:discounts ;
        rdfs:range syb:stores ;
        rdfs:label "store id" .

syb:lowqty a rdf:Property ;
        rdfs:domain syb:discounts ;
        rdfs:range xsd:integer ;
        rdfs:label "min quantity" .

syb:highqty a rdf:Property ;
        rdfs:domain syb:discounts ;
        rdfs:range xsd:integer ;
        rdfs:label "max quantity" .

syb:discount a rdf:Property ;
        rdfs:domain syb:discounts ;
        rdfs:range xsd:decimal ;
        rdfs:label "min quantity" .

syb:salesdetail a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/sybasepubs2> ;
        rdfs:label "sales details" ;
        rdfs:comment "Sybase Pubs2 sales detail table" .

syb:store_id a rdf:Property ;
        rdfs:domain syb:salesdetail ;
        rdfs:range syb:stores ;
        rdfs:label "store id" .

syb:ord_num a rdf:Property ;
        rdfs:domain syb:salesdetail ;
        rdfs:range syb:sales ;
        rdfs:label "order number" .

syb:title_id a rdf:Property ;
        rdfs:domain syb:salesdetail ;
        rdfs:range syb:titles ;
        rdfs:label "title id" .

syb:qty a rdf:Property ;
```

```
        rdfs:domain syb:salesdetail ;
        rdfs:range xsd:integer ;
        rdfs:label "quantity" .

syb:discount a rdf:Property ;
        rdfs:domain syb:salesdetail ;
        rdfs:range xsd:decimal ;
        rdfs:label "discount" .

syb:publishers a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/sybasepubs2> ;
        rdfs:label "Publishers" ;
        rdfs:comment "Sybase Pubs2 publishers table" .

syb:pub_id a rdf:Property ;
        rdfs:domain syb:publishers ;
        rdfs:range xsd:string ;
        rdfs:label "publisher id" .

syb:pub_name a rdf:Property ;
        rdfs:domain syb:publishers ;
        rdfs:range xsd:string ;
        rdfs:label "publisher name" .

syb:city a rdf:Property ;
        rdfs:domain syb:publishers ;
        rdfs:range xsd:string ;
        rdfs:label "city" .

syb:state a rdf:Property ;
        rdfs:domain syb:publishers ;
        rdfs:range xsd:string ;
        rdfs:label "state" .

syb:titleauthor a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/sybasepubs2> ;
        rdfs:label "title author" ;
        rdfs:comment "Sybase Pubs2 titleauthor table" .

syb:au_id a rdf:Property ;
        rdfs:domain syb:titleauthor ;
        rdfs:range syb:authors ;
        rdfs:label "author id" .

syb:title_id a rdf:Property ;
        rdfs:domain syb:titleauthor ;
        rdfs:range syb:titles ;
        rdfs:label "title id" .

syb:au_ord a rdf:Property ;
        rdfs:domain syb:titleauthor ;
        rdfs:range xsd:integer ;
        rdfs:label "author order" .

syb:royaltyper a rdf:Property ;
        rdfs:domain syb:titleauthor ;
        rdfs:range xsd:integer ;
        rdfs:label "royalty per book" .

syb:roysched a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/sybasepubs2> ;
        rdfs:label "Royalty Schedule" ;
        rdfs:comment "Sybase Pubs2 roysched table" .

syb:title_id a rdf:Property ;
        rdfs:domain syb:roysched ;
        rdfs:range syb:titles ;
        rdfs:label "title id" .

syb:lorange a rdf:Property ;
        rdfs:domain syb:roysched ;
        rdfs:range xsd:integer ;
        rdfs:label "low range" .
```

```
syb:hirange a rdf:Property ;
        rdfs:domain syb:roysched ;
        rdfs:range xsd:integer ;
        rdfs:label "high range" .

syb:royalty a rdf:Property ;
        rdfs:domain syb:roysched ;
        rdfs:range xsd:integer ;
        rdfs:label "royalty" .

syb:sales a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/sybasepubs2> ;
        rdfs:label "Sales" ;
        rdfs:comment "Sybase Pubs2 sales table" .

syb:stor_id a rdf:Property ;
        rdfs:domain syb:sales ;
        rdfs:range xsd:string ;
        rdfs:label "store id" .

syb:ord_num a rdf:Property ;
        rdfs:domain syb:sales ;
        rdfs:range xsd:string ;
        rdfs:label "order number" .

syb:date a rdf:Property ;
        rdfs:domain syb:sales ;
        rdfs:range xsd:dateTime ;
        rdfs:label "date" .
', '', 'http://example.com/schemas/sybasepubs2', 0);

-------------------------------------------------------------

----------- Create IRI Classes -------------

SPARQL

        create iri class <http://example.com/schemas/sybasepubs2/titles_iri>
        "http://^{URIQADefaultHost}^/sybasepubs2/titles/%s_%s#this"
        (in title_id varchar not null, in title varchar not null) .

        create iri class <http://example.com/schemas/sybasepubs2/authors_iri>
        "http://^{URIQADefaultHost}^/sybasepubs2/authors/%s#this"
        (in au_id varchar not null) .

        create iri class <http://example.com/schemas/sybasepubs2/stores_iri>
        "http://^{URIQADefaultHost}^/sybasepubs2/stores/%s#this"
        (in stor_id varchar not null) .

        create iri class <http://example.com/schemas/sybasepubs2/au_pix_iri>
        "http://^{URIQADefaultHost}^/sybasepubs2/au_pix/%s#this"
        (in au_id varchar not null) .

        create iri class <http://example.com/schemas/sybasepubs2/discounts_iri>
        "http://^{URIQADefaultHost}^/sybasepubs2/discounts/%s#this"
        (in discounttype varchar not null) .

        create iri class <http://example.com/schemas/sybasepubs2/salesdetail_iri>
        "http://^{URIQADefaultHost}^/sybasepubs2/salesdetail/%s_%s_%s#this"
        (in stor_id varchar not null, in ord_num varchar not null, in title_id varchar not null) .

        create iri class <http://example.com/schemas/sybasepubs2/publishers_iri>
        "http://^{URIQADefaultHost}^/sybasepubs2/publishers/%s#this"
        (in pub_id varchar not null) .

        create iri class <http://example.com/schemas/sybasepubs2/titleauthor_iri>
        "http://^{URIQADefaultHost}^/sybasepubs2/titleauthor/%s_%s#this"
        (in au_id varchar not null, in title_id varchar not null) .

        create iri class <http://example.com/schemas/sybasepubs2/roysched_iri>
        "http://^{URIQADefaultHost}^/sybasepubs2/roysched/%s#this"
        (in title_id varchar not null) .
```

```
        create iri class <http://example.com/schemas/sybasepubs2/sales_iri>
        "http://^{URIQADefaultHost}^/sybasepubs2/sales/%s_%s#this"
        (in stor_id varchar not null, in ord_num varchar not null) .

;

-----------------------------------------------------------------

------------ Create Quad Store ----------------------------------

SPARQL

prefix syb: <http://example.com/schemas/sybasepubs2/>

alter quad storage virtrdf:DefaultQuadStorage
  from pubs2.syb.au_pix as au_pix_tbl
  from pubs2.syb.authors as authors_tbl
  from pubs2.syb.discounts as discounts_tbl
  from pubs2.syb.publishers as publishers_tbl
  from pubs2.syb.roysched as roysched_tbl
  from pubs2.syb.sales as sales_tbl
  from pubs2.syb.salesdetail as salesdetail_tbl
  from pubs2.syb.stores as stores_tbl
  from pubs2.syb.titleauthor as titleauthor_tbl
  from pubs2.syb.titles as titles_tbl
{
  create virtrdf:sybasepubs2 as
      graph <http://example.com/sybasepubs2>
  {
        syb:au_pix_iri (au_pix_tbl.au_id) a syb:au_pix as virtrdf:au_pix_id;
        syb:au_id au_pix_tbl.au_id as virtrdf:au_pix_au_id;
        syb:format_type au_pix_tbl.format_type as virtrdf:au_pix_format_type;
        syb:bytesize au_pix_tbl.bytesize as virtrdf:au_pix_bytesize;
        syb:pixwidth_hor au_pix_tbl.pixwidth_hor as virtrdf:au_pix_pixwidth_hor;
        syb:pixwidth_vert au_pix_tbl.pixwidth_vert as virtrdf:au_pix_pixwidth_vert ;
        syb:has_author syb:authors_iri(authors_tbl.au_id) where (^{authors_tbl.}^.au_id = ^{au_pix_tbl.}^

        syb:authors_iri (authors_tbl.au_id) a syb:authors as virtrdf:authors_pk ;
        syb:au_id authors_tbl.au_id as virtrdf:authors_au_id;
        syb:au_lname authors_tbl.au_lname as virtrdf:authors_au_lname;
        syb:au_fname authors_tbl.au_fname as virtrdf:authors_au_fname;
        syb:phone authors_tbl.phone  as virtrdf:authors_phone;
        syb:address authors_tbl.address  as virtrdf:authors_address;
        syb:city authors_tbl.city as virtrdf:authors_city;
        syb:state authors_tbl.state  as virtrdf:authors_state;
        syb:country authors_tbl.country as virtrdf:authors_country;
        syb:postalcode authors_tbl.postalcode as virtrdf:authors_postalcode;
        syb:has_title syb:titleauthor_iri(titleauthor_tbl.au_id, titleauthor_tbl.title_id) where (^{title
        syb:has_pix syb:au_pix_iri(au_pix_tbl.au_id) where (^{au_pix_tbl.}^.au_id = ^{authors_tbl.}^.au_i

        syb:discounts_iri (discounts_tbl.stor_id) a syb:discounts as virtrdf:discounts_pk;
        syb:discounttype discounts_tbl.discounttype as virtrdf:discounts_discounttype;
        syb:stor_id syb:stores_iri(stores_tbl.stor_id) where (^{stores_tbl.}^.stor_id = ^{stores_tbl.}^.s
        syb:lowqty discounts_tbl.lowqty as virtrdf:discounts_lowqty;
        syb:highqty discounts_tbl.highqty as virtrdf:discounts_highqty;
        syb:discount discounts_tbl.discount as virtrdf:discounts_discount .

        syb:publishers_iri (publishers_tbl.pub_id) a syb:publishers as virtrdf:publishers_pk;
        syb:pub_id syb:titles_iri(titles_tbl.title_id, titles_tbl.pub_id) where (^{titles_tbl.}^.pub_id =
        syb:pub_name publishers_tbl.pub_name as virtrdf:publisherss_pub_name;
        syb:city publishers_tbl.city as virtrdf:publisherss_city;
        syb:state publishers_tbl.state as virtrdf:publisherss_state .

        syb:roysched_iri (roysched_tbl.title_id) a syb:roysched as virtrdf:roysched_pk;
        syb:title_id syb:titleauthor_iri(titleauthor_tbl.au_id, titleauthor_tbl.title_id) where (^{titlea
        syb:lorange roysched_tbl.lorange as virtrdf:roysched_lorange;
        syb:hirange roysched_tbl.hirange as virtrdf:roysched_hirange;
        syb:royalty roysched_tbl.royalty as virtrdf:roysched_royalty .

        syb:sales_iri (sales_tbl.stor_id, sales_tbl.ord_num) a syb:sales as virtrdf:sales_pk;
        syb:stor_id sales_tbl.stor_id as virtrdf:sales_stor_id;
        syb:ord_num sales_tbl.ord_num as virtrdf:sales_ord_num;
```

```
        syb:date sales_tbl.date as virtrdf:sales_date;
        syb:has_salesdetail syb:salesdetail_iri(salesdetail_tbl.stor_id, salesdetail_tbl.ord_num, salesde
        syb:has_stores syb:stores_iri(stores_tbl.stor_id) where (^{stores_tbl.}^.stor_id = ^{sales_tbl.}^

        syb:salesdetail_iri (salesdetail_tbl.stor_id, salesdetail_tbl.ord_num, salesdetail_tbl.title_id)
        syb:stor_id salesdetail_tbl.stor_id as virtrdf:salesdetail_stor_id;
        syb:ord_num salesdetail_tbl.ord_num as virtrdf:salesdetail_ord_num;
        syb:title_id salesdetail_tbl.title_id as virtrdf:salesdetail_title_id;
        syb:qty salesdetail_tbl.qty as virtrdf:salesdeail_qty;
        syb:discount salesdetail_tbl.discount as virtrdf:salesdetail_discount;
        syb:has_title  syb:titles_iri (titles_tbl.title_id, titles_tbl.pub_id) where (^{titles_tbl.}^.tit
        syb:has_sales syb:sales_iri(sales_tbl.stor_id, sales_tbl.ord_num) where (^{salesdetail_tbl.}^.sto

        syb:stores_iri (stores_tbl.stor_id) a syb:stores as virtrdf:stores_pk;
        syb:stor_id stores_tbl.stor_id as virtrdf:stores_stor_id;
        syb:stor_name stores_tbl.stor_name as virtrdf:stores_stor_name;
        syb:stor_address stores_tbl.stor_address as virtrdf:stores_stor_address;
        syb:city stores_tbl.city as virtrdf:stores_city;
        syb:state stores_tbl.state as virtrdf:stores_state;
        syb:country stores_tbl.country as virtrdf:stores_country;
        syb:postalcode stores_tbl.postalcode as virtrdf:stores_postalcode;
        syb:payterms stores_tbl.payterms as virtrdf:stores_payterms;
        syb:has_sales syb:sales_iri(sales_tbl.stor_id, sales_tbl.ord_num) where (^{sales_tbl.}^.stor_id =

        syb:titleauthor_iri (titleauthor_tbl.au_id, titleauthor_tbl.title_id) a syb:titleauthor as virtrd
        syb:au_id titleauthor_tbl.au_id as virtrdf:titleauthor_au_id;
        syb:title_id titleauthor_tbl.title_id as virtrdf:titleauthor_title_id;
        syb:au_ord titleauthor_tbl.au_ord as virtrdf:titleauthor_au_ord;
        syb:royaltyper titleauthor_tbl.royaltyper as virtrdf:titleauthor_royaltyper;
        syb:has_author syb:authors_iri(authors_tbl.au_id) where (^{authors_tbl.}^.au_id = ^{titleauthor_t
        syb:has_titles syb:titles_iri(titles_tbl.title_id, titles_tbl.pub_id) where (^{titles_tbl.}^.titl

        syb:titles_iri (titles_tbl.title_id, titles_tbl.pub_id) a syb:titles as virtrdf:titles_pk;
        syb:title_id titles_tbl.title_id as virtrdf:titles_title_idd;
        syb:title titles_tbl.title as virtrdf:titles_title;
        syb:type titles_tbl.type as virtrdf:titles_type;
        syb:pub_id titles_tbl.pub_id as virtrdf:titles_pub_id;
        syb:price titles_tbl.price as virtrdf:titles_price;
        syb:advance titles_tbl.advance as virtrdf:titles_advance;
        syb:total_sales titles_tbl.total_sales as virtrdf:titles_total_sales;
        syb:notes titles_tbl.notes as virtrdf:titles_notes;
        syb:pubdate titles_tbl.pubdate as virtrdf:titles_pubdate;
        syb:contract titles_tbl.contract as virtrdf:titles_contract;
        syb:has_titleauthor syb:titleauthor_iri(titleauthor_tbl.au_id, titleauthor_tbl.title_id) where (^
        syb:has_salesdetail syb:salesdetail_iri (salesdetail_tbl.stor_id, salesdetail_tbl.ord_num, salesd

  } .
} .
;

delete from db.dba.url_rewrite_rule_list where urrl_list like 'sybasepubs2_rule%';
delete from db.dba.url_rewrite_rule where urr_rule like 'sybasepubs2_rule%';

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'sybasepubs2_rule1',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/about/html/http/^{URIQADefaultHost}^%s',
    vector('path'),
    null,
    '(text/html)|(\\*/\\*)',
    0,
    303
    );

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'sybasepubs2_rule2',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
```

```
    '/sparql?query=DESCRIBE+%%3Chttp%%3A//localhost%%3A8890%U%%23this%%3E+%%3Chttp%%3A//localhost%%3A8890
    vector('path', 'path', '*accept*'),
    null,
    '(text/rdf.n3)|(application/rdf.xml)',
    0,
    null
    );

DB.DBA.URLREWRITE_CREATE_RULELIST (
    'sybasepubs2_rule_list1',
    1,
    vector (
                'sybasepubs2_rule1',
                'sybasepubs2_rule2'
          ));

-- ensure a VD for the IRIs which begins with /
VHOST_REMOVE (lpath=>'/sybasepubs2');

VHOST_DEFINE (
        lpath=>'/sybasepubs2',
        ppath=>'/DAV/sybasepubs2/',
        is_dav=>1,
        vsp_user=>'dba',
        is_brws=>0,
        opts=>vector ('url_rewrite', 'sybasepubs2_rule_list1')
        );

delete from db.dba.url_rewrite_rule_list where urrl_list like 'sybase_schemas_rule%';
delete from db.dba.url_rewrite_rule where urr_rule like 'sybase_schemas_rule%';

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'sybase_schemas_rule1',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/about/html/http/^{URIQADefaultHost}^%s',
    vector('path'),
    null,
    '(text/html)|(\\*/\\*)',
    0,
    303
    );

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'sybase_schemas_rule2',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/sparql?query=CONSTRUCT+{+%%3Chttp%%3A//localhost%%3A8890%U%%3E+%%3Fp+%%3Fo+}%%0D%%0AFROM+%%3Chttp%%
    vector('path','path','*accept*'),
    null,
    '(text/rdf.n3)|(application/rdf.xml)',
    0,
    null
    );

DB.DBA.URLREWRITE_CREATE_RULELIST (
    'sybase_schemas_rule_list1',
    1,
    vector (
                'sybase_schemas_rule1',
                'sybase_schemas_rule2'
          ));

-- ensure a VD for the IRIs which begins with /
VHOST_REMOVE (lpath=>'/schema/sybasepubs2');

VHOST_DEFINE (
        lpath=>'/schemas/sybasepubs2',
        ppath=>'/DAV/schemas/sybasepubs2/',
```

```
        is_dav=>1,
        vsp_user=>'dba',
        is_brws=>0,
        opts=>vector ('url_rewrite', 'sybase_schemas_rule_list1')
        );

DB.DBA.XML_SET_NS_DECL ('hr', 'http://^{URIQADefaultHost}^/schemas/sybasepubs2', 2);
```

16.8.9. Virtuoso's Northwind based Demo Database (Tutorials variant) to RDF

```
use DB;

DB.DBA.exec_no_error('UPDATE WS.WS.SYS_DAV_RES set RES_TYPE=\'image/jpeg\' where RES_FULL_PATH like \'/DA
;

DB.DBA.exec_no_error('UPDATE WS.WS.SYS_DAV_RES set RES_TYPE=\'image/jpeg\' where RES_FULL_PATH like \'/DA
;

GRANT SELECT ON "Demo"."demo"."Products" TO "SPARQL";
GRANT SELECT ON "Demo"."demo"."Suppliers" TO "SPARQL";
GRANT SELECT ON "Demo"."demo"."Shippers" TO "SPARQL";
GRANT SELECT ON "Demo"."demo"."Categories" TO "SPARQL";
GRANT SELECT ON "Demo"."demo"."Customers" TO "SPARQL";
GRANT SELECT ON "Demo"."demo"."Employees" TO "SPARQL";
GRANT SELECT ON "Demo"."demo"."Orders" TO "SPARQL";
GRANT SELECT ON "Demo"."demo"."Order_Details" TO "SPARQL";
GRANT SELECT ON "Demo"."demo"."Countries" TO "SPARQL";
GRANT SELECT ON "Demo"."demo"."Provinces" TO "SPARQL";

SPARQL
prefix tut_northwind: <http://demo.openlinksw.com/schemas/tutorial/northwind#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix sioc: <http://rdfs.org/sioc/ns#>
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix wgs: <http://www.w3.org/2003/01/geo/wgs84_pos#>
drop quad map graph iri("http://^{URIQADefaultHost}^/tutorial/Northwind") .
;

SPARQL
prefix tut_northwind: <http://demo.openlinksw.com/schemas/tutorial/northwind#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix sioc: <http://rdfs.org/sioc/ns#>
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix wgs: <http://www.w3.org/2003/01/geo/wgs84_pos#>
drop quad map virtrdf:TutorialNorthwindDemo .
;

create function DB.DBA.TUT_NORTHWIND_ID_TO_IRI(in _prefix varchar,in _id varchar)
{
  declare iri, uriqa_host any;
  uriqa_host := cfg_item_value(virtuoso_ini_path(), 'URIQA','DefaultHost');
  iri := 'http://' || uriqa_host || '/tutorial/Northwind/' || _prefix || '/' || _id || '#this';
  return sprintf ('http://%s/DAV/VAD/tutorial/rdfview/rd_v_1/RDFData/All/iid%%20(%d).rdf', uriqa_host, ir
}
;

create function DB.DBA.TUT_NORTHWIND_IRI_TO_ID(in _iri varchar)
{
    declare parts any;
    parts := sprintf_inverse (_iri, 'http://%s/DAV/VAD/tutorial/rdfview/rd_v_1/RDFData/All/iid (%d).rdf',
    if (parts is not null)
    {
        declare uriqa_host, iri any;
        uriqa_host := cfg_item_value(virtuoso_ini_path(), 'URIQA','DefaultHost');
        if (parts[0] = uriqa_host)
        {
            iri := id_to_iri(iri_id_from_num(parts[1]));
            parts := sprintf_inverse (iri, 'http://%s/tutorial/Northwind/%s/%s#this', 1 );
            if (parts[0] = uriqa_host)
            {
                return parts[2];
```

```
            }
        }
    }
    return NULL;
}
;

create function DB.DBA.TUT_CATEGORY_IRI (in _id integer) returns varchar
{
    return TUT_NORTHWIND_ID_TO_IRI('Category', cast(_id as varchar));
}
;

create function DB.DBA.TUT_CATEGORY_IRI_INVERSE (in _iri varchar) returns integer
{
    return atoi(DB.DBA.TUT_NORTHWIND_IRI_TO_ID(_iri));
};

create function DB.DBA.TUT_SHIPPER_IRI (in _id integer) returns varchar
{
    return TUT_NORTHWIND_ID_TO_IRI('Shipper', cast(_id as varchar));
}
;

create function DB.DBA.TUT_SHIPPER_IRI_INVERSE (in _iri varchar) returns integer
{
    return atoi(DB.DBA.TUT_NORTHWIND_IRI_TO_ID(_iri));
};

create function DB.DBA.TUT_SUPPLIER_IRI (in _id integer) returns varchar
{
    return TUT_NORTHWIND_ID_TO_IRI('Supplier', cast(_id as varchar));
}
;

create function DB.DBA.TUT_SUPPLIER_IRI_INVERSE (in _iri varchar) returns integer
{
    return atoi(DB.DBA.TUT_NORTHWIND_IRI_TO_ID(_iri));
};

create function DB.DBA.TUT_PRODUCT_IRI (in _id integer) returns varchar
{
    return TUT_NORTHWIND_ID_TO_IRI('Product', cast(_id as varchar));
}
;

create function DB.DBA.TUT_PRODUCT_IRI_INVERSE (in _iri varchar) returns integer
{
    return atoi(DB.DBA.TUT_NORTHWIND_IRI_TO_ID(_iri));
};

create function DB.DBA.TUT_CUSTOMER_IRI (in _id varchar) returns varchar
{
    return TUT_NORTHWIND_ID_TO_IRI('Customer', _id);
}
;

create function DB.DBA.TUT_CUSTOMER_IRI_INVERSE (in _iri varchar) returns varchar
{
    return DB.DBA.TUT_NORTHWIND_IRI_TO_ID(_iri);
};

create function DB.DBA.TUT_EMPLOYEE_IRI (in _id integer) returns varchar
{
    return TUT_NORTHWIND_ID_TO_IRI('Employee', cast(_id as varchar));
}
;

create function DB.DBA.TUT_EMPLOYEE_IRI_INVERSE (in _iri varchar) returns integer
{
    return atoi(DB.DBA.TUT_NORTHWIND_IRI_TO_ID(_iri));
};
```

```
create function DB.DBA.TUT_ORDER_IRI (in _id integer) returns varchar
{
    return TUT_NORTHWIND_ID_TO_IRI('Order', cast(_id as varchar));
}
;

create function DB.DBA.TUT_ORDER_IRI_INVERSE (in _iri varchar) returns integer
{
    return atoi(DB.DBA.TUT_NORTHWIND_IRI_TO_ID(_iri));
};

create function DB.DBA.TUT_CUSTOMERCONTACT_IRI (in _id integer) returns varchar
{
    return TUT_NORTHWIND_ID_TO_IRI('CustomerContact', cast(_id as varchar));
}
;

create function DB.DBA.TUT_CUSTOMERCONTACT_IRI_INVERSE (in _iri varchar) returns integer
{
    return atoi(DB.DBA.TUT_NORTHWIND_IRI_TO_ID(_iri));
};

create function DB.DBA.TUT_ORDERLINE_IRI (in _id1 integer, in _id2 integer) returns varchar
{
    return TUT_NORTHWIND_ID_TO_IRI('OrderLine', sprintf('%d/%d', _id1, _id2));
}
;

create function DB.DBA.TUT_ORDERLINE_IRI_INV_1 (in _iri varchar) returns integer
{
    return atoi(DB.DBA.TUT_NORTHWIND_IRI_TO_ID(_iri));
};

create function DB.DBA.TUT_ORDERLINE_IRI_INV_2 (in _iri varchar) returns integer
{
    return atoi(DB.DBA.TUT_NORTHWIND_IRI_TO_ID(_iri));
};

create function DB.DBA.TUT_PROVINCE_IRI (in _id1 varchar, in _id2 varchar) returns varchar
{
    return TUT_NORTHWIND_ID_TO_IRI('Province', sprintf('%s/%s', _id1, _id2));
}
;

create function DB.DBA.TUT_PROVINCE_IRI_INV_1 (in _iri varchar) returns varchar
{
    return DB.DBA.TUT_NORTHWIND_IRI_TO_ID(_iri);
};

create function DB.DBA.TUT_PROVINCE_IRI_INV_2 (in _iri varchar) returns varchar
{
    return DB.DBA.TUT_NORTHWIND_IRI_TO_ID(_iri);
};

create function DB.DBA.TUT_COUNTRY_IRI (in _id varchar) returns varchar
{
    return TUT_NORTHWIND_ID_TO_IRI('Country', _id);
}
;

create function DB.DBA.TUT_COUNTRY_IRI_INVERSE (in _iri varchar) returns varchar
{
    return DB.DBA.TUT_NORTHWIND_IRI_TO_ID(_iri);
};

create function DB.DBA.TUT_FLAG_IRI (in _id varchar) returns varchar
{
    return TUT_NORTHWIND_ID_TO_IRI('Flag', _id);
}
;

create function DB.DBA.TUT_FLAG_IRI_INVERSE (in _iri varchar) returns varchar
{
```

```
    return DB.DBA.TUT_NORTHWIND_IRI_TO_ID(_iri);
};

create function DB.DBA.TUT_EMPLOYEEPHOTO_IRI (in _id integer) returns varchar
{
    return TUT_NORTHWIND_ID_TO_IRI('EmployeePhoto', cast(_id as varchar));
}
;

create function DB.DBA.TUT_EMPLOYEEPHOTO_IRI_INVERSE (in _iri varchar) returns integer
{
    return atoi(DB.DBA.TUT_NORTHWIND_IRI_TO_ID(_iri));
};

create function DB.DBA.TUT_CATEGORYPHOTO_IRI (in _id integer) returns varchar
{
    return TUT_NORTHWIND_ID_TO_IRI('CategoryPhoto', cast(_id as varchar));
}
;

create function DB.DBA.TUT_CATEGORYPHOTO_IRI_INVERSE (in _iri varchar) returns integer
{
    return atoi(DB.DBA.TUT_NORTHWIND_IRI_TO_ID(_iri));
};

grant execute on DB.DBA.TUT_CATEGORY_IRI to "SPARQL";
grant execute on DB.DBA.TUT_CATEGORY_IRI_INVERSE to "SPARQL";
grant execute on DB.DBA.TUT_SHIPPER_IRI to "SPARQL";
grant execute on DB.DBA.TUT_SHIPPER_IRI_INVERSE to "SPARQL";
grant execute on DB.DBA.TUT_SUPPLIER_IRI to "SPARQL";
grant execute on DB.DBA.TUT_SUPPLIER_IRI_INVERSE to "SPARQL";
grant execute on DB.DBA.TUT_PRODUCT_IRI to "SPARQL";
grant execute on DB.DBA.TUT_PRODUCT_IRI_INVERSE to "SPARQL";
grant execute on DB.DBA.TUT_CUSTOMER_IRI to "SPARQL";
grant execute on DB.DBA.TUT_CUSTOMER_IRI_INVERSE to "SPARQL";
grant execute on DB.DBA.TUT_EMPLOYEE_IRI to "SPARQL";
grant execute on DB.DBA.TUT_EMPLOYEE_IRI_INVERSE to "SPARQL";
grant execute on DB.DBA.TUT_ORDER_IRI to "SPARQL";
grant execute on DB.DBA.TUT_ORDER_IRI_INVERSE to "SPARQL";
grant execute on DB.DBA.TUT_CUSTOMERCONTACT_IRI to "SPARQL";
grant execute on DB.DBA.TUT_CUSTOMERCONTACT_IRI_INVERSE to "SPARQL";
grant execute on DB.DBA.TUT_ORDERLINE_IRI to "SPARQL";
grant execute on DB.DBA.TUT_ORDERLINE_IRI_INV_1 to "SPARQL";
grant execute on DB.DBA.TUT_ORDERLINE_IRI_INV_2 to "SPARQL";
grant execute on DB.DBA.TUT_PROVINCE_IRI to "SPARQL";
grant execute on DB.DBA.TUT_PROVINCE_IRI_INV_1 to "SPARQL";
grant execute on DB.DBA.TUT_PROVINCE_IRI_INV_2 to "SPARQL";
grant execute on DB.DBA.TUT_COUNTRY_IRI to "SPARQL";
grant execute on DB.DBA.TUT_COUNTRY_IRI_INVERSE to "SPARQL";
grant execute on DB.DBA.TUT_FLAG_IRI to "SPARQL";
grant execute on DB.DBA.TUT_FLAG_IRI_INVERSE to "SPARQL";
grant execute on DB.DBA.TUT_EMPLOYEEPHOTO_IRI to "SPARQL";
grant execute on DB.DBA.TUT_EMPLOYEEPHOTO_IRI_INVERSE to "SPARQL";
grant execute on DB.DBA.TUT_CATEGORYPHOTO_IRI to "SPARQL";
grant execute on DB.DBA.TUT_CATEGORYPHOTO_IRI_INVERSE to "SPARQL";

SPARQL
prefix tut_northwind: <http://demo.openlinksw.com/schemas/tutorial/northwind#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix sioc: <http://rdfs.org/sioc/ns#>
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix wgs: <http://www.w3.org/2003/01/geo/wgs84_pos#>
create iri class tut_northwind:Category "http://^{URIQADefaultHost}^/tutorial/Northwind/Category/%d#this"
create iri class tut_northwind:Shipper "http://^{URIQADefaultHost}^/tutorial/Northwind/Shipper/%d#this" (
create iri class tut_northwind:Supplier "http://^{URIQADefaultHost}^/tutorial/Northwind/Supplier/%d#this"
create iri class tut_northwind:Product   "http://^{URIQADefaultHost}^/tutorial/Northwind/Product/%d#this"
create iri class tut_northwind:Customer "http://^{URIQADefaultHost}^/tutorial/Northwind/Customer/%U#this"
create iri class tut_northwind:Employee "http://^{URIQADefaultHost}^/tutorial/Northwind/Employee/%U%U%d#t
create iri class tut_northwind:Order "http://^{URIQADefaultHost}^/tutorial/Northwind/Order/%d#this" (in o
create iri class tut_northwind:CustomerContact "http://^{URIQADefaultHost}^/tutorial/Northwind/CustomerCo
create iri class tut_northwind:OrderLine "http://^{URIQADefaultHost}^/tutorial/Northwind/OrderLine/%d/%d#
create iri class tut_northwind:Province "http://^{URIQADefaultHost}^/tutorial/Northwind/Province/%U/%U#th
```

```
create iri class tut_northwind:Country "http://^{URIQADefaultHost}^/tutorial/Northwind/Country/%U#this" (
create iri class tut_northwind:Flag "http://^{URIQADefaultHost}^%U#this" (in flag_path varchar not null)
create iri class tut_northwind:dbpedia_iri "http://dbpedia.org/resource/%U" (in uname varchar not null) .
create iri class tut_northwind:EmployeePhoto "http://^{URIQADefaultHost}^/DAV/VAD/demo/sql/EMP%d#this" (i
create iri class tut_northwind:CategoryPhoto "http://^{URIQADefaultHost}^/DAV/VAD/demo/sql/CAT%d#this" (i
;

SPARQL
prefix tut_northwind: <http://demo.openlinksw.com/schemas/tutorial/northwind#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix sioc: <http://rdfs.org/sioc/ns#>
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix wgs: <http://www.w3.org/2003/01/geo/wgs84_pos#>
create iri class tut_northwind:customercontact_iri using
    function DB.DBA.TUT_CUSTOMERCONTACT_IRI (in customer_id varchar) returns varchar,
    function DB.DBA.TUT_CUSTOMERCONTACT_IRI_INVERSE (in customer_iri varchar) returns varchar.
create iri class tut_northwind:category_iri using
    function DB.DBA.TUT_CATEGORY_IRI (in customer_id integer) returns varchar,
    function DB.DBA.TUT_CATEGORY_IRI_INVERSE (in customer_iri varchar) returns integer.
create iri class tut_northwind:shipper_iri using
    function DB.DBA.TUT_SHIPPER_IRI (in customer_id integer) returns varchar,
    function DB.DBA.TUT_SHIPPER_IRI_INVERSE (in customer_iri varchar) returns integer.
create iri class tut_northwind:supplier_iri using
    function DB.DBA.TUT_SUPPLIER_IRI (in customer_id varchar) returns varchar,
    function DB.DBA.TUT_SUPPLIER_IRI_INVERSE (in customer_iri varchar) returns varchar.
create iri class tut_northwind:product_iri using
    function DB.DBA.TUT_PRODUCT_IRI (in customer_id integer) returns varchar,
    function DB.DBA.TUT_PRODUCT_IRI_INVERSE (in customer_iri varchar) returns integer.
create iri class tut_northwind:customer_iri using
    function DB.DBA.TUT_CUSTOMER_IRI (in customer_id varchar) returns varchar,
    function DB.DBA.TUT_CUSTOMER_IRI_INVERSE (in customer_iri varchar) returns varchar.
create iri class tut_northwind:employee_iri using
    function DB.DBA.TUT_EMPLOYEE_IRI (in customer_id integer) returns varchar,
    function DB.DBA.TUT_EMPLOYEE_IRI_INVERSE (in customer_iri varchar) returns integer.
create iri class tut_northwind:order_iri using
    function DB.DBA.TUT_ORDER_IRI (in customer_id integer) returns varchar,
    function DB.DBA.TUT_ORDER_IRI_INVERSE (in customer_iri varchar) returns integer.
create iri class tut_northwind:orderline_iri using
    function DB.DBA.TUT_ORDERLINE_IRI (in customer_id integer, in customer_id2 integer) returns varchar,
    function DB.DBA.TUT_ORDERLINE_IRI_INV_1 (in customer_iri varchar) returns integer,
    function DB.DBA.TUT_ORDERLINE_IRI_INV_2 (in customer_iri varchar) returns integer.
create iri class tut_northwind:province_iri using
    function DB.DBA.TUT_PROVINCE_IRI (in customer_id varchar, in customer_id2 varchar) returns varchar,
    function DB.DBA.TUT_PROVINCE_IRI_INV_1 (in customer_iri varchar) returns varchar,
    function DB.DBA.TUT_PROVINCE_IRI_INV_2 (in customer_iri varchar) returns varchar.
create iri class tut_northwind:country_iri using
    function DB.DBA.TUT_COUNTRY_IRI (in customer_id varchar) returns varchar,
    function DB.DBA.TUT_COUNTRY_IRI_INVERSE (in customer_iri varchar) returns varchar.
create iri class tut_northwind:employeephoto_iri using
    function DB.DBA.TUT_EMPLOYEEPHOTO_IRI (in customer_id integer) returns varchar,
    function DB.DBA.TUT_EMPLOYEEPHOTO_IRI_INVERSE (in customer_iri varchar) returns integer.
create iri class tut_northwind:categoryphoto_iri using
    function DB.DBA.TUT_CATEGORYPHOTO_IRI (in customer_id integer) returns varchar,
    function DB.DBA.TUT_CATEGORYPHOTO_IRI_INVERSE (in customer_iri varchar) returns integer.
create iri class tut_northwind:flag_iri using
    function DB.DBA.TUT_FLAG_IRI (in customer_id varchar) returns varchar,
    function DB.DBA.TUT_FLAG_IRI_INVERSE (in customer_iri varchar) returns varchar.
;

SPARQL
prefix tut_northwind: <http://demo.openlinksw.com/schemas/tutorial/northwind#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix sioc: <http://rdfs.org/sioc/ns#>
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix wgs: <http://www.w3.org/2003/01/geo/wgs84_pos#>
alter quad storage virtrdf:DefaultQuadStorage
from Demo.demo.Products as products
from Demo.demo.Suppliers as suppliers
from Demo.demo.Shippers as shippers
from Demo.demo.Categories as categories
from Demo.demo.Customers as customers
```

```
from Demo.demo.Employees as employees
from Demo.demo.Orders as orders
from Demo.demo.Order_Details as order_lines
from Demo.demo.Countries as countries
from Demo.demo.Provinces as provinces
where (^{suppliers.}^.Country = ^{countries.}^.Name)
where (^{customers.}^.Country = ^{countries.}^.Name)
where (^{employees.}^.Country = ^{countries.}^.Name)
where (^{orders.}^.ShipCountry = ^{countries.}^.Name)
{
        create virtrdf:TutorialNorthwindDemo as graph iri ("http://^{URIQADefaultHost}^/tutorial/Northwin
        {
                tut_northwind:CustomerContact (customers.CustomerID)
                        a foaf:Person
                                as virtrdf:tutCustomerContact-foaf_Person .

                tut_northwind:CustomerContact (customers.CustomerID)
                        a tut_northwind:CustomerContact
                                as virtrdf:tutCustomerContact-CustomerContact;
                        foaf:name customers.ContactName
                                as virtrdf:tutCustomerContact-contact_name ;
                        foaf:phone customers.Phone
                                as virtrdf:tutCustomerContact-foaf_phone ;
                        tut_northwind:is_contact_at tut_northwind:Customer (customers.CustomerID)
                                as virtrdf:tutCustomerContact-is_contact_at ;
                        tut_northwind:country tut_northwind:Country (customers.Country)
                                as virtrdf:tutCustomerContact-country ;
                        rdfs:isDefinedBy tut_northwind:customercontact_iri (customers.CustomerID) ;
                        rdfs:isDefinedBy tut_northwind:CustomerContact (customers.CustomerID) .

                tut_northwind:Country (customers.Country)
                        tut_northwind:is_country_of
                tut_northwind:CustomerContact (customers.CustomerID) as virtrdf:tutCustomerContact-is_cou

                tut_northwind:Product (products.ProductID)
                        a tut_northwind:Product
                                as virtrdf:tutProduct-ProductID ;
                        tut_northwind:has_category tut_northwind:Category (products.CategoryID)
                                as virtrdf:tutProduct-product_has_category ;
                        tut_northwind:has_supplier tut_northwind:Supplier (products.SupplierID)
                                as virtrdf:tutProduct-product_has_supplier ;
                        tut_northwind:productName products.ProductName
                                as virtrdf:tutProduct-name_of_product ;
                        tut_northwind:quantityPerUnit products.QuantityPerUnit
                                as virtrdf:tutProduct-quantity_per_unit ;
                        tut_northwind:unitPrice products.UnitPrice
                                as virtrdf:tutProduct-unit_price ;
                        tut_northwind:unitsInStock products.UnitsInStock
                                as virtrdf:tutProduct-units_in_stock ;
                        tut_northwind:unitsOnOrder products.UnitsOnOrder
                                as virtrdf:tutProduct-units_on_order ;
                        tut_northwind:reorderLevel products.ReorderLevel
                                as virtrdf:tutProduct-reorder_level ;
                        tut_northwind:discontinued products.Discontinued
                                as virtrdf:tutProduct-discontinued ;
                        rdfs:isDefinedBy tut_northwind:product_iri (products.ProductID) ;
                        rdfs:isDefinedBy tut_northwind:Product (products.ProductID).

                tut_northwind:Category (products.CategoryID)
                        tut_northwind:category_of tut_northwind:Product (products.ProductID) as virtrdf:t

                tut_northwind:Supplier (products.SupplierID)
                        tut_northwind:supplier_of tut_northwind:Product (products.ProductID) as virtrdf:t

                tut_northwind:Supplier (suppliers.SupplierID)
                        a tut_northwind:Supplier
                                as virtrdf:tutSupplier-SupplierID ;
                        tut_northwind:companyName suppliers.CompanyName
                                as virtrdf:tutSupplier-company_name ;
                        tut_northwind:contactName suppliers.ContactName
                                as virtrdf:tutSupplier-contact_name ;
                        tut_northwind:contactTitle suppliers.ContactTitle
                                as virtrdf:tutSupplier-contact_title ;
```

```
                tut_northwind:address suppliers.Address
                        as virtrdf:tutSupplier-address ;
                tut_northwind:city suppliers.City
                        as virtrdf:tutSupplier-city ;
                tut_northwind:dbpedia_city tut_northwind:dbpedia_iri(suppliers.City)
                        as virtrdf:tutSupplier-dbpedia_city ;
                tut_northwind:region suppliers.Region
                        as virtrdf:tutSupplier-region ;
                tut_northwind:postalCode suppliers.PostalCode
                        as virtrdf:tutSupplier-postal_code ;
                tut_northwind:country tut_northwind:Country(suppliers.Country)
                        as virtrdf:tutSupplier-country ;
                tut_northwind:phone suppliers.Phone
                        as virtrdf:tutSupplier-phone ;
                tut_northwind:fax suppliers.Fax
                        as virtrdf:tutSupplier-fax ;
                tut_northwind:homePage suppliers.HomePage
                        as virtrdf:tutSupplier-home_page ;
                rdfs:isDefinedBy tut_northwind:supplier_iri (suppliers.SupplierID) ;
                rdfs:isDefinedBy tut_northwind:Supplier (suppliers.SupplierID).

        tut_northwind:Country (suppliers.Country)
                tut_northwind:is_country_of
        tut_northwind:Supplier (suppliers.SupplierID) as virtrdf:tutSupplier-is_country_of .

        tut_northwind:Category (categories.CategoryID)
                a tut_northwind:Category
                        as virtrdf:tutCategory-CategoryID ;
                tut_northwind:categoryName categories.CategoryName
                        as virtrdf:tutCategory-home_page ;
                tut_northwind:description categories.Description
                        as virtrdf:tutCategory-description ;
                foaf:img tut_northwind:CategoryPhoto(categories.CategoryID)
                        as virtrdf:tutCategory-categories.CategoryPhoto ;
                rdfs:isDefinedBy tut_northwind:category_iri (categories.CategoryID) ;
                rdfs:isDefinedBy tut_northwind:Category (categories.CategoryID).

        tut_northwind:CategoryPhoto(categories.CategoryID)
                a tut_northwind:CategoryPhoto
                        as virtrdf:tutCategory-categories.CategoryPhotoID ;
                rdfs:isDefinedBy tut_northwind:categoryphoto_iri (categories.CategoryID) ;
                rdfs:isDefinedBy tut_northwind:CategoryPhoto(categories.CategoryID).

        tut_northwind:Shipper (shippers.ShipperID)
                a tut_northwind:Shipper
                        as virtrdf:tutShipper-ShipperID ;
                tut_northwind:companyName shippers.CompanyName
                        as virtrdf:tutShipper-company_name ;
                tut_northwind:phone shippers.Phone
                        as virtrdf:tutShipper-phone ;
                rdfs:isDefinedBy tut_northwind:shipper_iri (shippers.ShipperID) ;
                rdfs:isDefinedBy tut_northwind:Shipper (shippers.ShipperID).

        tut_northwind:Customer (customers.CustomerID)
                a  tut_northwind:Customer
                        as virtrdf:tutCustomer-CustomerID2 ;
                a  foaf:Organization
                        as virtrdf:tutCustomer-CustomerID ;
                foaf:name customers.CompanyName
                        as virtrdf:tutCustomer-foaf_name ;
                tut_northwind:companyName customers.CompanyName
                        as virtrdf:tutCustomer-company_name ;
                tut_northwind:has_contact tut_northwind:CustomerContact (customers.CustomerID)
                        as virtrdf:tutCustomer-contact ;
                tut_northwind:country tut_northwind:Country (customers.Country)
                        as virtrdf:tutCustomer-country ;
                tut_northwind:contactName customers.ContactName
                        as virtrdf:tutCustomer-contact_name ;
                tut_northwind:contactTitle customers.ContactTitle
                        as virtrdf:tutCustomer-contact_title ;
                tut_northwind:address customers.Address
                        as virtrdf:tutCustomer-address ;
                tut_northwind:city customers.City
```

```
                as virtrdf:tutCustomer-city ;
        tut_northwind:dbpedia_city tut_northwind:dbpedia_iri(customers.City)
                as virtrdf:tutCustomer-dbpedia_city ;
        tut_northwind:region customers.Region
                as virtrdf:tutCustomer-region ;
        tut_northwind:PostalCode customers.PostalCode
                as virtrdf:tutCustomer-postal_code ;
        foaf:phone customers.Phone
                as virtrdf:tutCustomer-foaf_phone ;
        tut_northwind:phone customers.Phone
                as virtrdf:tutCustomer-phone ;
        tut_northwind:fax customers.Fax
                as virtrdf:tutCustomer-fax ;
        rdfs:isDefinedBy tut_northwind:customer_iri (customers.CustomerID) ;
        rdfs:isDefinedBy tut_northwind:Customer (customers.CustomerID).

tut_northwind:Country (customers.Country)
        tut_northwind:is_country_of
tut_northwind:Customer (customers.CustomerID) as virtrdf:tutCustomer-is_country_of .

tut_northwind:Employee (employees.FirstName, employees.LastName, employees.EmployeeID)
        a tut_northwind:Employee
                as virtrdf:tutEmployee-EmployeeID2 ;
        a foaf:Person
                as virtrdf:tutEmployee-EmployeeID ;
        foaf:surname employees.LastName
                as virtrdf:tutEmployee-foaf_last_name ;
        tut_northwind:lastName employees.LastName
                as virtrdf:tutEmployee-last_name ;
        foaf:firstName employees.FirstName
                as virtrdf:tutEmployee-foaf_first_name ;
        tut_northwind:firstName employees.FirstName
                as virtrdf:tutEmployee-first_name ;
        foaf:title employees.Title
                as virtrdf:tutEmployee-title ;
        tut_northwind:titleOfCourtesy employees.TitleOfCourtesy
                as virtrdf:tutEmployee-title_of_courtesy ;
        foaf:birthday employees.BirthDate
                as virtrdf:tutEmployee-foaf_birth_date ;
        tut_northwind:birthday employees.BirthDate
                as virtrdf:tutEmployee-birth_date ;
        tut_northwind:hireDate employees.HireDate
                as virtrdf:tutEmployee-hire_date ;
        tut_northwind:address employees.Address
                as virtrdf:tutEmployee-address ;
        tut_northwind:city employees.City
                as virtrdf:tutEmployee-city ;
        tut_northwind:dbpedia_city tut_northwind:dbpedia_iri(employees.City)
                as virtrdf:tutEmployee-dbpedia_city ;
        tut_northwind:region employees.Region
                as virtrdf:tutEmployee-region ;
        tut_northwind:postalCode employees.PostalCode
                as virtrdf:tutEmployee-postal_code ;
        tut_northwind:country tut_northwind:Country (employees.Country)
                as virtrdf:tutEmployee-country ;
        foaf:phone employees.HomePhone
                as virtrdf:tutEmployee-home_phone ;
        tut_northwind:extension employees.Extension
                as virtrdf:tutEmployee-extension ;
        tut_northwind:notes employees.Notes
                as virtrdf:tutEmployee-notes ;
        tut_northwind:reportsTo tut_northwind:Employee(employees.FirstName, employees.Las
                as virtrdf:tutEmployee-reports_to ;
        foaf:img tut_northwind:EmployeePhoto(employees.EmployeeID)
                as virtrdf:tutEmployee-employees.EmployeePhoto ;
        rdfs:isDefinedBy tut_northwind:employee_iri (employees.EmployeeID) ;
        rdfs:isDefinedBy tut_northwind:Employee (employees.FirstName, employees.LastName,

tut_northwind:EmployeePhoto(employees.EmployeeID)
        a tut_northwind:EmployeePhoto
                as virtrdf:tut_Employee-employees.EmployeePhotoId ;
        rdfs:isDefinedBy tut_northwind:employeephoto_iri (employees.EmployeeID) ;
        rdfs:isDefinedBy tut_northwind:EmployeePhoto (employees.EmployeeID).
```

```
tut_northwind:Employee (employees.FirstName, employees.LastName, orders.EmployeeID)
        tut_northwind:is_salesrep_of
tut_northwind:Order (orders.OrderID) where (^{orders.}^.EmployeeID = ^{employees.}^.Emplo

tut_northwind:Country (employees.Country)
        tut_northwind:is_country_of
tut_northwind:Employee (employees.FirstName, employees.LastName, employees.EmployeeID) as

tut_northwind:Order (orders.OrderID)
        a tut_northwind:Order
                as virtrdf:tutOrder-Order ;
        tut_northwind:has_customer tut_northwind:Customer (orders.CustomerID)
                as virtrdf:tutOrder-order_has_customer ;
        tut_northwind:has_salesrep tut_northwind:Employee (employees.FirstName, employees
                as virtrdf:tutCustomer-has_salesrep ;
        tut_northwind:has_employee tut_northwind:Employee (employees.FirstName, employees
                as virtrdf:tutOrder-order_has_employee ;
        tut_northwind:orderDate orders.OrderDate
                as virtrdf:tutOrder-order_date ;
        tut_northwind:requiredDate orders.RequiredDate
                as virtrdf:tutOrder-required_date ;
        tut_northwind:shippedDate orders.ShippedDate
                as virtrdf:tutOrder-shipped_date ;
        tut_northwind:order_ship_via tut_northwind:Shipper (orders.ShipVia)
                as virtrdf:tutOrder-order_ship_via ;
        tut_northwind:freight orders.Freight
                as virtrdf:tutOrder-freight ;
        tut_northwind:shipName orders.ShipName
                as virtrdf:tutOrder-ship_name ;
        tut_northwind:shipAddress orders.ShipAddress
                as virtrdf:tutOrder-ship_address ;
        tut_northwind:shipCity orders.ShipCity
                as virtrdf:tutOrder-ship_city ;
        tut_northwind:dbpedia_shipCity tut_northwind:dbpedia_iri(orders.ShipCity)
                as virtrdf:tutOrder-ship_dbpedia_city ;
        tut_northwind:shipRegion orders.ShipRegion
                as virtrdf:tutOrder-ship_region ;
        tut_northwind:shipPostal_code orders.ShipPostalCode
                as virtrdf:tutOrder-ship_postal_code ;
        tut_northwind:shipCountry tut_northwind:Country(orders.ShipCountry)
                as virtrdf:tutship_country ;
        rdfs:isDefinedBy tut_northwind:order_iri (orders.OrderID) ;
        rdfs:isDefinedBy tut_northwind:Order (orders.OrderID).

tut_northwind:Country (orders.ShipCountry)
        tut_northwind:is_ship_country_of
tut_northwind:Order (orders.OrderID) as virtrdf:tutOrder-is_country_of .

tut_northwind:Customer (orders.CustomerID)
        tut_northwind:has_order tut_northwind:Order (orders.OrderID) as virtrdf:tutOrder-

tut_northwind:Shipper (orders.ShipVia)
        tut_northwind:ship_order tut_northwind:Order (orders.OrderID) as virtrdf:tutOrder

tut_northwind:OrderLine (order_lines.OrderID, order_lines.ProductID)
        a tut_northwind:OrderLine
                as virtrdf:tutOrderLine-OrderLines ;
        tut_northwind:has_order_id tut_northwind:Order (order_lines.OrderID)
                as virtrdf:tutorder_lines_has_order_id ;
        tut_northwind:has_product_id tut_northwind:Product (order_lines.ProductID)
                as virtrdf:tutorder_lines_has_product_id ;
        tut_northwind:unitPrice order_lines.UnitPrice
                as virtrdf:tutOrderLine-unit_price ;
        tut_northwind:quantity order_lines.Quantity
                as virtrdf:tutOrderLine-quantity ;
        tut_northwind:discount order_lines.Discount
                as virtrdf:tutOrderLine-discount ;
        rdfs:isDefinedBy tut_northwind:orderline_iri (order_lines.OrderID, order_lines.Pr
        rdfs:isDefinedBy tut_northwind:OrderLine (order_lines.OrderID, order_lines.Produc

tut_northwind:Order (orders.OrderID)
        tut_northwind:is_order_of
```

```
                tut_northwind:OrderLine (order_lines.OrderID, order_lines.ProductID) where (^{orders.}^.O

                tut_northwind:Product (products.ProductID)
                        tut_northwind:is_product_of
                tut_northwind:OrderLine (order_lines.OrderID, order_lines.ProductID) where (^{products.}^

                tut_northwind:Country (countries.Name)
                        a tut_northwind:Country
                                as virtrdf:tutCountry-Type2 ;
                        a wgs:SpatialThing
                                as virtrdf:tutCountry-Type ;
                        owl:sameAs tut_northwind:dbpedia_iri (countries.Name) ;
                        tut_northwind:name countries.Name
                                as virtrdf:tutCountry-Name ;
                        tut_northwind:code countries.Code
                                as virtrdf:tutCountry-Code ;
                        tut_northwind:smallFlagDAVResourceName countries.SmallFlagDAVResourceName
                                as virtrdf:tutCountry-SmallFlagDAVResourceName ;
                        tut_northwind:largeFlagDAVResourceName countries.LargeFlagDAVResourceName
                                as virtrdf:tutCountry-LargeFlagDAVResourceName ;
                        tut_northwind:smallFlagDAVResourceURI tut_northwind:Flag(countries.SmallFlagDAVRe
                                as virtrdf:tutCountry-SmallFlagDAVResourceURI ;
                        tut_northwind:largeFlagDAVResourceURI tut_northwind:Flag(countries.LargeFlagDAVRe
                                as virtrdf:tutCountry-LargeFlagDAVResourceURI ;
                        wgs:lat countries.Lat
                                as virtrdf:tutCountry-Lat ;
                        wgs:long countries.Lng
                                as virtrdf:tutCountry-Lng ;
                        rdfs:isDefinedBy tut_northwind:country_iri (countries.Name) ;
                        rdfs:isDefinedBy tut_northwind:Country (countries.Name).

                tut_northwind:Country (countries.Name)
                        tut_northwind:has_province
                tut_northwind:Province (provinces.CountryCode, provinces.Province) where (^{provinces.}^.

                tut_northwind:Province (provinces.CountryCode, provinces.Province)
                        a tut_northwind:Province
                                as virtrdf:tutProvince-Provinces ;
                        tut_northwind:has_country_code provinces.CountryCode
                                as virtrdf:tuthas_country_code ;
                        tut_northwind:provinceName provinces.Province
                                as virtrdf:tutProvince-ProvinceName ;
                        rdfs:isDefinedBy tut_northwind:province_iri (provinces.CountryCode, provinces.Pro
                        rdfs:isDefinedBy tut_northwind:Province (provinces.CountryCode, provinces.Provinc

                tut_northwind:Province (provinces.CountryCode, provinces.Province)
                        tut_northwind:is_province_of
                tut_northwind:Country (countries.Name) where  (^{countries.}^.Code = ^{provinces.}^.Count
        } .
} .
;

DELETE FROM db.dba.url_rewrite_rule_list where urrl_list like 'tut_nw%';
DELETE FROM db.dba.url_rewrite_rule where urr_rule like 'tut_nw%';

create procedure DB.DBA.install_run ()
{
        declare file_text, uriqa varchar;
        uriqa := registry_get('URIQADefaultHost');
        file_text := (SELECT blob_to_string (RES_CONTENT) FROM WS.WS.SYS_DAV_RES where RES_FULL_PATH='/DA
        file_text := replace(file_text, 'URIQA_MACRO', concat('http://', uriqa, '/tutorial/Northwind'));
        update WS.WS.SYS_DAV_RES set RES_CONTENT=file_text where RES_FULL_PATH='/DAV/VAD/tutorial/rdfview
}
;

DB.DBA.install_run()
;

drop procedure DB.DBA.install_run
;

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'tut_nw_rule2',
```

```
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/sparql?query=CONSTRUCT+{+%%3Chttp%%3A//^{URIQADefaultHost}^%U%%23this%%3E+%%3Fp+%%3Fo+}+FROM+%%3Cht
    vector('path', 'path', '*accept*'),
    null,
    '(text/rdf.n3)|(application/rdf.xml)',
    0,
    null
    );

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'tut_nw_rule1',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/rdfbrowser/index.html?uri=http%%3A//^{URIQADefaultHost}^%U%%23this',
    vector('path'),
    null,
    '(text/html)|(\\*/\\*)',
    0,
    303
    );

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'tut_nw_rule3',
    1,
    '(/[^#]*)/\x24',
    vector('path'),
    1,
    '%s',
    vector('path'),
    null,
    null,
    0,
    null
    );

create procedure DB.DBA.REMOVE_TUT_DEMO_RDF_DET()
{
  declare colid int;
  colid := DAV_SEARCH_ID('/DAV/VAD/tutorial/rdfview/rd_v_1/', 'C');
  if (colid < 0)
    return;
  update WS.WS.SYS_DAV_COL set COL_DET=null where COL_ID = colid;
}
;

DB.DBA.REMOVE_TUT_DEMO_RDF_DET();

drop procedure DB.DBA.REMOVE_TUT_DEMO_RDF_DET;

create procedure DB.DBA.TUT_NORTHWIND_MAKE_RDF_DET()
{
    declare uriqa_str varchar;
    uriqa_str := cfg_item_value(virtuoso_ini_path(), 'URIQA','DefaultHost');
    uriqa_str := 'http://' || uriqa_str || '/tutorial/Northwind';
    DB.DBA."RDFData_MAKE_DET_COL" ('/DAV/VAD/tutorial/rdfview/rd_v_1/RDFData/', uriqa_str, NULL);
    VHOST_REMOVE (lpath=>'/tutorial/Northwind/data/rdf');
    DB.DBA.VHOST_DEFINE (lpath=>'/tutorial/Northwind/data/rdf', ppath=>'/DAV/VAD/tutorial/rdfview/rd_v_1/
}
;

DB.DBA.TUT_NORTHWIND_MAKE_RDF_DET();

drop procedure DB.DBA.TUT_NORTHWIND_MAKE_RDF_DET;

-- procedure to convert path to DET resource name
create procedure DB.DBA.TUT_NORTHWIND_DET_REF (in par varchar, in fmt varchar, in val varchar)
{
  declare res, iri any;
```

```
  declare uriqa_str varchar;
  uriqa_str := cfg_item_value(virtuoso_ini_path(), 'URIQA','DefaultHost');
  uriqa_str := 'http://' || uriqa_str || '/tutorial/Northwind';
  iri := uriqa_str || val;
  res := sprintf ('iid (%d).rdf', iri_id_num (iri_to_id (iri)));
  return sprintf (fmt, res);
}
;

DB.DBA.URLREWRITE_CREATE_REGEX_RULE ('tut_nw_rdf', 1,
    '/tutorial/Northwind/(.*)', vector('path'), 1,
    '/tutorial/Northwind/data/rdf/%U', vector('path'),
    'DB.DBA.TUT_NORTHWIND_DET_REF',
    'application/rdf.xml',
    2,
    303);

DB.DBA.URLREWRITE_CREATE_RULELIST (
    'tut_nw_rule_list1',
    1,
    vector (
                'tut_nw_rule1',
                'tut_nw_rule2',
                'tut_nw_rule3',
                'tut_nw_rdf'
          ));

VHOST_REMOVE (lpath=>'/tutorial/Northwind');
DB.DBA.VHOST_DEFINE (lpath=>'/tutorial/Northwind', ppath=>'/DAV/VAD/tutorial/rdfview/rd_v_1/', vsp_user=>
          is_brws=>0, opts=>vector ('url_rewrite', 'tut_nw_rule_list1'));

create procedure DB.DBA.LOAD_TUTNW_ONTOLOGY_FROM_DAV()
{
  declare content, urihost varchar;
  whenever not found goto endpoint;
  SELECT cast (RES_CONTENT as varchar) into content FROM WS.WS.SYS_DAV_RES where RES_FULL_PATH = '/DAV/VA
  if (content is null or content = '')
    goto endpoint;
  DB.DBA.RDF_LOAD_RDFXML (content, 'http://demo.openlinksw.com/schemas/tutorial/northwind#', 'http://demo
  if (urihost = 'demo.openlinksw.com')
  {
    DB.DBA.VHOST_REMOVE (lpath=>'/schemas/tutorial/northwind#');
    DB.DBA.VHOST_DEFINE (lpath=>'/schemas/tutorial/northwind#', ppath=>'/DAV/VAD/tutorial/rdfview/rd_v_1/
    DB.DBA.VHOST_REMOVE (lpath=>'/schemas/tutorial/northwind');
    DB.DBA.VHOST_DEFINE (lpath=>'/schemas/tutorial/northwind', ppath=>'/DAV/VAD/tutorial/rdfview/rd_v_1/r
  }
  endpoint:
  ;
}
;

DB.DBA.LOAD_TUTNW_ONTOLOGY_FROM_DAV()
;

drop procedure DB.DBA.LOAD_TUTNW_ONTOLOGY_FROM_DAV
;

create procedure DB.DBA.LOAD_TUTNW_ONTOLOGY_FROM_DAV2()
{
  declare urihost varchar;
  sparql base <http://demo.openlinksw.com/schemas/tutorial/northwind#> load bif:concat ("http://", bif:re
   into graph <http://demo.openlinksw.com/schemas/TutorialNorthwindOntology/1.0/>;
  urihost := cfg_item_value(virtuoso_ini_path(), 'URIQA','DefaultHost');
  if (urihost = 'demo.openlinksw.com')
  {
    DB.DBA.VHOST_REMOVE (lpath=>'/tutorial/northwind#');
    DB.DBA.VHOST_DEFINE (lpath=>'/tutorial/northwind#', ppath=>'/DAV/VAD/tutorial/rdfview/rd_v_1/rd_v_1.o
  }
}
;

--DB.DBA.LOAD_TUTNW_ONTOLOGY_FROM_DAV2();
```

```
drop procedure DB.DBA.LOAD_TUTNW_ONTOLOGY_FROM_DAV2
;

DB.DBA.XML_SET_NS_DECL ('tut_northwind', 'http://demo.openlinksw.com/schemas/tutorial/northwind#', 2);
```

16.8.10. SQL Server's Northwind Demo Database

```
use DB;

DB.DBA.exec_stmt ('UPDATE WS.WS.SYS_DAV_RES set RES_TYPE=\'image/jpeg\' where RES_FULL_PATH like \'/DAV/V
;

DB.DBA.exec_stmt ('UPDATE WS.WS.SYS_DAV_RES set RES_TYPE=\'image/jpeg\' where RES_FULL_PATH like \'/DAV/V
;

GRANT SELECT ON "Demo"."demo"."Products" TO "SPARQL";
GRANT SELECT ON "Demo"."demo"."Suppliers" TO "SPARQL";
GRANT SELECT ON "Demo"."demo"."Shippers" TO "SPARQL";
GRANT SELECT ON "Demo"."demo"."Categories" TO "SPARQL";
GRANT SELECT ON "Demo"."demo"."Customers" TO "SPARQL";
GRANT SELECT ON "Demo"."demo"."Employees" TO "SPARQL";
GRANT SELECT ON "Demo"."demo"."Orders" TO "SPARQL";
GRANT SELECT ON "Demo"."demo"."Order_Details" TO "SPARQL";
GRANT SELECT ON "Demo"."demo"."Countries" TO "SPARQL";
GRANT SELECT ON "Demo"."demo"."Provinces" TO "SPARQL";

SPARQL drop quad map graph iri("http://^{URIQADefaultHost}^/Northwind") .
;

SPARQL drop quad map virtrdf:NorthwindDemo .
;

SPARQL
prefix northwind: <http://demo.openlinksw.com/schemas/northwind#>
drop iri class northwind:Category .
drop iri class northwind:CategoryDoc .
drop iri class northwind:Shipper .
drop iri class northwind:ShipperDoc .
drop iri class northwind:Supplier .
drop iri class northwind:SupplierDoc .
drop iri class northwind:Product .
drop iri class northwind:ProductDoc .
drop iri class northwind:Customer .
drop iri class northwind:CustomerDoc .
drop iri class northwind:Employee .
drop iri class northwind:EmployeeDoc .
drop iri class northwind:Order .
drop iri class northwind:OrderDoc .
drop iri class northwind:CustomerContact .
drop iri class northwind:CustomerContactDoc .
drop iri class northwind:OrderLine .
drop iri class northwind:OrderLineDoc .
drop iri class northwind:Province .
drop iri class northwind:ProvinceDoc .
drop iri class northwind:Country .
drop iri class northwind:CountryDoc .
drop iri class northwind:Flag .
drop iri class northwind:FlagDoc .
drop iri class northwind:dbpedia_iri2 .
drop iri class northwind:EmployeePhoto .
drop iri class northwind:CategoryPhoto .

drop iri class northwind:category_iri .
drop iri class northwind:categorydoc_iri .
drop iri class northwind:shipper_iri .
drop iri class northwind:shipperdoc_iri .
drop iri class northwind:supplier_iri .
drop iri class northwind:supplierdoc_iri .
drop iri class northwind:product_iri .
drop iri class northwind:productdoc_iri .
drop iri class northwind:customer_iri .
drop iri class northwind:customerdoc_iri .
drop iri class northwind:employee_iri .
```

```
drop iri class northwind:employeedoc_iri .
drop iri class northwind:order_iri .
drop iri class northwind:orderdoc_iri .
drop iri class northwind:customercontact_iri .
drop iri class northwind:customercontactdoc_iri .
drop iri class northwind:orderline_iri .
drop iri class northwind:orderlinedoc_iri .
drop iri class northwind:province_iri .
drop iri class northwind:provincedoc_iri .
drop iri class northwind:country_iri .
drop iri class northwind:countrydoc_iri .
drop iri class northwind:employeephoto_iri .
drop iri class northwind:categoryphoto_iri .
drop iri class northwind:flag_iri .
drop iri class northwind:flagdoc_iri .
;

SPARQL
prefix northwind: <http://demo.openlinksw.com/schemas/northwind#>

create iri class northwind:Category "http://^{URIQADefaultHost}^/Northwind/Category/%d#this" (in category
create iri class northwind:CategoryDoc "http://^{URIQADefaultHost}^/Northwind/Category/%d" (in category_i
create iri class northwind:Shipper "http://^{URIQADefaultHost}^/Northwind/Shipper/%d#this" (in shipper_id
create iri class northwind:ShipperDoc "http://^{URIQADefaultHost}^/Northwind/Shipper/%d" (in shipper_id i
create iri class northwind:Supplier "http://^{URIQADefaultHost}^/Northwind/Supplier/%d#this" (in supplier
create iri class northwind:SupplierDoc "http://^{URIQADefaultHost}^/Northwind/Supplier/%d" (in supplier_i
create iri class northwind:Product   "http://^{URIQADefaultHost}^/Northwind/Product/%d#this" (in product_
create iri class northwind:ProductDoc   "http://^{URIQADefaultHost}^/Northwind/Product/%d" (in product_id
create iri class northwind:Customer "http://^{URIQADefaultHost}^/Northwind/Customer/%U#this" (in customer
create iri class northwind:CustomerDoc "http://^{URIQADefaultHost}^/Northwind/Customer/%U" (in customer_i
create iri class northwind:Employee "http://^{URIQADefaultHost}^/Northwind/Employee/%U_%U_%d#this" (in em
create iri class northwind:EmployeeDoc "http://^{URIQADefaultHost}^/Northwind/Employee/%U_%U_%d" (in empl
create iri class northwind:Order "http://^{URIQADefaultHost}^/Northwind/Order/%d#this" (in order_id integ
create iri class northwind:OrderDoc "http://^{URIQADefaultHost}^/Northwind/Order/%d" (in order_id integer
create iri class northwind:CustomerContact "http://^{URIQADefaultHost}^/Northwind/CustomerContact/%U#this
create iri class northwind:CustomerContactDoc "http://^{URIQADefaultHost}^/Northwind/CustomerContact/%U"
create iri class northwind:OrderLine "http://^{URIQADefaultHost}^/Northwind/OrderLine/%d/%d#this" (in ord
create iri class northwind:OrderLineDoc "http://^{URIQADefaultHost}^/Northwind/OrderLine/%d/%d" (in order
create iri class northwind:Province "http://^{URIQADefaultHost}^/Northwind/Province/%U/%U#this" (in count
create iri class northwind:ProvinceDoc "http://^{URIQADefaultHost}^/Northwind/Province/%U/%U" (in country
create iri class northwind:Country "http://^{URIQADefaultHost}^/Northwind/Country/%U#this" (in country_na
create iri class northwind:CountryDoc "http://^{URIQADefaultHost}^/Northwind/Country/%U" (in country_name
create iri class northwind:Flag "http://^{URIQADefaultHost}^%U#this" (in flag_path varchar not null) .
create iri class northwind:FlagDoc "http://^{URIQADefaultHost}^%U" (in flag_path varchar not null) .
create iri class northwind:dbpedia_iri2 "http://dbpedia.org/resource/%U" (in uname varchar not null) .
create iri class northwind:EmployeePhoto "http://^{URIQADefaultHost}^/DAV/VAD/demo/sql/EMP%d#this" (in em
create iri class northwind:CategoryPhoto "http://^{URIQADefaultHost}^/DAV/VAD/demo/sql/CAT%d#this" (in ca
create iri class northwind:Phone "tel:%s" (in phone_number varchar) .
create iri class northwind:Fax "fax:%s" (in fax_number varchar) .
;

SPARQL
prefix northwind: <http://demo.openlinksw.com/schemas/northwind#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix sioc: <http://rdfs.org/sioc/ns#>
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix wgs: <http://www.w3.org/2003/01/geo/wgs84_pos#>

alter quad storage virtrdf:DefaultQuadStorage
FROM Demo.demo.Products as products
FROM Demo.demo.Suppliers as suppliers
FROM Demo.demo.Shippers as shippers
FROM Demo.demo.Categories as categories
FROM Demo.demo.Customers as customers
FROM Demo.demo.Employees as employees
FROM Demo.demo.Orders as orders
FROM Demo.demo.Order_Details as order_lines
FROM Demo.demo.Countries as countries
FROM Demo.demo.Provinces as provinces
where (^{suppliers.}^.Country = ^{countries.}^.Name)
where (^{customers.}^.Country = ^{countries.}^.Name)
where (^{employees.}^.Country = ^{countries.}^.Name)
```

```
where (^{orders.}^.ShipCountry = ^{countries.}^.Name)
{
        create virtrdf:NorthwindDemo as graph iri ("http://^{URIQADefaultHost}^/Northwind") option (exclu
        {
                northwind:CustomerContact (customers.CustomerID)
                        a foaf:Person
                                as virtrdf:CustomerContact-foaf_Person .

                northwind:CustomerContact (customers.CustomerID)
                        a northwind:CustomerContact
                                as virtrdf:CustomerContact-CustomerContact;
                        foaf:name customers.ContactName
                                as virtrdf:CustomerContact-contact_name ;
                        foaf:phone northwind:Phone (customers.Phone)
                                as virtrdf:CustomerContact-foaf_phone ;
                        northwind:is_contact_at northwind:Customer (customers.CustomerID)
                                as virtrdf:CustomerContact-is_contact_at ;
                        northwind:country northwind:Country (customers.Country)
                                as virtrdf:CustomerContact-country ;
                        rdfs:isDefinedBy northwind: .

                northwind:CustomerContactDoc (customers.CustomerID)
                        a northwind:CustomerContactDoc
                                as virtrdf:CustomerContactDoc-CustomerID ;
                        a foaf:Document
                                as virtrdf:CustomerContactDoc-foaf_DocCustomerID ;
                        foaf:primaryTopic northwind:CustomerContact (customers.CustomerID)
                                as virtrdf:CustomerContactDoc-foaf_primarytopic ;
                        rdfs:isDefinedBy northwind: .

                northwind:Country (customers.Country)
                        northwind:is_country_of
                northwind:CustomerContact (customers.CustomerID) as virtrdf:CustomerContact-is_country_of

                northwind:Product (products.ProductID)
                        a northwind:Product
                                as virtrdf:Product-ProductID ;
                        northwind:has_category northwind:Category (products.CategoryID)
                                as virtrdf:Product-product_has_category ;
                        northwind:has_supplier northwind:Supplier (products.SupplierID)
                                as virtrdf:Product-product_has_supplier ;
                        northwind:productName products.ProductName
                                as virtrdf:Product-name_of_product ;
                        northwind:quantityPerUnit products.QuantityPerUnit
                                as virtrdf:Product-quantity_per_unit ;
                        northwind:unitPrice products.UnitPrice
                                as virtrdf:Product-unit_price ;
                        northwind:unitsInStock products.UnitsInStock
                                as virtrdf:Product-units_in_stock ;
                        northwind:unitsOnOrder products.UnitsOnOrder
                                as virtrdf:Product-units_on_order ;
                        northwind:reorderLevel products.ReorderLevel
                                as virtrdf:Product-reorder_level ;
                        northwind:discontinued products.Discontinued
                                as virtrdf:Product-discontinued ;
                        rdfs:isDefinedBy northwind: .

                northwind:ProductDoc (products.ProductID)
                        a northwind:ProductDoc
                                as virtrdf:ProductDoc-ProductID ;
                        a foaf:Document
                                as virtrdf:ProductDoc-foaf_DocProductID ;
                        foaf:primaryTopic northwind:Product (products.ProductID)
                                as virtrdf:ProductDoc-foaf_primarytopic ;
                        rdfs:isDefinedBy northwind: .

                northwind:Category (products.CategoryID)
                        northwind:category_of northwind:Product (products.ProductID) as virtrdf:Product-c

                northwind:Supplier (products.SupplierID)
                        northwind:supplier_of northwind:Product (products.ProductID) as virtrdf:Product-s

                northwind:Supplier (suppliers.SupplierID)
```

```
        a northwind:Supplier
                as virtrdf:Supplier-SupplierID ;
        northwind:companyName suppliers.CompanyName
                as virtrdf:Supplier-company_name ;
        northwind:contactName suppliers.ContactName
                as virtrdf:Supplier-contact_name ;
        northwind:contactTitle suppliers.ContactTitle
                as virtrdf:Supplier-contact_title ;
        northwind:address suppliers.Address
                as virtrdf:Supplier-address ;
        northwind:city northwind:dbpedia_iri2(suppliers.City)
                as virtrdf:Supplier-dbpediacity ;
        northwind:region suppliers.Region
                as virtrdf:Supplier-region ;
        northwind:postalCode suppliers.PostalCode
                as virtrdf:Supplier-postal_code ;
        northwind:country northwind:Country(suppliers.Country)
                as virtrdf:Supplier-country ;
        northwind:phone northwind:Phone (suppliers.Phone)
                as virtrdf:Supplier-phone ;
        northwind:fax northwind:Fax (suppliers.Fax)
                as virtrdf:Supplier-fax ;
        northwind:homePage suppliers.HomePage
                as virtrdf:Supplier-home_page ;
        rdfs:isDefinedBy northwind: .

northwind:SupplierDoc (suppliers.SupplierID)
        a northwind:SupplierDoc
                as virtrdf:SupplierDoc-SupplierID ;
        a foaf:Document
                as virtrdf:SupplierDoc-foaf_DocSupplierID ;
        foaf:primaryTopic northwind:Supplier (suppliers.SupplierID)
                as virtrdf:SupplierDoc-foaf_primarytopic ;
        rdfs:isDefinedBy northwind: .

northwind:Country (suppliers.Country)
        northwind:is_country_of
northwind:Supplier (suppliers.SupplierID) as virtrdf:Supplier-is_country_of .

northwind:Category (categories.CategoryID)
        a northwind:Category
                as virtrdf:Category-CategoryID ;
        northwind:categoryName categories.CategoryName
                as virtrdf:Category-home_page ;
        northwind:description categories.Description
                as virtrdf:Category-description ;
        foaf:img northwind:CategoryPhoto(categories.CategoryID)
                as virtrdf:Category-categories.CategoryPhoto ;
        rdfs:isDefinedBy northwind: .

northwind:CategoryDoc (categories.CategoryID)
        a northwind:CategoryDoc
                as virtrdf:CategoryDoc-CategoryID ;
        a foaf:Document
                as virtrdf:CategoryDoc-foaf_DocCategoryID ;
        foaf:primaryTopic northwind:Category (categories.CategoryID)
                as virtrdf:CategoryDoc-foaf_primarytopic ;
        rdfs:isDefinedBy northwind: .

northwind:CategoryPhoto(categories.CategoryID)
        a northwind:CategoryPhoto
                as virtrdf:Category-categories.CategoryPhotoID ;
        rdfs:isDefinedBy northwind: .

northwind:Shipper (shippers.ShipperID)
        a northwind:Shipper
                as virtrdf:Shipper-ShipperID ;
        northwind:companyName shippers.CompanyName
                as virtrdf:Shipper-company_name ;
        northwind:phone northwind:Phone (shippers.Phone)
                as virtrdf:Shipper-phone ;
        rdfs:isDefinedBy northwind: .
```

```
northwind:ShipperDoc (shippers.ShipperID)
        a northwind:ShipperDoc
                as virtrdf:ShipperDoc-ShipperID ;
        a foaf:Document
                as virtrdf:ShipperDoc-foaf_DocShipperID ;
        foaf:primaryTopic northwind:Shipper (shippers.ShipperID)
                as virtrdf:ShipperDoc-foaf_primarytopic ;
        rdfs:isDefinedBy northwind: .

northwind:Customer (customers.CustomerID)
        a  northwind:Customer
                as virtrdf:Customer-CustomerID2 ;
        a  foaf:Organization
                as virtrdf:Customer-CustomerID ;
        foaf:name customers.CompanyName
                as virtrdf:Customer-foaf_name ;
        northwind:companyName customers.CompanyName
                as virtrdf:Customer-company_name ;
        northwind:has_contact northwind:CustomerContact (customers.CustomerID)
                as virtrdf:Customer-contact ;
        northwind:country northwind:Country (customers.Country)
                as virtrdf:Customer-country ;
        northwind:contactName customers.ContactName
                as virtrdf:Customer-contact_name ;
        northwind:contactTitle customers.ContactTitle
                as virtrdf:Customer-contact_title ;
        northwind:address customers.Address
                as virtrdf:Customer-address ;
        northwind:city northwind:dbpedia_iri2(customers.City)
                as virtrdf:Customer-dbpediacity ;
        northwind:region customers.Region
                as virtrdf:Customer-region ;
        northwind:PostalCode customers.PostalCode
                as virtrdf:Customer-postal_code ;
        foaf:phone northwind:Phone (customers.Phone)
                as virtrdf:Customer-foaf_phone ;
        northwind:phone northwind:Phone (customers.Phone)
                as virtrdf:Customer-phone ;
        northwind:fax northwind:Fax (customers.Fax)
                as virtrdf:Customer-fax ;
        rdfs:isDefinedBy northwind: .

northwind:CustomerDoc (customers.CustomerID)
        a  northwind:CustomerDoc
                as virtrdf:CustomerDoc-CustomerID2 ;
        a  foaf:Document
                as virtrdf:CustomerDoc-CustomerID3 ;
        foaf:primaryTopic northwind:Customer (customers.CustomerID)
                as virtrdf:CustomerDoc-foaf_primarytopic ;
        rdfs:isDefinedBy northwind: .

northwind:Country (customers.Country)
        northwind:is_country_of
northwind:Customer (customers.CustomerID) as virtrdf:Customer-is_country_of .

northwind:Employee (employees.FirstName, employees.LastName, employees.EmployeeID)
        a northwind:Employee
                as virtrdf:Employee-EmployeeID2 ;
        a foaf:Person
                as virtrdf:Employee-EmployeeID ;
        foaf:surname employees.LastName
                as virtrdf:Employee-foaf_last_name ;
        northwind:lastName employees.LastName
                as virtrdf:Employee-last_name ;
        foaf:firstName employees.FirstName
                as virtrdf:Employee-foaf_first_name ;
        northwind:firstName employees.FirstName
                as virtrdf:Employee-first_name ;
        foaf:title employees.Title
                as virtrdf:Employee-title ;
        northwind:titleOfCourtesy employees.TitleOfCourtesy
                as virtrdf:Employee-title_of_courtesy ;
        foaf:birthday employees.BirthDate
```

```
                as virtrdf:Employee-foaf_birth_date ;
        northwind:birthday employees.BirthDate
                as virtrdf:Employee-birth_date ;
        northwind:hireDate employees.HireDate
                as virtrdf:Employee-hire_date ;
        northwind:address employees.Address
                as virtrdf:Employee-address ;
        northwind:city northwind:dbpedia_iri2(employees.City)
                as virtrdf:Employee-dbpediacity ;
        northwind:region employees.Region
                as virtrdf:Employee-region ;
        northwind:postalCode employees.PostalCode
                as virtrdf:Employee-postal_code ;
        northwind:country northwind:Country(employees.Country)
                as virtrdf:Employee-country ;
        foaf:phone employees.HomePhone
                as virtrdf:Employee-home_phone ;
        northwind:extension employees.Extension
                as virtrdf:Employee-extension ;
        northwind:notes employees.Notes
                as virtrdf:Employee-notes ;
        northwind:reportsTo northwind:Employee(employees.FirstName, employees.LastName, e
                as virtrdf:Employee-reports_to ;
        foaf:img northwind:EmployeePhoto(employees.EmployeeID)
                as virtrdf:Employee-employees.EmployeePhoto ;
        rdfs:isDefinedBy northwind: .

northwind:EmployeeDoc (employees.FirstName, employees.LastName, employees.EmployeeID)
        a  northwind:EmployeeDoc
                as virtrdf:EmployeeDoc-EmployeeID2 ;
        a  foaf:Document
                as virtrdf:EmployeeDoc-EmployeeID3 ;
        foaf:primaryTopic northwind:Employee (employees.FirstName, employees.LastName, em
                as virtrdf:EmployeeDoc-foaf_primarytopic ;
        rdfs:isDefinedBy northwind: .

northwind:EmployeePhoto(employees.EmployeeID)
        a northwind:EmployeePhoto
                as virtrdf:Employee-employees.EmployeePhotoId ;
        rdfs:isDefinedBy northwind: .

northwind:Employee (employees.FirstName, employees.LastName, orders.EmployeeID)
        northwind:is_salesrep_of
northwind:Order (orders.OrderID) where (^{orders.}^.EmployeeID = ^{employees.}^.EmployeeI

northwind:Country (employees.Country)
        northwind:is_country_of
northwind:Employee (employees.FirstName, employees.LastName, employees.EmployeeID) as vir

northwind:Order (orders.OrderID)
        a northwind:Order
                as virtrdf:Order-Order ;
        northwind:has_customer northwind:Customer (orders.CustomerID)
                as virtrdf:Order-order_has_customer ;
        northwind:has_salesrep northwind:Employee (employees.FirstName, employees.LastNam
                as virtrdf:Customer-has_salesrep ;
        northwind:has_employee northwind:Employee (employees.FirstName, employees.LastNam
                as virtrdf:Order-order_has_employee ;
        northwind:orderDate orders.OrderDate
                as virtrdf:Order-order_date ;
        northwind:requiredDate orders.RequiredDate
                as virtrdf:Order-required_date ;
        northwind:shippedDate orders.ShippedDate
                as virtrdf:Order-shipped_date ;
        northwind:order_ship_via northwind:Shipper (orders.ShipVia)
                as virtrdf:Order-order_ship_via ;
        northwind:freight orders.Freight
                as virtrdf:Order-freight ;
        northwind:shipName orders.ShipName
                as virtrdf:Order-ship_name ;
        northwind:shipAddress orders.ShipAddress
                as virtrdf:Order-ship_address ;
        northwind:shipCity northwind:dbpedia_iri2(orders.ShipCity)
```

```
                as virtrdf:Order-dbpediaship_city ;
        northwind:shipRegion orders.ShipRegion
                as virtrdf:Order-ship_region ;
        northwind:shipPostal_code orders.ShipPostalCode
                as virtrdf:Order-ship_postal_code ;
        northwind:shipCountry northwind:Country(orders.ShipCountry)
                as virtrdf:ship_country ;
        rdfs:isDefinedBy northwind: .

northwind:OrderDoc (orders.OrderID)
        a  northwind:OrderDoc
                as virtrdf:OrderDoc-OrderID2 ;
        a  foaf:Document
                as virtrdf:OrderDoc-OrderID3 ;
        foaf:primaryTopic northwind:Order (orders.OrderID)
                as virtrdf:OrderDoc-foaf_primarytopic ;
        rdfs:isDefinedBy northwind: .

northwind:Country (orders.ShipCountry)
        northwind:is_ship_country_of
northwind:Order (orders.OrderID) as virtrdf:Order-is_country_of .

northwind:Customer (orders.CustomerID)
        northwind:has_order northwind:Order (orders.OrderID) as virtrdf:Order-has_order .

northwind:Shipper (orders.ShipVia)
        northwind:ship_order northwind:Order (orders.OrderID) as virtrdf:Order-ship_order

northwind:OrderLine (order_lines.OrderID, order_lines.ProductID)
        a northwind:OrderLine
                as virtrdf:OrderLine-OrderLines ;
        northwind:has_order_id northwind:Order (order_lines.OrderID)
                as virtrdf:order_lines_has_order_id ;
        northwind:has_product_id northwind:Product (order_lines.ProductID)
                as virtrdf:order_lines_has_product_id ;
        northwind:unitPrice order_lines.UnitPrice
                as virtrdf:OrderLine-unit_price ;
        northwind:quantity order_lines.Quantity
                as virtrdf:OrderLine-quantity ;
        northwind:discount order_lines.Discount
                as virtrdf:OrderLine-discount ;
        rdfs:isDefinedBy northwind: .

northwind:OrderLineDoc (order_lines.OrderID, order_lines.ProductID)
        a  northwind:OrderLineDoc
                as virtrdf:OrderLineDoc-OrderLineID2 ;
        a  foaf:Document
                as virtrdf:OrderLineDoc-OrderLineID3 ;
        foaf:primaryTopic northwind:OrderLine (order_lines.OrderID, order_lines.ProductID
                as virtrdf:OrderLineDoc-foaf_primarytopic ;
        rdfs:isDefinedBy northwind: .

northwind:Order (orders.OrderID)
        northwind:is_order_of
northwind:OrderLine (order_lines.OrderID, order_lines.ProductID) where (^{orders.}^.Order

northwind:Product (products.ProductID)
        northwind:is_product_of
northwind:OrderLine (order_lines.OrderID, order_lines.ProductID) where (^{products.}^.Pro

northwind:Country (countries.Name)
        a northwind:Country
                as virtrdf:Country-Type2 ;
        a wgs:SpatialThing
                as virtrdf:Country-Type ;
        owl:sameAs northwind:dbpedia_iri2 (countries.Name) ;
        northwind:name countries.Name
                as virtrdf:Country-Name ;
        northwind:code countries.Code
                as virtrdf:Country-Code ;
        northwind:smallFlagDAVResourceName countries.SmallFlagDAVResourceName
                as virtrdf:Country-SmallFlagDAVResourceName ;
        northwind:largeFlagDAVResourceName countries.LargeFlagDAVResourceName
```

```
                        as virtrdf:Country-LargeFlagDAVResourceName ;
                northwind:smallFlagDAVResourceURI northwind:Flag(countries.SmallFlagDAVResourceUR
                        as virtrdf:Country-SmallFlagDAVResourceURI ;
                northwind:largeFlagDAVResourceURI northwind:Flag(countries.LargeFlagDAVResourceUR
                        as virtrdf:Country-LargeFlagDAVResourceURI ;
                wgs:lat countries.Lat
                        as virtrdf:Country-Lat ;
                wgs:long countries.Lng
                        as virtrdf:Country-Lng ;
                rdfs:isDefinedBy northwind: .

        northwind:CountryDoc (countries.Name)
                a  northwind:CountryDoc
                        as virtrdf:CountryDoc-CountryID2 ;
                a  foaf:Document
                        as virtrdf:CountryDoc-CountryID3 ;
                foaf:primaryTopic northwind:Country (countries.Name)
                        as virtrdf:CountryDoc-foaf_primarytopic ;
                rdfs:isDefinedBy northwind: .

        northwind:Country (countries.Name)
                northwind:has_province
        northwind:Province (provinces.CountryCode, provinces.Province) where (^{provinces.}^.Coun

        northwind:Province (provinces.CountryCode, provinces.Province)
                a northwind:Province
                        as virtrdf:Province-Provinces ;
                owl:sameAs northwind:dbpedia_iri2 (provinces.Province) ;
                northwind:has_country_code provinces.CountryCode
                        as virtrdf:has_country_code ;
                northwind:provinceName provinces.Province
                        as virtrdf:Province-ProvinceName ;
                rdfs:isDefinedBy northwind: .

        northwind:ProvinceDoc (provinces.CountryCode, provinces.Province)
                a  northwind:ProvinceDoc
                        as virtrdf:ProvinceDoc-ProvinceID2 ;
                a  foaf:Document
                        as virtrdf:ProvinceDoc-ProvinceID3 ;
                foaf:primaryTopic northwind:Province (provinces.CountryCode, provinces.Province)
                        as virtrdf:ProvinceDoc-foaf_primarytopic ;
                rdfs:isDefinedBy northwind: .

        northwind:Province (provinces.CountryCode, provinces.Province)
                northwind:is_province_of
        northwind:Country (countries.Name) where  (^{countries.}^.Code = ^{provinces.}^.CountryCo
    }.
}.
;

delete FROM DB.DBA.URL_REWRITE_RULE_LIST where urrl_list like 'demo_nw%';
delete FROM DB.DBA.URL_REWRITE_RULE where urr_rule like 'demo_nw%';

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'demo_nw_rule2',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/sparql?query=DESCRIBE+%%3Chttp%%3A//^{URIQADefaultHost}^%U%%23this%%3E+%%3Chttp%%3A//^{URIQADefault
    vector('path', 'path', '*accept*'),
    null,
    '(text/rdf.n3)|(application/rdf.xml)',
    0,
    null
    );

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'demo_nw_rule1',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
```

```
    '/about/html/http/^{URIQADefaultHost}^%s',
    vector('path'),
    null,
    '(text/html)|(\\*/\\*)',
    0,
    303
    );

DB.DBA.URLREWRITE_CREATE_RULELIST (
    'demo_nw_rule_list1',
    1,
    vector (
                'demo_nw_rule1',
                'demo_nw_rule2'
          ));

VHOST_REMOVE (lpath=>'/Northwind');
DB.DBA.VHOST_DEFINE (lpath=>'/Northwind', ppath=>'/DAV/home/demo/', vsp_user=>'dba', is_dav=>1,
          is_brws=>0, opts=>vector ('url_rewrite', 'demo_nw_rule_list1'));

create procedure DB.DBA.LOAD_NW_ONTOLOGY_FROM_DAV()
{
  declare content1, urihost varchar;
  SELECT cast (RES_CONTENT as varchar) into content1 FROM WS.WS.SYS_DAV_RES where RES_FULL_PATH = '/DAV/V
  DB.DBA.RDF_LOAD_RDFXML (content1, 'http://demo.openlinksw.com/schemas/northwind#', 'http://demo.openlin
  urihost := cfg_item_value(virtuoso_ini_path(), 'URIQA','DefaultHost');
  if (urihost = 'demo.openlinksw.com')
  {
    DB.DBA.VHOST_REMOVE (lpath=>'/schemas/northwind');
    DB.DBA.VHOST_DEFINE (lpath=>'/schemas/northwind', ppath=>'/DAV/VAD/demo/sql/nw.owl', vsp_user=>'dba',
    DB.DBA.VHOST_REMOVE (lpath=>'/schemas/northwind#');
    DB.DBA.VHOST_DEFINE (lpath=>'/schemas/northwind#', ppath=>'/DAV/VAD/demo/sql/nw.owl', vsp_user=>'dba'
  }
};

DB.DBA.LOAD_NW_ONTOLOGY_FROM_DAV();
drop procedure DB.DBA.LOAD_NW_ONTOLOGY_FROM_DAV;

DB.DBA.XML_SET_NS_DECL ('northwind', 'http://demo.openlinksw.com/schemas/northwind#', 2);
```

Basic Northwind Ontology

```
<?xml version="1.0" encoding="utf-8"?>

<!--
 -
 -
 -  This file is part of the OpenLink Software Virtuoso Open-Source (VOS)
 -  project.
 -
 -  Copyright (C) 1998-2024 OpenLink Software
 -
 -  This project is free software; you can redistribute it and/or modify it
 -  under the terms of the GNU General Public License as published by the
 -  Free Software Foundation; only version 2 of the License, dated June 1991.
 -
 -  This program is distributed in the hope that it will be useful, but
 -  WITHOUT ANY WARRANTY; without even the implied warranty of
 -  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 -  General Public License for more details.
 -
 -  You should have received a copy of the GNU General Public License along
 -  with this program; if not, write to the Free Software Foundation, Inc.,
 -  51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
 -
-->

<rdf:RDF xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:owl ="http://www.w3.org/2002/07/owl#"
         xmlns:virtrdf="http://www.openlinksw.com/schemas/virtrdf#"
         xml:base="http://demo.openlinksw.com/schemas/northwind#">
  <owl:Ontology rdf:about="http://demo.openlinksw.com/schemas/northwind#">
```

```
      <rdfs:label>Northwind</rdfs:label>
      <rdfs:comment>Northwind database classes and properties</rdfs:comment>
      <virtrdf:catName>Northwind</virtrdf:catName>
      <virtrdf:version>1.00</virtrdf:version>
</owl:Ontology>

<rdfs:Class rdf:ID="Product">
  <rdfs:label>Product</rdfs:label>
</rdfs:Class>
<rdf:Property rdf:ID="has_category">
  <rdfs:range rdf:resource="#Category"/>
  <rdfs:domain rdf:resource="#Product"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>Category</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="has_supplier">
  <rdfs:range rdf:resource="#Supplier"/>
  <rdfs:domain rdf:resource="#Product"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>Supplier</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="ProductName">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Product"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>ProductName</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="QuantityPerUnit">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Product"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>QuantityPerUnit</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="UnitPrice">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#double"/>
  <rdfs:domain rdf:resource="#Product"/>
  <rdfs:domain rdf:resource="#OrderLine"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>UnitPrice</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="UnitsInStock">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
  <rdfs:domain rdf:resource="#Product"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>UnitsInStock</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="UnitsOnOrder">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
  <rdfs:domain rdf:resource="#Product"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>UnitsOnOrder</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="ReorderLevel">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
  <rdfs:domain rdf:resource="#Product"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>ReorderLevel</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="Discontinued">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
  <rdfs:domain rdf:resource="#Product"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>Discontinued</rdfs:label>
</rdf:Property>

<rdfs:Class rdf:ID="Supplier">
  <rdfs:label>Supplier</rdfs:label>
</rdfs:Class>
<rdf:Property rdf:ID="CompanyName">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:subPropertyOf rdf:resource="http://xmlns.com/foaf/0.1/name"/>
  <rdfs:domain rdf:resource="#Supplier"/>
  <rdfs:domain rdf:resource="#Shipper"/>
```

```
    <rdfs:domain rdf:resource="#Customer"/>
    <rdfs:cardinality>1</rdfs:cardinality>
    <rdfs:label>CompanyName</rdfs:label>
  </rdf:Property>
  <rdf:Property rdf:ID="ContactName">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:subPropertyOf rdf:resource="http://xmlns.com/foaf/0.1/name"/>
    <rdfs:domain rdf:resource="#Supplier"/>
    <rdfs:domain rdf:resource="#Customer"/>
    <rdfs:domain rdf:resource="#CustomerContact"/>
    <rdfs:cardinality>1</rdfs:cardinality>
    <rdfs:label>ContactName</rdfs:label>
  </rdf:Property>
  <rdf:Property rdf:ID="ContactTitle">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:subPropertyOf rdf:resource="http://xmlns.com/foaf/0.1/title"/>
    <rdfs:domain rdf:resource="#Supplier"/>
    <rdfs:domain rdf:resource="#Customer"/>
    <rdfs:cardinality>1</rdfs:cardinality>
    <rdfs:label>ContactTitle</rdfs:label>
  </rdf:Property>
  <rdf:Property rdf:ID="Address">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Supplier"/>
    <rdfs:domain rdf:resource="#Customer"/>
    <rdfs:domain rdf:resource="#Employee"/>
    <rdfs:cardinality>1</rdfs:cardinality>
    <rdfs:label>Address</rdfs:label>
  </rdf:Property>
  <rdf:Property rdf:ID="City">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Supplier"/>
    <rdfs:domain rdf:resource="#Customer"/>
    <rdfs:domain rdf:resource="#Employee"/>
    <rdfs:cardinality>1</rdfs:cardinality>
    <rdfs:label>City</rdfs:label>
  </rdf:Property>
  <rdf:Property rdf:ID="Region">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Supplier"/>
    <rdfs:domain rdf:resource="#Customer"/>
    <rdfs:domain rdf:resource="#Employee"/>
    <rdfs:cardinality>1</rdfs:cardinality>
    <rdfs:label>Region</rdfs:label>
  </rdf:Property>
  <rdf:Property rdf:ID="PostalCode">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Supplier"/>
    <rdfs:domain rdf:resource="#Customer"/>
    <rdfs:domain rdf:resource="#Employee"/>
    <rdfs:cardinality>1</rdfs:cardinality>
    <rdfs:label>PostalCode</rdfs:label>
  </rdf:Property>
  <rdf:Property rdf:ID="CountryName">
    <rdfs:range rdf:resource="#Country"/>
    <rdfs:domain rdf:resource="#Supplier"/>
    <rdfs:domain rdf:resource="#Customer"/>
    <rdfs:domain rdf:resource="#CustomerContact"/>
    <rdfs:domain rdf:resource="#Employee"/>
    <rdfs:cardinality>1</rdfs:cardinality>
    <rdfs:label>Country</rdfs:label>
  </rdf:Property>
  <rdf:Property rdf:ID="Phone">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:subPropertyOf rdf:resource="http://xmlns.com/foaf/0.1/phone"/>
    <rdfs:domain rdf:resource="#Supplier"/>
    <rdfs:domain rdf:resource="#Shipper"/>
    <rdfs:domain rdf:resource="#Customer"/>
    <rdfs:domain rdf:resource="#CustomerContact"/>
    <rdfs:domain rdf:resource="#Employee"/>
    <rdfs:cardinality>1</rdfs:cardinality>
    <rdfs:label>Phone</rdfs:label>
  </rdf:Property>
```

```
<rdf:Property rdf:ID="Fax">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Supplier"/>
  <rdfs:domain rdf:resource="#Customer"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>Fax</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="HomePage">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Supplier"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>HomePage</rdfs:label>
</rdf:Property>

<rdfs:Class rdf:ID="Category">
  <rdfs:label>Category</rdfs:label>
</rdfs:Class>
<rdf:Property rdf:ID="CategoryName">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Category"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>CategoryName</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="Description">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Category"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>Description</rdfs:label>
</rdf:Property>

<rdfs:Class rdf:ID="Shipper">
  <rdfs:label>Shipper</rdfs:label>
</rdfs:Class>

<rdfs:Class rdf:ID="CustomerContact">
  <rdfs:label>CustomerContact</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Customer">
  <rdfs:label>Customer</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://xmlns.com/foaf/0.1/Organization"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Employee">
  <rdfs:label>Employee</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
</rdfs:Class>
<rdf:Property rdf:ID="LastName">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:subPropertyOf rdf:resource="http://xmlns.com/foaf/0.1/surname"/>
  <rdfs:domain rdf:resource="#Employee"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>LastName</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="FirstName">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:subPropertyOf rdf:resource="http://xmlns.com/foaf/0.1/firstName"/>
  <rdfs:domain rdf:resource="#Employee"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>FirstName</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="Title">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:subPropertyOf rdf:resource="http://xmlns.com/foaf/0.1/title"/>
  <rdfs:domain rdf:resource="#Employee"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>Title</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="TitleOfCourtesy">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Employee"/>
  <rdfs:cardinality>1</rdfs:cardinality>
```

```
  <rdfs:label>TitleOfCourtesy</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="BirthDate">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:subPropertyOf rdf:resource="http://xmlns.com/foaf/0.1/birthday"/>
  <rdfs:domain rdf:resource="#Employee"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>BirthDate</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="HireDate">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Employee"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>HireDate</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="Extension">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Employee"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>Extension</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="Notes">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Employee"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>Notes</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="ReportsTo">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
  <rdfs:domain rdf:resource="#Employee"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>ReportsTo</rdfs:label>
</rdf:Property>

<rdfs:Class rdf:ID="Order">
  <rdfs:label>Order</rdfs:label>
</rdfs:Class>
<rdf:Property rdf:ID="has_customer">
  <rdfs:range rdf:resource="#Customer"/>
  <rdfs:domain rdf:resource="#Order"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>Customer</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="has_employee">
  <rdfs:range rdf:resource="#Employee"/>
  <rdfs:domain rdf:resource="#Order"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>Employee</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="OrderDate">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Order"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>OrderDate</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="RequiredDate">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Order"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>RequiredDate</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="ShippedDate">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Order"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>ShippedDate</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="order_ship_via">
  <rdfs:range rdf:resource="#Shipper"/>
  <rdfs:domain rdf:resource="#Order"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>Shipper</rdfs:label>
</rdf:Property>
```

```
<rdf:Property rdf:ID="Freight">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#double"/>
  <rdfs:domain rdf:resource="#Order"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>Freight</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="ShipName">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Order"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>ShipName</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="ShipAddress">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Order"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>ShipAddress</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="ShipCity">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Order"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>ShipCity</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="ShipRegion">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Order"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>ShipRegion</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="ShipPostalCode">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Order"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>ShipPostalCode</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="ShipCountry">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Order"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>ShipCountry</rdfs:label>
</rdf:Property>

<rdfs:Class rdf:ID="OrderLine">
  <rdfs:label>OrderLine</rdfs:label>
</rdfs:Class>
<rdf:Property rdf:ID="has_order_id">
  <rdfs:range rdf:resource="#Order"/>
  <rdfs:domain rdf:resource="#OrderLine"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>Order</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="has_product_id">
  <rdfs:range rdf:resource="#Product"/>
  <rdfs:domain rdf:resource="#OrderLine"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>Product</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="Quantity">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
  <rdfs:domain rdf:resource="#OrderLine"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>Quantity</rdfs:label>
</rdf:Property>
<rdf:Property rdf:ID="Discount">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#double"/>
  <rdfs:domain rdf:resource="#OrderLine"/>
  <rdfs:cardinality>1</rdfs:cardinality>
  <rdfs:label>Discount</rdfs:label>
</rdf:Property>

<rdfs:Class rdf:ID="Country">
  <rdfs:label>Country</rdfs:label>
```

```
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2003/01/geo/wgs84_pos#SpatialThing"/>
  </rdfs:Class>
  <rdf:Property rdf:ID="Name">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Country"/>
    <rdfs:cardinality>1</rdfs:cardinality>
    <rdfs:label>Name</rdfs:label>
  </rdf:Property>
  <rdf:Property rdf:ID="Code">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Country"/>
    <rdfs:cardinality>1</rdfs:cardinality>
    <rdfs:label>Code</rdfs:label>
  </rdf:Property>
  <rdf:Property rdf:ID="SmallFlagDAVResourceName">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Country"/>
    <rdfs:cardinality>1</rdfs:cardinality>
    <rdfs:label>SmallFlagDAVResourceName</rdfs:label>
  </rdf:Property>
  <rdf:Property rdf:ID="LargeFlagDAVResourceName">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Country"/>
    <rdfs:cardinality>1</rdfs:cardinality>
    <rdfs:label>LargeFlagDAVResourceName</rdfs:label>
  </rdf:Property>
  <rdf:Property rdf:ID="SmallFlagDAVResourceURI">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Country"/>
    <rdfs:cardinality>1</rdfs:cardinality>
    <rdfs:label>SmallFlagDAVResourceURI</rdfs:label>
  </rdf:Property>
  <rdf:Property rdf:ID="LargeFlagDAVResourceURI">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Country"/>
    <rdfs:cardinality>1</rdfs:cardinality>
    <rdfs:label>LargeFlagDAVResourceURI</rdfs:label>
  </rdf:Property>
  <rdf:Property rdf:ID="Lat">
    <rdfs:range rdf:resource="ttp://www.w3.org/2003/01/geo/wgs84_pos#lat"/>
    <rdfs:domain rdf:resource="#Country"/>
    <rdfs:cardinality>1</rdfs:cardinality>
    <rdfs:label>Lat</rdfs:label>
  </rdf:Property>
  <rdf:Property rdf:ID="Lng">
    <rdfs:range rdf:resource="ttp://www.w3.org/2003/01/geo/wgs84_pos#lng"/>
    <rdfs:domain rdf:resource="#Country"/>
    <rdfs:cardinality>1</rdfs:cardinality>
    <rdfs:label>Lng</rdfs:label>
  </rdf:Property>

  <rdfs:Class rdf:ID="Province">
    <rdfs:label>Province</rdfs:label>
  </rdfs:Class>
  <rdf:Property rdf:ID="has_country_code">
    <rdfs:range rdf:resource="#Country"/>
    <rdfs:domain rdf:resource="#Provinces"/>
    <rdfs:cardinality>1</rdfs:cardinality>
    <rdfs:label>Country Code</rdfs:label>
  </rdf:Property>
  <rdf:Property rdf:ID="ProvinceName">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Province"/>
    <rdfs:cardinality>1</rdfs:cardinality>
    <rdfs:label>ProvinceName</rdfs:label>
  </rdf:Property>

</rdf:RDF>
```

16.8.11. Oracle Demonstration 'HR' Database

Live links to a sample instance

- list all employee URIs
- description of employee 105

Script to set up your own instance

```
-- Setup script for Linked Data Views of Oracle 10 Human Resources Sample Database --

GRANT SELECT ON HR.orama.COUNTRIES TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON HR.orama.REGIONS TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON HR.orama.DEPARTMENTS TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON HR.orama.LOCATIONS TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON HR.orama.EMPLOYEES TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON HR.orama.JOBS TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON HR.orama.JOB_HISTORY TO "SPARQL", "SPARQL_UPDATE";

-----------------------------------------------------------------

-------- Create rdfs:Class definitions ----------------------------

ttlp (
'
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

@prefix hr: <http://example.com/schemas/oraclehr/> .

hr:countries a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/oraclehr> ;
        rdfs:label "COUNTRIES" ;
        rdfs:comment "Oracle HR COUNTRIES table" .

hr:country_id a rdf:Property ;
        rdfs:domain hr:countries ;
        rdfs:range xsd:string ;
        rdfs:label "COUNTRY ID" .

hr:country_name a rdf:Property ;
        rdfs:domain hr:countries ;
        rdfs:range xsd:string ;
        rdfs:label "COUNTRY NAME" .

hr:region_id a rdf:Property ;
        rdfs:domain hr:countries ;
        rdfs:range hr:regions ;
        rdfs:label "REGION ID" .

hr:regions a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/oraclehr> ;
        rdfs:label "REGIONS" ;
        rdfs:comment "Oracle HR REGIONS table" .

hr:region_id a rdf:Property ;
        rdfs:domain hr:regions ;
        rdfs:range xsd:integer ;
        rdfs:label "REGION ID" .

hr:region_name a rdf:Property ;
        rdfs:domain hr:regions ;
        rdfs:range xsd:string ;
        rdfs:label "REGION NAME" .

hr:departments a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/oraclehr> ;
        rdfs:label "DEPARTMENTS" ;
        rdfs:comment "Oracle HR DEPARTMENT table" .
```

```
hr:department_id a rdf:Property ;
        rdfs:domain hr:departments ;
        rdfs:range xsd:integer ;
        rdfs:label "DEPARTMENT ID" .

hr:department_name a rdf:Property ;
        rdfs:domain hr:departments ;
        rdfs:range xsd:string ;
        rdfs:comment "DEPARTMENT NAME" .

hr:manager_id a rdf:Property ;
        rdfs:domain hr:departments ;
        rdfs:range hr:employees ;
        rdfs:comment "MANAGER ID" .

hr:location_id a rdf:Property ;
        rdfs:domain hr:departments ;
        rdfs:range hr:locations ;
        rdfs:comment "LOCATION ID" .

hr:employees a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/oraclehr> ;
        rdfs:label "employees" ;
        rdfs:comment "Oracle HR EMPLOYEES table" .

hr:employee_id a rdf:Property ;
        rdfs:domain hr:employees;
        rdfs:range xsd:integer ;
        rdfs:label "EMPLOYEE ID" .

hr:first_name a rdf:Property ;
        rdfs:domain hr:employees;
        rdfs:range xsd:string ;
        rdfs:label "FIRST NAME" .

hr:last_name a rdf:Property ;
        rdfs:domain hr:employees ;
        rdfs:range xsd:string ;
        rdfs:label "LAST NAME" .

hr:email a rdf:Property ;
        rdfs:domain hr:employees;
        rdfs:range xsd:string ;
        rdfs:label "EMAIL" .

hr:phone_number a rdf:Property ;
        rdfs:domain hr:employees ;
        rdfs:range xsd:string ;
        rdfs:label "PHONE NUMBER" .

hr:hire_date a rdf:Property ;
        rdfs:domain hr:employees ;
        rdfs:range xsd:date ;
        rdfs:label "HIRE DATE" .

hr:job_id a rdf:Property ;
        rdfs:domain hr:employees;
        rdfs:range hr:jobs ;
        rdfs:label "JOB ID" .

hr:salary a rdf:Property ;
        rdfs:domain hr:employees ;
        rdfs:range xsd:integer ;
        rdfs:label "SALARY" .

hr:commission_pct a rdf:Property ;
        rdfs:domain hr:employees ;
        rdfs:range xsd:integer ;
        rdfs:label "COMMISSION PCT" .

hr:manager_id a rdf:Property ;
        rdfs:domain hr:employees ;
        rdfs:range xsd:string ;
```

```
        rdfs:label "MANAGER ID" .

hr:department_id a rdf:Property ;
        rdfs:domain hr:employees ;
        rdfs:range hr:departments ;
        rdfs:label "DEPARTMENT ID" .

hr:jobs a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/oraclehr> ;
        rdfs:label "JOBS" ;
        rdfs:comment "Oracle HR JOBS table" .

hr:job_id a rdf:Property ;

        rdfs:domain hr:jobs ;
        rdfs:range xsd:string ;
        rdfs:label "JOB ID" .

hr:job_title a rdf:Property ;
        rdfs:domain hr:jobs ;
        rdfs:range xsd:string ;
        rdfs:label "JOB TITLE" .

hr:min_salary a rdf:Property ;
        rdfs:domain hr:jobs ;
        rdfs:range xsd:number;
        rdfs:label "MIN SALARY" .

hr:max_salary a rdf:Property ;
        rdfs:domain hr:jobs ;
        rdfs:range xsd:number;
        rdfs:label "MAXSALARY" .

hr:job_history a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/oraclehr> ;
        rdfs:label "JOB HISTORY" ;
        rdfs:comment "Oracle HR JOB HISTORY table" .

hr:employee_id a rdf:Property ;
        rdfs:domain hr:job_history ;
        rdfs:range hr:employees ;
        rdfs:label "EMPLOYEE ID" .

hr:start_date a rdf:Property ;
        rdfs:domain hr:job_history ;
        rdfs:range xsd:date ;
        rdfs:label "START DATE" .

hr:end_date a rdf:Property ;
        rdfs:domain hr:job_history ;
        rdfs:range xsd:date ;
        rdfs:label "END DATE" .

hr:job_id a rdf:Property ;
        rdfs:domain hr:job_history ;
        rdfs:range hr:jobs ;
        rdfs:label "JOB ID" .

hr:department_id a rdf:Property ;
        rdfs:domain hr:job_history ;
        rdfs:range hr:departments ;
        rdfs:label "DEPARTMENT ID" .

hr:locations a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/oraclehr> ;
        rdfs:label "LOCATIONS" ;
        rdfs:comment "Oracle HR JOB LOCATIONS table" .

hr:location_id a rdf:Property ;
        rdfs:domain hr:locations ;
        rdfs:range xsd:number ;
        rdfs:label "LOCATION ID" .
```

```
hr:street_address a rdf:Property ;
        rdfs:domain hr:locations ;
        rdfs:range xsd:string ;
        rdfs:label "STREET ADDRESS" .

hr:postal_code a rdf:Property ;
        rdfs:domain hr:locations ;
        rdfs:range xsd:string ;
        rdfs:label "POSTAL CODE" .

hr:city a rdf:Property ;
        rdfs:domain hr:locations ;
        rdfs:range xsd:string ;
        rdfs:label "CITY" .

hr:state_province a rdf:Property ;
        rdfs:domain hr:locations ;
        rdfs:range xsd:string ;
        rdfs:label "STATE PROVINCE" .

hr:country_id a rdf:Property ;
        rdfs:domain hr:locations ;
        rdfs:range hr:countries ;
        rdfs:label "COUNTRY" .
', '', 'http://example.com/schemas/oraclehr', 0);

-----------------------------------------------------------------

----------- Create IRI Classes -------------

create function DB.DBA.JOB_HISTORY (in EMPLOYEE_ID integer, in
START_DATE date) returns varchar
{
  return sprintf_or_null
('http://example.com/oraclehr/job_history/%d_%s#this',
  EMPLOYEE_ID, cast (START_DATE as varchar) );
}
;

create function DB.DBA.JOB_HISTORY_INV_1 (in id varchar) returns integer
{
  return sprintf_inverse (id,
'http://example.com/oraclehr/job_history/%d_%s#this',
2)[0];
}
;

create function DB.DBA.JOB_HISTORY_INV_2 (in id varchar) returns date
{
  declare exit handler for sqlstate '*' { return NULL; };
  return cast (sprintf_inverse (id,
'http://example.com/oraclehr/job_history/%d_%s#this',
2)[1] as date);
}
;

GRANT EXECUTE ON DB.DBA.JOB_HISTORY TO "SPARQL", "SPARQL_UPDATE";
GRANT EXECUTE ON DB.DBA.JOB_HISTORY_URI_INV_1 TO "SPARQL", "SPARQL_UPDATE";
GRANT EXECUTE ON DB.DBA.JOB_HISTORY_URI_INV_2 TO "SPARQL", "SPARQL_UPDATE";

SPARQL

        create iri class <http://example.com/schemas/oraclehr/countries_iri>
        "http://^{URIQADefaultHost}^/oraclehr/countries/%s#this"
        (in COUNTRY_ID varchar not null) .

        create iri class <http://example.com/schemas/oraclehr/regions_iri>
        "http://^{URIQADefaultHost}^/oraclehr/regions/%d#this"
            (in REGION_ID integer not null) .

        create iri class <http://example.com/schemas/oraclehr/departments_iri>
        "http://^{URIQADefaultHost}^/oraclehr/departments/%d#this"
            (in DEPARTMENT_ID integer not null) .
```

```
        create iri class <http://example.com/schemas/oraclehr/employees_iri>
        "http://^{URIQADefaultHost}^/oraclehr/employees/%d#this"
                (in EMPLOYEE_ID integer not null) .

        create iri class <http://example.com/schemas/oraclehr/jobs_iri>
        "http://^{URIQADefaultHost}^/oraclehr/jobs/%s#this"
            (in JOB_ID varchar not null) .

        create iri class <http://example.com/schemas/oraclehr/job_history_iri>
                using function DB.DBA.JOB_HISTORY (in EMPLOYEE_ID integer not null, in
                START_DATE date not null) returns varchar not null,
                function DB.DBA.JOB_HISTORY_INV_1 (in id varchar) returns integer,
                function DB.DBA.JOB_HISTORY_INV_2 (in id varchar) returns date
                option (bijection, returns
                "http://example.com/oraclehr/job_history/%d_%s#this") .

        create iri class <http://example.com/schemas/oraclehr/locations_iri>
        "http://^{URIQADefaultHost}^/oraclehr/locations/%d#this"
            (in LOCATION_ID integer not null) .
;

-----------------------------------------------------------------

------------- Create Quad Store -----------------------------------

SPARQL

prefix hr: <http://example.com/schemas/oraclehr/>

alter quad storage virtrdf:DefaultQuadStorage
  FROM HR.orama.COUNTRIES as countries_tbl
  FROM HR.orama.REGIONS as regions_tbl
  FROM HR.orama.DEPARTMENTS as departments_tbl
  FROM HR.orama.EMPLOYEES as employees_tbl
  FROM HR.orama.EMPLOYEES as employees_tbl_1            ### alias required to represent recursive FK rela
  FROM HR.orama.JOBS as jobs_tbl
  FROM HR.orama.JOB_HISTORY as job_history_tbl
  FROM HR.orama.LOCATIONS as locations_tbl
{
  create virtrdf:oraclehr as
      graph <http://example.com/oraclehr>
  {
        hr:countries_iri(countries_tbl.COUNTRY_ID) a hr:countries  as virtrdf:countires_country_id ;
        hr:country_name countries_tbl.COUNTRY_NAME  as virtrdf:countries_country_name ;
        hr:region_id hr:regions_iri(regions_tbl.REGION_ID) where (^{countries_tbl.}^.REGION_ID = ^{region

        hr:regions_iri(regions_tbl.REGION_ID) a hr:regions as virtrdf:regions_region_id ;
        hr:region_name regions_tbl.REGION_NAME  as virtrdf:regions_region_name .

        hr:departments_iri(departments_tbl.DEPARTMENT_ID) a hr:departments as virtrdf:departments_departm
        hr:department_name departments_tbl.DEPARTMENT_NAME as virtrdf:departments_department_name ;
        hr:location_id hr:locations_iri(locations_tbl.LOCATION_ID) where (^{departments_tbl.}^.LOCATION_I
        hr:manager_id hr:employees_iri(employees_tbl.EMPLOYEE_ID) where (^{departments_tbl.}^.MANAGER_ID

        hr:employees_iri(employees_tbl.EMPLOYEE_ID) a hr:employees as virtrdf:employees_employee_id ;
        hr:department_id hr:departments_iri(departments_tbl.DEPARTMENT_ID) where (^{employees_tbl.}^.DEPA
        hr:job_id hr:jobs_iri(jobs_tbl.JOB_ID) where (^{employees_tbl.}^.JOB_ID = ^{jobs_tbl.}^.JOB_ID) a
        hr:manager_id employees_tbl.MANAGER_ID as virtrdf:employees_manager_id ;
        hr:commissin_pct employees_tbl.COMMISSION_PCT as virtrdf:employees_commission_pct ;
        hr:email employees_tbl.EMAIL as virtrdf:employees_email ;
        hr:first_name employees_tbl.FIRST_NAME as virtrdf:employees_first_name ;
        hr:hire_date employees_tbl.HIRE_DATE as virtrdf:employees_hire_date ;
        hr:last_name employees_tbl.LAST_NAME as virtrdf:employees_last_name ;
        hr:phone_number employees_tbl.PHONE_NUMBER as virtrdf:employees_phone_number ;
        hr:salary employees_tbl.SALARY as virtrdf:employees_salary ;
        hr:has_job_history hr:job_history_iri(job_history_tbl.EMPLOYEE_ID, job_history_tbl.START_DATE) wh
        hr:has_manager hr:employees_iri(employees_tbl_1.EMPLOYEE_ID) where (^{employees_tbl.}^.MANAGER_ID

        hr:locations_iri(locations_tbl.LOCATION_ID) a hr:locations as virtrdf:locations_location_id ;
        hr:country_id hr:countries_iri(countries_tbl.COUNTRY_ID) where (^{locations_tbl.}^.COUNTRY_ID = ^
        hr:city locations_tbl.CITY as virtrdf:locations_city ;
        hr:postal_code locations_tbl.POSTAL_CODE as virtrdf:locations_postal_code ;
```

```
        hr:state_province locations_tbl.STATE_PROVINCE as virtrdf:locations_state_province ;
        hr:street_address locations_tbl.STREET_ADDRESS as virtrdf:locations_street_address .

        hr:jobs_iri(jobs_tbl.JOB_ID) a hr:jobs as virtrdf:jobs_job_id ;
        hr:job_title jobs_tbl.JOB_TITLE as virtrdf:jobs_job_title ;
        hr:max_salary jobs_tbl.MAX_SALARY as virtrdf:jobs_max_salary ;
        hr:min_salary jobs_tbl.MIN_SALARY as virtrdf:jobs_min_salary .

        hr:job_history_iri(job_history_tbl.EMPLOYEE_ID, job_history_tbl.START_DATE) a hr:job_history as v
        hr:employee_id hr:employees_iri(employees_tbl.EMPLOYEE_ID) where (^{job_history_tbl.}^.EMPLOYEE_I
        hr:department_id hr:departments_iri(departments_tbl.DEPARTMENT_ID) where (^{job_history_tbl.}^.DE
        hr:job_id hr:jobs_iri(jobs_tbl.JOB_ID) where (^{job_history_tbl.}^.JOB_ID = ^{jobs_tbl.}^.JOB_ID)
        hr:start_date job_history_tbl.START_DATE as virtrdf:job_history_start_date ;
        hr:end_date job_history_tbl.END_DATE as virtrdf:job_history_end_date .

  } .
} .
;

delete from db.dba.url_rewrite_rule_list where urrl_list like 'oraclehr_rule%';
delete from db.dba.url_rewrite_rule where urr_rule like 'oraclehr_rule%';

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'oraclehr_rule1',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/about/html/http/^{URIQADefaultHost}^%s',
    vector('path'),
    null,
    '(text/html)|(\\*/\\*)',
    0,
    303
    );

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'oraclehr_rule2',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/sparql?query=DESCRIBE+%%3Chttp%%3A//localhost%%3A8890%U%%23this%%3E+%%3Chttp%%3A//localhost%%3A8890
    vector('path', 'path', '*accept*'),
    null,
    '(text/rdf.n3)|(application/rdf.xml)',
    0,
    null
    );

DB.DBA.URLREWRITE_CREATE_RULELIST (
    'oraclehr_rule_list1',
    1,
    vector (
                'oraclehr_rule1',
                'oraclehr_rule2'
          ));

-- ensure a VD for the IRIs which begins with /
VHOST_REMOVE (lpath=>'/oraclehr');

VHOST_DEFINE (
        lpath=>'/oraclehr',
        ppath=>'/DAV/oraclehr/',
        is_dav=>1,
        vsp_user=>'dba',
        is_brws=>0,
        opts=>vector ('url_rewrite', 'oraclehr_rule_list1')
        );

delete from db.dba.url_rewrite_rule_list where urrl_list like 'oracle_schemas_rule%';
delete from db.dba.url_rewrite_rule where urr_rule like 'oracle_schemas_rule%';
```

```
DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'oracle_schemas_rule1',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/about/html/http/^{URIQADefaultHost}^%s',
    vector('path'),
    null,
    '(text/html)|(\\*/\\*)',
    0,
    303
    );

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'oracle_schemas_rule2',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/sparql?query=CONSTRUCT+{+%%3Chttp%%3A//localhost%%3A8890%U%%3E+%%3Fp+%%3Fo+}%%0D%%0AFROM+%%3Chttp%%
    vector('path','path','*accept*'),
    null,
    '(text/rdf.n3)|(application/rdf.xml)',
    0,
    null
    );

DB.DBA.URLREWRITE_CREATE_RULELIST (
    'oracle_schemas_rule_list1',
    1,
    vector (
                'oracle_schemas_rule1',
                'oracle_schemas_rule2'
          ));

-- ensure a VD for the IRIs which begins with /
VHOST_REMOVE (lpath=>'/schema/oraclehr');

VHOST_DEFINE (
        lpath=>'/schemas/oraclehr',
        ppath=>'/DAV/schemas/oraclehr/',
        is_dav=>1,
        vsp_user=>'dba',
        is_brws=>0,
        opts=>vector ('url_rewrite', 'oracle_schemas_rule_list1')
        );

DB.DBA.XML_SET_NS_DECL ('hr', 'http://^{URIQADefaultHost}^/schemas/oraclehr/', 2);
```

16.8.12. Oracle using the demonstration 'Human Resources' database

Live links to a sample instance

- list all employee URIs
- description of employee 105

Script to set up your own instance

```
-- Setup script for Linked Data Views of Oracle 10 Human Resources Sample Database --

GRANT SELECT ON HR.orama.COUNTRIES TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON HR.orama.REGIONS TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON HR.orama.DEPARTMENTS TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON HR.orama.LOCATIONS TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON HR.orama.EMPLOYEES TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON HR.orama.JOBS TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON HR.orama.JOB_HISTORY TO "SPARQL", "SPARQL_UPDATE";

------------------------------------------------------------------------
```

```
-------- Create rdfs:Class definitions ----------------------------

ttlp (
'
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

@prefix hr: <http://example.com/schemas/oraclehr/> .

hr:countries a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/oraclehr> ;
        rdfs:label "COUNTRIES" ;
        rdfs:comment "Oracle HR COUNTRIES table" .

hr:country_id a rdf:Property ;
        rdfs:domain hr:countries ;
        rdfs:range xsd:string ;
        rdfs:label "COUNTRY ID" .

hr:country_name a rdf:Property ;
        rdfs:domain hr:countries ;
        rdfs:range xsd:string ;
        rdfs:label "COUNTRY NAME" .

hr:region_id a rdf:Property ;
        rdfs:domain hr:countries ;
        rdfs:range hr:regions ;
        rdfs:label "REGION ID" .

hr:regions a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/oraclehr> ;
        rdfs:label "REGIONS" ;
        rdfs:comment "Oracle HR REGIONS table" .

hr:region_id a rdf:Property ;
        rdfs:domain hr:regions ;
        rdfs:range xsd:integer ;
        rdfs:label "REGION ID" .

hr:region_name a rdf:Property ;
        rdfs:domain hr:regions ;
        rdfs:range xsd:string ;
        rdfs:label "REGION NAME" .

hr:departments a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/oraclehr> ;
        rdfs:label "DEPARTMENTS" ;
        rdfs:comment "Oracle HR DEPARTMENT table" .

hr:department_id a rdf:Property ;
        rdfs:domain hr:departments ;
        rdfs:range xsd:integer ;
        rdfs:label "DEPARTMENT ID" .

hr:department_name a rdf:Property ;
        rdfs:domain hr:departments ;
        rdfs:range xsd:string ;
        rdfs:comment "DEPARTMENT NAME" .

hr:manager_id a rdf:Property ;
        rdfs:domain hr:departments ;
        rdfs:range hr:employees ;
        rdfs:comment "MANAGER ID" .

hr:location_id a rdf:Property ;
        rdfs:domain hr:departments ;
        rdfs:range hr:locations ;
        rdfs:comment "LOCATION ID" .

hr:employees a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/oraclehr> ;
        rdfs:label "employees" ;
```

```
        rdfs:comment "Oracle HR EMPLOYEES table" .

hr:employee_id a rdf:Property ;
        rdfs:domain hr:employees;
        rdfs:range xsd:integer ;
        rdfs:label "EMPLOYEE ID" .

hr:first_name a rdf:Property ;
        rdfs:domain hr:employees;
        rdfs:range xsd:string ;
        rdfs:label "FIRST NAME" .

hr:last_name a rdf:Property ;
        rdfs:domain hr:employees ;
        rdfs:range xsd:string ;
        rdfs:label "LAST NAME" .

hr:email a rdf:Property ;
        rdfs:domain hr:employees;
        rdfs:range xsd:string ;
        rdfs:label "EMAIL" .

hr:phone_number a rdf:Property ;
        rdfs:domain hr:employees ;
        rdfs:range xsd:string ;
        rdfs:label "PHONE NUMBER" .

hr:hire_date a rdf:Property ;
        rdfs:domain hr:employees ;
        rdfs:range xsd:date ;
        rdfs:label "HIRE DATE" .

hr:job_id a rdf:Property ;
        rdfs:domain hr:employees;
        rdfs:range hr:jobs ;
        rdfs:label "JOB ID" .

hr:salary a rdf:Property ;
        rdfs:domain hr:employees ;
        rdfs:range xsd:integer ;
        rdfs:label "SALARY" .

hr:commission_pct a rdf:Property ;
        rdfs:domain hr:employees ;
        rdfs:range xsd:integer ;
        rdfs:label "COMMISSION PCT" .

hr:manager_id a rdf:Property ;
        rdfs:domain hr:employees ;
        rdfs:range xsd:string ;
        rdfs:label "MANAGER ID" .

hr:department_id a rdf:Property ;
        rdfs:domain hr:employees ;
        rdfs:range hr:departments ;
        rdfs:label "DEPARTMENT ID" .

hr:jobs a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/oraclehr> ;
        rdfs:label "JOBS" ;
        rdfs:comment "Oracle HR JOBS table" .

hr:job_id a rdf:Property ;

        rdfs:domain hr:jobs ;
        rdfs:range xsd:string ;
        rdfs:label "JOB ID" .

hr:job_title a rdf:Property ;
        rdfs:domain hr:jobs ;
        rdfs:range xsd:string ;
        rdfs:label "JOB TITLE" .
```

```
hr:min_salary a rdf:Property ;
        rdfs:domain hr:jobs ;
        rdfs:range xsd:number;
        rdfs:label "MIN SALARY" .

hr:max_salary a rdf:Property ;
        rdfs:domain hr:jobs ;
        rdfs:range xsd:number;
        rdfs:label "MAXSALARY" .

hr:job_history a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/oraclehr> ;
        rdfs:label "JOB HISTORY" ;
        rdfs:comment "Oracle HR JOB HISTORY table" .

hr:employee_id a rdf:Property ;
        rdfs:domain hr:job_history ;
        rdfs:range hr:employees ;
        rdfs:label "EMPLOYEE ID" .

hr:start_date a rdf:Property ;
        rdfs:domain hr:job_history ;
        rdfs:range xsd:date ;
        rdfs:label "START DATE" .

hr:end_date a rdf:Property ;
        rdfs:domain hr:job_history ;
        rdfs:range xsd:date ;
        rdfs:label "END DATE" .

hr:job_id a rdf:Property ;
        rdfs:domain hr:job_history ;
        rdfs:range hr:jobs ;
        rdfs:label "JOB ID" .

hr:department_id a rdf:Property ;
        rdfs:domain hr:job_history ;
        rdfs:range hr:departments ;
        rdfs:label "DEPARTMENT ID" .

hr:locations a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/oraclehr> ;
        rdfs:label "LOCATIONS" ;
        rdfs:comment "Oracle HR JOB LOCATIONS table" .

hr:location_id a rdf:Property ;
        rdfs:domain hr:locations ;
        rdfs:range xsd:number ;
        rdfs:label "LOCATION ID" .

hr:street_address a rdf:Property ;
        rdfs:domain hr:locations ;
        rdfs:range xsd:string ;
        rdfs:label "STREET ADDRESS" .

hr:postal_code a rdf:Property ;
        rdfs:domain hr:locations ;
        rdfs:range xsd:string ;
        rdfs:label "POSTAL CODE" .

hr:city a rdf:Property ;
        rdfs:domain hr:locations ;
        rdfs:range xsd:string ;
        rdfs:label "CITY" .

hr:state_province a rdf:Property ;
        rdfs:domain hr:locations ;
        rdfs:range xsd:string ;
        rdfs:label "STATE PROVINCE" .

hr:country_id a rdf:Property ;
        rdfs:domain hr:locations ;
        rdfs:range hr:countries ;
```

```
        rdfs:label "COUNTRY" .
', '', 'http://example.com/schemas/oraclehr', 0);

-----------------------------------------------------------------

----------- Create IRI Classes -------------

create function DB.DBA.JOB_HISTORY (in EMPLOYEE_ID integer, in
START_DATE date) returns varchar
{
  return sprintf_or_null
('http://example.com/oraclehr/job_history/%d_%s#this',
  EMPLOYEE_ID, cast (START_DATE as varchar) );
}
;

create function DB.DBA.JOB_HISTORY_INV_1 (in id varchar) returns integer
{
  return sprintf_inverse (id,
'http://example.com/oraclehr/job_history/%d_%s#this',
2)[0];
}
;

create function DB.DBA.JOB_HISTORY_INV_2 (in id varchar) returns date
{
  declare exit handler for sqlstate '*' { return NULL; };
  return cast (sprintf_inverse (id,
'http://example.com/oraclehr/job_history/%d_%s#this',
2)[1] as date);
}
;

GRANT EXECUTE ON DB.DBA.JOB_HISTORY TO "SPARQL", "SPARQL_UPDATE";
GRANT EXECUTE ON DB.DBA.JOB_HISTORY_URI_INV_1 TO "SPARQL", "SPARQL_UPDATE";
GRANT EXECUTE ON DB.DBA.JOB_HISTORY_URI_INV_2 TO "SPARQL", "SPARQL_UPDATE";

SPARQL

        create iri class <http://example.com/schemas/oraclehr/countries_iri>
        "http://^{URIQADefaultHost}^/oraclehr/countries/%s#this"
        (in COUNTRY_ID varchar not null) .

        create iri class <http://example.com/schemas/oraclehr/regions_iri>
        "http://^{URIQADefaultHost}^/oraclehr/regions/%d#this"
            (in REGION_ID integer not null) .

        create iri class <http://example.com/schemas/oraclehr/departments_iri>
        "http://^{URIQADefaultHost}^/oraclehr/departments/%d#this"
            (in DEPARTMENT_ID integer not null) .

        create iri class <http://example.com/schemas/oraclehr/employees_iri>
        "http://^{URIQADefaultHost}^/oraclehr/employees/%d#this"
                (in EMPLOYEE_ID integer not null) .

        create iri class <http://example.com/schemas/oraclehr/jobs_iri>
        "http://^{URIQADefaultHost}^/oraclehr/jobs/%s#this"
            (in JOB_ID varchar not null) .

        create iri class <http://example.com/schemas/oraclehr/job_history_iri>
                using function DB.DBA.JOB_HISTORY (in EMPLOYEE_ID integer not null, in
                START_DATE date not null) returns varchar not null,
                function DB.DBA.JOB_HISTORY_INV_1 (in id varchar) returns integer,
                function DB.DBA.JOB_HISTORY_INV_2 (in id varchar) returns date
                option (bijection, returns
                "http://example.com/oraclehr/job_history/%d_%s#this") .

        create iri class <http://example.com/schemas/oraclehr/locations_iri>
        "http://^{URIQADefaultHost}^/oraclehr/locations/%d#this"
            (in LOCATION_ID integer not null) .
;

-----------------------------------------------------------------
```

```
------------- Create Quad Store -----------------------------------

SPARQL

prefix hr: <http://example.com/schemas/oraclehr/>

alter quad storage virtrdf:DefaultQuadStorage
  from HR.orama.COUNTRIES as countries_tbl
  from HR.orama.REGIONS as regions_tbl
  from HR.orama.DEPARTMENTS as departments_tbl
  from HR.orama.EMPLOYEES as employees_tbl
  from HR.orama.EMPLOYEES as employees_tbl_1            ### alias required to represent recursive FK rela
  from HR.orama.JOBS as jobs_tbl
  from HR.orama.JOB_HISTORY as job_history_tbl
  from HR.orama.LOCATIONS as locations_tbl
{
  create virtrdf:oraclehr as
      graph <http://example.com/oraclehr>
  {
        hr:countries_iri(countries_tbl.COUNTRY_ID) a hr:countries  as virtrdf:countires_country_id ;
        hr:country_name countries_tbl.COUNTRY_NAME  as virtrdf:countries_country_name ;
        hr:region_id hr:regions_iri(regions_tbl.REGION_ID) where (^{countries_tbl.}^.REGION_ID = ^{region

        hr:regions_iri(regions_tbl.REGION_ID) a hr:regions as virtrdf:regions_region_id ;
        hr:region_name regions_tbl.REGION_NAME  as virtrdf:regions_region_name .

        hr:departments_iri(departments_tbl.DEPARTMENT_ID) a hr:departments as virtrdf:departments_departm
        hr:department_name departments_tbl.DEPARTMENT_NAME as virtrdf:departments_department_name ;
        hr:location_id hr:locations_iri(locations_tbl.LOCATION_ID) where (^{departments_tbl.}^.LOCATION_I
        hr:manager_id hr:employees_iri(employees_tbl.EMPLOYEE_ID) where (^{departments_tbl.}^.MANAGER_ID

        hr:employees_iri(employees_tbl.EMPLOYEE_ID) a hr:employees as virtrdf:employees_employee_id ;
        hr:department_id hr:departments_iri(departments_tbl.DEPARTMENT_ID) where (^{employees_tbl.}^.DEPA
        hr:job_id hr:jobs_iri(jobs_tbl.JOB_ID) where (^{employees_tbl.}^.JOB_ID = ^{jobs_tbl.}^.JOB_ID) a
        hr:manager_id employees_tbl.MANAGER_ID as virtrdf:employees_manager_id ;
        hr:commissin_pct employees_tbl.COMMISSION_PCT as virtrdf:employees_commission_pct ;
        hr:email employees_tbl.EMAIL as virtrdf:employees_email ;
        hr:first_name employees_tbl.FIRST_NAME as virtrdf:employees_first_name ;
        hr:hire_date employees_tbl.HIRE_DATE as virtrdf:employees_hire_date ;
        hr:last_name employees_tbl.LAST_NAME as virtrdf:employees_last_name ;
        hr:phone_number employees_tbl.PHONE_NUMBER as virtrdf:employees_phone_number ;
        hr:salary employees_tbl.SALARY as virtrdf:employees_salary ;
        hr:has_job_history hr:job_history_iri(job_history_tbl.EMPLOYEE_ID, job_history_tbl.START_DATE) wh
        hr:has_manager hr:employees_iri(employees_tbl_1.EMPLOYEE_ID) where (^{employees_tbl.}^.MANAGER_ID

        hr:locations_iri(locations_tbl.LOCATION_ID) a hr:locations as virtrdf:locations_location_id ;
        hr:country_id hr:countries_iri(countries_tbl.COUNTRY_ID) where (^{locations_tbl.}^.COUNTRY_ID = ^
        hr:city locations_tbl.CITY as virtrdf:locations_city ;
        hr:postal_code locations_tbl.POSTAL_CODE as virtrdf:locations_postal_code ;
        hr:state_province locations_tbl.STATE_PROVINCE as virtrdf:locations_state_province ;
        hr:street_address locations_tbl.STREET_ADDRESS as virtrdf:locations_street_address .

        hr:jobs_iri(jobs_tbl.JOB_ID) a hr:jobs as virtrdf:jobs_job_id ;
        hr:job_title jobs_tbl.JOB_TITLE as virtrdf:jobs_job_title ;
        hr:max_salary jobs_tbl.MAX_SALARY as virtrdf:jobs_max_salary ;
        hr:min_salary jobs_tbl.MIN_SALARY as virtrdf:jobs_min_salary .

        hr:job_history_iri(job_history_tbl.EMPLOYEE_ID, job_history_tbl.START_DATE) a hr:job_history as v
        hr:employee_id hr:employees_iri(employees_tbl.EMPLOYEE_ID) where (^{job_history_tbl.}^.EMPLOYEE_I
        hr:department_id hr:departments_iri(departments_tbl.DEPARTMENT_ID) where (^{job_history_tbl.}^.DE
        hr:job_id hr:jobs_iri(jobs_tbl.JOB_ID) where (^{job_history_tbl.}^.JOB_ID = ^{jobs_tbl.}^.JOB_ID)
        hr:start_date job_history_tbl.START_DATE as virtrdf:job_history_start_date ;
        hr:end_date job_history_tbl.END_DATE as virtrdf:job_history_end_date .

  } .
} .
;

delete from db.dba.url_rewrite_rule_list where urrl_list like 'oraclehr_rule%';
delete from db.dba.url_rewrite_rule where urr_rule like 'oraclehr_rule%';

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
```

```
    'oraclehr_rule1',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/about/html/http/^{URIQADefaultHost}^%s',
    vector('path'),
    null,
    '(text/html)|(\\*/\\*)',
    0,
    303
    );

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'oraclehr_rule2',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/sparql?query=DESCRIBE+%3Chttp%3A//localhost%3A8890%U%23this%3E+%3Chttp%3A//localhost%3A8890
    vector('path', 'path', '*accept*'),
    null,
    '(text/rdf.n3)|(application/rdf.xml)',
    0,
    null
    );

DB.DBA.URLREWRITE_CREATE_RULELIST (
    'oraclehr_rule_list1',
    1,
    vector (
                'oraclehr_rule1',
                'oraclehr_rule2'
          ));

-- ensure a VD for the IRIs which begins with /
VHOST_REMOVE (lpath=>'/oraclehr');

VHOST_DEFINE (
        lpath=>'/oraclehr',
        ppath=>'/DAV/oraclehr/',
        is_dav=>1,
        vsp_user=>'dba',
        is_brws=>0,
        opts=>vector ('url_rewrite', 'oraclehr_rule_list1')
        );

delete from db.dba.url_rewrite_rule_list where urrl_list like 'oracle_schemas_rule%';
delete from db.dba.url_rewrite_rule where urr_rule like 'oracle_schemas_rule%';

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'oracle_schemas_rule1',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/about/html/http/^{URIQADefaultHost}^%s',
    vector('path'),
    null,
    '(text/html)|(\\*/\\*)',
    0,
    303
    );

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'oracle_schemas_rule2',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/sparql?query=CONSTRUCT+{+%3Chttp%3A//localhost%3A8890%U%3E+%3Fp+%3Fo+}%0D%0AFROM+%3Chttp%
    vector('path','path','*accept*'),
    null,
```

```
    '(text/rdf.n3)|(application/rdf.xml)',
    0,
    null
    );

DB.DBA.URLREWRITE_CREATE_RULELIST (
    'oracle_schemas_rule_list1',
    1,
    vector (
                'oracle_schemas_rule1',
                'oracle_schemas_rule2'
          ));

-- ensure a VD for the IRIs which begins with /
VHOST_REMOVE (lpath=>'/schema/oraclehr');

VHOST_DEFINE (
        lpath=>'/schemas/oraclehr',
        ppath=>'/DAV/schemas/oraclehr/',
        is_dav=>1,
        vsp_user=>'dba',
        is_brws=>0,
        opts=>vector ('url_rewrite', 'oracle_schemas_rule_list1')
        );

DB.DBA.XML_SET_NS_DECL ('hr', 'http://^{URIQADefaultHost}^/schemas/oraclehr/', 2);
```

16.8.13. DB2 using the demonstration 'Sample' database

Version defined using explicit host - example.com

```
-- $Id$
-- Setup script for Linked Data Views of portions of DB2 SAMPLE database included
-- in DB2 Express Edition v9.5
--
-- The script assumes external DB2 tables are linked into Virtuoso using
-- local schema name db2sample.

DB..vd_remote_data_source ('db2ma-smpl', '', '<uid>','<pwd>);

ATTACH TABLE  "DB2ADMIN"."ACT"      PRIMARY KEY ("ACTNO")              AS "DB"."db2sample"."ACT"      FRO
ATTACH TABLE  "DB2ADMIN"."DEPARTMENT"      PRIMARY KEY ("DEPTNO")              AS "DB"."db2sample"."DEPAR
ATTACH TABLE  "DB2ADMIN"."EMPLOYEE"      PRIMARY KEY ("EMPNO")              AS "DB"."db2sample"."EMPLOYEE
ATTACH TABLE  "DB2ADMIN"."EMPPROJACT"      PRIMARY KEY ("EMPNO", "PROJNO", "ACTNO", "EMSTDATE")
ATTACH TABLE  "DB2ADMIN"."EMP_RESUME"      PRIMARY KEY ("EMPNO", "RESUME_FORMAT")              AS "DB"."db
ATTACH TABLE  "DB2ADMIN"."PROJACT"      PRIMARY KEY ("PROJNO", "ACTNO", "ACSTDATE")              AS "DB".
ATTACH TABLE  "DB2ADMIN"."PROJECT"      PRIMARY KEY ("PROJNO")              AS "DB"."db2sample"."PROJECT"

COMMIT WORK;

GRANT SELECT ON DB.db2sample.ACT TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON DB.db2sample.DEPARTMENT TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON DB.db2sample.EMPLOYEE TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON DB.db2sample.EMPPROJACT TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON DB.db2sample.EMP_RESUME TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON DB.db2sample.PROJACT TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON DB.db2sample.PROJECT TO "SPARQL", "SPARQL_UPDATE";

create function DB.DBA.PROJ_ACT_IRI (
  in proj_no varchar,
  in act_no integer,
  in ac_st_date date
  ) returns varchar
{
  declare _act_no, _datetime, _date any;
  _act_no := cast(act_no as varchar);
  _datetime := cast(ac_st_date as varchar);
  _date := left(_datetime, 10);
  return sprintf('http://example.com/db2sample/proj_act/%s_%s_%s#this',
    proj_no, _act_no, _date);
};
```

```
create function
DB.DBA.PROJ_ACT_IRI_INV_1 (in proj_act_iri varchar) returns varchar
{
  declare exit handler for sqlstate '*' { return NULL; };
  declare parts any;
  parts := sprintf_inverse (proj_act_iri,
   'http://example.com/db2sample/proj_act/%s_%s_%s#this', 1);
  if (parts is not null)
  {
    return parts[0];
  }
  return NULL;
};

create function
DB.DBA.PROJ_ACT_IRI_INV_2 (in proj_act_iri varchar) returns integer
{
  declare exit handler for sqlstate '*' { return NULL; };
  declare parts any;
  parts := sprintf_inverse (proj_act_iri,
   'http://example.com/db2sample/proj_act/%s_%s_%s#this', 1);
  if (parts is not null)
  {
    return cast(parts[1] as integer);
  }
  return NULL;
};

create function
DB.DBA.PROJ_ACT_IRI_INV_3 (in proj_act_iri varchar) returns date
{
  declare exit handler for sqlstate '*' { return NULL; };
  declare parts any;
  parts := sprintf_inverse (proj_act_iri,
   'http://example.com/db2sample/proj_act/%s_%s_%s#this', 1);
  if (parts is not null)
  {
    return cast(parts[2] as date);
  }
  return NULL;
};

create function DB.DBA.EMP_PROJ_ACT_IRI (
  in emp_no varchar,
  in proj_no varchar,
  in act_no integer,
  in emp_start_date date
  ) returns varchar
{
  declare _act_no, _datetime, _date any;
  _act_no := cast(act_no as varchar);
  _datetime := cast(emp_start_date as varchar);
  _date := left(_datetime, 10);
  return sprintf(
      'http://example.com/db2sample/emp_proj_act/%s_%s_%s_%s#this',
    emp_no, proj_no, _act_no, _date);
};

create function
DB.DBA.EMP_PROJ_ACT_IRI_INV_1 (in emp_proj_act_iri varchar) returns varchar
{
  declare exit handler for sqlstate '*' { return NULL; };
  declare parts any;
  parts := sprintf_inverse (emp_proj_act_iri,
   'http://example.com/db2sample/emp_proj_act/%s_%s_%s_%s#this', 1);
  if (parts is not null)
  {
    return parts[0];
  }
  return NULL;
};

create function
```

```
DB.DBA.EMP_PROJ_ACT_IRI_INV_2 (in emp_proj_act_iri varchar) returns varchar
{
  declare exit handler for sqlstate '*' { return NULL; };
  declare parts any;
  parts := sprintf_inverse (emp_proj_act_iri,
   'http://example.com/db2sample/emp_proj_act/%s_%s_%s_%s#this', 1);
  if (parts is not null)
  {
    return parts[1];
  }
  return NULL;
};

create function
DB.DBA.EMP_PROJ_ACT_IRI_INV_3 (in emp_proj_act_iri varchar) returns integer
{
  declare exit handler for sqlstate '*' { return NULL; };
  declare parts any;
  parts := sprintf_inverse (emp_proj_act_iri,
   'http://example.com/db2sample/emp_proj_act/%s_%s_%s_%s#this', 1);
  if (parts is not null)
  {
    return cast(parts[2] as integer);
  }
  return NULL;
};

create function
DB.DBA.EMP_PROJ_ACT_IRI_INV_4 (in emp_proj_act_iri varchar) returns date
{
  declare exit handler for sqlstate '*' { return NULL; };
  declare parts any;
  parts := sprintf_inverse (emp_proj_act_iri,
   'http://example.com/db2sample/emp_proj_act/%s_%s_%s_%s#this', 1);
  if (parts is not null)
  {
    return cast(parts[3] as date);
  }
  return NULL;
};

grant execute on DB.DBA.PROJ_ACT_IRI to "SPARQL", "SPARQL_UPDATE";
grant execute on DB.DBA.PROJ_ACT_IRI_INV_1 to "SPARQL", "SPARQL_UPDATE";
grant execute on DB.DBA.PROJ_ACT_IRI_INV_2 to "SPARQL", "SPARQL_UPDATE";
grant execute on DB.DBA.PROJ_ACT_IRI_INV_3 to "SPARQL", "SPARQL_UPDATE";

grant execute on DB.DBA.EMP_PROJ_ACT_IRI to "SPARQL", "SPARQL_UPDATE";
grant execute on DB.DBA.EMP_PROJ_ACT_IRI_INV_1 to "SPARQL", "SPARQL_UPDATE";
grant execute on DB.DBA.EMP_PROJ_ACT_IRI_INV_2 to "SPARQL", "SPARQL_UPDATE";
grant execute on DB.DBA.EMP_PROJ_ACT_IRI_INV_3 to "SPARQL", "SPARQL_UPDATE";
grant execute on DB.DBA.EMP_PROJ_ACT_IRI_INV_4 to "SPARQL", "SPARQL_UPDATE";

SPARQL drop graph <http://example.com/schemas/db2sample> ;
SPARQL drop graph <http://example.com/db2sample> ;

SPARQL drop quad map virtrdf:db2sample ;

---------------------------
-- RDFS class definitions
ttlp (
'
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

@prefix opl: <http://example.com/schemas/db2sample/> .

opl:Act a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/db2sample> ;
        rdfs:label "Act" ;
        rdfs:comment "Activity" .

# ACTNO SMALLINT PRIMARY KEY
```

```
opl:act_no a rdf:Property ;
        rdfs:domain opl:Act ;
        rdfs:range xsd:integer ;
        rdfs:label "Activity number" .

# ACTKWD VARCHAR(6)
opl:act_kwd a rdf:Property ;
        rdfs:domain opl:Act ;
        rdfs:range xsd:string ;
        rdfs:label "Activity keyword" .

# ACTDESC VARCHAR(20)
opl:act_desc a rdf:Property ;
        rdfs:domain opl:Act ;
        rdfs:range xsd:string ;
        rdfs:label "Activity description" .

#####
opl:Department a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/db2sample> ;
        rdfs:label "Department" ;
        rdfs:comment "Department" .

# DEPTNO VARCHAR(3) PRIMARY KEY
opl:dept_no a rdf:Property ;
        rdfs:domain opl:Department ;
        rdfs:range xsd:string ;
        rdfs:label "Department number" .

# DEPTNAME VARCHAR(36)
opl:dept_name a rdf:Property ;
        rdfs:domain opl:Department ;
        rdfs:range xsd:string ;
        rdfs:label "Department name" .

# MGRNO CHAR(6)
opl:dept_manager a rdf:Property ;
        rdfs:domain opl:Department ;
        rdfs:range xsd:Employee ;
        rdfs:label "Department manager" .

# ADMRDEPT CHAR(3)
opl:supervising_dept a rdf:Property ;
        rdfs:domain opl:Department ;
        rdfs:range opl:Department ;
        rdfs:label "Department reported to" .

# LOCATION CHAR(6)
opl:location a rdf:Property ;
        rdfs:domain opl:Department ;
        rdfs:range xsd:string ;
        rdfs:label "Location" .

opl:employee_collection a rdf:Property ;
        rdfs:domain opl:Department ;
        rdfs:range opl:Employee ;
        rdfs:label "Department employees" .

opl:dept_project_collection a rdf:Property ;
        rdfs:domain opl:Department ;
        rdfs:range opl:Project ;
        rdfs:label "Department projects" .

#####
opl:Employee a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/db2sample> ;
        rdfs:label "Employee" ;
        rdfs:comment "Employee" .

# EMPNO VARCHAR(6) PRIMARY KEY
opl:emp_no a rdf:Property ;
        rdfs:domain opl:Employee;
        rdfs:range xsd:string ;
```

```
        rdfs:label "Employee number" .

# FIRSTNME VARCHAR(12)
opl:first_name a rdf:Property ;
        rdfs:domain opl:Employee ;
        rdfs:range xsd:string ;
        rdfs:label "First name" .

# MIDINIT VARCHAR(1)
opl:middle_initial a rdf:Property ;
        rdfs:domain opl:Employee ;
        rdfs:range xsd:string ;
        rdfs:label "Middle initial" .

# LASTNAME VARCHAR(15)
opl:last_name a rdf:Property ;
        rdfs:domain opl:Employee ;
        rdfs:range xsd:string ;
        rdfs:label "Last name" .

# WORKDEPT VARCHAR(3)
opl:work_dept a rdf:Property ;
        rdfs:domain opl:Employee ;
        rdfs:range opl:Department ;
        rdfs:label "Work department" .

# PHONENO VARCHAR(4)
opl:phone_no a rdf:Property ;
        rdfs:domain opl:Employee ;
        rdfs:range xsd:string ;
        rdfs:label "Phone number" .

# HIREDATE DATE
opl:hire_date a rdf:Property ;
        rdfs:domain opl:Employee ;
        rdfs:range xsd:date;
        rdfs:label "Hire date" .

# JOB VARCHAR(8)
opl:job a rdf:Property ;
        rdfs:domain opl:Employee ;
        rdfs:range xsd:string ;
        rdfs:label "Job" .

# EDLEVEL SMALLINT
opl:education_level a rdf:Property ;
        rdfs:domain opl:Employee ;
        rdfs:range xsd:integer ;
        rdfs:label "Education level" .

# SEX VARCHAR(1)
opl:gender a rdf:Property ;
        rdfs:domain opl:Employee ;
        rdfs:range xsd:string ;
        rdfs:label "Gender" .

# BIRTHDATE DATE
opl:date_of_birth a rdf:Property ;
        rdfs:domain opl:Employee ;
        rdfs:range xsd:date ;
        rdfs:label "Date of birth" .

# SALARY DECIMAL(9,2)
opl:salary a rdf:Property ;
        rdfs:domain opl:Employee ;
        rdfs:range xsd:decimal ;
        rdfs:label "Salary" .

# BONUS DECIMAL(9,2)
opl:bonus a rdf:Property ;
        rdfs:domain opl:Employee ;
        rdfs:range xsd:decimal ;
        rdfs:label "Bonus" .
```

```
# COMM DECIMAL(9,2)
opl:commission a rdf:Property ;
        rdfs:domain opl:Employee ;
        rdfs:range xsd:decimal ;
        rdfs:label "Commission" .

opl:resume_collection a rdf:Property ;
        rdfs:domain opl:Employee ;
        rdfs:range opl:EmployeeResume ;
        rdfs:label "Employee resumes" .

opl:projects_responsible_for_collection a rdf:Property ;
        rdfs:domain opl:Employee ;
        rdfs:range opl:Project ;
        rdfs:label "responsible for project" .

opl:activity_collection a rdf:Property ;
        rdfs:domain opl:Employee ;
        rdfs:range opl:EmpProjAct ;
        rdfs:label "project activities" .

#####
opl:EmpProjAct a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/db2sample> ;
        rdfs:label "EmpProjAct" ;
        rdfs:comment "Employee project activity" .

# EMPNO VARCHAR(6) PRIMARY KEY
opl:epa_emp_no a rdf:Property ;
        rdfs:domain opl:EmpProjAct ;
        rdfs:range xsd:string ;
        rdfs:label "Employee number" .

# PROJNO VARCHAR(6) PRIMARY KEY
opl:epa_proj_no a rdf:Property ;
        rdfs:domain opl:EmpProjAct ;
        rdfs:range xsd:string ;
        rdfs:label "Project number" .

# ACTNO SMALLINT PRIMARY KEY
opl:epa_act_no a rdf:Property ;
        rdfs:domain opl:EmpProjAct ;
        rdfs:range xsd:string ;
        rdfs:label "Activity number" .

# EMSTDATE DATE PRIMARY KEY
opl:emp_start_date a rdf:Property ;
        rdfs:domain opl:EmpProjAct ;
        rdfs:range xsd:date ;
        rdfs:label "Employee activity start date" .

# EMPTIME DECIMAL(5,2)
opl:emp_time a rdf:Property ;
        rdfs:domain opl:EmpProjAct ;
        rdfs:range xsd:decimal ;
        rdfs:label "Employee time" .

# EMENDATE DATE PRIMARY KEY
opl:emp_end_date a rdf:Property ;
        rdfs:domain opl:EmpProjAct ;
        rdfs:range xsd:date ;
        rdfs:label "Employee activity end date" .

opl:assigned_to a rdf:Property ;
        rdfs:domain opl:EmpProjAct ;
        rdfs:range opl:Employee ;
        rdfs:label "Assigned to" .

opl:project_activity a rdf:Property ;
        rdfs:domain opl:EmpProjAct ;
        rdfs:range opl:ProjAct ;
        rdfs:label "Project activity" .
```

```
#####
opl:EmployeeResume a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/db2sample> ;
        rdfs:label "EmployeeResume" ;
        rdfs:comment "Employee resume" .

# EMPNO VARCHAR(6) PRIMARY KEY
opl:er_emp_no a rdf:Property ;
        rdfs:domain opl:EmployeeResume ;
        rdfs:range xsd:string ;
        rdfs:label "Employee number" .

# RESUME_FORMAT VARCHAR(10) PRIMARY KEY
opl:resume_format a rdf:Property ;
        rdfs:domain opl:EmployeeResume ;
        rdfs:range xsd:string ;
        rdfs:label "Resume format" .

# RESUME VARCHAR(5120)
opl:resume a rdf:Property ;
        rdfs:domain opl:EmployeeResume ;
        rdfs:range xsd:string ;
        rdfs:label "Resume" .

opl:resume_of a rdf:Property ;
        rdfs:domain opl:EmployeeResume ;
        rdfs:range opl:Employee ;
        rdfs:label "Resume subject" .

#####
opl:ProjAct a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/db2sample> ;
        rdfs:label "ProjAct" ;
        rdfs:comment "Project activity" .

# PROJNO VARCHAR(6) PRIMARY KEY
opl:pa_proj_no a rdf:Property ;
        rdfs:domain opl:ProjAct ;
        rdfs:range xsd:string ;
        rdfs:label "Project number" .

# ACTNO SMALLINT PRIMARY KEY
opl:pa_act_no a rdf:Property ;
        rdfs:domain opl:ProjAct ;
        rdfs:range xsd:string ;
        rdfs:label "Activity number" .

# ACSTDATE DATE PRIMARY KEY
opl:ac_st_date a rdf:Property ;
        rdfs:domain opl:ProjAct ;
        rdfs:range xsd:date ;
        rdfs:label "Activity start date" .

# ACSTAFF DECIMAL(5,2)
opl:ac_staff a rdf:Property ;
        rdfs:domain opl:ProjAct ;
        rdfs:range xsd:decimal ;
        rdfs:label "Acstaff" .

# ACENDATE DATE
opl:ac_en_date a rdf:Property ;
        rdfs:domain opl:ProjAct ;
        rdfs:range xsd:date ;
        rdfs:label "Activity end date" .

opl:project a rdf:Property ;
        rdfs:domain opl:ProjAct ;
        rdfs:range opl:Project ;
        rdfs:label "Project" .

opl:activity a rdf:Property ;
        rdfs:domain opl:ProjAct ;
```

```
        rdfs:range opl:Act ;
        rdfs:label "Activity" .

opl:employee_activity_collection a rdf:Property ;
        rdfs:domain opl:ProjAct ;
        rdfs:range opl:EmpProjAct ;
        rdfs:label "Employee activity collection" .

#####
opl:Project a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/db2sample> ;
        rdfs:label "Project" ;
        rdfs:comment "Project" .

# PROJNO VARCHAR(6) PRIMARY KEY
opl:proj_no a rdf:Property ;
        rdfs:domain opl:Project ;
        rdfs:range xsd:string ;
        rdfs:label "Project number" .

# PROJNAME VARCHAR(24)
opl:proj_name a rdf:Property ;
        rdfs:domain opl:Project ;
        rdfs:range xsd:string ;
        rdfs:label "Project name" .

# DEPTNO CHAR(3)
opl:is_project_of_department a rdf:Property ;
        rdfs:domain opl:Project ;
        rdfs:range opl:Department ;
        rdfs:label "is project of department" .

# RESPEMP VARCHAR(6)
opl:resp_emp a rdf:Property ;
        rdfs:domain opl:Project ;
        rdfs:range opl:Employee ;
        rdfs:label "Employee responsible" .

# PRSTAFF DECIMAL(5,2)
opl:pr_staff a rdf:Property ;
        rdfs:domain opl:Project ;
        rdfs:range xsd:decimal ;
        rdfs:label "PrStaff" .

# PRSTDATE DATE
opl:pr_st_date a rdf:Property ;
        rdfs:domain opl:Project ;
        rdfs:range xsd:date ;
        rdfs:label "Project start date" .

# PRENDATE DATE
opl:pr_en_date a rdf:Property ;
        rdfs:domain opl:Project ;
        rdfs:range xsd:date ;
        rdfs:label "Project end date" .

# MAJPROJ VARCHAR(6)
opl:maj_proj a rdf:Property ;
        rdfs:domain opl:Project ;
        rdfs:range xsd:string ;
        rdfs:label "MajProj" .

opl:proj_activity_collection a rdf:Property ;
        rdfs:domain opl:Project ;
        rdfs:range opl:ProjAct ;
        rdfs:label "Project activities" .

', '', 'http://example.com/schemas/db2sample', 0);

--------------------------

SPARQL
prefix opl: <http://example.com/schemas/db2sample/>
```

```
create iri class
<http://example.com/schemas/db2sample/act_iri>
        "http://example.com/db2sample/act/%d#this"
        (
         in act_no integer not null
        ) .

create iri class
<http://example.com/schemas/db2sample/department_iri>
        "http://example.com/db2sample/department/%s#this"
        (
         in dept_no varchar not null
        ) .

create iri class
<http://example.com/schemas/db2sample/employee_iri>
        "http://example.com/db2sample/employee/%s#this"
        (
         in emp_no varchar not null
        ) .

create iri class opl:emp_proj_act_iri using
        function DB.DBA.EMP_PROJ_ACT_IRI (
                in emp_no varchar,
                in proj_no varchar,
                in act_no integer,
                in emp_start_date date
                ) returns varchar,
        function DB.DBA.EMP_PROJ_ACT_IRI_INV_1 (in emp_proj_act_iri varchar)
                returns varchar ,
        function DB.DBA.EMP_PROJ_ACT_IRI_INV_2 (in emp_proj_act_iri varchar)
                returns varchar ,
        function DB.DBA.EMP_PROJ_ACT_IRI_INV_3 (in emp_proj_act_iri varchar)
                returns integer ,
        function DB.DBA.EMP_PROJ_ACT_IRI_INV_4 (in emp_proj_act_iri varchar)
                returns date
        option (bijection, returns
          "http://example.com/db2sample/emp_proj_act/%s_%s_%s_%s#this")
        .

create iri class
<http://example.com/schemas/db2sample/employee_resume_iri>
        "http://example.com/db2sample/employee_resume/%s_%s#this"
        (
         in emp_no varchar not null,
         in resume_format varchar not null
        ) .

create iri class opl:proj_act_iri using
        function DB.DBA.PROJ_ACT_IRI (
                in proj_no varchar,
                in act_no integer,
                in ac_st_date date
                ) returns varchar,
        function DB.DBA.PROJ_ACT_IRI_INV_1 (in proj_act_iri varchar)
                returns varchar ,
        function DB.DBA.PROJ_ACT_IRI_INV_2 (in proj_act_iri varchar)
                returns integer ,
        function DB.DBA.PROJ_ACT_IRI_INV_3 (in proj_act_iri varchar)
                returns date
        option (bijection, returns
                "http://example.com/db2sample/proj_act/%s_%s_%s#this")
        .

create iri class
<http://example.com/schemas/db2sample/project_iri>
        "http://example.com/db2sample/project/%s#this"
        (
         in proj_no varchar not null
        ) .
;
```

```
SPARQL
prefix opl: <http://example.com/schemas/db2sample/>

alter quad storage virtrdf:DefaultQuadStorage
from DB.db2sample.ACT as act_tbl
from DB.db2sample.DEPARTMENT as dept_tbl
from DB.db2sample.EMPLOYEE as emp_tbl
from DB.db2sample.EMPPROJACT as emp_proj_act_tbl
from DB.db2sample.EMP_RESUME as emp_resume_tbl
from DB.db2sample.PROJACT as proj_act_tbl
from DB.db2sample.PROJECT as project_tbl
{
        create virtrdf:db2sample as
                graph <http://example.com/db2sample>
        {
        opl:act_iri(act_tbl.ACTNO) a opl:Act
                as virtrdf:act_id ;
        opl:act_no act_tbl.ACTNO
                as virtrdf:act_act_no ;
        opl:act_kwd act_tbl.ACTKWD
                as virtrdf:act_act_kwd ;
        opl:act_desc act_tbl.ACTDESC
                as virtrdf:act_act_desc .

        opl:department_iri(dept_tbl.DEPTNO) a opl:Department
                as virtrdf:dept_id ;
        opl:dept_no dept_tbl.DEPTNO
                as virtrdf:dept_dept_no ;
        opl:dept_name dept_tbl.DEPTNAME
                as virtrdf:dept_dept_name ;
        opl:dept_manager opl:employee_iri(dept_tbl.MGRNO)
                as virtrdf:dept_mgr_no ;
        opl:supervising_dept opl:department_iri(dept_tbl.ADMRDEPT)
                as virtrdf:dept_supervising_dept ;
        opl:location dept_tbl.LOCATION
                as virtrdf:dept_location ;
        opl:employee_collection opl:employee_iri(emp_tbl.EMPNO)
                where (^{emp_tbl.}^.WORKDEPT = ^{dept_tbl.}^.DEPTNO)
                as virtrdf:dept_employee_collection ;
        opl:dept_project_collection opl:project_iri(project_tbl.PROJNO)
                where (^{project_tbl.}^.DEPTNO = ^{dept_tbl.}^.DEPTNO)
                as virtrdf:dept_project_collection .

        opl:employee_iri(emp_tbl.EMPNO) a opl:Employee
                as virtrdf:employee_id ;
        opl:emp_no emp_tbl.EMPNO
                as virtrdf:employee_emp_no ;
        opl:first_name emp_tbl.FIRSTNME
                as virtrdf:employee_first_name ;
        opl:middle_initial emp_tbl.MIDINIT
                as virtrdf:employee_middle_initial ;
        opl:last_name emp_tbl.LASTNAME
                as virtrdf:employee_last_name ;
        opl:work_dept opl:department_iri(emp_tbl.WORKDEPT)
                as virtrdf:employee_work_dept ;
        opl:phone_no emp_tbl.PHONENO
                as virtrdf:employee_phone_no ;
        opl:hire_date emp_tbl.HIREDATE
                as virtrdf:employee_hire_date ;
        opl:job emp_tbl.JOB
                as virtrdf:employee_job ;
        opl:education_level emp_tbl.EDLEVEL
                as virtrdf:employee_education_level ;
        opl:gender emp_tbl.SEX
                as virtrdf:employee_gender ;
        opl:date_of_birth emp_tbl.BIRTHDATE
                as virtrdf:employee_date_of_birth ;
        opl:salary emp_tbl.SALARY
                as virtrdf:employee_salary ;
        opl:bonus emp_tbl.BONUS
                as virtrdf:employee_bonus ;
        opl:commission emp_tbl.COMM
                as virtrdf:employee_commission ;
```

```
opl:resume_collection opl:employee_resume_iri(
        emp_resume_tbl.EMPNO,
        emp_resume_tbl.RESUME_FORMAT
        )
        where (^{emp_tbl.}^.EMPNO = ^{emp_resume_tbl.}^.EMPNO)
        as virtrdf:employee_resume_collection ;
opl:projects_responsible_for_collection
        opl:project_iri(project_tbl.PROJNO)
        where (^{project_tbl.}^.RESPEMP = ^{emp_tbl.}^.EMPNO)
        as virtrdf:employee_projects_responsible_for_collection ;
opl:activity_collection opl:emp_proj_act_iri(
        emp_proj_act_tbl.EMPNO,
        emp_proj_act_tbl.PROJNO,
        emp_proj_act_tbl.ACTNO,
        emp_proj_act_tbl.EMSTDATE
        )
        where (^{emp_tbl.}^.EMPNO = ^{emp_proj_act_tbl.}^.EMPNO)
        as virtrdf:employee_activity_collection .

opl:emp_proj_act_iri(
        emp_proj_act_tbl.EMPNO,
        emp_proj_act_tbl.PROJNO,
        emp_proj_act_tbl.ACTNO,
        emp_proj_act_tbl.EMSTDATE
        ) a opl:EmpProjAct
        as virtrdf:empprojact_id ;
opl:epa_emp_no emp_proj_act_tbl.EMPNO
        as virtrdf:empprojact_emp_no ;
opl:epa_proj_no emp_proj_act_tbl.PROJNO
        as virtrdf:empprojact_proj_no ;
opl:epa_act_no emp_proj_act_tbl.ACTNO
        as virtrdf:empprojact_act_no ;
opl:emp_start_date emp_proj_act_tbl.EMSTDATE
        as virtrdf:empprojact_emp_start_date ;
opl:emp_time emp_proj_act_tbl.EMPTIME
        as virtrdf:empprojact_emp_time ;
opl:emp_end_date emp_proj_act_tbl.EMENDATE
        as virtrdf:empprojact_emp_end_date ;
opl:assigned_to opl:employee_iri(emp_proj_act_tbl.EMPNO)
        as virtrdf:empprojact_assigned_to ;
opl:project_activity opl:proj_act_iri(
        emp_proj_act_tbl.PROJNO,
        emp_proj_act_tbl.ACTNO,
        emp_proj_act_tbl.EMSTDATE
        )
        as virtrdf:empprojact_project_activity .

opl:employee_resume_iri(
        emp_resume_tbl.EMPNO,
        emp_resume_tbl.RESUME_FORMAT
        ) a opl:EmployeeResume
        as virtrdf:employee_resume_id ;
opl:er_emp_no emp_resume_tbl.EMPNO
        as virtrdf:employee_resume_emp_no ;
opl:resume_format emp_resume_tbl.RESUME_FORMAT
        as virtrdf:employee_resume_resume_format ;
opl:resume emp_resume_tbl.RESUME
        as virtrdf:employee_resume_resume ;
opl:resume_of opl:employee_iri(emp_resume_tbl.EMPNO)
        as virtrdf:employee_resume_resume_of .

opl:proj_act_iri(
        proj_act_tbl.PROJNO,
        proj_act_tbl.ACTNO,
        proj_act_tbl.ACSTDATE
        ) a opl:ProjAct
        as virtrdf:projact_id;
opl:pa_proj_no proj_act_tbl.PROJNO
        as virtrdf:projact_proj_no ;
opl:pa_act_no proj_act_tbl.ACTNO
        as virtrdf:projact_act_no ;
opl:ac_st_date proj_act_tbl.ACSTDATE
        as virtrdf:projact_ac_st_date ;
```

```
        opl:ac_staff proj_act_tbl.ACSTAFF
                as virtrdf:projact_ac_staff ;
        opl:ac_en_date proj_act_tbl.ACENDATE
                as virtrdf:projact_ac_en_date ;
        opl:project opl:project_iri(proj_act_tbl.PROJNO)
                as virtrdf:projact_project ;
        opl:activity opl:act_iri(proj_act_tbl.ACTNO)
                as virtrdf:projact_activity ;
        opl:employee_activity_collection opl:emp_proj_act_iri(
                emp_proj_act_tbl.EMPNO,
                emp_proj_act_tbl.PROJNO,
                emp_proj_act_tbl.ACTNO,
                emp_proj_act_tbl.EMSTDATE
                )
                where (
                ^{proj_act_tbl.}^.PROJNO = ^{emp_proj_act_tbl.}^.PROJNO AND
                ^{proj_act_tbl.}^.ACTNO = ^{emp_proj_act_tbl.}^.ACTNO AND
                ^{proj_act_tbl.}^.ACSTDATE = ^{emp_proj_act_tbl.}^.EMSTDATE
                )
                as virtrdf:project_employee_activity_collection .

        opl:project_iri(project_tbl.PROJNO) a opl:Project
                as virtrdf:project_id ;
        opl:proj_no project_tbl.PROJNO
                as virtrdf:project_proj_no ;
        opl:proj_name project_tbl.PROJNAME
                as virtrdf:project_proj_name ;
        opl:is_project_of_department opl:department_iri(project_tbl.DEPTNO)
                as virtrdf:project_is_project_of_department ;
        opl:resp_emp opl:employee_iri(project_tbl.RESPEMP)
                as virtrdf:project_resp_emp ;
        opl:pr_staff project_tbl.PRSTAFF
                as virtrdf:project_pr_staff ;
        opl:pr_st_date project_tbl.PRSTDATE
                as virtrdf:project_pr_st_date ;
        opl:pr_en_date project_tbl.PRENDATE
                as virtrdf:project_pr_en_date ;
        opl:maj_proj project_tbl.MAJPROJ
                as virtrdf:project_maj_proj ;
        opl:proj_activity_collection opl:proj_act_iri(
                proj_act_tbl.PROJNO,
                proj_act_tbl.ACTNO,
                proj_act_tbl.ACSTDATE
                )
                where (^{project_tbl.}^.PROJNO = ^{proj_act_tbl.}^.PROJNO)
                as virtrdf:project_activity_collection .
        } .
} .
;

delete from db.dba.url_rewrite_rule_list where urrl_list like 'db2sample_rule%';
delete from db.dba.url_rewrite_rule where urr_rule like 'db2sample_rule%';

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'db2sample_rule1',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/about/html/http/example.com%s',
    vector('path'),
    null,
    '(text/html)|(\\*/\\*)',
    0,
    303
    );

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'db2sample_rule2',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
```

```
    '/sparql?query=DESCRIBE+%%3Chttp%%3A//localhost%%3A8890%U%%23this%%3E+%%3Chttp%%3A//localhost%%3A8890

    vector('path','path','*accept*'),
    null,
    '(text/rdf.n3)|(application/rdf.xml)',
    0,
    null
    );

DB.DBA.URLREWRITE_CREATE_RULELIST (
    'db2sample_rule_list1',
    1,
    vector (
                'db2sample_rule1',
                'db2sample_rule2'
          ));

-- ensure a VD for the IRIs which begins with /
VHOST_REMOVE (lpath=>'/db2sample');

VHOST_DEFINE (
        lpath=>'/db2sample',
        ppath=>'/DAV/db2sample/',
        vsp_user=>'dba',
        is_dav=>1,
        is_brws=>0,
        opts=>vector ('url_rewrite', 'db2sample_rule_list1')
        );
delete from db.dba.url_rewrite_rule_list where urrl_list like 'db2sample_schema_rule%';
delete from db.dba.url_rewrite_rule where urr_rule like 'db2sample_schema_rule%';

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'db2sample_schema_rule1',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/about/html/http/example.com%U',
    vector('path'),
    null,
    '(text/html)|(\\*/\\*)',
    0,
    303
    );

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'db2sample_schema_rule2',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/sparql?query=CONSTRUCT+{+%%3Chttp%%3A//localhost%%3A8890%U%%3E+%%3Fp+%%3Fo+}%%0D%%0AFROM+%%3Chttp%%
    vector('path','path','*accept*'),
    null,
    '(text/rdf.n3)|(application/rdf.xml)',
    0,
    null
    );

DB.DBA.URLREWRITE_CREATE_RULELIST (
    'db2sample_schema_rule_list1',
    1,
    vector (
                'db2sample_schema_rule1',
                'db2sample_schema_rule2'
          ));

-- ensure a VD for the IRIs which begins with /
VHOST_REMOVE (lpath=>'/schemas/db2sample');

VHOST_DEFINE (
        lpath=>'/schemas/db2sample',
        ppath=>'/DAV/schemas_db2sample/',
```

```
        vsp_user=>'dba',
        is_dav=>1,
        is_brws=>0,
        opts=>vector ('url_rewrite', 'db2sample_schema_rule_list1')
        );
```

16.8.14. Informix using demonstration 'Stores' database

```
DB..vd_remote_data_source ('inf10_stores_demo_rdf', '', '<uid>','<pwd>');

ATTACH TABLE  "informix"."call_type"  PRIMARY KEY ("call_code")                  AS "stores_demo_rdf"."in
ATTACH TABLE  "informix"."catalog"    PRIMARY KEY ("catalog_num")               AS "stores_demo_rdf"."in
ATTACH TABLE  "informix"."cust_calls" PRIMARY KEY ("customer_num", "call_dtime") AS "stores_demo_rdf"."in
ATTACH TABLE  "informix"."customer"   PRIMARY KEY ("customer_num")              AS "stores_demo_rdf"."in
ATTACH TABLE  "informix"."items"      PRIMARY KEY ("item_num", "order_num")     AS "stores_demo_rdf"."in
ATTACH TABLE  "informix"."manufact"   PRIMARY KEY ("manu_code")                 AS "stores_demo_rdf"."in
ATTACH TABLE  "informix"."msgs"       PRIMARY KEY ("lang", "number", "message") AS "stores_demo_rdf"."in
ATTACH TABLE  "informix"."orders"     PRIMARY KEY ("order_num")                 AS "stores_demo_rdf"."in
ATTACH TABLE  "informix"."state"      PRIMARY KEY ("code", "sname")             AS "stores_demo_rdf"."in
ATTACH TABLE  "informix"."stock"      PRIMARY KEY ("stock_num", "manu_code")    AS "stores_demo_rdf"."in

COMMIT WORK;

GRANT SELECT ON stores_demo_rdf.inf10_stores_demo_rdf.items      TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON stores_demo_rdf.inf10_stores_demo_rdf.catalog    TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON stores_demo_rdf.inf10_stores_demo_rdf.msgs       TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON stores_demo_rdf.inf10_stores_demo_rdf.state      TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON stores_demo_rdf.inf10_stores_demo_rdf.orders     TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON stores_demo_rdf.inf10_stores_demo_rdf.stock      TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON stores_demo_rdf.inf10_stores_demo_rdf.customer   TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON stores_demo_rdf.inf10_stores_demo_rdf.call_type  TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON stores_demo_rdf.inf10_stores_demo_rdf.manufact   TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON stores_demo_rdf.inf10_stores_demo_rdf.cust_calls TO "SPARQL", "SPARQL_UPDATE";
GRANT SPARQL_UPDATE to "SPARQL";

create function DB.DBA.CUST_CALLS_IRI (in customer_num integer, in call_dtime datetime) returns varchar
{
        declare _call_dtime any;
        _call_dtime := cast(call_dtime as varchar);
        return sprintf('http://example.com/informix/stores_demo/cust_calls/%d_%U#this', customer_num, _ca
};

create function DB.DBA.CUST_CALLS_IRI_INV_1 (in cust_calls_iri varchar) returns integer
{
        declare parts any;
        parts := sprintf_inverse(cust_calls_iri, 'http://example.com/informix/stores_demo/cust_calls/%d_%
        if(parts is not null)
        {
                return parts[0];
        }
        return NULL;
};

create function DB.DBA.CUST_CALLS_IRI_INV_2 (in cust_calls_iri varchar) returns datetime
{
        declare parts any;
        parts := sprintf_inverse(cust_calls_iri, 'http://example.com/informix/stores_demo/cust_calls/%d_%
        if(parts is not null)
        {
                return parts[1];
        }
        return NULL;
};

grant execute on DB.DBA.CUST_CALLS_IRI to "SPARQL", "SPARQL_UPDATE";
grant execute on DB.DBA.CUST_CALLS_IRI_INV_1 to "SPARQL", "SPARQL_UPDATE";
grant execute on DB.DBA.CUST_CALLS_IRI_INV_2 to "SPARQL", "SPARQL_UPDATE";

-------- Create rdfs:Class definitions ----------------------------

ttlp (
'
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
```

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

@prefix items:   <http://example.com/schemas/informix/stores_demo/items/> .
@prefix catalog: <http://example.com/schemas/informix/stores_demo/catalog/> .
@prefix stock:   <http://example.com/schemas/informix/stores_demo/stock/> .
@prefix msgs:    <http://example.com/schemas/informix/stores_demo/msgs/> .
@prefix state:   <http://example.com/schemas/informix/stores_demo/state/> .
@prefix orders:  <http://example.com/schemas/informix/stores_demo/orders/> .
@prefix manuf:   <http://example.com/schemas/informix/stores_demo/manufact/> .
@prefix cust:    <http://example.com/schemas/informix/stores_demo/customer/> .
@prefix callt:   <http://example.com/schemas/informix/stores_demo/call_type/> .
@prefix custc:   <http://example.com/schemas/informix/stores_demo/cust_calls/> .

items:Items a rdfs:Class ;
        rdfs:label "Items" ;
        rdfs:comment "Informix SD items table" .

items:item_num a rdf:Property ;
        rdfs:domain items:Items ;
        rdfs:range xsd:integer ;
        rdfs:label "ITEM NUMBER" .

items:quantity a rdf:Property ;
        rdfs:domain items:Items ;
        rdfs:range xsd:integer ;
        rdfs:label "QUANTITY" .

items:total_price a rdf:Property ;
        rdfs:domain items:Items ;
        rdfs:range xsd:decimal ;
        rdfs:label "TOTAL PRICE" .

items:order_num_fk a rdf:Property ;
        rdfs:domain items:Items ;
        rdfs:range orders:Orders ;
        rdfs:label "ORDER NUMBER" .

items:stock_num_fk a rdf:Property ;
        rdfs:domain items:Items ;
        rdfs:range stock:Stock ;
        rdfs:label "STOCK NUMBER" .

items:manu_code_fk a rdf:Property ;
        rdfs:domain items:Items ;
        rdfs:range stock:Stock ;
        rdfs:label "MANUAL CODE" .

catalog:Catalog a rdfs:Class ;
        rdfs:label "Catalog" ;
        rdfs:comment "Informix SD catalog table" .

catalog:manu_code a rdf:Property ;
        rdfs:domain catalog:Catalog ;
        rdfs:range xsd:integer ;
        rdfs:label "MANUAL CODE" .

catalog:cat_descr a rdf:Property ;
        rdfs:domain catalog:Catalog ;
        rdfs:range xsd:string ;
        rdfs:label "CATALOG DESCRIPTION" .

catalog:cat_picture a rdf:Property ;
        rdfs:domain catalog:Catalog ;
        rdfs:range xsd:byte ;
        rdfs:label "CATALOG PICTURE" .

catalog:cat_advert a rdf:Property ;
        rdfs:domain catalog:Catalog ;
        rdfs:range xsd:string ;
        rdfs:label "CATALOG ADVERT" .

catalog:catalog_num_fk a rdf:Property ;
```

```
        rdfs:domain catalog:Catalog ;
        rdfs:range stock:Stock ;
        rdfs:label "CATALOG NUMBER" .

catalog:stock_num_fk a rdf:Property ;
        rdfs:domain catalog:Catalog ;
        rdfs:range stock:Stock ;
        rdfs:label "STOCK NUMBER" .

msgs:Msgs a rdfs:Class ;
        rdfs:label "Msgs" ;
        rdfs:comment "Informix SD msgs table" .

msgs:lang a rdf:Property ;
        rdfs:domain msgs:Msgs ;
        rdfs:range xsd:string ;
        rdfs:label "LANGUAGE" .

msgs:number a rdf:Property ;
        rdfs:domain msgs:Msgs ;
        rdfs:range xsd:integer ;
        rdfs:label "NUMBER" .

msgs:message a rdf:Property ;
        rdfs:domain msgs:Msgs ;
        rdfs:range xsd:string ;
        rdfs:label "MESSAGE" .

state:State a rdfs:Class ;
        rdfs:label "State" ;
        rdfs:comment "Informix SD state table" .

state:code a rdf:Property ;
        rdfs:domain state:State ;
        rdfs:range xsd:string ;
        rdfs:label "STATE CODE" .

state:sname a rdf:Property ;
        rdfs:domain state:State ;
        rdfs:range xsd:string ;
        rdfs:label "STATE NAME" .

orders:Orders a rdfs:Class ;
        rdfs:label "Orders" ;
        rdfs:comment "Informix SD orders table" .

orders:order_num a rdf:Property ;
        rdfs:domain orders:Orders ;
        rdfs:range xsd:integer ;
        rdfs:label "ORDER NUMBER" .

orders:order_date a rdf:Property ;
        rdfs:domain orders:Orders ;
        rdfs:range xsd:date ;
        rdfs:label "ORDER DATE" .

orders:ship_instruct a rdf:Property ;
        rdfs:domain orders:Orders ;
        rdfs:range xsd:string ;
        rdfs:label "SHIPPING INSTRUCTION" .

orders:backlog a rdf:Property ;
        rdfs:domain orders:Orders ;
        rdfs:range xsd:string ;
        rdfs:label "BACKLOG" .

orders:po_num a rdf:Property ;
        rdfs:domain orders:Orders ;
        rdfs:range xsd:string ;
        rdfs:label "PURCHASE ORDER NUMBER" .

orders:ship_date a rdf:Property ;
        rdfs:domain orders:Orders ;
```

```
        rdfs:range xsd:date ;
        rdfs:label "SHIPPING DATE" .

orders:ship_weight a rdf:Property ;
        rdfs:domain orders:Orders ;
        rdfs:range xsd:decimal ;
        rdfs:label "SHIPPING WEIGHT" .

orders:ship_charge a rdf:Property ;
        rdfs:domain orders:Orders ;
        rdfs:range xsd:decimal ;
        rdfs:label "SHIPPING CHARGE" .

orders:paid_date a rdf:Property ;
        rdfs:domain orders:Orders ;
        rdfs:range xsd:date ;
        rdfs:label "PAID DATE" .

orders:customer_num_fk a rdf:Property ;
        rdfs:domain orders:Orders ;
        rdfs:range cust:Customer ;
        rdfs:label "CUSTOMER NUMBER" .

stock:Stock a rdfs:Class ;
        rdfs:label "Stock" ;
        rdfs:comment "Informix SD stock table" .

stock:stock_num a rdf:Property ;
        rdfs:domain stock:Stock ;
        rdfs:range xsd:integer ;
        rdfs:label "STOCK NUMBER" .

stock:description a rdf:Property ;
        rdfs:domain stock:Stock ;
        rdfs:range xsd:string ;
        rdfs:label "DESCRIPTION" .

stock:unit_price a rdf:Property ;
        rdfs:domain stock:Stock ;
        rdfs:range xsd:decimal ;
        rdfs:label "UNIT PRICE" .

stock:unit a rdf:Property ;
        rdfs:domain stock:Stock ;
        rdfs:range xsd:string ;
        rdfs:label "UNIT" .

stock:unit_descr a rdf:Property ;
        rdfs:domain stock:Stock ;
        rdfs:range xsd:decimal ;
        rdfs:label "UNIT DESCRIPTION" .

stock:manu_code_fk a rdf:Property ;
        rdfs:domain stock:Stock ;
        rdfs:range manuf:Manufact ;
        rdfs:label "MANUAL CODE" .

cust:Customer a rdfs:Class ;
        rdfs:label "Customer" ;
        rdfs:comment "Informix SD customer table" .

cust:customer_num a rdf:Property ;
        rdfs:domain cust:Customer ;
        rdfs:range xsd:integer ;
        rdfs:label "CUSTOMER NUMBER" .

cust:fname a rdf:Property ;
        rdfs:domain cust:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "FIRST NAME" .

cust:lname a rdf:Property ;
        rdfs:domain cust:Customer ;
```

```
        rdfs:range xsd:string ;
        rdfs:label "LAST NAME" .

cust:company a rdf:Property ;
        rdfs:domain cust:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "COMPANY" .

cust:address1 a rdf:Property ;
        rdfs:domain cust:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "ADDRESS1" .

cust:address2 a rdf:Property ;
        rdfs:domain cust:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "ADDRESS2" .

cust:city a rdf:Property ;
        rdfs:domain cust:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "CITY" .

cust:state a rdf:Property ;
        rdfs:domain cust:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "STATE" .

cust:zipcode a rdf:Property ;
        rdfs:domain cust:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "ZIP CODE" .

cust:phone a rdf:Property ;
        rdfs:domain cust:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "PHONE NUMBER" .

callt:Call_type a rdfs:Class ;
        rdfs:label "Call_type" ;
        rdfs:comment "Informix SD call_type table" .

callt:call_code a rdf:Property ;
        rdfs:domain callt:Call_type ;
        rdfs:range xsd:string ;
        rdfs:label "CALL CODE" .

callt:code_descr a rdf:Property ;
        rdfs:domain callt:Call_type ;
        rdfs:range xsd:string ;
        rdfs:label "CODE DESCRIPTION" .

manuf:Manufact a rdfs:Class ;
        rdfs:label "Manufact" ;
        rdfs:comment "Informix SD manufact table" .

manuf:manu_code a rdf:Property ;
        rdfs:domain manuf:Manufact ;
        rdfs:range xsd:string ;
        rdfs:label "MANUFACTURE CODE" .

manuf:manu_name a rdf:Property ;
        rdfs:domain manuf:Manufact ;
        rdfs:range xsd:string ;
        rdfs:label "MANUFACTURE NAME" .

manuf:lead_time a rdf:Property ;
        rdfs:domain manuf:Manufact ;
        rdfs:range xsd:integer ;
        rdfs:label "LEAD TIME" .

custc:Cust_calls a rdfs:Class ;
        rdfs:label "Cust_calls" ;
```

```
        rdfs:comment "Informix SD cust_calls table" .

custc:call_dtime a rdf:Property ;
        rdfs:domain manuf:Cust_calls ;
        rdfs:range xsd:datetime ;
        rdfs:label "CALL TIME" .

custc:user_id a rdf:Property ;
        rdfs:domain manuf:Cust_calls ;
        rdfs:range xsd:string ;
        rdfs:label "USER ID" .

custc:call_descr a rdf:Property ;
        rdfs:domain manuf:Cust_calls ;
        rdfs:range xsd:string ;
        rdfs:label "CALL DESCRIPTION" .

custc:res_dtime a rdf:Property ;
        rdfs:domain manuf:Cust_calls ;
        rdfs:range xsd:datetime ;
        rdfs:label "RES TIME" .

custc:res_descr a rdf:Property ;
        rdfs:domain manuf:Cust_calls ;
        rdfs:range xsd:string ;
        rdfs:label "RES DESCRIPTION" .

custc:customer_num_fk a rdf:Property ;
        rdfs:domain manuf:Cust_calls ;
        rdfs:range cust:Customer ;
        rdfs:label "CUSTOMER NUM" .

custc:call_code_fk a rdf:Property ;
        rdfs:domain manuf:Cust_calls ;
        rdfs:range callt:Call_type ;
        rdfs:label "CALL CODE" .

', '', 'http://example.com/schemas/informix/stores_demo', 0);

----------- Create IRI Classes -------------

SPARQL

prefix items:   <http://example.com/schemas/informix/stores_demo/items/>
prefix catalog: <http://example.com/schemas/informix/stores_demo/catalog/>
prefix stock:   <http://example.com/schemas/informix/stores_demo/stock/>
prefix msgs:    <http://example.com/schemas/informix/stores_demo/msgs/>
prefix state:   <http://example.com/schemas/informix/stores_demo/state/>
prefix orders:  <http://example.com/schemas/informix/stores_demo/orders/>
prefix manuf:   <http://example.com/schemas/informix/stores_demo/manufact/>
prefix cust:    <http://example.com/schemas/informix/stores_demo/customer/>
prefix callt:   <http://example.com/schemas/informix/stores_demo/call_type/>
prefix custc:   <http://example.com/schemas/informix/stores_demo/cust_calls/>

create iri class items:items_iri
        "http://example.com/informix/stores_demo/items/%d_%d#this"
        (in item_num integer not null, in order_num integer not null) .

create iri class catalog:catalog_iri
        "http://example.com/informix/stores_demo/catalog/%d#this"
        (in catalog_num integer not null) .

create iri class msgs:msgs_iri
        "http://example.com/informix/stores_demo/msgs/%U_%d_%U#this"
        (in _lang varchar not null, in number integer not null, in message varchar not null) .

create iri class state:state_iri
        "http://example.com/informix/stores_demo/state/%U#this"
        (in code varchar not null) .

create iri class orders:orders_iri
        "http://example.com/informix/stores_demo/orders/%d#this"
        (in order_num integer not null) .
```

```
create iri class stock:stock_iri
        "http://example.com/informix/stores_demo/stock/%d_%U#this"
        (in stock_num integer not null, in manu_code varchar not null) .

create iri class cust:customer_iri
        "http://example.com/informix/stores_demo/customer/%d#this"
        (in customer_num integer not null) .

create iri class callt:call_type_iri
        "http://example.com/informix/stores_demo/call_type/%U#this"
        (in call_code varchar not null) .

create iri class manuf:manufact_iri
        "http://example.com/informix/stores_demo/manufact/%U#this"
        (in manu_code varchar not null) .

create iri class custc:cust_calls_iri using
        function DB.DBA.CUST_CALLS_IRI (in customer_num integer, in call_dtime datetime) returns varchar,
        function DB.DBA.CUST_CALLS_IRI_INV_1 (in cust_calls_iri varchar) returns integer,
        function DB.DBA.CUST_CALLS_IRI_INV_2 (in cust_calls_iri varchar) returns datetime .
;

------------- Create Quad Store -----------------------------------

SPARQL

prefix items:   <http://example.com/schemas/informix/stores_demo/items/>
prefix catalog: <http://example.com/schemas/informix/stores_demo/catalog/>
prefix stock:   <http://example.com/schemas/informix/stores_demo/stock/>
prefix msgs:    <http://example.com/schemas/informix/stores_demo/msgs/>
prefix state:   <http://example.com/schemas/informix/stores_demo/state/>
prefix orders:  <http://example.com/schemas/informix/stores_demo/orders/>
prefix manuf:   <http://example.com/schemas/informix/stores_demo/manufact/>
prefix cust:    <http://example.com/schemas/informix/stores_demo/customer/>
prefix callt:   <http://example.com/schemas/informix/stores_demo/call_type/>
prefix custc:   <http://example.com/schemas/informix/stores_demo/cust_calls/>

alter quad storage virtrdf:DefaultQuadStorage
  from stores_demo_rdf.inf10_stores_demo_rdf.items      as items_tbl
  from stores_demo_rdf.inf10_stores_demo_rdf.catalog    as catalog_tbl
  from stores_demo_rdf.inf10_stores_demo_rdf.msgs       as msgs_tbl
  from stores_demo_rdf.inf10_stores_demo_rdf.state      as state_tbl
  from stores_demo_rdf.inf10_stores_demo_rdf.orders     as orders_tbl
  from stores_demo_rdf.inf10_stores_demo_rdf.stock      as stock_tbl
  from stores_demo_rdf.inf10_stores_demo_rdf.customer   as customer_tbl
  from stores_demo_rdf.inf10_stores_demo_rdf.call_type  as call_type_tbl
  from stores_demo_rdf.inf10_stores_demo_rdf.manufact   as manufact_tbl
  from stores_demo_rdf.inf10_stores_demo_rdf.cust_calls as cust_calls_tbl
{
  create virtrdf:informix_stores_demo as graph <http://example.com/informix/stores_demo>
  {
    items:items_iri (items_tbl.item_num, items_tbl.order_num) a items:Items as virtrdf:items_pk ;
    items:item_num    items_tbl.item_num       as virtrdf:items_item_num ;
    items:order_num   items_tbl.order_num      as virtrdf:items_order_num ;
    items:stock_num   items_tbl.stock_num      as virtrdf:items_stock_num ;
    items:manu_code   items_tbl.manu_code      as virtrdf:items_manu_code ;
    items:quantity    items_tbl.quantity       as virtrdf:items_quantity ;
    items:total_price items_tbl.total_price    as virtrdf:items_total_price ;
    items:from_order  orders:orders_iri (orders_tbl.order_num) where (^{items_tbl.}^.order_num = ^{orders
    items:has_stock   stock:stock_iri (stock_tbl.stock_num, stock_tbl.manu_code) where (^{items_tbl.}^.st

    catalog:catalog_iri (catalog_tbl.catalog_num) a catalog:Catalog as virtrdf:catalog_num;
    catalog:stock_num   catalog_tbl.stock_num    as virtrdf:catalog_stock_num ;
    catalog:manu_code   catalog_tbl.manu_code    as virtrdf:catalog_manu_code ;
    catalog:cat_descr   catalog_tbl.cat_descr    as virtrdf:catalog_cat_descr ;
    catalog:cat_picture catalog_tbl.cat_picture  as virtrdf:catalog_cat_picture ;
    catalog:cat_advert  catalog_tbl.cat_advert   as virtrdf:catalog_cat_advert ;
    catalog:has_stock   stock:stock_iri (stock_tbl.stock_num, stock_tbl.manu_code) where (^{catalog_tbl.}

    msgs:msgs_iri (msgs_tbl.lang, msgs_tbl.number, msgs_tbl.message) a msgs:Msgs as virtrdf:msgs_pk ;
    msgs:lang     msgs_tbl.lang    as virtrdf:msgs_lang ;
    msgs:number   msgs_tbl.number  as virtrdf:msgs_number ;
```

```
    msgs:message  msgs_tbl.message as virtrdf:msgs_message .

    state:state_iri (state_tbl.code) a state:State as virtrdf:code ;
    state:code   state_tbl.code   as virtrdf:state_code ;
    state:sname  state_tbl.sname  as virtrdf:state_sname .

    orders:orders_iri (orders_tbl.order_num) a orders:Orders as virtrdf:order_num ;
    orders:order_num     orders_tbl.order_num     as virtrdf:orders_order_num ;
    orders:order_date    orders_tbl.order_date    as virtrdf:orders_order_date ;
    orders:customer_num  orders_tbl.customer_num  as virtrdf:orders_customer_num ;
    orders:ship_instruct orders_tbl.ship_instruct as virtrdf:orders_ship_instruct ;
    orders:backlog       orders_tbl.backlog       as virtrdf:orders_backlog ;
    orders:po_num        orders_tbl.po_num        as virtrdf:orders_po_num ;
    orders:ship_date     orders_tbl.ship_date     as virtrdf:orders_ship_date ;
    orders:ship_weight   orders_tbl.ship_weight   as virtrdf:orders_ship_weight ;
    orders:ship_charge   orders_tbl.ship_charge   as virtrdf:orders_ship_charge ;
    orders:paid_date     orders_tbl.paid_date     as virtrdf:orders_paid_date ;
    orders:has_customer cust:customer_iri (customer_tbl.customer_num) where (^{orders_tbl.}^.customer_num
    orders:has_item     items:items_iri (items_tbl.item_num, items_tbl.order_num) where (^{orders_tbl.}^.

    stock:stock_iri (stock_tbl.stock_num, stock_tbl.manu_code) a stock:Stock as virtrdf:stock_pk ;
    stock:stock_num    stock_tbl.stock_num    as virtrdf:stock_stock_num ;
    stock:manu_code    stock_tbl.manu_code    as virtrdf:stock_manu_code ;
    stock:description  stock_tbl.description  as virtrdf:stock_description ;
    stock:unit_price   stock_tbl.unit_price   as virtrdf:stock_unit_price ;
    stock:unit         stock_tbl.unit         as virtrdf:stock_unit ;
    stock:unit_descr   stock_tbl.unit_descr   as virtrdf:stock_unit_descr ;
    stock:manufactured_by manuf:manufact_iri (manufact_tbl.manu_code) where (^{stock_tbl.}^.manu_code = ^
    stock:in_catalog  catalog:catalog_iri (catalog_tbl.catalog_num) where (^{stock_tbl.}^.stock_num = ^{c
    stock:in_item     items:items_iri (items_tbl.item_num, items_tbl.order_num) where (^{stock_tbl.}^.sto

    cust:customer_iri (customer_tbl.customer_num) a cust:Customer as virtrdf:customer_num ;
    cust:customer_num  customer_tbl.customer_num  as virtrdf:customer_customer_num ;
    cust:fname         customer_tbl.fname         as virtrdf:customer_fname ;
    cust:lname         customer_tbl.lname         as virtrdf:customer_lname ;
    cust:company       customer_tbl.company       as virtrdf:customer_company ;
    cust:address1      customer_tbl.address1      as virtrdf:customer_address1 ;
    cust:address2      customer_tbl.address2      as virtrdf:customer_address2 ;
    cust:city          customer_tbl.city          as virtrdf:customer_city ;
    cust:state         customer_tbl.state         as virtrdf:customer_state ;
    cust:zipcode       customer_tbl.zipcode       as virtrdf:customer_zipcode ;
    cust:phone         customer_tbl.phone         as virtrdf:customer_phone ;
    cust:placed_order orders:orders_iri (orders_tbl.order_num) where (^{customer_tbl.}^.customer_num = ^{
    cust:made_call    custc:cust_calls_iri (cust_calls_tbl.customer_num, cust_calls_tbl.call_dtime ) wher

    callt:call_type_iri (call_type_tbl.call_code) a callt:Call_type as virtrdf:call_code ;
    callt:call_code   call_type_tbl.call_code as virtrdf:call_type_call_code ;
    callt:code_descr  call_type_tbl.code_descr as virtrdf:call_type_code_descr ;
    callt:call_is_type  custc:cust_calls_iri (cust_calls_tbl.customer_num, cust_calls_tbl.call_dtime) whe

    manuf:manufact_iri (manufact_tbl.manu_code) a manuf:Manufact as virtrdf:manu_code ;
    manuf:manu_code     manufact_tbl.manu_code   as virtrdf:manufact_tbl_manu_code ;
    manuf:manu_name     manufact_tbl.manu_name   as virtrdf:manufact_tbl_manu_name ;
    manuf:lead_time     manufact_tbl.lead_time   as virtrdf:manufact_tbl_lead_time ;
    manuf:manufactures stock:stock_iri (stock_tbl.stock_num, stock_tbl.manu_code) where (^{manufact_tbl.}

    custc:cust_calls_iri  (cust_calls_tbl.customer_num, cust_calls_tbl.call_dtime) a custc:Cust_calls as
    custc:user_id    cust_calls_tbl.user_id      as virtrdf:cust_calls_user_id ;
    custc:call_code  cust_calls_tbl.call_code    as virtrdf:cust_calls_call_code ;
    custc:call_descr cust_calls_tbl.call_descr   as virtrdf:cust_calls_call_descr ;
    custc:res_dtime  cust_calls_tbl.res_dtime    as virtrdf:cust_calls_res_dtime ;
    custc:res_descr  cust_calls_tbl.res_descr    as virtrdf:cust_calls_res_descr ;
    custc:made_by_customer cust:customer_iri   (customer_tbl.customer_num) where (^{cust_calls_tbl.}^.cus
    custc:is_call_type     callt:call_type_iri (call_type_tbl.call_code)   where (^{cust_calls_tbl.}^.cal

  } .
} .
;

delete from db.dba.url_rewrite_rule_list where urrl_list like 'informix_sd_rule%';
delete from db.dba.url_rewrite_rule where urr_rule like 'informix_sd_rule%';

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
```

```
    'informix_sd_rule1',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/about/html/http/^{URIQADefaultHost}^%s',
    vector('path'),
    null,
    '(text/html)|(\\*/\\*)',
    0,
    303
    );

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'informix_sd_rule2',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/sparql?query=DESCRIBE+%3Chttp%3A//localhost%3A8890%U%23this%3E+%3Chttp%3A//localhost%3A8890
    vector('path', 'path', '*accept*'),
    null,
    '(text/rdf.n3)|(application/rdf.xml)',
    0,
    null
    );

DB.DBA.URLREWRITE_CREATE_RULELIST (
    'informix_sd_rule_list1',
    1,
    vector (
                'informix_sd_rule1',
                'informix_sd_rule2'
          ));

-- ensure a VD for the IRIs which begins with /
VHOST_REMOVE (lpath=>'/informix/stores_demo');

VHOST_DEFINE (
        lpath=>'/informix/stores_demo',
        ppath=>'/DAV/informix/stores_demo/',
        is_dav=>1,
        vsp_user=>'dba',
        is_brws=>0,
        opts=>vector ('url_rewrite', 'informix_sd_rule_list1')
        );

delete from db.dba.url_rewrite_rule_list where urrl_list like 'informix_sd_schemas_rule%';
delete from db.dba.url_rewrite_rule where urr_rule like 'informix_sd_schemas_rule%';

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'informix_sd_schemas_rule1',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/about/html/http/^{URIQADefaultHost}^%s',
    vector('path'),
    null,
    '(text/html)|(\\*/\\*)',
    0,
    303
    );

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'informix_sd_schemas_rule2',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/sparql?query=CONSTRUCT+{+%3Chttp%3A//localhost%3A8890%U%3E+%3Fp+%3Fo+}%0D%0AFROM+%3Chttp%
    vector('path','path','*accept*'),
    null,
```

```
    '(text/rdf.n3)|(application/rdf.xml)',
    0,
    null
    );

DB.DBA.URLREWRITE_CREATE_RULELIST (
    'informix_sd_schemas_rule_list1',
    1,
    vector (
                'informix_sd_schemas_rule1',
                'informix_sd_schemas_rule2'
          ));

-- ensure a VD for the IRIs which begins with /
VHOST_REMOVE (lpath=>'/schemas/informix/stores_demo');

VHOST_DEFINE (
        lpath=>'/schemas/informix/stores_demo',
        ppath=>'/DAV/schemas/informix/stores_demo/',
        is_dav=>1,
        vsp_user=>'dba',
        is_brws=>0,
        opts=>vector ('url_rewrite', 'informix_sd_schemas_rule_list1')
        );

DB.DBA.XML_SET_NS_DECL ('items',   'http://^{URIQADefaultHost}^/schemas/informix/stores_demo/items/', 2);
DB.DBA.XML_SET_NS_DECL ('catalog', 'http://^{URIQADefaultHost}^/schemas/informix/stores_demo/catalog/', 2
DB.DBA.XML_SET_NS_DECL ('stock',   'http://^{URIQADefaultHost}^/schemas/informix/stores_demo/stock/', 2);
DB.DBA.XML_SET_NS_DECL ('msgs',    'http://^{URIQADefaultHost}^/schemas/informix/stores_demo/msgs/', 2);
DB.DBA.XML_SET_NS_DECL ('state',   'http://^{URIQADefaultHost}^/schemas/informix/stores_demo/state/', 2);
DB.DBA.XML_SET_NS_DECL ('orders',  'http://^{URIQADefaultHost}^/schemas/informix/stores_demo/orders/', 2)
DB.DBA.XML_SET_NS_DECL ('manuf',   'http://^{URIQADefaultHost}^/schemas/informix/stores_demo/manufact/',
DB.DBA.XML_SET_NS_DECL ('cust',    'http://^{URIQADefaultHost}^/schemas/informix/stores_demo/customer/',
DB.DBA.XML_SET_NS_DECL ('callt',   'http://^{URIQADefaultHost}^/schemas/informix/stores_demo/call_type/',
DB.DBA.XML_SET_NS_DECL ('custc',   'http://^{URIQADefaultHost}^/schemas/informix/stores_demo/cust_calls/'
```

16.8.15. Ingres using demonstration 'Tutorial' database

```
-- Setup script for Linked Data Views of Ingres R3 Tutorial Sample Database --

DB..vd_remote_data_source ('ingiima-tut', '', '<uid>','<pwd>');

ATTACH TABLE  "ingres"."book_list"      PRIMARY KEY ("book_no")              AS "TUT"."ingiima"."book_lis
ATTACH TABLE  "ingres"."book_orders"      PRIMARY KEY ("order_no")             AS "TUT"."ingiima"."book_
ATTACH TABLE  "ingres"."cust_info"      PRIMARY KEY ("cust_no")              AS "TUT"."ingiima"."cust_inf
ATTACH TABLE  "ingres"."cust_orders"      PRIMARY KEY ("order_no")             AS "TUT"."ingiima"."cust_

COMMIT WORK;

GRANT SELECT ON TUT.ingiima.book_list TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON TUT.ingiima.book_orders TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON TUT.ingiima.cust_info TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON TUT.ingiima.cust_orders TO "SPARQL", "SPARQL_UPDATE";

-----------------------------------------------------------------

-------- Create rdfs:Class definitions ----------------------------

ttlp (
'
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

@prefix tut: <http://example.com/schemas/ingrestut/> .

tut:book_list a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/ingrestut> ;
        rdfs:label "book_list" ;
        rdfs:comment "Ingres Tutorial Database book_list table" .

tut:book_no a rdf:Property ;
        rdfs:domain tut:book_list ;
```

```
        rdfs:range xsd:integer ;
        rdfs:label "Book No" .

tut:title a rdf:Property ;
        rdfs:domain tut:book_list ;
        rdfs:range xsd:string ;
        rdfs:label "Title" .

tut:author a rdf:Property ;
        rdfs:domain tut:book_list ;
        rdfs:range xsd:string ;
        rdfs:label "Author" .

tut:price a rdf:Property ;
        rdfs:domain tut:book_list ;
        rdfs:range xsd:money;
        rdfs:label "Price" .

tut:category a rdf:Property ;
        rdfs:domain tut:book_list ;
        rdfs:range xsd:string ;
        rdfs:label "Category" .

tut:stock a rdf:Property ;
        rdfs:domain tut:book_list ;
        rdfs:range xsd:integer ;
        rdfs:label "Stock" .

tut:dist_no a rdf:Property ;
        rdfs:domain tut:book_list ;
        rdfs:range xsd:integer ;
        rdfs:label "Dist No" .

tut:book_orders a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/ingrestut> ;
        rdfs:label "Book Orders" ;
        rdfs:comment "Ingres Tutorial Database book_orders table" .

tut:order_no a rdf:Property ;
        rdfs:domain tut:book_orders ;
        rdfs:range xsd:integer ;
         rdfs:label "Order No" .

tut:book_no_no a rdf:Property ;
        rdfs:domain tut:book_orders ;
        rdfs:range tut:book_list ;
         rdfs:label "Book No" .

tut:sale_price a rdf:Property ;
        rdfs:domain tut:book_orders ;
        rdfs:range xsd:money ;
         rdfs:label "Sale Price" .

tut:quantity a rdf:Property ;
        rdfs:domain tut:book_orders ;
        rdfs:range xsd:integer ;
         rdfs:label "Quantity" .

tut:extension a rdf:Property ;
        rdfs:domain tut:book_orders ;
        rdfs:range xsd:money ;
         rdfs:label "Extension" .

tut:cust_info a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/ingrestut> ;
        rdfs:label "Customer Information" ;
        rdfs:comment "Ingres Tutorial Database cust_info table" .

tut:cust_no a rdf:Property ;
        rdfs:domain tut:cust_info ;
        rdfs:range xsd:integer ;
         rdfs:label "Customer No" .
```

```
tut:name a rdf:Property ;
        rdfs:domain tut:cust_info ;
        rdfs:range xsd:string ;
         rdfs:label "Name" .

tut:company a rdf:Property ;
        rdfs:domain tut:cust_info ;
        rdfs:range xsd:string ;
         rdfs:label "Company" .

tut:street a rdf:Property ;
        rdfs:domain tut:cust_info ;
        rdfs:range xsd:string;
         rdfs:label "Street" .

tut:city a rdf:Property ;
        rdfs:domain tut:cust_info ;
        rdfs:range xsd:string;
         rdfs:label "City" .

tut:state a rdf:Property ;
        rdfs:domain tut:cust_info ;
        rdfs:range xsd:string;
         rdfs:label "State" .

tut:city a rdf:Property ;
        rdfs:domain tut:cust_info ;
        rdfs:range xsd:string;
         rdfs:label "City" .

tut:state a rdf:Property ;
        rdfs:domain tut:cust_info ;
        rdfs:range xsd:string;
         rdfs:label "State" .

tut:zip a rdf:Property ;
        rdfs:domain tut:cust_info ;
        rdfs:range xsd:string;
         rdfs:label "Zip Code" .

tut:card_no a rdf:Property ;
        rdfs:domain tut:cust_info ;
        rdfs:range xsd:string;
         rdfs:label "Card No" .

tut:exp_date a rdf:Property ;
        rdfs:domain tut:cust_info ;
        rdfs:range xsd:date;
         rdfs:label "Expire Date" .

tut:ship_to a rdf:Property ;
        rdfs:domain tut:cust_info ;
        rdfs:range xsd:string;
         rdfs:label "Ship To" .

tut:cust_orders a rdfs:Class ;
        rdfs:isDefinedBy <http://example.com/schemas/ingrestut> ;
        rdfs:label "Customer Orders" ;
        rdfs:comment "Ingres Tutorial Database cust_orders table" .

tut:order_no a rdf:Property ;
        rdfs:domain tut:cust_orders ;
        rdfs:range tut:book_orders ;
         rdfs:label "Order No" .

tut:book_no a rdf:Property ;
        rdfs:domain tut:cust_orders ;
        rdfs:range tut:cust_info ;
         rdfs:label "Book No" .

tut:order_date a rdf:Property ;
        rdfs:domain tut:cust_orders ;
        rdfs:range xsd:date ;
```

```
         rdfs:label "Order Date" .

tut:status a rdf:Property ;
        rdfs:domain tut:cust_orders ;
        rdfs:range xsd:string ;
         rdfs:label "Status" .

tut:order_total a rdf:Property ;
        rdfs:domain tut:cust_orders ;
        rdfs:range xsd:money ;
         rdfs:label "Order Total" .
', '', 'http://example.com/schemas/ingrestut', 0);

-----------------------------------------------------------------

----------- Create IRI Classes -------------

SPARQL

        create iri class <http://example.com/schemas/ingrestut/book_list_iri>
        "http://^{URIQADefaultHost}^/ingrestut/book_list/%d#this"
        (in book_no integer not null) .

        create iri class <http://example.com/schemas/ingrestut/book_orders_iri>
        "http://^{URIQADefaultHost}^/ingrestut/book_orders/%d_%d#this"
         (in order_no integer not null, in book_no integer not null ) .

        create iri class <http://example.com/schemas/ingrestut/cust_info_iri>
        "http://^{URIQADefaultHost}^/ingrestut/cust_info/%d#this"
        (in cust_no integer not null) .

        create iri class <http://example.com/schemas/ingrestut/cust_orders_iri>
        "http://^{URIQADefaultHost}^/ingrestut/cust_orders/%d#this"
        (in order_no integer not null) .

        ;

-------------------------------------------------------------------

------------- Create Quad Store ----------------------------------

SPARQL

prefix tut:     <http://example.com/schemas/ingrestut/>

alter quad storage virtrdf:DefaultQuadStorage
  from TUT.ingiima.book_list as book_list_tbl
  from TUT.ingiima.book_orders as book_orders_tbl
  from TUT.ingiima.cust_info as cust_info_tbl
  from TUT.ingiima.cust_orders as cust_orders_tbl
{
  create virtrdf:ingrestut as
      graph <http://example.com/ingrestut>
  {
        tut:book_list_iri(book_list_tbl.book_no) a tut:book_list
               as virtrdf:book_list_book_no ;
        tut:title book_list_tbl.title
             as virtrdf:book_list_title;
        tut:author book_list_tbl.author
              as virtrdf:book_list_author;
        tut:price book_list_tbl.price
              as virtrdf:book_list_price;
        tut:category book_list_tbl.category
              as virtrdf:book_list_category;
        tut:stock book_list_tbl.stock
              as virtrdf:book_list_stock;
        tut:dist_no book_list_tbl.dist_no
              as virtrdf:book_list_dist_no .

        tut:book_orders_iri(book_orders_tbl.order_no, book_orders_tbl.book_no) a tut:book_orders
                as virtrdf:book_orders_pk;
        tut:order_no book_orders_tbl.order_no
                as virtrdf:book_orders_order_no;
```

```
        tut:book_no tut:book_list_iri(book_list_tbl.book_no)
                where(^{book_orders_tbl.}^.book_no = ^{book_list_tbl.}^.book_no)
                as virtrdf:book_orders_book_no;
        tut:sale_price book_orders_tbl.sale_price
                as virtrdf:book_orders_sale_price;
        tut:quantity book_orders_tbl.quantity
                as virtrdf:book_orders_quantity;
        tut:extension book_orders_tbl.extension
                as virtrdf:book_orders_extension .

        tut:cust_info_iri(cust_info_tbl.cust_no) a tut:cust_info
                as virtrdf:cust_info_cust_no;
        tut:name cust_info_tbl.name
                as virtrdf:cust_info_name;
        tut:company cust_info_tbl.company
                as virtrdf:cust_info_company;
        tut:street cust_info_tbl.street
                as virtrdf:cust_info_street;
        tut:city cust_info_tbl.city
                as virtrdf:cust_info_city;
        tut:state cust_info_tbl.state
                as virtrdf:cust_info_state;
        tut:zip cust_info_tbl.zip
                as virtrdf:cust_info_zip;
        tut:card_no cust_info_tbl.card_no
                as virtrdf:cust_info_card_no;
        tut:exp_date cust_info_tbl.exp_date
                as virtrdf:cust_info_exp_date;
        tut:ship_to cust_info_tbl.ship_to
                as virtrdf:cust_info_ship_to .

        tut:cust_orders_iri(cust_orders_tbl.order_no) a tut:cust_orders
                as virtrdf:cust_orders_order_no;

        tut:cust_no tut:cust_info_iri(cust_info_tbl.cust_no)
                where (^{cust_orders_tbl.}^.cust_no = ^{cust_info_tbl.}^.cust_no)
                as virtrdf:cust_orders_cust_no;
        tut:order_date cust_orders_tbl.order_date
                as virtrdf:cust_orders_order_date;
        tut:status cust_orders_tbl.status
                as virtrdf:cust_orders_status;
        tut:order_total cust_orders_tbl.order_total
                as virtrdf:cust_orders_order_total   .
  } .
} .
;

delete from db.dba.url_rewrite_rule_list where urrl_list like 'ingrestut_rule%';
delete from db.dba.url_rewrite_rule where urr_rule like 'ingrestut_rule%';

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'ingrestut_rule1',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/about/html/http/^{URIQADefaultHost}^%s',
    vector('path'),
    null,
    '(text/html)|(\\*/\\*)',
    0,
    303
    );

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'ingrestut_rule2',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/sparql?query=DESCRIBE+%%3Chttp%%3A//localhost%%3A8890%U%%23this%%3E+%%3Chttp%%3A//localhost%%3A8890
    vector('path', 'path', '*accept*'),
    null,
```

```
    '(text/rdf.n3)|(application/rdf.xml)',
    0,
    null
    );

DB.DBA.URLREWRITE_CREATE_RULELIST (
    'ingrestut_rule_list1',
    1,
    vector (
                'ingrestut_rule1',
                'ingrestut_rule2'
          ));

-- ensure a VD for the IRIs which begins with /
VHOST_REMOVE (lpath=>'/ingrestut');

VHOST_DEFINE (
        lpath=>'/ingrestut',
        ppath=>'/DAV/ingrestut/',
        is_dav=>1,
        vsp_user=>'dba',
        is_brws=>0,
        opts=>vector ('url_rewrite', 'ingrestut_rule_list1')
        );

delete from db.dba.url_rewrite_rule_list where urrl_list like 'ingres_schemas_rule%';
delete from db.dba.url_rewrite_rule where urr_rule like 'ingres_schemas_rule%';

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'ingres_schemas_rule1',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/about/html/http/^{URIQADefaultHost}^%s',
    vector('path'),
    null,
    '(text/html)|(\\*/\\*)',
    0,
    303
    );

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'ingres_schemas_rule2',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/sparql?query=CONSTRUCT+{+%%3Chttp%%3A//localhost%%3A8890%U%%3E+%%3Fp+%%3Fo+}%%0D%%0AFROM+%%3Chttp%%
    vector('path','path','*accept*'),
    null,
    '(text/rdf.n3)|(application/rdf.xml)',
    0,
    null
    );

DB.DBA.URLREWRITE_CREATE_RULELIST (
    'ingres_schemas_rule_list1',
    1,
    vector (
                'ingres_schemas_rule1',
                'ingres_schemas_rule2'
          ));

-- ensure a VD for the IRIs which begins with /
VHOST_REMOVE (lpath=>'/schema/ingrestut');

VHOST_DEFINE (
        lpath=>'/schemas/ingrestut',
        ppath=>'/DAV/schemas/ingrestut/',
        is_dav=>1,
        vsp_user=>'dba',
        is_brws=>0,
```

```
        opts=>vector ('url_rewrite', 'ingres_schemas_rule_list1')
        );

DB.DBA.XML_SET_NS_DECL ('tut', 'http://^{URIQADefaultHost}^/schemas/ingrestut/', 2);
```

16.8.16. Progress (SQL-89) using demonstration 'iSports' database

```
ATTACH TABLE  "ISPORTS_RDF"."Customer" PRIMARY KEY ("Cust-Num")
AS "isports_rdf"."pro91_isports_rdf"."Customer"
FROM 'pro91_isports_rdf';

ATTACH TABLE  "ISPORTS_RDF"."Invoice" PRIMARY KEY ("Invoice-Num")
AS "isports_rdf"."pro91_isports_rdf"."Invoice"
FROM 'pro91_isports_rdf';

ATTACH TABLE  "ISPORTS_RDF"."Item" PRIMARY KEY ("Item-num")
AS "isports_rdf"."pro91_isports_rdf"."Item"
FROM 'pro91_isports_rdf';

ATTACH TABLE  "ISPORTS_RDF"."Local-Default" PRIMARY KEY ("Country")
AS "isports_rdf"."pro91_isports_rdf"."Local-Default"
FROM 'pro91_isports_rdf';

ATTACH TABLE  "ISPORTS_RDF"."Order" PRIMARY KEY ("Order-num")
AS "isports_rdf"."pro91_isports_rdf"."Order"
FROM 'pro91_isports_rdf';

ATTACH TABLE  "ISPORTS_RDF"."Order-Line"  PRIMARY KEY ("Order-num", "Line-num")
AS "isports_rdf"."pro91_isports_rdf"."Order-Line"
FROM 'pro91_isports_rdf';

ATTACH TABLE  "ISPORTS_RDF"."Ref-Call" PRIMARY KEY ("Call-Num")
AS "isports_rdf"."pro91_isports_rdf"."Ref-Call"
FROM 'pro91_isports_rdf';

ATTACH TABLE  "ISPORTS_RDF"."Salesrep" PRIMARY KEY ("Sales-Rep")
AS "isports_rdf"."pro91_isports_rdf"."Salesrep"
FROM 'pro91_isports_rdf';

ATTACH TABLE  "ISPORTS_RDF"."State" PRIMARY KEY ("State")
AS "isports_rdf"."pro91_isports_rdf"."State"
FROM 'pro91_isports_rdf';

COMMIT WORK;

GRANT SELECT ON isports_rdf.pro91_isports_rdf.Customer        TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.pro91_isports_rdf."Order"         TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.pro91_isports_rdf.Item            TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.pro91_isports_rdf."Order-Line"    TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.pro91_isports_rdf.Invoice         TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.pro91_isports_rdf."Local-Default" TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.pro91_isports_rdf."Ref-Call"      TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.pro91_isports_rdf.Salesrep        TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.pro91_isports_rdf.State           TO "SPARQL", "SPARQL_UPDATE";

GRANT SPARQL_UPDATE to "SPARQL";

CREATE VIEW isports_rdf.pro91_isports_rdf.VCustomer      AS SELECT "Cust-Num" AS Cust_Num, Name, Address,
CREATE VIEW isports_rdf.pro91_isports_rdf.VOrder         AS SELECT "Order-num" AS Order_num, "Cust-Num" A
CREATE VIEW isports_rdf.pro91_isports_rdf.VItem          AS SELECT "Item-num" AS Item_num, "Item-Name" AS
CREATE VIEW isports_rdf.pro91_isports_rdf.VOrder_Line    AS SELECT "Order-num" AS Order_num, "Line-num" A
CREATE VIEW isports_rdf.pro91_isports_rdf.VInvoice       AS SELECT "Invoice-Num" AS Invoice_Num, "Cust-Nu
CREATE VIEW isports_rdf.pro91_isports_rdf.VLocal_Default AS SELECT Country, "Region1-Label" AS Region1_La
CREATE VIEW isports_rdf.pro91_isports_rdf.VRef_Call      AS SELECT "Call-Num" AS Call_Num, "Cust-Num" AS
CREATE VIEW isports_rdf.pro91_isports_rdf.VSalesrep      AS SELECT "Rep-Name" AS Rep_Name, Region, "Sales
CREATE VIEW isports_rdf.pro91_isports_rdf.VState         AS SELECT State, "State-Name" AS State_Name, Reg

GRANT SELECT ON isports_rdf.pro91_isports_rdf.VCustomer      TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.pro91_isports_rdf.VOrder         TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.pro91_isports_rdf.VItem          TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.pro91_isports_rdf.VOrder_Line    TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.pro91_isports_rdf.VInvoice       TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.pro91_isports_rdf.VLocal_Default TO "SPARQL", "SPARQL_UPDATE";
```

```
GRANT SELECT ON isports_rdf.pro91_isports_rdf.VRef_Call      TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.pro91_isports_rdf.VSalesrep      TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.pro91_isports_rdf.VState         TO "SPARQL", "SPARQL_UPDATE";

-------- Create rdfs:Class definitions ----------------------------

ttlp (
'
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

@prefix customer:     <http://example.com/schemas/progress/isports/customer/> .
@prefix order:        <http://example.com/schemas/progress/isports/order/> .
@prefix item:         <http://example.com/schemas/progress/isports/item/> .
@prefix orderline:    <http://example.com/schemas/progress/isports/order_line/> .
@prefix invoice:      <http://example.com/schemas/progress/isports/invoice/> .
@prefix localdefault: <http://example.com/schemas/progress/isports/local_default/> .
@prefix refcall:      <http://example.com/schemas/progress/isports/ref_call/> .
@prefix salesrep:     <http://example.com/schemas/progress/isports/salesrep/> .
@prefix state:        <http://example.com/schemas/progress/isports/state/> .

customer:Customer a rdfs:Class ;
        rdfs:label "Customer" ;
        rdfs:comment "Progress isports Customer table" .

customer:Cust-Num a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:integer ;
        rdfs:label "Cust-Num" .

customer:Name a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "Name" .

customer:Address a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "Address" .

customer:Address2 a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "Address2" .

customer:City a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "City" .

customer:State a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "State" .

customer:Country a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "Country" .

customer:Phone a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "Phone" .

customer:Contact a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "Contact" .

customer:Sales-Rep a rdf:Property ;
        rdfs:domain customer:Customer ;
```

```
        rdfs:range xsd:string ;
        rdfs:label "Sales-Rep" .

customer:Comments a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "Comments" .

customer:Credit-Limit a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:decimal ;
        rdfs:label "Credit-Limit" .

customer:Balance a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:decimal ;
        rdfs:label "Balance" .

customer:Terms a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "Terms" .

customer:Discount a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:integer ;
        rdfs:label "Discount" .

customer:Postal-Code a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "Postal-Code" .

order:Order a rdfs:Class ;
        rdfs:label "Order" ;
        rdfs:comment "Progress isports Order table" .

order:Order-num a rdf:Property ;
        rdfs:domain order:Order ;
        rdfs:range xsd:integer ;
        rdfs:label "Order-num" .

order:Cust-Num a rdf:Property ;
        rdfs:domain order:Order ;
        rdfs:range xsd:integer ;
        rdfs:label "Cust-Num" .

order:Order-Date a rdf:Property ;
        rdfs:domain order:Order ;
        rdfs:range xsd:date ;
        rdfs:label "Order-Date" .

order:Ship-Date a rdf:Property ;
        rdfs:domain order:Order ;
        rdfs:range xsd:date ;
        rdfs:label "Ship-Date" .

order:Promise-Date a rdf:Property ;
        rdfs:domain order:Order ;
        rdfs:range xsd:date ;
        rdfs:label "Promise-Date" .

order:Carrier a rdf:Property ;
        rdfs:domain order:Order ;
        rdfs:range xsd:string ;
        rdfs:label "Carrier" .

order:Instructions a rdf:Property ;
        rdfs:domain order:Order ;
        rdfs:range xsd:string ;
        rdfs:label "Instructions" .

order:PO a rdf:Property ;
```

```
        rdfs:domain order:Order ;
        rdfs:range xsd:string ;
        rdfs:label "PO" .

order:Terms a rdf:Property ;
        rdfs:domain order:Order ;
        rdfs:range xsd:string ;
        rdfs:label "Terms" .

order:Sales-Rep a rdf:Property ;
        rdfs:domain order:Order ;
        rdfs:range xsd:string ;
        rdfs:label "Sales-Rep" .

item:Item a rdfs:Class ;
        rdfs:label "Item" ;
        rdfs:comment "Progress isports Item table" .

item:Item-num a rdf:Property ;
        rdfs:domain item:Item ;
        rdfs:range xsd:integer ;
        rdfs:label "Item-num" .

item:Item-Name a rdf:Property ;
        rdfs:domain item:Item ;
        rdfs:range xsd:string ;
        rdfs:label "Item-Name" .

item:Cat-Page a rdf:Property ;
        rdfs:domain item:Item ;
        rdfs:range xsd:integer ;
        rdfs:label "Cat-Page" .

item:Price a rdf:Property ;
        rdfs:domain item:Item ;
        rdfs:range xsd:decimal ;
        rdfs:label "Price" .

item:Cat-Description a rdf:Property ;
        rdfs:domain item:Item ;
        rdfs:range xsd:string ;
        rdfs:label "Cat-Description" .

item:On-hand a rdf:Property ;
        rdfs:domain item:Item ;
        rdfs:range xsd:integer ;
        rdfs:label "On-hand" .

item:Allocated a rdf:Property ;
        rdfs:domain item:Item ;
        rdfs:range xsd:integer ;
        rdfs:label "Allocated" .

item:Re-Order a rdf:Property ;
        rdfs:domain item:Item ;
        rdfs:range xsd:integer ;
        rdfs:label "Re-Order" .

item:On-Order a rdf:Property ;
        rdfs:domain item:Item ;
        rdfs:range xsd:integer ;
        rdfs:label "On-Order" .

orderline:Order-Line a rdfs:Class ;
        rdfs:label "Order-Line" ;
        rdfs:comment "Progress isports Order-Line table" .

orderline:Order-num a rdf:Property ;
        rdfs:domain orderline:Order-Line ;
        rdfs:range xsd:integer ;
        rdfs:label "Order-num" .

orderline:Line-num a rdf:Property ;
```

```
        rdfs:domain orderline:Order-Line ;
        rdfs:range xsd:integer ;
        rdfs:label "Line-num" .

orderline:Item-num a rdf:Property ;
        rdfs:domain orderline:Order-Line ;
        rdfs:range xsd:integer ;
        rdfs:label "Item-num" .

orderline:Price a rdf:Property ;
        rdfs:domain orderline:Order-Line ;
        rdfs:range xsd:decimal ;
        rdfs:label "Price" .

orderline:Qty a rdf:Property ;
        rdfs:domain orderline:Order-Line ;
        rdfs:range xsd:integer ;
        rdfs:label "Qty" .

orderline:Discount a rdf:Property ;
        rdfs:domain orderline:Order-Line ;
        rdfs:range xsd:integer ;
        rdfs:label "Discount" .

orderline:Extended-Price a rdf:Property ;
        rdfs:domain orderline:Order-Line ;
        rdfs:range xsd:decimal ;
        rdfs:label "Extended-Price" .

orderline:Backorder a rdf:Property ;
        rdfs:domain orderline:Order-Line ;
        rdfs:range xsd:byte ;
        rdfs:label "Backorder" .

invoice:Invoice a rdfs:Class ;
        rdfs:label "Invoice" ;
        rdfs:comment "Progress isports Invoice table" .

invoice:Invoice-Num a rdf:Property ;
        rdfs:domain invoice:Invoice ;
        rdfs:range xsd:integer ;
        rdfs:label "Invoice-Num" .

invoice:Cust-Num a rdf:Property ;
        rdfs:domain invoice:Invoice ;
        rdfs:range xsd:integer ;
        rdfs:label "Cust-Num" .

invoice:Invoice-Date a rdf:Property ;
        rdfs:domain invoice:Invoice ;
        rdfs:range xsd:date ;
        rdfs:label "Invoice-Date" .

invoice:Amount a rdf:Property ;
        rdfs:domain invoice:Invoice ;
        rdfs:range xsd:decimal ;
        rdfs:label "Amount" .

invoice:Total-Paid a rdf:Property ;
        rdfs:domain invoice:Invoice ;
        rdfs:range xsd:decimal ;
        rdfs:label "Total-Paid" .

invoice:Adjustment a rdf:Property ;
        rdfs:domain invoice:Invoice ;
        rdfs:range xsd:decimal ;
        rdfs:label "Adjustment" .

invoice:Order-Num a rdf:Property ;
        rdfs:domain invoice:Invoice ;
        rdfs:range xsd:integer ;
        rdfs:label "Order-Num" .
```

```
invoice:Ship-Charge a rdf:Property ;
        rdfs:domain invoice:Invoice ;
        rdfs:range xsd:decimal ;
        rdfs:label "Ship-Charge" .

localdefault:Local-Default a rdfs:Class ;
        rdfs:label "Local-Default" ;
        rdfs:comment "Progress isports Local-Default table" .

localdefault:Country a rdf:Property ;
        rdfs:domain localdefault:Local-Default ;
        rdfs:range xsd:string ;
        rdfs:label "Country" .

localdefault:Region1-Label a rdf:Property ;
        rdfs:domain localdefault:Local-Default ;
        rdfs:range xsd:string ;
        rdfs:label "Region1-Label" .

localdefault:Region2-Label a rdf:Property ;
        rdfs:domain localdefault:Local-Default ;
        rdfs:range xsd:string ;
        rdfs:label "Region2-Label" .

localdefault:Postal-Label a rdf:Property ;
        rdfs:domain localdefault:Local-Default ;
        rdfs:range xsd:string ;
        rdfs:label "Postal-Label" .

localdefault:Postal-Format a rdf:Property ;
        rdfs:domain localdefault:Local-Default ;
        rdfs:range xsd:string ;
        rdfs:label "Postal-Format" .

localdefault:Tel-Format a rdf:Property ;
        rdfs:domain localdefault:Local-Default ;
        rdfs:range xsd:string ;
        rdfs:label "Tel-Format" .

localdefault:Date-Format a rdf:Property ;
        rdfs:domain localdefault:Local-Default ;
        rdfs:range xsd:string ;
        rdfs:label "Date-Format" .

localdefault:Currency-Symbol a rdf:Property ;
        rdfs:domain localdefault:Local-Default ;
        rdfs:range xsd:string ;
        rdfs:label "Currency-Symbol" .

refcall:Ref-Call a rdfs:Class ;
        rdfs:label "Ref-Call" ;
        rdfs:comment "Progress isports Ref-Call table" .

refcall:Call-Num a rdf:Property ;
        rdfs:domain refcall:Ref-Call ;
        rdfs:range xsd:string ;
        rdfs:label "Call-Num" .

refcall:Cust-Num a rdf:Property ;
        rdfs:domain refcall:Ref-Call ;
        rdfs:range xsd:integer ;
        rdfs:label "Cust-Num" .

refcall:Call-Date a rdf:Property ;
        rdfs:domain refcall:Ref-Call ;
        rdfs:range xsd:date ;
        rdfs:label "Call-Date" .

refcall:Sales-Rep a rdf:Property ;
        rdfs:domain refcall:Ref-Call ;
        rdfs:range xsd:string ;
        rdfs:label "Sales-Rep" .
```

```
refcall:Parent a rdf:Property ;
        rdfs:domain refcall:Ref-Call ;
        rdfs:range xsd:string ;
        rdfs:label "Parent" .

refcall:Txt a rdf:Property ;
        rdfs:domain refcall:Ref-Call ;
        rdfs:range xsd:string ;
        rdfs:label "Txt" .

salesrep:Salesrep a rdfs:Class ;
        rdfs:label "Salesrep" ;
        rdfs:comment "Progress isports Salesrep table" .

salesrep:Sales-Rep a rdf:Property ;
        rdfs:domain salesrep:Salesrep ;
        rdfs:range xsd:string ;
        rdfs:label "Sales-Rep" .

salesrep:Rep-Name a rdf:Property ;
        rdfs:domain salesrep:Salesrep ;
        rdfs:range xsd:string ;
        rdfs:label "Rep-Name" .

salesrep:Region a rdf:Property ;
        rdfs:domain salesrep:Salesrep ;
        rdfs:range xsd:string ;
        rdfs:label "Region" .

salesrep:Month-Quota-1 a rdf:Property ;
        rdfs:domain salesrep:Salesrep ;
        rdfs:range xsd:string ;
        rdfs:label "Month-Quota@1" .

salesrep:Month-Quota-2 a rdf:Property ;
        rdfs:domain salesrep:Salesrep ;
        rdfs:range xsd:string ;
        rdfs:label "Month-Quota@2" .

salesrep:Month-Quota-3 a rdf:Property ;
        rdfs:domain salesrep:Salesrep ;
        rdfs:range xsd:string ;
        rdfs:label "Month-Quota@3" .

salesrep:Month-Quota-4 a rdf:Property ;
        rdfs:domain salesrep:Salesrep ;
        rdfs:range xsd:string ;
        rdfs:label "Month-Quota@4" .

salesrep:Month-Quota-5 a rdf:Property ;
        rdfs:domain salesrep:Salesrep ;
        rdfs:range xsd:string ;
        rdfs:label "Month-Quota@5" .

salesrep:Month-Quota-6 a rdf:Property ;
        rdfs:domain salesrep:Salesrep ;
        rdfs:range xsd:string ;
        rdfs:label "Month-Quota@6" .

salesrep:Month-Quota-7 a rdf:Property ;
        rdfs:domain salesrep:Salesrep ;
        rdfs:range xsd:string ;
        rdfs:label "Month-Quota@7" .

salesrep:Month-Quota-8 a rdf:Property ;
        rdfs:domain salesrep:Salesrep ;
        rdfs:range xsd:string ;
        rdfs:label "Month-Quota@8" .

salesrep:Month-Quota-9 a rdf:Property ;
        rdfs:domain salesrep:Salesrep ;
        rdfs:range xsd:string ;
        rdfs:label "Month-Quota@9" .
```

```
salesrep:Month-Quota-10 a rdf:Property ;
        rdfs:domain salesrep:Salesrep ;
        rdfs:range xsd:string ;
        rdfs:label "Month-Quota@10" .

salesrep:Month-Quota-11 a rdf:Property ;
        rdfs:domain salesrep:Salesrep ;
        rdfs:range xsd:string ;
        rdfs:label "Month-Quota@11" .

salesrep:Month-Quota-12 a rdf:Property ;
        rdfs:domain salesrep:Salesrep ;
        rdfs:range xsd:string ;
        rdfs:label "Month-Quota@12" .

state:State a rdfs:Class ;
        rdfs:label "State" ;
        rdfs:comment "Progress isports State table" .

state:State_ a rdf:Property ;
        rdfs:domain state:State ;
        rdfs:range xsd:string ;
        rdfs:label "State" .

state:State-Name a rdf:Property ;
        rdfs:domain state:State ;
        rdfs:range xsd:string ;
        rdfs:label "State-Name" .

state:Region a rdf:Property ;
        rdfs:domain state:State ;
        rdfs:range xsd:string ;
        rdfs:label "Region" .

', '', 'http://example.com/schemas/progress/isports', 0);

----------- Create IRI Classes -------------

SPARQL

prefix customer:     <http://example.com/schemas/progress/isports/customer/>
prefix order:        <http://example.com/schemas/progress/isports/order/>
prefix item:         <http://example.com/schemas/progress/isports/item/>
prefix orderline:    <http://example.com/schemas/progress/isports/order_line/>
prefix invoice:      <http://example.com/schemas/progress/isports/invoice/>
prefix localdefault: <http://example.com/schemas/progress/isports/local_default/>
prefix refcall:      <http://example.com/schemas/progress/isports/ref_call/>
prefix salesrep:     <http://example.com/schemas/progress/isports/salesrep/>
prefix state:        <http://example.com/schemas/progress/isports/state/>

create iri class customer:customer_iri
        "http://example.com/progress/isports/customer/%d#this"
        (in Cust_Num integer not null) .

create iri class order:order_iri
        "http://example.com/progress/isports/order/%d#this"
        (in Order_Num integer not null) .

create iri class item:item_iri
        "http://example.com/progress/isports/item/%d#this"
        (in Item_num integer not null) .

create iri class orderline:order-line_iri
        "http://example.com/progress/isports/order-line/%d_%d#this"
        (in Order_num integer not null, in Line_num integer not null) .

create iri class invoice:invoice_iri
        "http://example.com/progress/isports/invoice/%d#this"
        (in Invoice_Num integer not null) .

create iri class localdefault:local-default_iri
        "http://example.com/progress/isports/local-default/%U#this"
```

```
        (in Country varchar not null) .

create iri class refcall:ref-call_iri
        "http://example.com/progress/isports/ref-call/%U#this"
        (in Call_Num varchar not null) .

create iri class salesrep:salesrep_iri
        "http://example.com/progress/isports/salesrep/%U#this"
        (in Sales_Rep varchar not null) .

create iri class state:state_iri
        "http://example.com/progress/isports/state/%U#this"
        (in State varchar not null) .

;

------------- Create Quad Store -----------------------------------

SPARQL

prefix customer:     <http://example.com/schemas/progress/isports/customer/>
prefix order:        <http://example.com/schemas/progress/isports/order/>
prefix item:         <http://example.com/schemas/progress/isports/item/>
prefix orderline:    <http://example.com/schemas/progress/isports/order_line/>
prefix invoice:      <http://example.com/schemas/progress/isports/invoice/>
prefix localdefault: <http://example.com/schemas/progress/isports/local_default/>
prefix refcall:      <http://example.com/schemas/progress/isports/ref_call/>
prefix salesrep:     <http://example.com/schemas/progress/isports/salesrep/>
prefix state:        <http://example.com/schemas/progress/isports/state/>

alter quad storage virtrdf:DefaultQuadStorage
  from isports_rdf.pro91_isports_rdf.VCustomer      as Customer_tbl
  from isports_rdf.pro91_isports_rdf.VOrder         as Order_tbl
  from isports_rdf.pro91_isports_rdf.VItem          as Item_tbl
  from isports_rdf.pro91_isports_rdf.VOrder_Line    as Order_Line_tbl
  from isports_rdf.pro91_isports_rdf.VInvoice       as Invoice_tbl
  from isports_rdf.pro91_isports_rdf.VRef_Call      as Ref_Call_tbl
  from isports_rdf.pro91_isports_rdf.VRef_Call      as Ref_Call_tbl_1
  from isports_rdf.pro91_isports_rdf.VLocal_Default as Local_Default_tbl
  from isports_rdf.pro91_isports_rdf.VSalesrep      as Salesrep_tbl
  from isports_rdf.pro91_isports_rdf.VState         as State_tbl
{
  create virtrdf:progress_isports as graph <http://example.com/progress/isports>
  {
    customer:customer_iri (Customer_tbl.Cust_Num) a customer:Customer as virtrdf:customer_pk ;
    customer:Cust-Num     Customer_tbl.Cust_Num     as virtrdf:Customer_cust-num ;
    customer:Name         Customer_tbl.Name         as virtrdf:Customer_name ;
    customer:Address      Customer_tbl.Address      as virtrdf:Customer_address ;
    customer:Address2     Customer_tbl.Address2     as virtrdf:Customer_address2 ;
    customer:City         Customer_tbl.City         as virtrdf:Customer_city ;
    customer:State        Customer_tbl.State        as virtrdf:Customer_state ;
    customer:Country      Customer_tbl.Country      as virtrdf:Customer_country ;
    customer:Phone        Customer_tbl.Phone        as virtrdf:Customer_phone ;
    customer:Contact      Customer_tbl.Contact      as virtrdf:Customer_contact ;
    customer:Sales-Rep    Customer_tbl.Sales_Rep    as virtrdf:Customer_sales_rep ;
    customer:Comments     Customer_tbl.Comments     as virtrdf:Customer_comments ;
    customer:Credit-Limit Customer_tbl.Credit_Limit as virtrdf:Customer_credit-limit ;
    customer:Balance      Customer_tbl.Balance      as virtrdf:Customer_balance ;
    customer:Terms        Customer_tbl.Terms        as virtrdf:Customer_terms ;
    customer:Discount     Customer_tbl.Discount     as virtrdf:Customer_discount ;
    customer:Postal-Code  Customer_tbl.Postal_Code  as virtrdf:Customer_postal-code ;
    customer:from_state        state:state_iri (State_tbl.State)                          where ( ^{Cust
    customer:has_sales_rep     salesrep:salesrep_iri (Salesrep_tbl.Sales_Rep)             where ( ^{Cust
    customer:has_local_default localdefault:local-default_iri (Local_Default_tbl.Country) where ( ^{Cust
    customer:placed_order      order:order_iri (Order_tbl.Order_num)                      where ( ^{Cust
    customer:has_invoice       invoice:invoice_iri (Invoice_tbl.Invoice_Num)              where ( ^{Cust
    customer:ref_call          refcall:ref-call_iri (Ref_Call_tbl.Call_Num)               where ( ^{Cust

    order:order_iri (Order_tbl.Order_num) a order:Order as virtrdf:order_pk ;
    order:Order-num    Order_tbl.Order_num    as virtrdf:Order_order-num ;
    order:Cust-Num     Order_tbl.Cust_Num     as virtrdf:Order_cust_num ;
    order:Order-Date   Order_tbl.Order_Date   as virtrdf:Order_order-date ;
    order:Ship-Date    Order_tbl.Ship_Date    as virtrdf:Order_ship-date ;
```

```
order:Promise-Date Order_tbl.Promise_Date as virtrdf:Order_promise-date ;
order:Carrier      Order_tbl.Carrier      as virtrdf:Order_carrier ;
order:Instructions Order_tbl.Instructions as virtrdf:Order_instructions ;
order:PO           Order_tbl.PO           as virtrdf:Order_po ;
order:Terms        Order_tbl.Terms        as virtrdf:Order_terms ;
order:placed_by    customer:customer_iri (Customer_tbl.Cust_Num)                                  wher
order:Sales-Rep    salesrep:salesrep_iri (Salesrep_tbl.Sales_Rep)                                 wher
order:invoiced_on  invoice:invoice_iri (Invoice_tbl.Invoice_Num)                                  wher
order:has_lines    orderline:order-line_iri (Order_Line_tbl.Order_num, Order_Line_tbl.Line_num)  wher

item:item_iri (Item_tbl.Item_num) a item:Item as virtrdf:item_pk ;
item:Item-num        Item_tbl.Item_num        as virtrdf:item_item-num ;
item:Item-Name       Item_tbl.Item_Name       as virtrdf:Item_item-Name ;
item:Cat-Page        Item_tbl.Cat_Page        as virtrdf:Item_cat-page ;
item:Price           Item_tbl.Price           as virtrdf:Item_price ;
item:Cat-Description Item_tbl.Cat_Description as virtrdf:Item_cat-description ;
item:On-hand         Item_tbl.On_hand         as virtrdf:Item_on-hand ;
item:Allocated       Item_tbl.Allocated       as virtrdf:Item_allocated ;
item:Re-Order        Item_tbl.Re_Order        as virtrdf:Item_re-order ;
item:On-Order        Item_tbl.On_Order        as virtrdf:Item_on-order ;
item:order_line  orderline:order-line_iri (Order_Line_tbl.Order_num, Order_Line_tbl.Line_num)  where

orderline:order-line_iri (Order_Line_tbl.Order_num, Order_Line_tbl.Line_num) a orderline:Order-Line a
orderline:Line-num       Order_Line_tbl.Line_num       as virtrdf:Order-Line_line-num ;
orderline:Price          Order_Line_tbl.Price          as virtrdf:Order-Line_price ;
orderline:Qty            Order_Line_tbl.Qty            as virtrdf:Order-Line_qty ;
orderline:Discount       Order_Line_tbl.Discount       as virtrdf:Order-Line_discount ;
orderline:Extended-Price Order_Line_tbl.Extended_Price as virtrdf:Order-Line_extended-price ;
orderline:Backorder      Order_Line_tbl.Backorder      as virtrdf:Order-Line_backorder ;
orderline:Order-num  order:order_iri (Order_tbl.Order_num)  where ( ^{Order_Line_tbl.}^.Order_num = ^
orderline:Item-num   item:item_iri (Item_tbl.Item_num)      where ( ^{Order_Line_tbl.}^.Item_num  = ^

invoice:invoice_iri (Invoice_tbl.Invoice_Num) a invoice:Invoice as virtrdf:invoice_pk ;
invoice:Invoice-Num  Invoice_tbl.Invoice_Num  as virtrdf:Invoice_invoice-num ;
invoice:Cust-Num     Invoice_tbl.Cust_Num    as virtrdf:Invoice_cust_num ;
invoice:Invoice-Date Invoice_tbl.Invoice_Date as virtrdf:Invoice_invoice-date ;
invoice:Amount       Invoice_tbl.Amount       as virtrdf:Invoice_amount ;
invoice:Total-Paid   Invoice_tbl.Total_Paid   as virtrdf:Invoice_total-paid ;
invoice:Adjustment   Invoice_tbl.Adjustment   as virtrdf:Invoice_adjustment ;
invoice:Order-Num    Invoice_tbl.Order_Num    as virtrdf:Invoice_order-num ;
invoice:Ship-Charge  Invoice_tbl.Ship_Charge  as virtrdf:Invoice_ship-charge ;
invoice:invoiced_to  customer:customer_iri (Customer_tbl.Cust_Num)  where ( ^{Invoice_tbl.}^.Cust_Num
invoice:Order-Num    order:order_iri (Order_tbl.Order_num)          where ( ^{Invoice_tbl.}^.Order_Nu

localdefault:local-default_iri (Local_Default_tbl.Country) a localdefault:Local-Default as virtrdf:lo
localdefault:Country         Local_Default_tbl.Country as virtrdf:local-default_country ;
localdefault:Region1-Label   Local_Default_tbl.Region1_Label   as virtrdf:Local-Default_region1-label
localdefault:Region2-Label   Local_Default_tbl.Region2_Label   as virtrdf:Local-Default_region2-label
localdefault:Postal-Label    Local_Default_tbl.Postal_Label    as virtrdf:Local-Default_postal-label
localdefault:Postal-Format   Local_Default_tbl.Postal_Format   as virtrdf:Local-Default_postal-format
localdefault:Tel-Format      Local_Default_tbl.Tel_Format      as virtrdf:Local-Default_tel-format ;
localdefault:Date-Format     Local_Default_tbl.Date_Format     as virtrdf:Local-Default_date-format ;
localdefault:Currency-Symbol Local_Default_tbl.Currency_Symbol as virtrdf:Local-Default_currency-symb
localdefault:has_customer customer:customer_iri (Customer_tbl.Cust_Num) where ( ^{Local_Default_tbl.}

refcall:ref-call_iri (Ref_Call_tbl.Call_Num) a refcall:Ref-Call as virtrdf:ref-call_pk ;
refcall:Call-Num   Ref_Call_tbl.Call_Num   as virtrdf:Ref-Call_call-num ;
refcall:Cust-Num   Ref_Call_tbl.Cust_Num   as virtrdf:Ref-Call_cust-num ;
refcall:Call-Date  Ref_Call_tbl.Call_Date  as virtrdf:Ref-Call_call-date ;
refcall:Sales-Rep  Ref_Call_tbl.Sales_Rep  as virtrdf:Ref-sales-rep ;
refcall:Parent     Ref_Call_tbl.Parent     as virtrdf:Ref-Call_parent ;
refcall:Txt        Ref_Call_tbl.Txt        as virtrdf:Ref-Call_txt ;
refcall:made_to     customer:customer_iri (Customer_tbl.Cust_Num)   where  ( ^{Ref_Call_tbl.}^.Cust_N
refcall:made_by     salesrep:salesrep_iri (Salesrep_tbl.Sales_Rep)  where  ( ^{Ref_Call_tbl.}^.Sales_
refcall:has_parent  refcall:ref-call_iri (Ref_Call_tbl_1.Call_Num)    where  ( ^{Ref_Call_tbl.}^.Pare

salesrep:salesrep_iri (Salesrep_tbl.Sales_Rep) a salesrep:Salesrep as virtrdf:salesrep_pk ;
salesrep:Sales-Rep      Salesrep_tbl.Sales_Rep       as virtrdf:Salesrep_sales-rep ;
salesrep:Region         Salesrep_tbl.Region          as virtrdf:Salesrep_region ;
salesrep:Rep-Name       Salesrep_tbl.Rep_Name        as virtrdf:Salesrep_rep-name ;
salesrep:Month-Quota-1  Salesrep_tbl.Month_Quota_1   as virtrdf:Salesrep_month-quota-1 ;
salesrep:Month-Quota-2  Salesrep_tbl.Month_Quota_2   as virtrdf:Salesrep_month-quota-2 ;
salesrep:Month-Quota-3  Salesrep_tbl.Month_Quota_3   as virtrdf:Salesrep_month-quota-3 ;
```

```
    salesrep:Month-Quota-4  Salesrep_tbl.Month_Quota_4   as virtrdf:Salesrep_month-quota-4 ;
    salesrep:Month-Quota-5  Salesrep_tbl.Month_Quota_5   as virtrdf:Salesrep_month-quota-5 ;
    salesrep:Month-Quota-6  Salesrep_tbl.Month_Quota_6   as virtrdf:Salesrep_month-quota-6 ;
    salesrep:Month-Quota-7  Salesrep_tbl.Month_Quota_7   as virtrdf:Salesrep_month-quota-7 ;
    salesrep:Month-Quota-8  Salesrep_tbl.Month_Quota_8   as virtrdf:Salesrep_month-quota-8 ;
    salesrep:Month-Quota-9  Salesrep_tbl.Month_Quota_9   as virtrdf:Salesrep_month-quota-9 ;
    salesrep:Month-Quota-10 Salesrep_tbl.Month_Quota_10  as virtrdf:Salesrep_month-quota-10 ;
    salesrep:Month-Quota-11 Salesrep_tbl.Month_Quota_11  as virtrdf:Salesrep_month-quota-11 ;
    salesrep:Month-Quota-12 Salesrep_tbl.Month_Quota_12  as virtrdf:Salesrep_month-quota-12 ;
    salesrep:is_sales_rep_for customer:customer_iri (Customer_tbl.Cust_Num) where ( ^{Salesrep_tbl.}^.Sal
    salesrep:has_order        order:order_iri (Order_tbl.Order_num)         where ( ^{Salesrep_tbl.}^.Sal
    salesrep:manages_region   state:state_iri (State_tbl.State)             where ( ^{Salesrep_tbl.}^.Reg
    salesrep:made_call        refcall:ref-call_iri (Ref_Call_tbl.Call_Num)  where ( ^{Salesrep_tbl.}^.Sal

    state:state_iri (State_tbl.State) a state:State as virtrdf:state_pk ;
    state:State_     State_tbl.State        as virtrdf:State_state ;
    state:State-Name State_tbl.State_Name   as virtrdf:State_state-name ;
    state:Region     State_tbl.Region       as virtrdf:State_region ;
    state:has_customer   customer:customer_iri (Customer_tbl.Cust_Num)   where ( ^{State_tbl.}^.State  =
    state:has_sales_rep  salesrep:salesrep_iri (Salesrep_tbl.Sales_Rep)  where ( ^{State_tbl.}^.Region =

  } .
} .
;

delete from db.dba.url_rewrite_rule_list where urrl_list like 'progress_isports_rule%';
delete from db.dba.url_rewrite_rule where urr_rule like 'progress_isports_rule%';

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'progress_isports_rule1',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/about/html/http/^{URIQADefaultHost}^%s',
    vector('path'),
    null,
    '(text/html)|(\\*/\\*)',
    0,
    303
    );

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'progress_isports_rule2',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/sparql?query=DESCRIBE+%%3Chttp%%3A//localhost%%3A8890%U%%23this%%3E+%%3Chttp%%3A//localhost%%3A8890
    vector('path', 'path', '*accept*'),
    null,
    '(text/rdf.n3)|(application/rdf.xml)',
    0,
    null
    );

DB.DBA.URLREWRITE_CREATE_RULELIST (
    'progress_isports_rule_list1',
    1,
    vector (
                'progress_isports_rule1',
                'progress_isports_rule2'
          ));

-- ensure a VD for the IRIs which begins with /
VHOST_REMOVE (lpath=>'/progress/isports');

VHOST_DEFINE (
        lpath=>'/progress/isports',
        ppath=>'/DAV/progress/isports/',
        is_dav=>1,
        vsp_user=>'dba',
        is_brws=>0,
```

```
        opts=>vector ('url_rewrite', 'progress_isports_rule_list1')
        );

delete from db.dba.url_rewrite_rule_list where urrl_list like 'progress_isports_schemas_rule%';
delete from db.dba.url_rewrite_rule where urr_rule like 'progress_isports_schemas_rule%';

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'progress_isports_schemas_rule1',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/about/html/http/^{URIQADefaultHost}^%s',
    vector('path'),
    null,
    '(text/html)|(\\*/\\*)',
    0,
    303
    );

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'progress_isports_schemas_rule2',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/sparql?query=CONSTRUCT+{+%%3Chttp%%3A//localhost%%3A8890%U%%3E+%%3Fp+%%3Fo+}%%0D%%0AFROM+%%3Chttp%%
    vector('path','path','*accept*'),
    null,
    '(text/rdf.n3)|(application/rdf.xml)',
    0,
    null
    );

DB.DBA.URLREWRITE_CREATE_RULELIST (
    'progress_isports_schemas_rule_list1',
    1,
    vector (
                'progress_isports_schemas_rule1',
                'progress_isports_schemas_rule2'
          ));

-- ensure a VD for the IRIs which begins with /
VHOST_REMOVE (lpath=>'/schemas/progress/isports');

VHOST_DEFINE (
        lpath=>'/schemas/progress/isports',
        ppath=>'/DAV/schemas/progress/isports/',
        is_dav=>1,
        vsp_user=>'dba',
        is_brws=>0,
        opts=>vector ('url_rewrite', 'progress_isports_schemas_rule_list1')
        );

DB.DBA.XML_SET_NS_DECL ('customer',     'http://^{URIQADefaultHost}^/schemas/progress/isports/customer/',
DB.DBA.XML_SET_NS_DECL ('order',        'http://^{URIQADefaultHost}^/schemas/progress/isports/order/', 2)
DB.DBA.XML_SET_NS_DECL ('item',         'http://^{URIQADefaultHost}^/schemas/progress/isports/item/', 2);
DB.DBA.XML_SET_NS_DECL ('orderline',    'http://^{URIQADefaultHost}^/schemas/progress/isports/order_line/
DB.DBA.XML_SET_NS_DECL ('invoice',      'http://^{URIQADefaultHost}^/schemas/progress/isports/invoice/',
DB.DBA.XML_SET_NS_DECL ('localdefault', 'http://^{URIQADefaultHost}^/schemas/progress/isports/local_defau
DB.DBA.XML_SET_NS_DECL ('refcall',      'http://^{URIQADefaultHost}^/schemas/progress/isports/ref_call/',
DB.DBA.XML_SET_NS_DECL ('salesrep',     'http://^{URIQADefaultHost}^/schemas/progress/isports/salesrep/',
DB.DBA.XML_SET_NS_DECL ('state',        'http://^{URIQADefaultHost}^/schemas/progress/isports/state/', 2)
```

16.8.17. Progress (SQL-92) using demonstration 'iSports' database

```
ATTACH TABLE  "PUB"."Customer"      PRIMARY KEY ("Cust-Num")              AS "isports_rdf"."prs10_isports
ATTACH TABLE  "PUB"."Invoice"       PRIMARY KEY ("Invoice-Num")           AS "isports_rdf"."prs10_isports
ATTACH TABLE  "PUB"."Item"          PRIMARY KEY ("Item-num")              AS "isports_rdf"."prs10_isports
ATTACH TABLE  "PUB"."Local-Default" PRIMARY KEY ("Country")               AS "isports_rdf"."prs10_isports
ATTACH TABLE  "PUB"."Order"         PRIMARY KEY ("Order-num")             AS "isports_rdf"."prs10_isports
ATTACH TABLE  "PUB"."Order-Line"    PRIMARY KEY ("Order-num", "Line-num") AS "isports_rdf"."prs10_isports
ATTACH TABLE  "PUB"."Ref-Call"      PRIMARY KEY ("Call-Num")              AS "isports_rdf"."prs10_isports
```

```
ATTACH TABLE  "PUB"."Salesrep"      PRIMARY KEY ("Sales-Rep")            AS "isports_rdf"."prs10_isports
ATTACH TABLE  "PUB"."State"         PRIMARY KEY ("State")                AS "isports_rdf"."prs10_isports

COMMIT WORK;

GRANT SELECT ON isports_rdf.prs10_isports_rdf.Customer        TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.prs10_isports_rdf."Order"         TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.prs10_isports_rdf.Item            TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.prs10_isports_rdf."Order-Line"    TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.prs10_isports_rdf.Invoice         TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.prs10_isports_rdf."Local-Default" TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.prs10_isports_rdf."Ref-Call"      TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.prs10_isports_rdf.Salesrep        TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.prs10_isports_rdf.State           TO "SPARQL", "SPARQL_UPDATE";

GRANT SPARQL_UPDATE to "SPARQL";

CREATE VIEW isports_rdf.prs10_isports_rdf.VCustomer      AS SELECT "Cust-Num" AS Cust_Num, Name, Address,
CREATE VIEW isports_rdf.prs10_isports_rdf.VOrder         AS SELECT "Order-num" AS Order_num, "Cust-Num" A
CREATE VIEW isports_rdf.prs10_isports_rdf.VItem          AS SELECT "Item-num" AS Item_num, "Item-Name" AS
CREATE VIEW isports_rdf.prs10_isports_rdf.VOrder_Line    AS SELECT "Order-num" AS Order_num, "Line-num" A
CREATE VIEW isports_rdf.prs10_isports_rdf.VInvoice       AS SELECT "Invoice-Num" AS Invoice_Num, "Cust-Nu
CREATE VIEW isports_rdf.prs10_isports_rdf.VLocal_Default AS SELECT Country, "Region1-Label" AS Region1_La
CREATE VIEW isports_rdf.prs10_isports_rdf.VRef_Call      AS SELECT "Call-Num" AS Call_Num, "Cust-Num" AS
CREATE VIEW isports_rdf.prs10_isports_rdf.VSalesrep      AS SELECT "Rep-Name" AS Rep_Name, Region, "Sales
CREATE VIEW isports_rdf.prs10_isports_rdf.VState         AS SELECT State, "State-Name" AS State_Name, Reg

GRANT SELECT ON isports_rdf.prs10_isports_rdf.VCustomer      TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.prs10_isports_rdf.VOrder         TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.prs10_isports_rdf.VItem          TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.prs10_isports_rdf.VOrder_Line    TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.prs10_isports_rdf.VInvoice       TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.prs10_isports_rdf.VLocal_Default TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.prs10_isports_rdf.VRef_Call      TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.prs10_isports_rdf.VSalesrep      TO "SPARQL", "SPARQL_UPDATE";
GRANT SELECT ON isports_rdf.prs10_isports_rdf.VState         TO "SPARQL", "SPARQL_UPDATE";

-------- Create rdfs:Class definitions ----------------------------

ttlp (
'
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

@prefix customer:     <http://example.com/schemas/progress/isports/customer/> .
@prefix order:        <http://example.com/schemas/progress/isports/order/> .
@prefix item:         <http://example.com/schemas/progress/isports/item/> .
@prefix orderline:    <http://example.com/schemas/progress/isports/order_line/> .
@prefix invoice:      <http://example.com/schemas/progress/isports/invoice/> .
@prefix localdefault: <http://example.com/schemas/progress/isports/local_default/> .
@prefix refcall:      <http://example.com/schemas/progress/isports/ref_call/> .
@prefix salesrep:     <http://example.com/schemas/progress/isports/salesrep/> .
@prefix state:        <http://example.com/schemas/progress/isports/state/> .

customer:Customer a rdfs:Class ;
        rdfs:label "Customer" ;
        rdfs:comment "Progress isports Customer table" .

customer:Cust-Num a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:integer ;
        rdfs:label "Cust-Num" .

customer:Name a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "Name" .

customer:Address a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "Address" .
```

```
customer:Address2 a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "Address2" .

customer:City a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "City" .

customer:State a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "State" .

customer:Country a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "Country" .

customer:Phone a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "Phone" .

customer:Contact a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "Contact" .

customer:Sales-Rep a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "Sales-Rep" .

customer:Comments a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "Comments" .

customer:Credit-Limit a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:decimal ;
        rdfs:label "Credit-Limit" .

customer:Balance a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:decimal ;
        rdfs:label "Balance" .

customer:Terms a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "Terms" .

customer:Discount a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:integer ;
        rdfs:label "Discount" .

customer:Postal-Code a rdf:Property ;
        rdfs:domain customer:Customer ;
        rdfs:range xsd:string ;
        rdfs:label "Postal-Code" .

order:Order a rdfs:Class ;
        rdfs:label "Order" ;
        rdfs:comment "Progress isports Order table" .

order:Order-num a rdf:Property ;
        rdfs:domain order:Order ;
        rdfs:range xsd:integer ;
```

```
        rdfs:label "Order-num" .

order:Cust-Num a rdf:Property ;
        rdfs:domain order:Order ;
        rdfs:range xsd:integer ;
        rdfs:label "Cust-Num" .

order:Order-Date a rdf:Property ;
        rdfs:domain order:Order ;
        rdfs:range xsd:date ;
        rdfs:label "Order-Date" .

order:Ship-Date a rdf:Property ;
        rdfs:domain order:Order ;
        rdfs:range xsd:date ;
        rdfs:label "Ship-Date" .

order:Promise-Date a rdf:Property ;
        rdfs:domain order:Order ;
        rdfs:range xsd:date ;
        rdfs:label "Promise-Date" .

order:Carrier a rdf:Property ;
        rdfs:domain order:Order ;
        rdfs:range xsd:string ;
        rdfs:label "Carrier" .

order:Instructions a rdf:Property ;
        rdfs:domain order:Order ;
        rdfs:range xsd:string ;
        rdfs:label "Instructions" .

order:PO a rdf:Property ;
        rdfs:domain order:Order ;
        rdfs:range xsd:string ;
        rdfs:label "PO" .

order:Terms a rdf:Property ;
        rdfs:domain order:Order ;
        rdfs:range xsd:string ;
        rdfs:label "Terms" .

order:Sales-Rep a rdf:Property ;
        rdfs:domain order:Order ;
        rdfs:range xsd:string ;
        rdfs:label "Sales-Rep" .

item:Item a rdfs:Class ;
        rdfs:label "Item" ;
        rdfs:comment "Progress isports Item table" .

item:Item-num a rdf:Property ;
        rdfs:domain item:Item ;
        rdfs:range xsd:integer ;
        rdfs:label "Item-num" .

item:Item-Name a rdf:Property ;
        rdfs:domain item:Item ;
        rdfs:range xsd:string ;
        rdfs:label "Item-Name" .

item:Cat-Page a rdf:Property ;
        rdfs:domain item:Item ;
        rdfs:range xsd:integer ;
        rdfs:label "Cat-Page" .

item:Price a rdf:Property ;
        rdfs:domain item:Item ;
        rdfs:range xsd:decimal ;
        rdfs:label "Price" .

item:Cat-Description a rdf:Property ;
        rdfs:domain item:Item ;
```

```
        rdfs:range xsd:string ;
        rdfs:label "Cat-Description" .

item:On-hand a rdf:Property ;
        rdfs:domain item:Item ;
        rdfs:range xsd:integer ;
        rdfs:label "On-hand" .

item:Allocated a rdf:Property ;
        rdfs:domain item:Item ;
        rdfs:range xsd:integer ;
        rdfs:label "Allocated" .

item:Re-Order a rdf:Property ;
        rdfs:domain item:Item ;
        rdfs:range xsd:integer ;
        rdfs:label "Re-Order" .

item:On-Order a rdf:Property ;
        rdfs:domain item:Item ;
        rdfs:range xsd:integer ;
        rdfs:label "On-Order" .

orderline:Order-Line a rdfs:Class ;
        rdfs:label "Order-Line" ;
        rdfs:comment "Progress isports Order-Line table" .

orderline:Order-num a rdf:Property ;
        rdfs:domain orderline:Order-Line ;
        rdfs:range xsd:integer ;
        rdfs:label "Order-num" .

orderline:Line-num a rdf:Property ;
        rdfs:domain orderline:Order-Line ;
        rdfs:range xsd:integer ;
        rdfs:label "Line-num" .

orderline:Item-num a rdf:Property ;
        rdfs:domain orderline:Order-Line ;
        rdfs:range xsd:integer ;
        rdfs:label "Item-num" .

orderline:Price a rdf:Property ;
        rdfs:domain orderline:Order-Line ;
        rdfs:range xsd:decimal ;
        rdfs:label "Price" .

orderline:Qty a rdf:Property ;
        rdfs:domain orderline:Order-Line ;
        rdfs:range xsd:integer ;
        rdfs:label "Qty" .

orderline:Discount a rdf:Property ;
        rdfs:domain orderline:Order-Line ;
        rdfs:range xsd:integer ;
        rdfs:label "Discount" .

orderline:Extended-Price a rdf:Property ;
        rdfs:domain orderline:Order-Line ;
        rdfs:range xsd:decimal ;
        rdfs:label "Extended-Price" .

orderline:Backorder a rdf:Property ;
        rdfs:domain orderline:Order-Line ;
        rdfs:range xsd:byte ;
        rdfs:label "Backorder" .

invoice:Invoice a rdfs:Class ;
        rdfs:label "Invoice" ;
        rdfs:comment "Progress isports Invoice table" .

invoice:Invoice-Num a rdf:Property ;
        rdfs:domain invoice:Invoice ;
```

```
        rdfs:range xsd:integer ;
        rdfs:label "Invoice-Num" .

invoice:Cust-Num a rdf:Property ;
        rdfs:domain invoice:Invoice ;
        rdfs:range xsd:integer ;
        rdfs:label "Cust-Num" .

invoice:Invoice-Date a rdf:Property ;
        rdfs:domain invoice:Invoice ;
        rdfs:range xsd:date ;
        rdfs:label "Invoice-Date" .

invoice:Amount a rdf:Property ;
        rdfs:domain invoice:Invoice ;
        rdfs:range xsd:decimal ;
        rdfs:label "Amount" .

invoice:Total-Paid a rdf:Property ;
        rdfs:domain invoice:Invoice ;
        rdfs:range xsd:decimal ;
        rdfs:label "Total-Paid" .

invoice:Adjustment a rdf:Property ;
        rdfs:domain invoice:Invoice ;
        rdfs:range xsd:decimal ;
        rdfs:label "Adjustment" .

invoice:Order-Num a rdf:Property ;
        rdfs:domain invoice:Invoice ;
        rdfs:range xsd:integer ;
        rdfs:label "Order-Num" .

invoice:Ship-Charge a rdf:Property ;
        rdfs:domain invoice:Invoice ;
        rdfs:range xsd:decimal ;
        rdfs:label "Ship-Charge" .

localdefault:Local-Default a rdfs:Class ;
        rdfs:label "Local-Default" ;
        rdfs:comment "Progress isports Local-Default table" .

localdefault:Country a rdf:Property ;
        rdfs:domain localdefault:Local-Default ;
        rdfs:range xsd:string ;
        rdfs:label "Country" .

localdefault:Region1-Label a rdf:Property ;
        rdfs:domain localdefault:Local-Default ;
        rdfs:range xsd:string ;
        rdfs:label "Region1-Label" .

localdefault:Region2-Label a rdf:Property ;
        rdfs:domain localdefault:Local-Default ;
        rdfs:range xsd:string ;
        rdfs:label "Region2-Label" .

localdefault:Postal-Label a rdf:Property ;
        rdfs:domain localdefault:Local-Default ;
        rdfs:range xsd:string ;
        rdfs:label "Postal-Label" .

localdefault:Postal-Format a rdf:Property ;
        rdfs:domain localdefault:Local-Default ;
        rdfs:range xsd:string ;
        rdfs:label "Postal-Format" .

localdefault:Tel-Format a rdf:Property ;
        rdfs:domain localdefault:Local-Default ;
        rdfs:range xsd:string ;
        rdfs:label "Tel-Format" .

localdefault:Date-Format a rdf:Property ;
```

```
        rdfs:domain localdefault:Local-Default ;
        rdfs:range xsd:string ;
        rdfs:label "Date-Format" .

localdefault:Currency-Symbol a rdf:Property ;
        rdfs:domain localdefault:Local-Default ;
        rdfs:range xsd:string ;
        rdfs:label "Currency-Symbol" .

refcall:Ref-Call a rdfs:Class ;
        rdfs:label "Ref-Call" ;
        rdfs:comment "Progress isports Ref-Call table" .

refcall:Call-Num a rdf:Property ;
        rdfs:domain refcall:Ref-Call ;
        rdfs:range xsd:string ;
        rdfs:label "Call-Num" .

refcall:Cust-Num a rdf:Property ;
        rdfs:domain refcall:Ref-Call ;
        rdfs:range xsd:integer ;
        rdfs:label "Cust-Num" .

refcall:Call-Date a rdf:Property ;
        rdfs:domain refcall:Ref-Call ;
        rdfs:range xsd:date ;
        rdfs:label "Call-Date" .

refcall:Sales-Rep a rdf:Property ;
        rdfs:domain refcall:Ref-Call ;
        rdfs:range xsd:string ;
        rdfs:label "Sales-Rep" .

refcall:Parent a rdf:Property ;
        rdfs:domain refcall:Ref-Call ;
        rdfs:range xsd:string ;
        rdfs:label "Parent" .

refcall:Txt a rdf:Property ;
        rdfs:domain refcall:Ref-Call ;
        rdfs:range xsd:string ;
        rdfs:label "Txt" .

salesrep:Salesrep a rdfs:Class ;
        rdfs:label "Salesrep" ;
        rdfs:comment "Progress isports Salesrep table" .

salesrep:Sales-Rep a rdf:Property ;
        rdfs:domain salesrep:Salesrep ;
        rdfs:range xsd:string ;
        rdfs:label "Sales-Rep" .

salesrep:Rep-Name a rdf:Property ;
        rdfs:domain salesrep:Salesrep ;
        rdfs:range xsd:string ;
        rdfs:label "Rep-Name" .

salesrep:Region a rdf:Property ;
        rdfs:domain salesrep:Salesrep ;
        rdfs:range xsd:string ;
        rdfs:label "Region" .

salesrep:Month-Quota a rdf:Property ;
        rdfs:domain salesrep:Salesrep ;
        rdfs:range xsd:string ;
        rdfs:label "Month-Quota" .

state:State a rdfs:Class ;
        rdfs:label "State" ;
        rdfs:comment "Progress isports State table" .

state:State_ a rdf:Property ;
        rdfs:domain state:State ;
```

```
        rdfs:range xsd:string ;
        rdfs:label "State" .

state:State-Name a rdf:Property ;
        rdfs:domain state:State ;
        rdfs:range xsd:string ;
        rdfs:label "State-Name" .

state:Region a rdf:Property ;
        rdfs:domain state:State ;
        rdfs:range xsd:string ;
        rdfs:label "Region" .

', '', 'http://example.com/schemas/progress/isports', 0);

----------- Create IRI Classes -------------

SPARQL

prefix customer:     <http://example.com/schemas/progress/isports/customer/>
prefix order:        <http://example.com/schemas/progress/isports/order/>
prefix item:         <http://example.com/schemas/progress/isports/item/>
prefix orderline:    <http://example.com/schemas/progress/isports/order_line/>
prefix invoice:      <http://example.com/schemas/progress/isports/invoice/>
prefix localdefault: <http://example.com/schemas/progress/isports/local_default/>
prefix refcall:      <http://example.com/schemas/progress/isports/ref_call/>
prefix salesrep:     <http://example.com/schemas/progress/isports/salesrep/>
prefix state:        <http://example.com/schemas/progress/isports/state/>

create iri class customer:customer_iri
        "http://example.com/progress/isports/customer/%d#this"
        (in Cust_Num integer not null) .

create iri class order:order_iri
        "http://example.com/progress/isports/order/%d#this"
        (in Order_Num integer not null) .

create iri class item:item_iri
        "http://example.com/progress/isports/item/%d#this"
        (in Item_num integer not null) .

create iri class orderline:order-line_iri
        "http://example.com/progress/isports/order-line/%d_%d#this"
        (in Order_num integer not null, in Line_num integer not null) .

create iri class invoice:invoice_iri
        "http://example.com/progress/isports/invoice/%d#this"
        (in Invoice_Num integer not null) .

create iri class localdefault:local-default_iri
        "http://example.com/progress/isports/local-default/%U#this"
        (in Country varchar not null) .

create iri class refcall:ref-call_iri
        "http://example.com/progress/isports/ref-call/%U#this"
        (in Call_Num varchar not null) .

create iri class salesrep:salesrep_iri
        "http://example.com/progress/isports/salesrep/%U#this"
        (in Sales_Rep varchar not null) .

create iri class state:state_iri
        "http://example.com/progress/isports/state/%U#this"
        (in State varchar not null) .

;

------------- Create Quad Store -----------------------------------

SPARQL

prefix customer:     <http://example.com/schemas/progress/isports/customer/>
prefix order:        <http://example.com/schemas/progress/isports/order/>
```

```
prefix item:         <http://example.com/schemas/progress/isports/item/>
prefix orderline:    <http://example.com/schemas/progress/isports/order_line/>
prefix invoice:      <http://example.com/schemas/progress/isports/invoice/>
prefix localdefault: <http://example.com/schemas/progress/isports/local_default/>
prefix refcall:      <http://example.com/schemas/progress/isports/ref_call/>
prefix salesrep:     <http://example.com/schemas/progress/isports/salesrep/>
prefix state:        <http://example.com/schemas/progress/isports/state/>

alter quad storage virtrdf:DefaultQuadStorage
  from isports_rdf.prs10_isports_rdf.VCustomer      as Customer_tbl
  from isports_rdf.prs10_isports_rdf.VOrder         as Order_tbl
  from isports_rdf.prs10_isports_rdf.VItem          as Item_tbl
  from isports_rdf.prs10_isports_rdf.VOrder_Line    as Order_Line_tbl
  from isports_rdf.prs10_isports_rdf.VInvoice       as Invoice_tbl
  from isports_rdf.prs10_isports_rdf.VRef_Call      as Ref_Call_tbl
  from isports_rdf.prs10_isports_rdf.VRef_Call      as Ref_Call_tbl_1  -- Additional Ref_Call_tbl_1 alias
  from isports_rdf.prs10_isports_rdf.VLocal_Default as Local_Default_tbl
  from isports_rdf.prs10_isports_rdf.VSalesrep      as Salesrep_tbl
  from isports_rdf.prs10_isports_rdf.VState         as State_tbl
{
  create virtrdf:progress_isports as graph <http://example.com/progress/isports>
  {
    customer:customer_iri (Customer_tbl.Cust_Num) a customer:Customer as virtrdf:customer_pk ;
    customer:Cust-Num     Customer_tbl.Cust_Num     as virtrdf:Customer_cust-num ;
    customer:Name         Customer_tbl.Name         as virtrdf:Customer_name ;
    customer:Address      Customer_tbl.Address      as virtrdf:Customer_address ;
    customer:Address2     Customer_tbl.Address2     as virtrdf:Customer_address2 ;
    customer:City         Customer_tbl.City         as virtrdf:Customer_city ;
    customer:State        Customer_tbl.State        as virtrdf:Customer_state ;
    customer:Country      Customer_tbl.Country      as virtrdf:Customer_country ;
    customer:Phone        Customer_tbl.Phone        as virtrdf:Customer_phone ;
    customer:Contact      Customer_tbl.Contact      as virtrdf:Customer_contact ;
    customer:Sales-Rep    Customer_tbl.Sales_Rep    as virtrdf:Customer_sales_rep ;
    customer:Comments     Customer_tbl.Comments     as virtrdf:Customer_comments ;
    customer:Credit-Limit Customer_tbl.Credit_Limit as virtrdf:Customer_credit-limit ;
    customer:Balance      Customer_tbl.Balance      as virtrdf:Customer_balance ;
    customer:Terms        Customer_tbl.Terms        as virtrdf:Customer_terms ;
    customer:Discount     Customer_tbl.Discount     as virtrdf:Customer_discount ;
    customer:Postal-Code  Customer_tbl.Postal_Code  as virtrdf:Customer_postal-code ;
    customer:from_state        state:state_iri (State_tbl.State)                            where ( ^{Cust
    customer:has_sales_rep     salesrep:salesrep_iri (Salesrep_tbl.Sales_Rep)               where ( ^{Cust
    customer:has_local_default localdefault:local-default_iri (Local_Default_tbl.Country)  where ( ^{Cust
    customer:placed_order      order:order_iri (Order_tbl.Order_num)                        where ( ^{Cust
    customer:has_invoice       invoice:invoice_iri (Invoice_tbl.Invoice_Num)                where ( ^{Cust
    customer:ref_call          refcall:ref-call_iri (Ref_Call_tbl.Call_Num)                 where ( ^{Cust

    order:order_iri (Order_tbl.Order_num) a order:Order as virtrdf:order_pk ;
    order:Order-num    Order_tbl.Order_num    as virtrdf:Order_order-num ;
    order:Cust-Num     Order_tbl.Cust_Num     as virtrdf:Order_cust_num ;
    order:Order-Date   Order_tbl.Order_Date   as virtrdf:Order_order-date ;
    order:Ship-Date    Order_tbl.Ship_Date    as virtrdf:Order_ship-date ;
    order:Promise-Date Order_tbl.Promise_Date as virtrdf:Order_promise-date ;
    order:Carrier      Order_tbl.Carrier      as virtrdf:Order_carrier ;
    order:Instructions Order_tbl.Instructions as virtrdf:Order_instructions ;
    order:PO           Order_tbl.PO           as virtrdf:Order_po ;
    order:Terms        Order_tbl.Terms        as virtrdf:Order_terms ;
    order:placed_by    customer:customer_iri (Customer_tbl.Cust_Num)                                  wher
    order:Sales-Rep    salesrep:salesrep_iri (Salesrep_tbl.Sales_Rep)                                 wher
    order:invoiced_on  invoice:invoice_iri (Invoice_tbl.Invoice_Num)                                  wher
    order:has_lines    orderline:order-line_iri (Order_Line_tbl.Order_num, Order_Line_tbl.Line_num)  wher

    item:item_iri (Item_tbl.Item_num) a item:Item as virtrdf:item_pk ;
    item:Item-num        Item_tbl.Item_num        as virtrdf:item_item-num ;
    item:Item-Name       Item_tbl.Item_Name       as virtrdf:Item_item-Name ;
    item:Cat-Page        Item_tbl.Cat_Page        as virtrdf:Item_cat-page ;
    item:Price           Item_tbl.Price           as virtrdf:Item_price ;
    item:Cat-Description Item_tbl.Cat_Description as virtrdf:Item_cat-description ;
    item:On-hand         Item_tbl.On_hand         as virtrdf:Item_on-hand ;
    item:Allocated       Item_tbl.Allocated       as virtrdf:Item_allocated ;
    item:Re-Order        Item_tbl.Re_Order        as virtrdf:Item_re-order ;
    item:On-Order        Item_tbl.On_Order        as virtrdf:Item_on-order ;
    item:order_line  orderline:order-line_iri (Order_Line_tbl.Order_num, Order_Line_tbl.Line_num)  where
```

```
    orderline:order-line_iri (Order_Line_tbl.Order_num, Order_Line_tbl.Line_num) a orderline:Order-Line a
    orderline:Line-num       Order_Line_tbl.Line_num       as virtrdf:Order-Line_line-num ;
    orderline:Price          Order_Line_tbl.Price          as virtrdf:Order-Line_price ;
    orderline:Qty            Order_Line_tbl.Qty            as virtrdf:Order-Line_qty ;
    orderline:Discount       Order_Line_tbl.Discount       as virtrdf:Order-Line_discount ;
    orderline:Extended-Price Order_Line_tbl.Extended_Price as virtrdf:Order-Line_extended-price ;
    orderline:Backorder      Order_Line_tbl.Backorder      as virtrdf:Order-Line_backorder ;
    orderline:Order-num  order:order_iri (Order_tbl.Order_num)  where ( ^{Order_Line_tbl.}^.Order_num = ^
    orderline:Item-num   item:item_iri (Item_tbl.Item_num)      where ( ^{Order_Line_tbl.}^.Item_num  = ^

    invoice:invoice_iri (Invoice_tbl.Invoice_Num) a invoice:Invoice as virtrdf:invoice_pk ;
    invoice:Invoice-Num  Invoice_tbl.Invoice_Num  as virtrdf:Invoice_invoice-num ;
    invoice:Cust-Num     Invoice_tbl.Cust_Num    as virtrdf:Invoice_cust_num ;
    invoice:Invoice-Date Invoice_tbl.Invoice_Date as virtrdf:Invoice_invoice-date ;
    invoice:Amount       Invoice_tbl.Amount       as virtrdf:Invoice_amount ;
    invoice:Total-Paid   Invoice_tbl.Total_Paid   as virtrdf:Invoice_total-paid ;
    invoice:Adjustment   Invoice_tbl.Adjustment   as virtrdf:Invoice_adjustment ;
    invoice:Order-Num    Invoice_tbl.Order_Num    as virtrdf:Invoice_order-num ;
    invoice:Ship-Charge  Invoice_tbl.Ship_Charge  as virtrdf:Invoice_ship-charge ;
    invoice:invoiced_to  customer:customer_iri (Customer_tbl.Cust_Num)  where ( ^{Invoice_tbl.}^.Cust_Num
    invoice:Order-Num    order:order_iri (Order_tbl.Order_num)          where ( ^{Invoice_tbl.}^.Order_Nu

    localdefault:local-default_iri (Local_Default_tbl.Country) a localdefault:Local-Default as virtrdf:lo
    localdefault:Country         Local_Default_tbl.Country as virtrdf:local-default_country ;
    localdefault:Region1-Label   Local_Default_tbl.Region1_Label   as virtrdf:Local-Default_region1-label
    localdefault:Region2-Label   Local_Default_tbl.Region2_Label   as virtrdf:Local-Default_region2-label
    localdefault:Postal-Label    Local_Default_tbl.Postal_Label    as virtrdf:Local-Default_postal-label
    localdefault:Postal-Format   Local_Default_tbl.Postal_Format   as virtrdf:Local-Default_postal-format
    localdefault:Tel-Format      Local_Default_tbl.Tel_Format      as virtrdf:Local-Default_tel-format ;
    localdefault:Date-Format     Local_Default_tbl.Date_Format     as virtrdf:Local-Default_date-format ;
    localdefault:Currency-Symbol Local_Default_tbl.Currency_Symbol as virtrdf:Local-Default_currency-symb
    localdefault:has_customer customer:customer_iri (Customer_tbl.Cust_Num) where ( ^{Local_Default_tbl.}

    refcall:ref-call_iri (Ref_Call_tbl.Call_Num) a refcall:Ref-Call as virtrdf:ref-call_pk ;
    refcall:Call-Num   Ref_Call_tbl.Call_Num   as virtrdf:Ref-Call_call-num ;
    refcall:Cust-Num   Ref_Call_tbl.Cust_Num   as virtrdf:Ref-Call_cust-num ;
    refcall:Call-Date  Ref_Call_tbl.Call_Date  as virtrdf:Ref-Call_call-date ;
    refcall:Sales-Rep  Ref_Call_tbl.Sales_Rep  as virtrdf:Ref-sales-rep ;
    refcall:Parent     Ref_Call_tbl.Parent     as virtrdf:Ref-Call_parent ;
    refcall:Txt        Ref_Call_tbl.Txt        as virtrdf:Ref-Call_txt ;
    refcall:made_to     customer:customer_iri (Customer_tbl.Cust_Num)   where  ( ^{Ref_Call_tbl.}^.Cust_N
    refcall:made_by     salesrep:salesrep_iri (Salesrep_tbl.Sales_Rep)  where  ( ^{Ref_Call_tbl.}^.Sales_
    refcall:has_parent  refcall:ref-call_iri (Ref_Call_tbl_1.Call_Num)  where  ( ^{Ref_Call_tbl.}^.Parent

    salesrep:salesrep_iri (Salesrep_tbl.Sales_Rep) a salesrep:Salesrep as virtrdf:salesrep_pk ;
    salesrep:Sales-Rep   Salesrep_tbl.Sales_Rep   as virtrdf:Salesrep_sales-rep ;
    salesrep:Region      Salesrep_tbl.Region      as virtrdf:Salesrep_region ;
    salesrep:Rep-Name    Salesrep_tbl.Rep_Name    as virtrdf:Salesrep_rep-name ;
    salesrep:Month-Quota Salesrep_tbl.Month_Quota as virtrdf:Salesrep_month-quota ;
    salesrep:is_sales_rep_for customer:customer_iri (Customer_tbl.Cust_Num) where ( ^{Salesrep_tbl.}^.Sal
    salesrep:has_order        order:order_iri (Order_tbl.Order_num)         where ( ^{Salesrep_tbl.}^.Sal
    salesrep:manages_region   state:state_iri (State_tbl.State)             where ( ^{Salesrep_tbl.}^.Reg
    salesrep:made_call        refcall:ref-call_iri (Ref_Call_tbl.Call_Num)  where ( ^{Salesrep_tbl.}^.Sal

    state:state_iri (State_tbl.State) a state:State as virtrdf:state_pk ;
    state:State_     State_tbl.State        as virtrdf:State_state ;
    state:State-Name State_tbl.State_Name   as virtrdf:State_state-name ;
    state:Region     State_tbl.Region       as virtrdf:State_region ;
    state:has_customer   customer:customer_iri (Customer_tbl.Cust_Num)   where ( ^{State_tbl.}^.State  =
    state:has_sales_rep  salesrep:salesrep_iri (Salesrep_tbl.Sales_Rep)  where ( ^{State_tbl.}^.Region =

  } .
} .
;

-- Setup re-write rules that enable de-referencing of RDF based descriptions of
-- iSports Entities

-- Cleanup old rules

delete from db.dba.url_rewrite_rule_list where urrl_list like 'progress_isports_rule%';
delete from db.dba.url_rewrite_rule where urr_rule like 'progress_isports_rule%';
```

```
-- Create rules for handling HTML representation of Entity (resource) description requests

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'progress_isports_rule1',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/about/html/http/^{URIQADefaultHost}^%s',
    vector('path'),
    null,
    '(text/html)|(\\*/\\*)',
    0,
    303
    );

-- Create rules for handling RDF based representations (N3 or RDF/XML) of Entity (resource) descriptions

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'progress_isports_rule2',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/sparql?query=DESCRIBE+%%3Chttp%%3A//localhost%%3A8890%U%%23this%%3E+%%3Chttp%%3A//localhost%%3A8890
    vector('path', 'path', '*accept*'),
    null,
    '(text/rdf.n3)|(application/rdf.xml)',
    0,
    null
    );

DB.DBA.URLREWRITE_CREATE_RULELIST (
    'progress_isports_rule_list1',
    1,
    vector (
                'progress_isports_rule1',
                'progress_isports_rule2'
          ));

-- Setup OWL ontology data space that describes iSports entities

-- Create Virtual Directory access point

VHOST_REMOVE (lpath=>'/progress/isports');

VHOST_DEFINE (
        lpath=>'/progress/isports',
        ppath=>'/DAV/progress/isports/',
        is_dav=>1,
        vsp_user=>'dba',
        is_brws=>0,
        opts=>vector ('url_rewrite', 'progress_isports_rule_list1')
        );

delete from db.dba.url_rewrite_rule_list where urrl_list like 'progress_isports_schemas_rule%';
delete from db.dba.url_rewrite_rule where urr_rule like 'progress_isports_schemas_rule%';

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'progress_isports_schemas_rule1',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/about/html/http/^{URIQADefaultHost}^%s',
    vector('path'),
    null,
    '(text/html)|(\\*/\\*)',
    0,
    303
    );

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
```

```
    'progress_isports_schemas_rule2',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/sparql?query=CONSTRUCT+{+%%3Chttp%%3A//localhost%%3A8890%U%%3E+%%3Fp+%%3Fo+}%%0D%%0AFROM+%%3Chttp%%
    vector('path','path','*accept*'),
    null,
    '(text/rdf.n3)|(application/rdf.xml)',
    0,
    null
    );

DB.DBA.URLREWRITE_CREATE_RULELIST (
    'progress_isports_schemas_rule_list1',
    1,
    vector (
                'progress_isports_schemas_rule1',
                'progress_isports_schemas_rule2'
          ));

-- ensure a VD for the IRIs which begins with /
VHOST_REMOVE (lpath=>'/schemas/progress/isports');

VHOST_DEFINE (
        lpath=>'/schemas/progress/isports',
        ppath=>'/DAV/schemas/progress/isports/',
        is_dav=>1,
        vsp_user=>'dba',
        is_brws=>0,
        opts=>vector ('url_rewrite', 'progress_isports_schemas_rule_list1')
        );

DB.DBA.XML_SET_NS_DECL ('customer',     'http://^{URIQADefaultHost}^/schemas/progress/isports/customer/',
DB.DBA.XML_SET_NS_DECL ('order',        'http://^{URIQADefaultHost}^/schemas/progress/isports/order/', 2)
DB.DBA.XML_SET_NS_DECL ('item',         'http://^{URIQADefaultHost}^/schemas/progress/isports/item/', 2);
DB.DBA.XML_SET_NS_DECL ('orderline',    'http://^{URIQADefaultHost}^/schemas/progress/isports/order_line/
DB.DBA.XML_SET_NS_DECL ('invoice',      'http://^{URIQADefaultHost}^/schemas/progress/isports/invoice/',
DB.DBA.XML_SET_NS_DECL ('localdefault', 'http://^{URIQADefaultHost}^/schemas/progress/isports/local_defau
DB.DBA.XML_SET_NS_DECL ('refcall',      'http://^{URIQADefaultHost}^/schemas/progress/isports/ref_call/',
DB.DBA.XML_SET_NS_DECL ('salesrep',     'http://^{URIQADefaultHost}^/schemas/progress/isports/salesrep/',
DB.DBA.XML_SET_NS_DECL ('state',        'http://^{URIQADefaultHost}^/schemas/progress/isports/state/', 2)
```

16.9. RDF Insert Methods in Virtuoso

16.9.1. Using API functions

- Using the DB.DBA.TTLP() function
    - Note: use this function for loading Turtle
- Using the DB.DBA.TTLP_MT() function
    - Note: use this function for loading triples from file on multiple threads
- Using the DB.DBA.RDF_LOAD_RDFXML_MT() function
    - Note: Use this function for loading large resources when transactional integrity is not important (loading of a single resource may take more than one transaction)
- Using the DB.DBA.RDF_TTL2HASH() function
    - Note: use this function to get dictionary of triples in 'long valmode'.
- Using the DB.DBA.RDF_LOAD_RDFXML() function
    - For loading RDF/XML, the best way is to split the data to be loaded into multiple streams and load these in parallel using this function.

See more details for loading Performance Tuning specifics.

16.9.2. SPARQL endpoint REST API

With POST can be accomplished SPARQL Insert/Update etc.

The result is in the rdf_quad.

With GET Methods you can get the triples which are saved.

Examples:

Example 1:

Create a DAV collection xx for user demo with password demo.

Execute the following command:

```
curl -i -d "INSERT {<http://demo.openlinksw.com/DAV/home/demo_about.rdf>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://rdfs.org/sioc/ns#User>}" -u "demo:demo"
-H "Content-Type: application/sparql-query" http://example.com/DAV/xx/yy
```

The response should be:

```
HTTP/1.1 201 Created
Server: Virtuoso/05.00.3023 (Win32) i686-generic-win-32  VDB
Connection: Keep-Alive
Content-Type: text/html; charset=ISO-8859-1
Date: Fri, 28 Dec 2007 12:50:12 GMT
Accept-Ranges: bytes
MS-Author-Via: SPARQL
Content-Length: 0
```

The result in the DAV/xx location will be a new WebDAV resource with name "yy" containing the inserted RDF:

```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
<rdf:Description
rdf:about="http://demo.openlinksw.com/DAV/home/demo_about.rdf">
<ns0pred:type xmlns:ns0pred="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
rdf:resource="http://rdfs.org/sioc/ns#User"/>
</rdf:Description>
</rdf:RDF>
```

Example 2:

Create a DAV collection, for ex. with name "test" for user (for ex. demo).

Execute the following command:

```
curl -i -d "INSERT IN GRAPH <http://mygraph.com>
{ <http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#this>
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://rdfs.org/sioc/ns#User> .
  <http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#this>
  <http://www.w3.org/2000/01/rdf-schema#label>
  <Kingsley Uyi Idehen> .
  <http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#this>
  <http://rdfs.org/sioc/ns#creator_of>

<http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%20%5
  } " -u "demo:demo" -H "Content-Type: application/sparql-query" http://example.com/DAV/home/demo/test/my
```

As result the response will be:

```
HTTP/1.1 201 Created
Server: Virtuoso/05.00.3023 (Win32) i686-generic-win-32  VDB
Connection: Keep-Alive
```

```
Content-Type: text/html; charset=ISO-8859-1
Date: Thu, 20 Dec 2007 16:25:25 GMT
Accept-Ranges: bytes
MS-Author-Via: SPARQL
Content-Length: 0
```

Now let's check the inserted triples. Go to the sparql endpoint, i.e. http://example.com/sparql and:

1. Enter for Default Graph URI:

   ```
   http://mygraph.com
   ```
2. Enter in the Query area:

   ```
   SELECT * WHERE {?s ?p ?o}
   ```
3. Click the button "Run Query"
4. As result will be shown the inserted triples:

   ```
   s                                                                    p
   http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#this   http://www.w3.org/1999/02
   http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#this   http://www.w3.org/2000/01
   http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#this   http://rdfs.org/sioc/ns#c
   ```

16.9.3. HTTP PUT using Content-Type: application/rdf+xml

The URI in a PUT request identifies the entity enclosed with the request. Therefore using HTTP PUT is a more useful and meaningful command than using POST (which is more about submitting data to a script).

Example:

Suppose there is myfoaf.rdf file with the following content:

```
<rdf:RDF xmlns="http://www.example/jose/foaf.rdf#"
    xmlns:foaf="http://xmlns.com/foaf/0.1/"
    xmlns:log="http://www.w3.org/2000/10/swap/log#"
    xmlns:myfoaf="http://www.example/jose/foaf.rdf#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

    <foaf:Person rdf:about="http://www.example/jose/foaf.rdf#jose">
        <foaf:homepage rdf:resource="http://www.example/jose/"/>
        <foaf:knows rdf:resource="http://www.example/jose/foaf.rdf#juan"/>
        <foaf:name>Jose Jimen~ez</foaf:name>
        <foaf:nick>Jo</foaf:nick>
        <foaf:workplaceHomepage rdf:resource="http://www.corp.example/"/>
    </foaf:Person>

    <foaf:Person rdf:about="http://www.example/jose/foaf.rdf#juan">
        <foaf:mbox rdf:resource="mailto:juan@mail.example"/>
    </foaf:Person>

    <foaf:Person rdf:about="http://www.example/jose/foaf.rdf#julia">
        <foaf:mbox rdf:resource="mailto:julia@mail.example"/>
    </foaf:Person>

    <rdf:Description rdf:about="http://www.example/jose/foaf.rdf#kendall">
        <foaf:knows rdf:resource="http://www.example/jose/foaf.rdf#edd"/>
    </rdf:Description>
</rdf:RDF>
```

Now let's upload the myfoaf.rdf file to destination server demo.openlinksw.com for user demo:

```
curl -T myfoaf.rdf http://demo.openlinksw.com/DAV/home/demo/rdf_sink/myfoaf.rdf -u demo:demo
```

As result the response should be:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML>
<HEAD>
  <TITLE>201 Created</TITLE>
```

```
</HEAD>
<BODY>
  <H1>Created</H1>
  Resource /DAV/home/demo/rdf_sink/ myfoaf.rdf has been created.
</BODY>
</HTML>
```

Then you can execute:

```
curl -F "query=SELECT DISTINCT ?p FROM <http://demo.openlinksw.com/DAV/home/demo/rdf_sink/> WHERE {?s ?p
```

The result should be:

```
<?xml version="1.0" ?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instan
 <head>
  <variable name="p"/>
 </head>
 <results distinct="false" ordered="true">
  <result>
   <binding name="p"><uri>http://www.w3.org/1999/02/22-rdf-syntax-ns#type</uri></binding>
  </result>
  <result>
   <binding name="p"><uri>http://xmlns.com/foaf/0.1/nick</uri></binding>
  </result>
  <result>
   <binding name="p"><uri>http://xmlns.com/foaf/0.1/name</uri></binding>
  </result>
  <result>
   <binding name="p"><uri>http://xmlns.com/foaf/0.1/homepage</uri></binding>
  </result>
  <result>
   <binding name="p"><uri>http://xmlns.com/foaf/0.1/knows</uri></binding>
  </result>
  <result>
   <binding name="p"><uri>http://xmlns.com/foaf/0.1/workplaceHomepage</uri></binding>
  </result>
  <result>
   <binding name="p"><uri>http://xmlns.com/foaf/0.1/mbox</uri></binding>
  </result>
 </results>
</sparql>
```

Other examples with curl:

```
curl -F "query=SELECT distinct ?Concept FROM <http://dbpedia.org> WHERE {?s a ?Concept} limit 10" http://

curl -F "query=SELECT distinct ?Concept FROM <http://example.com/dataspace/person/kidehen> WHERE {?s a ?C

curl -F "query=SELECT distinct ?Concept FROM <http://data.openlinksw.com/oplweb/product_family/virtuoso>

curl -F "query=SELECT distinct ?Concept FROM <http://openlinksw.com/dataspace/organization/openlink> WHER
```

16.9.4. SPARQL Insert using LOAD

SPARQL INSERT operation can be done using the LOAD features:

```
SPARQL INSERT INTO <..> { .... } [[FROM ...] { ... }]

SPARQL LOAD <x> [INTO <y>]

-- <ResourceURL> will be the Graph IRI of the loaded data:
SPARQL LOAD <ResourceURL>
```

Examples:

1. Load from ISQL:

```
SPARQL insert in graph <http://mygraph.com>
{
  <http://example.com/dataspace/Kingsley#this>
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://rdfs.org/sioc/ns#User> .

  <http://example.com/dataspace/Kingsley#this>
  <http://rdfs.org/sioc/ns#id>
  <Kingsley> .

  <http://example.com/dataspace/Caroline#this>
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://rdfs.org/sioc/ns#User> .

 <http://example.com/dataspace/Caroline#this>
  <http://rdfs.org/sioc/ns#id>
  <Caroline> .

   <http://example.com/dataspace/Matt#this>
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://rdfs.org/sioc/ns#User> .

  <http://example.com/dataspace/Matt#this>
  <http://rdfs.org/sioc/ns#id>
  <Matt> .

   <http://example.com/dataspace/demo#this>
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://rdfs.org/sioc/ns#User> .

  <http://example.com/dataspace/demo#this>
  <http://rdfs.org/sioc/ns#id>
  <demo> .};
```

2. Load from .rq file:

   a. Create DAV collection which is visible to public, for ex: http://example.com/DAV/tmp
   b. Upload to the DAV collection the following file for ex. with name listall.rq and with the following content:

```
SPARQL
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX sioc: <http://rdfs.org/sioc/ns#>
SELECT ?x ?p ?o
FROM <http://mygraph.com>
WHERE
  {
    ?x rdf:type sioc:User .
    ?x ?p ?o.
    ?x sioc:id ?id .
    FILTER REGEX(str(?id), "^King")
  }
ORDER BY ?x
```

   c. Execute from ISQL the following command:

```
SQL>SPARQL
load bif:concat ("http://", bif:registry_get("URIQADefaultHost"), "/DAV/tmp/listall.rq") ir
```

   As result should be shown:

```
callret-0
VARCHAR
_______________________________________________________________________________

Load <http://example.com/DAV/tmp/listall.rq> into graph <http://myNewGraph.com> -- done

1 Rows. -- 321 msec.
```

3. Load from Resource URL:

```
SQL> SPARQL LOAD <http://www.w3.org/People/Berners-Lee/card#i>;
callret-0
VARCHAR
```

```
Load <http://www.w3.org/People/Berners-Lee/card#i> into graph <http://www.w3.org/People/Berners-Le

1 Rows. -- 703 msec.
SQL>
```

16.9.5. SPARQL Insert via /sparql endpoint

SPARQL INSERT operation can be sent to a web service endpoint as a single statement and executed in sequence.

Example:

Using the Virtuoso ISQL tool or using the /sparql UI at http://host:port/sparql, execute the following:

1. Insert into graph http://BookStore.com 3 triples:

```
SQL>SPARQL insert in graph <http://BookStore.com>
{ <http://www.dajobe.org/foaf.rdf#i> <http://purl.org/dc/elements/1.1/date> <1999-04-01T00:00:00>
  <http://www.w3.org/People/Berners-Lee/card#i> <http://purl.org/dc/elements/1.1/date> <1998-05-03
  <http://www.w3.org/People/Connolly/#me> <http://purl.org/dc/elements/1.1/date> <2001-02-08T00:00
```

2. As result will be shown the message:

```
SQL>Insert into <http://BookStore.com>
3 triples -- done
```

3. Next we will select all triples from the graph http://BookStore.com:

```
SQL>SPARQL SELECT * FROM <http://BookStore.com> WHERE {?s ?p ?o};
```

4. As result will be shown:

```
s                                                 p                                          o
VARCHAR                                           VARCHAR                                    VARCHAR
_______________________________________________________________________________________________

http://www.w3.org/People/Berners-Lee/card#i       http://purl.org/dc/elements/1.1/date       1998-05-03T
http://www.w3.org/People/Connolly/#me             http://purl.org/dc/elements/1.1/date       2001-02-08T
http://www.dajobe.org/foaf.rdf#i                  http://purl.org/dc/elements/1.1/date       1999-04-01T

3 Rows. -- 0 msec.
```

5. Now let's insert into graph another http://NewBookStore.com graph's values:

```
SQL>SPARQL
PREFIX dc:  <http://purl.org/dc/elements/1.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
INSERT INTO GRAPH <http://NewBookStore.com> { ?book ?p ?v }
WHERE
  { GRAPH  <http://BookStore.com>
   { ?book dc:date ?date
     FILTER ( xsd:dateTime(?date) < xsd:dateTime("2000-01-01T00:00:00")).
     ?book ?p ?v.
   }
  };
```

6. As result will be shown:

```
callret-0
VARCHAR
_______________________________________________________________________________

Insert into <http://NewBookStore.com>, 2 triples -- done
```

7. Finally we will check the triples from the graph NewBookStore.com:

```
SQL> SPARQL
SELECT *
FROM <http://NewBookStore.com>
WHERE {?s ?p ?o};
```

8. As result will be shown:

```
s                                                p                                          o
VARCHAR                                          VARCHAR                                    VARCHAR
_______________________________________________________________________________________________

http://www.w3.org/People/Berners-Lee/card#i      http://purl.org/dc/elements/1.1/date       1998-05-03T00
http://www.dajobe.org/foaf.rdf#i                 http://purl.org/dc/elements/1.1/date       1999-04-01T00

2 Rows. -- 10 msec.
```

16.9.6. SPARQL Insert via SPARQL endpoint REST API and ODS wiki

With HTTP Post and ODS wiki can be written an rdf document and respectively to be performed over it INSERT/UPDATE action.

You can write to a file using SIOC terms for ODS-Wiki

You can check with sparql the inserted / updated triples in the Quad Store.

Example:

Suppose there is ODS user test3 with ODS password 1, which has testWiki wiki instance.

Execute the following:

```
curl -i -d "INSERT {<http://example.com/dataspace/test3/wiki/testWiki> <http://atomowl.org/ontologies/ato
```

As result we should have 2 files created:

- In the user DAV folder "DAV/home/test3/wiki/testWiki/" will be created a file "MyTest" with type "application/sparql-query". You can view the content of this file from the Conductor UI or from the user's Briefcase UI, path "DAV/home/test3/wiki/testWiki". Its content will be:

```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:rdfs="http://www.w3.org/200
<rdf:Description rdf:about="http://example.com/dataspace/test3/wiki/testWiki"><ns0pred:entry xmlns
<rdf:Description rdf:about="http://example.com/dataspace/test3/wiki/testWiki/MyTest"><ns0pred:labe
<rdf:Description rdf:about="http://example.com/dataspace/test3/wiki/testWiki/MyTest"><ns0pred:type
<rdf:Description rdf:about="http://example.com/dataspace/test3/wiki/testWiki/MyTest"><ns0pred:type
<rdf:Description rdf:about="http://example.com/dataspace/test3/wiki/testWiki/MyTest"><ns0pred:type
<rdf:Description rdf:about="http://example.com/dataspace/test3/wiki/testWiki/MyTest"><ns0pred:has_
<rdf:Description rdf:about="http://example.com/dataspace/test3/wiki/testWiki"><ns0pred:container_o
<rdf:Description rdf:about="http://example.com/dataspace/test3/wiki/testWiki"><ns0pred:contains xm
<rdf:Description rdf:about="http://example.com/dataspace/test3/wiki/testWiki/MyTest"><ns0pred:cont
<rdf:Description rdf:about="http://example.com/dataspace/test3/wiki/testWiki/MyTest"><ns0pred:topi
<rdf:Description rdf:about="http://example.com/dataspace/test3/wiki/testWiki/MyTest"><ns0pred:sour
</rdf:RDF>
```

- To the user's wiki instance will be added a new WikiWord "MyTest" with content the value of the SIOC term attribute "content":

```
<http://example.com/dataspace/test3/wiki/testWiki/MyTest> <http://rdfs.org/sioc/ns#content> <test>
i.e. the content will be "test".
```

Now let's check what data was inserted in the Quad Store:

1. Go to the sparql endpoint, i.e. for ex. to http://example.com/sparql
2. Enter for Default Graph URI:

```
http://example.com/DAV/home/test3/wiki/testWiki/MyTest
```

3. Enter for Query text:

```
SELECT * WHERE {?s ?p ?o}
```

4. Click the "Run Query" button.
5. As result will be shown the inserted triples:

```
s                                                                p
http://example.com/dataspace/test3/wiki/testWiki                 http://rdfs.org/sioc/ns#container_of
```

```
http://example.com/dataspace/test3/wiki/testWiki            http://atomowl.org/ontologies/atomrdf#
http://example.com/dataspace/test3/wiki/testWiki            http://atomowl.org/ontologies/atomrdf#
http://example.com/dataspace/test3/wiki/testWiki/MyTest  http://www.w3.org/1999/02/22-rdf-syntax-n
http://example.com/dataspace/test3/wiki/testWiki/MyTest  http://www.w3.org/1999/02/22-rdf-syntax-n
http://example.com/dataspace/test3/wiki/testWiki/MyTest  http://www.w3.org/1999/02/22-rdf-syntax-n
http://example.com/dataspace/test3/wiki/testWiki/MyTest  http://www.w3.org/2000/01/rdf-schema#labe
http://example.com/dataspace/test3/wiki/testWiki/MyTest  http://rdfs.org/sioc/ns#has_container
http://example.com/dataspace/test3/wiki/testWiki/MyTest  http://rdfs.org/sioc/ns#content
http://example.com/dataspace/test3/wiki/testWiki/MyTest  http://rdfs.org/sioc/ns#topic
http://example.com/dataspace/test3/wiki/testWiki/MyTest  http://atomowl.org/ontologies/atomrdf#sou
```

16.9.7. Using WebDAV

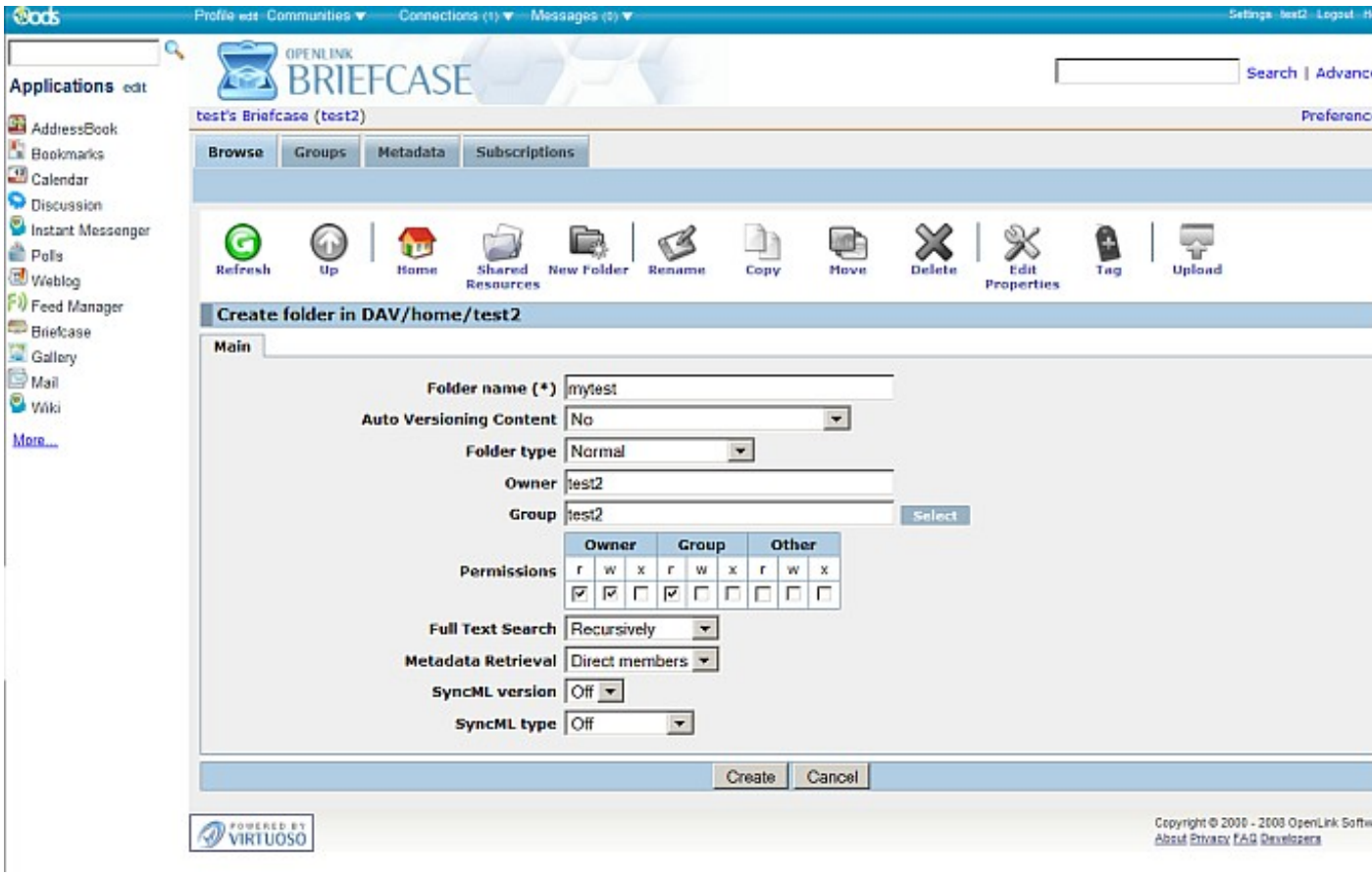
Example using WebDAV (mount folder to DAV and dump; if this is the rdf_sink the Quad Store is updated automatically, or you can load from DAV manually to quad store)

Example:

Example 1: Using ODS Briefcase

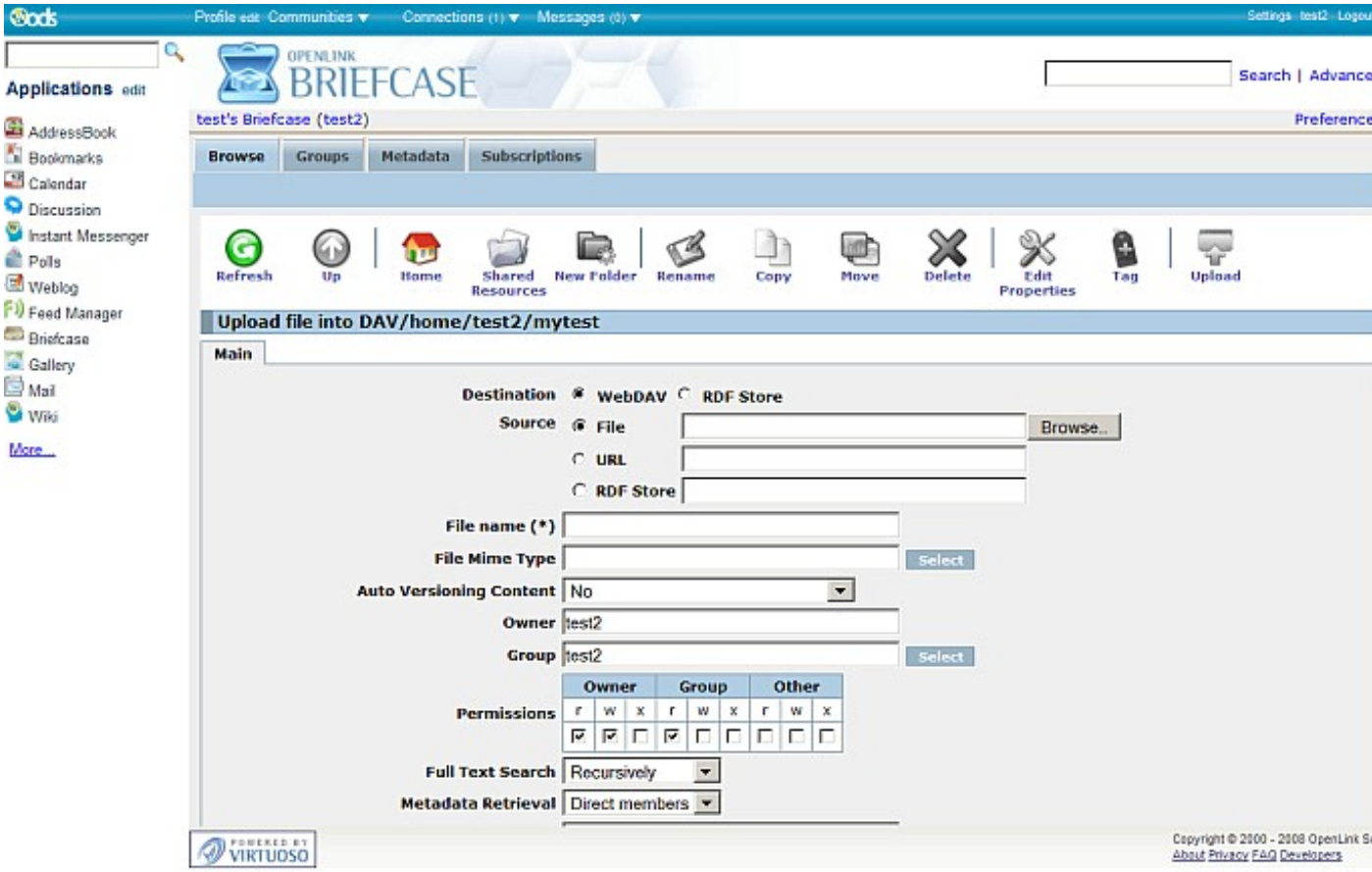
1. Go to your ods location, for ex. http://example.com/ods
2. Register user, for ex. user test1
3. Login if not already in ods
4. Go to ODS -> Briefcase
5. Go to ODS -> Briefcase
6. Click the "New folder" icon from the Main Briefcase horizontal navigation
7. Enter for name for ex. "mytest" and click the "Create" button.

Figure 16.81. Using Briefcase UI



8. Go to folder "mytest" and click the click the "Upload" icon from the Main Briefcase horizontal navigation
9. Enter for name for ex. "mytest" and click the "Create" button.

Figure 16.82. Using Briefcase UI



10. In the shown form set:

    - Destination: RDF Store
    - RDF graph name for ex. with the value: http://example.com/DAV/home/test2/mytest/
    - Select URL or File. For ex. you can select the following file with name jose.rdf:

```
<rdf:RDF xmlns="http://www.example/jose/foaf.rdf#"
    xmlns:foaf="http://xmlns.com/foaf/0.1/"
    xmlns:log="http://www.w3.org/2000/10/swap/log#"
    xmlns:myfoaf="http://www.example/jose/foaf.rdf#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

    <foaf:Person rdf:about="http://www.example/jose/foaf.rdf#jose">
        <foaf:homepage rdf:resource="http://www.example/jose/"/>
        <foaf:knows rdf:resource="http://www.example/jose/foaf.rdf#juan"/>
        <foaf:name>Jose Jimen~ez</foaf:name>
        <foaf:nick>Jo</foaf:nick>
        <foaf:workplaceHomepage rdf:resource="http://www.corp.example/"/>
    </foaf:Person>

    <foaf:Person rdf:about="http://www.example/jose/foaf.rdf#juan">
        <foaf:mbox rdf:resource="mailto:juan@mail.example"/>
    </foaf:Person>

    <foaf:Person rdf:about="http://www.example/jose/foaf.rdf#julia">
        <foaf:mbox rdf:resource="mailto:julia@mail.example"/>
    </foaf:Person>

    <rdf:Description rdf:about="http://www.example/jose/foaf.rdf#kendall">

        <foaf:knows rdf:resource="http://www.example/jose/foaf.rdf#edd"/>
    </rdf:Description>
</rdf:RDF>
```

11. You can also perform the steps from above by uploading the file in the rdf_sink folder i.e. in Briefcase it will be with this path: DAV/home/test2/rdf_sink and respectively the "RDF graph name" will have this value: http://host:port/DAV/home/username/rdf_sink/

Execute from ISQL or from the SPARQL endpoint the following query:

```
SELECT * FROM <http://example.com/DAV/home/test2/mytest/>
WHERE {?s ?p ?o}
```

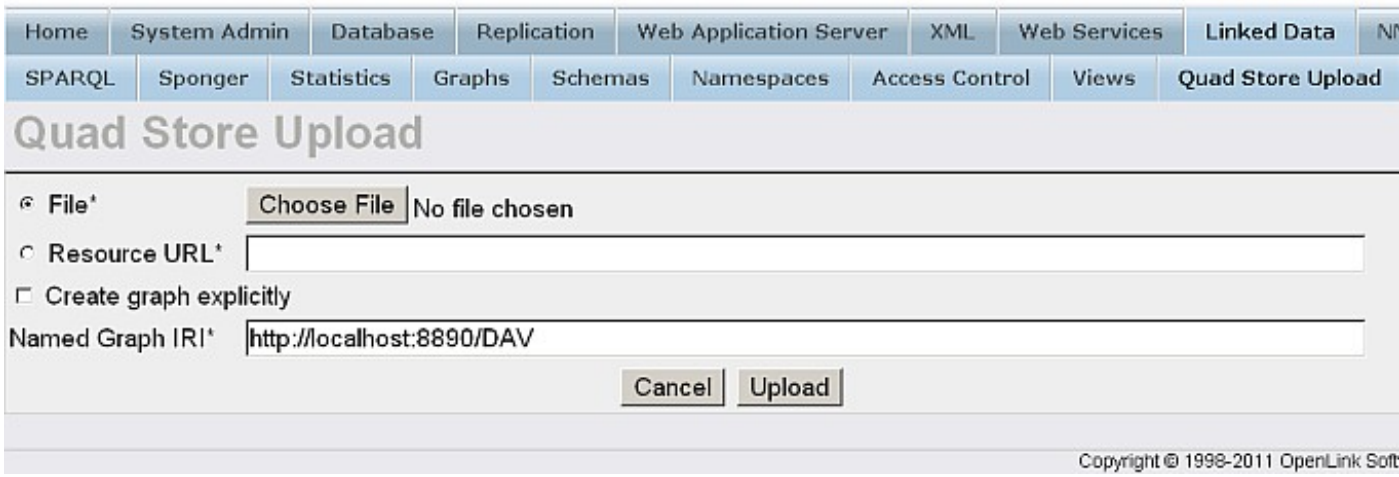
As result should be shown:

```
s                                        p                                                o
http://www.example/jose/foaf.rdf#jose    http://www.w3.org/1999/02/22-rdf-syntax-ns#type  http://xmlns.
http://www.example/jose/foaf.rdf#jose    http://xmlns.com/foaf/0.1/nick                   Jo
http://www.example/jose/foaf.rdf#jose    http://xmlns.com/foaf/0.1/name                   Jose Jimen~ez
http://www.example/jose/foaf.rdf#jose    http://xmlns.com/foaf/0.1/knows                  http://www.ex
http://www.example/jose/foaf.rdf#jose    http://xmlns.com/foaf/0.1/homepage               http://www.ex
http://www.example/jose/foaf.rdf#jose    http://xmlns.com/foaf/0.1/workplaceHomepage      http://www.co
http://www.example/jose/foaf.rdf#kendall http://xmlns.com/foaf/0.1/knows                  http://www.ex
http://www.example/jose/foaf.rdf#julia   http://www.w3.org/1999/02/22-rdf-syntax-ns#type  http://xmlns.
http://www.example/jose/foaf.rdf#julia   http://xmlns.com/foaf/0.1/mbox                   mailto:julia@
http://www.example/jose/foaf.rdf#juan    http://www.w3.org/1999/02/22-rdf-syntax-ns#type  http://xmlns.
http://www.example/jose/foaf.rdf#juan    http://xmlns.com/foaf/0.1/mbox                   mailto:juan@m
```

Example 2: Using Conductor UI

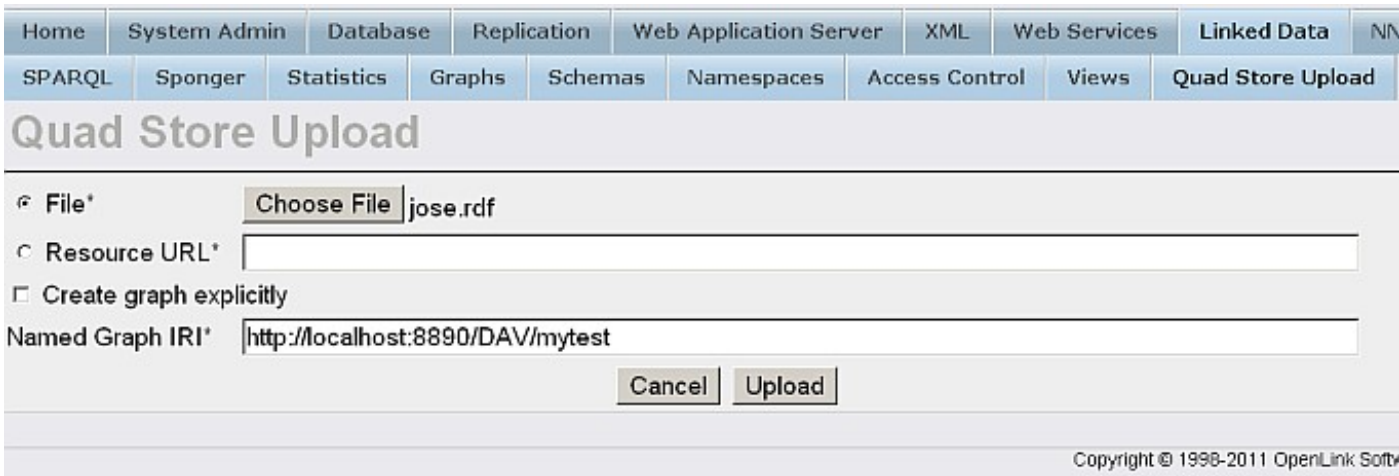
1. Go to Conductor UI, for ex. at http://example.com/conductor
2. Login as dba user
3. Go to Linked Data -> Quad Store Upload

Figure 16.83. Quad Store Upload



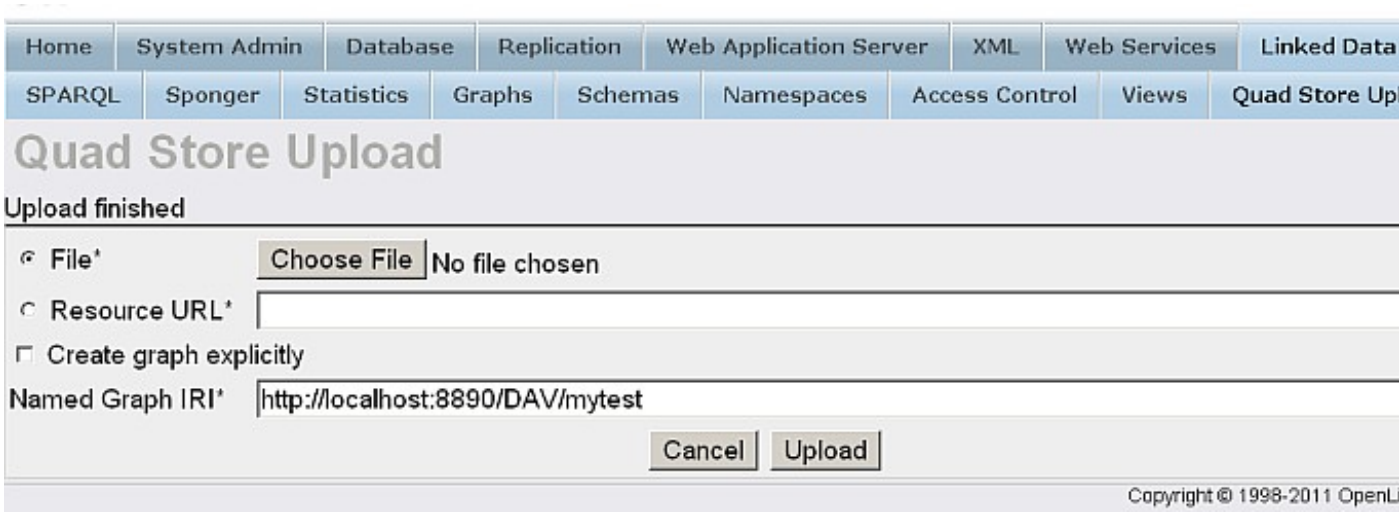
4. In the shown form click the "Browse" button in order to select a file, for ex. the file jose.rdf and set the "RDF IRI*"

Figure 16.84. Quad Store Upload



5. Click the "Upload" button.

Figure 16.85. Quad Store Upload



16.9.8. Using Virtuoso Crawler

Using Virtuoso Crawler (which includes the Sponger options so you crawl non-RDF but get RDF and this can go to the Quad Store).

Example:

1. Go to Conductor UI. For ex. at http://example.com/conductor :

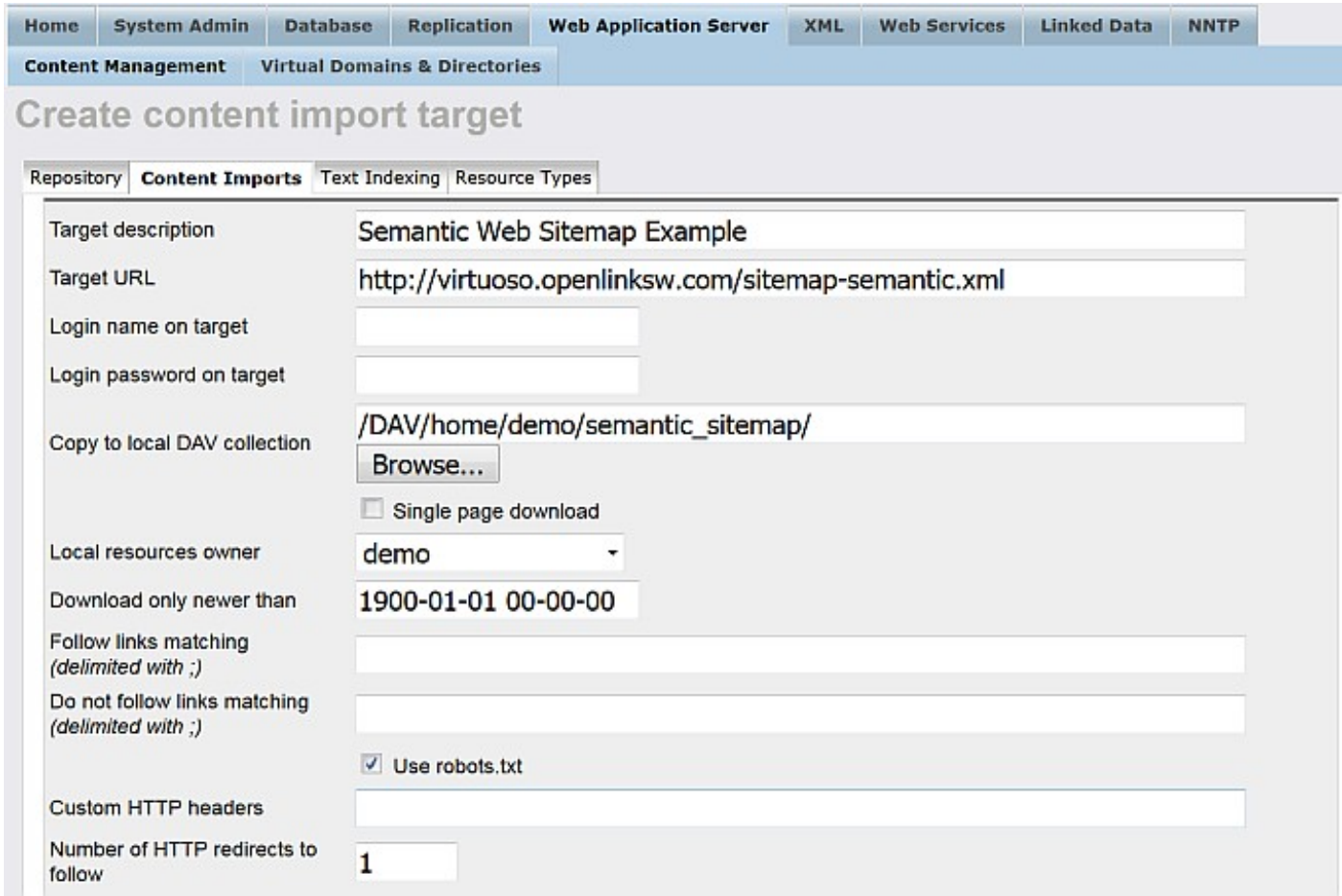
Figure 16.86. Using Virtuoso Crawler

2. Enter admin user credentials:

Figure 16.87. Using Virtuoso Crawler

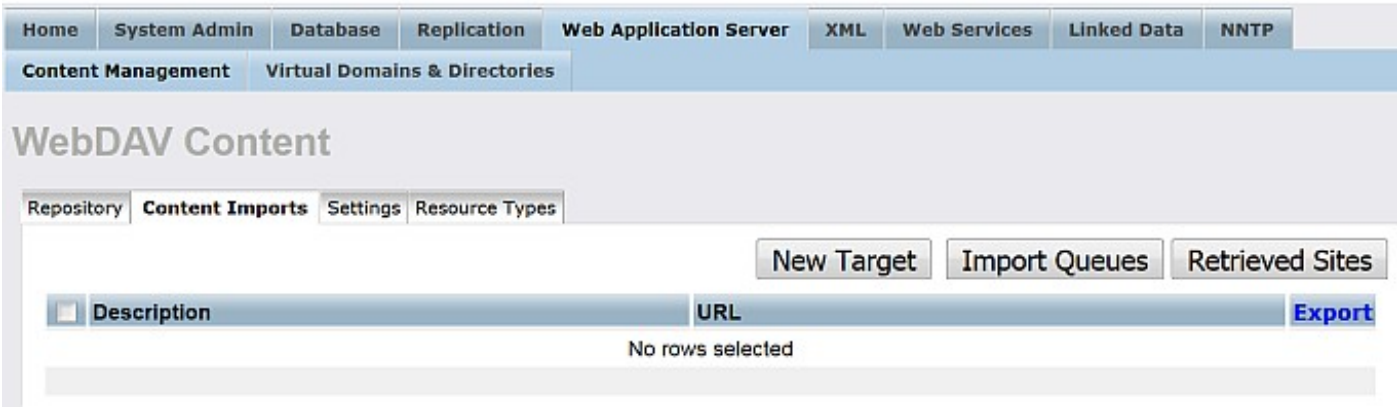
3. Go to tab Web Application Server:

Figure 16.88. Using Virtuoso Crawler



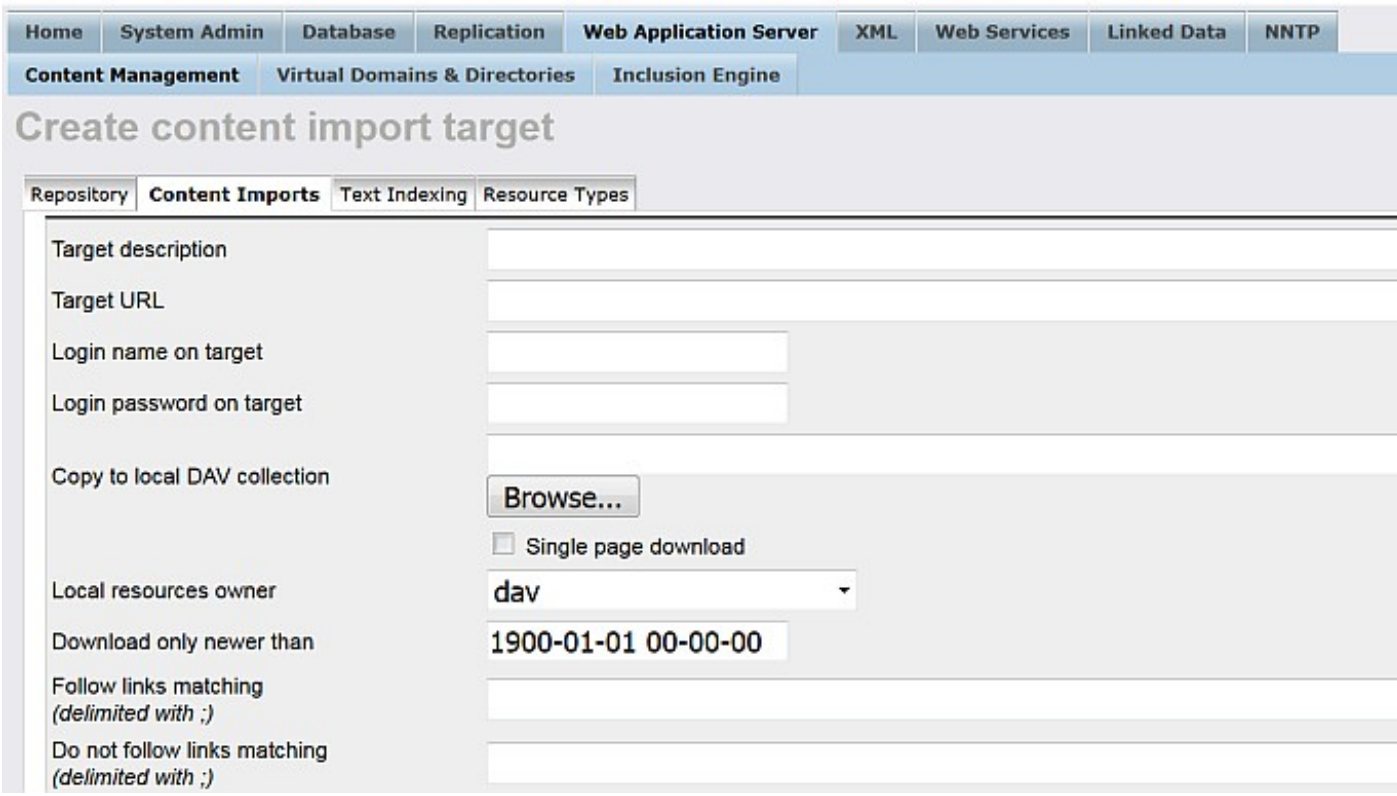
4. Go to tab Content Imports:

Figure 16.89. Using Virtuoso Crawler



5. Click the "New Target" button:

Figure 16.90. Using Virtuoso Crawler



6. In the shown form set respectively:

   a. "Target description": Tim Berners-Lee's electronic Business Card
   b. "Target URL": http://www.w3.org/People/Berners-Lee/ ;
   c. "Copy to local DAV collection " for ex.: /DAV/home/demo/my-crawling/ ;
   d. Choose from the list "Local resources owner": demo ;
   e. Leave checked by default the check-box "Store documents locally". -- Note: if "Store document locally" is not checked, then in this case no documents will be save as DAV resource and the specified DAV folder from above will not be used ;
   f. Check the check-box with label "Store metadata" ;
   g. Specify which cartridges to be involved by hatching their check-box ;
   h. Note: when selected "Convert Link", then all HREFs in the local stored content will be relative.

Figure 16.91. Using Virtuoso Crawler

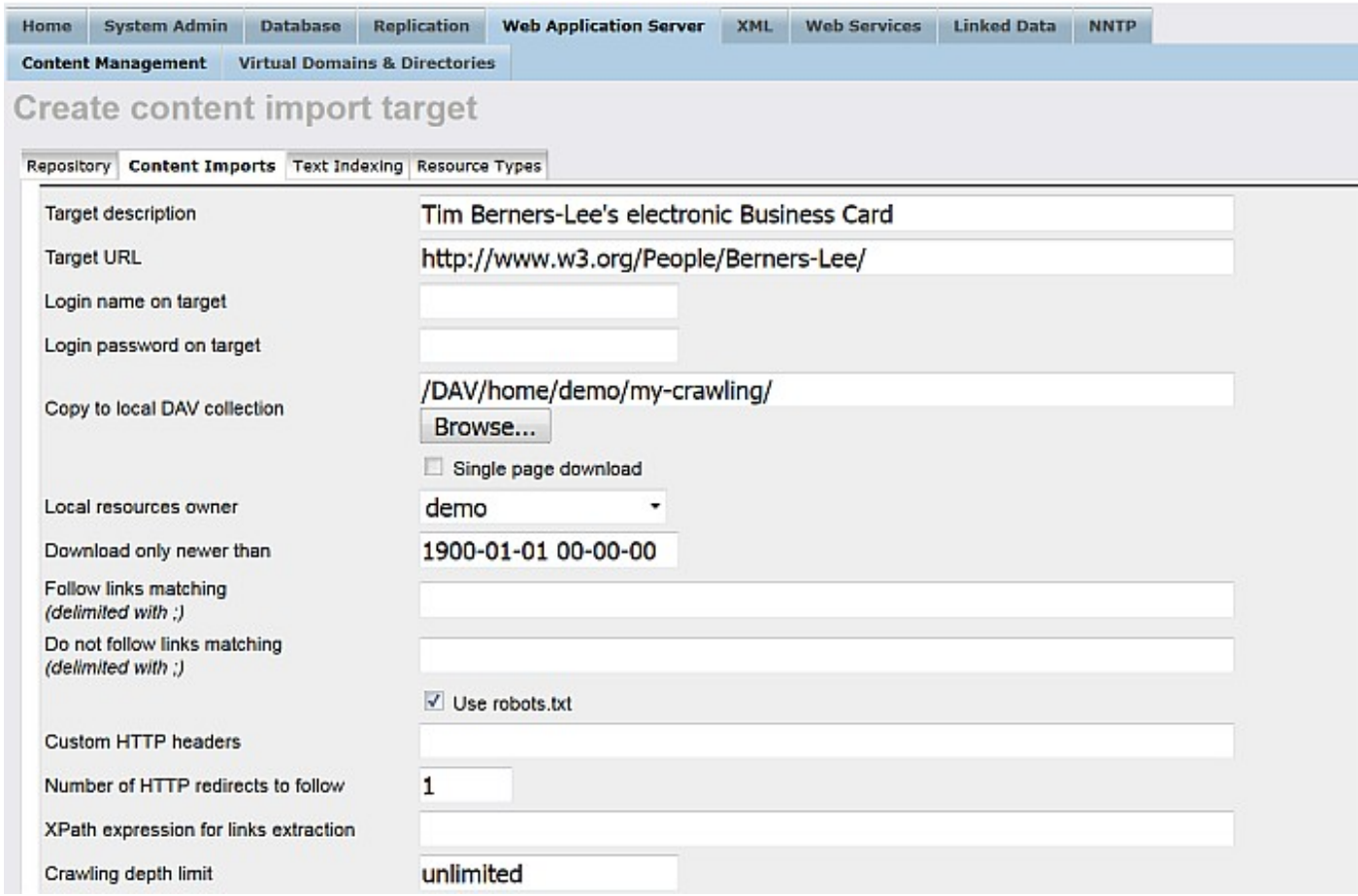
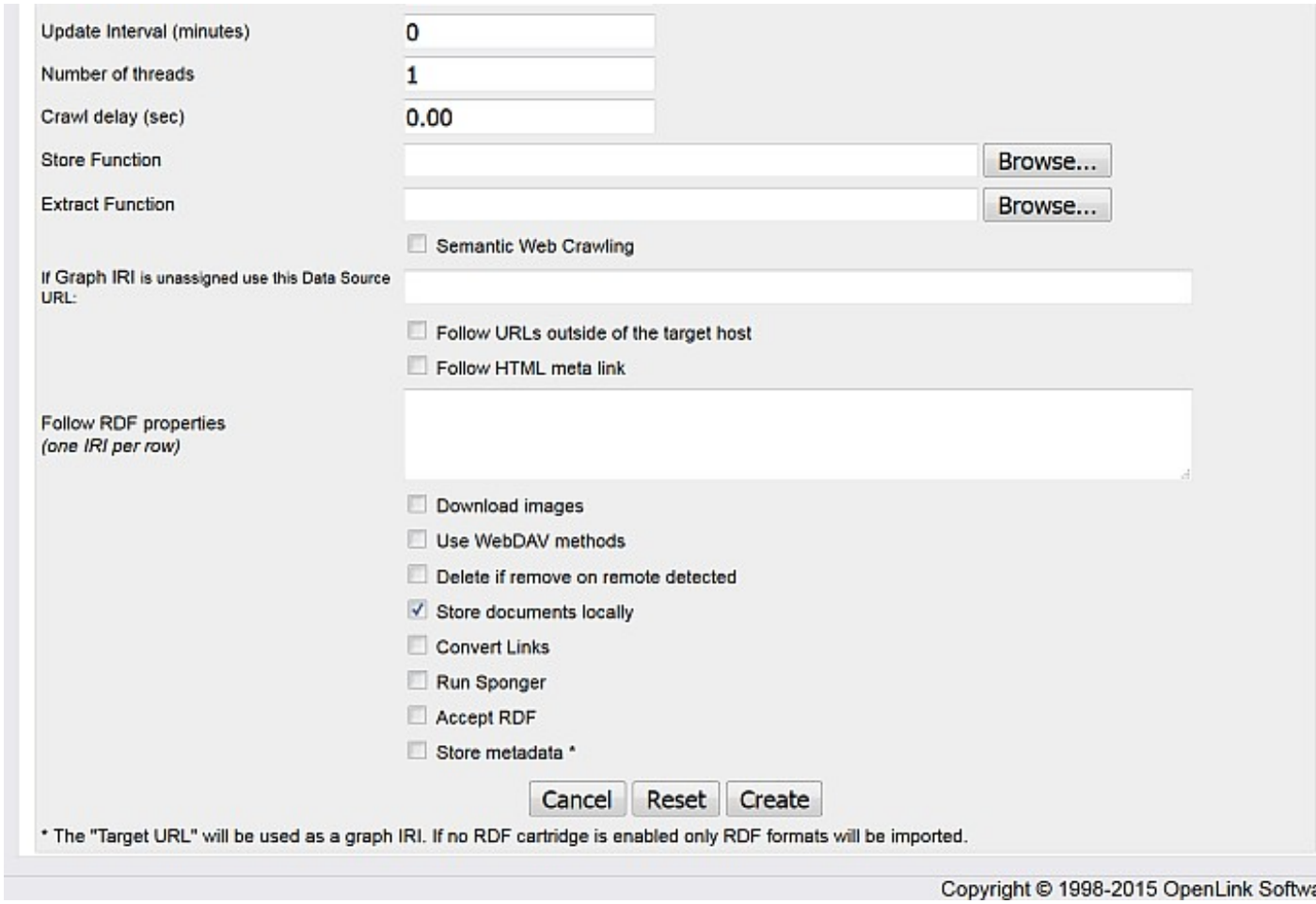
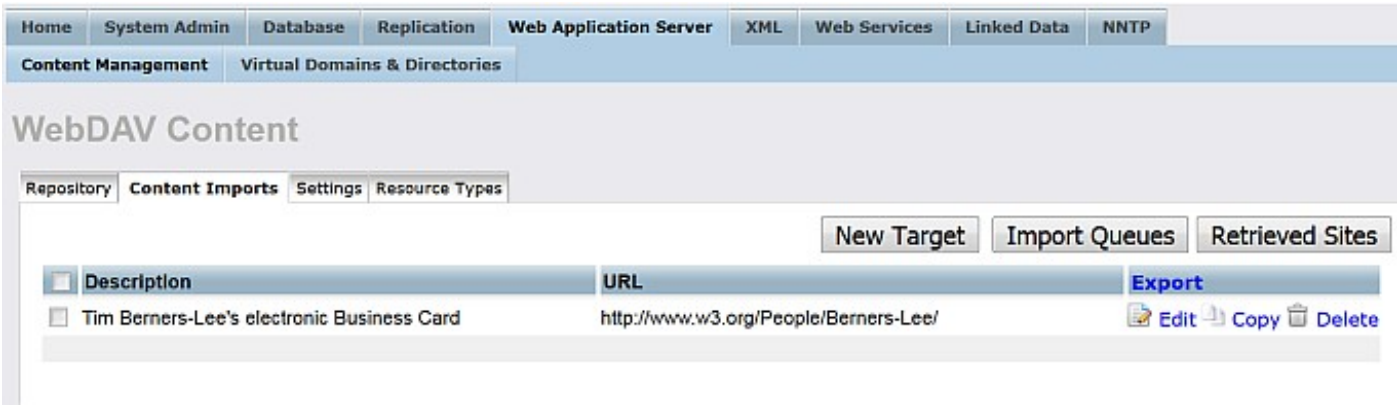


Figure 16.92. Using Virtuoso Crawler



7. Click the button "Create":

Figure 16.93. Using Virtuoso Crawler

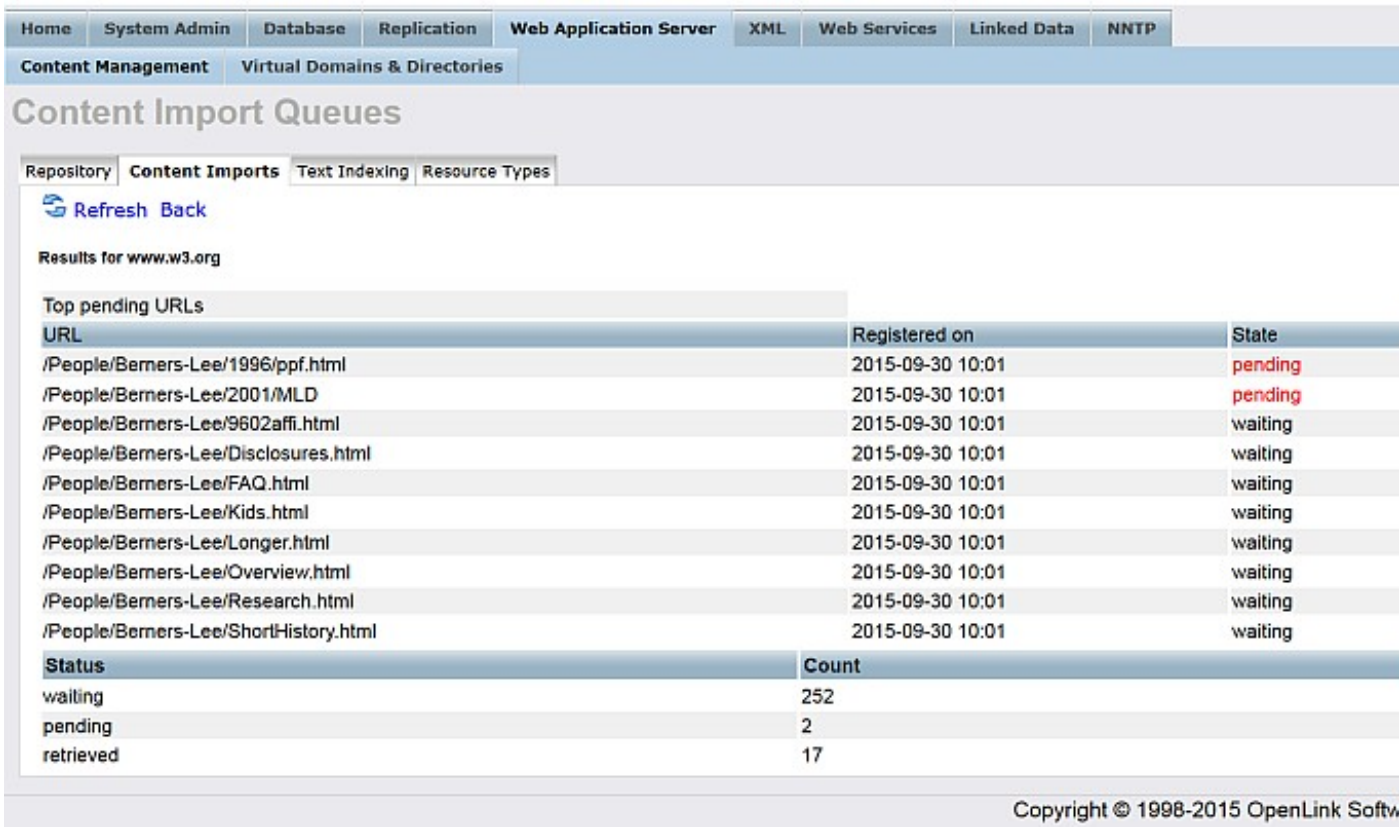


8. Click the button "Import Queues":

Figure 16.94. Using Virtuoso Crawler

9. For "Robot target" with label "Tim Berners-Lee's electronic Business Card" click "Run".
10. As result should be shown the number of the pages retrieved.

Figure 16.95. Using Virtuoso Crawler



Example: Use of schedular to interface Virtuoso Quad Store with PTSW using the following program:

```
create procedure PTSW_CRAWL ()
{
  declare xt, xp any;
  declare content, headers any;

  content := http_get ('http://pingthesemanticweb.com/export/', headers);
  xt := xtree_doc (content);
  xp := xpath_eval ('//rdfdocument/@url', xt, 0);
  foreach (any x in xp) do
    {
      x := cast (x as varchar);
      dbg_obj_print (x);
      {
        declare exit handler for sqlstate '*' {
          log_message (sprintf ('PTSW crawler can not load : %s', x));
        };
        sparql load ?:x into graph ?:x;
        update DB.DBA.SYS_HTTP_SPONGE set HS_LOCAL_IRI = x, HS_EXPIRATION = null WHERE HS_LOCAL_IRI = 'de
        commit work;
      }
    }
}
;

insert soft SYS_SCHEDULED_EVENT (SE_SQL, SE_START, SE_INTERVAL, SE_NAME)
        values ('DB.DBA.PTSW_CRAWL ()', cast (stringtime ('0:0') as DATETIME), 60, 'PTSW Crawling');
```

See Also:

Other Methods to Set Up the Content Crawler for RDF gathering.

16.9.9. Using SPARQL Query and Sponger (i.e. we Fetch the Network Resources in the FROM Clause or values for the graph-uri parameter in SPARQL protocol URLs)

Example:

Execute the following query:

```
SQL>SPARQL
SELECT ?id
FROM NAMED <http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%
 OPTION (get:soft "soft", get:method "GET")
WHERE { GRAPH ?g { ?id a ?o } }
limit 10;
```

As result will be shown the retrieved triples:

```
id
VARCHAR
_______________________________________________________________________________

http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#this
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%20%5B
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%20%5B
http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#this
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%20%5B
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%20%5B
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%20%5B
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%20%5B
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%20%5B
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%20%5B

10 Rows. -- 20 msec.
```

16.9.10. Using Virtuoso PL APIs

Basic Sponger Cartridge Example

In the example script we implement a basic mapper which maps a text/plain mime type to an imaginary ontology, which extends the class Document from FOAF with properties 'txt:UniqueWords' and 'txt:Chars', where the prefix 'txt:' we specify as 'urn:txt:v0.0:'.

```
use DB;

create procedure DB.DBA.RDF_LOAD_TXT_META
        (
         in graph_iri varchar,
         in new_origin_uri varchar,
         in dest varchar,
         inout ret_body any,
         inout aq any,
         inout ps any,
         inout ser_key any
         )
{
  declare words, chars int;
  declare vtb, arr, subj, ses, str any;
  declare ses any;
  -- if any error we just say nothing can be done
  declare exit handler for sqlstate '*'
    {
      return 0;
    };
  subj := coalesce (dest, new_origin_uri);
  vtb := vt_batch ();
  chars := length (ret_body);
  -- using the text index procedures we get a list of words
  vt_batch_feed (vtb, ret_body, 1);
  arr := vt_batch_strings_array (vtb);
  -- the list has 'word' and positions array , so we must divide by 2
  words := length (arr) / 2;
  ses := string_output ();
  -- we compose a N3 literal
  http (sprintf ('<%s> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/Docum
  http (sprintf ('<%s> <urn:txt:v0.0:UniqueWords> "%d" .\n', subj, words), ses);
  http (sprintf ('<%s> <urn:txt:v0.0:Chars> "%d" .\n', subj, chars), ses);
```

```
  str := string_output_string (ses);
  -- we push the N3 text into the local store
  DB.DBA.TTLP (str, new_origin_uri, subj);
  return 1;
}
;

--
DELETE FROM DB.DBA.SYS_RDF_MAPPERS WHERE RM_HOOK = 'DB.DBA.RDF_LOAD_TXT_META';

INSERT SOFT DB.DBA.SYS_RDF_MAPPERS (RM_PATTERN, RM_TYPE, RM_HOOK, RM_KEY, RM_DESCRIPTION)
    VALUES ('(text/plain)', 'MIME', 'DB.DBA.RDF_LOAD_TXT_META', null, 'Text Files (demo)');

-- here we set order to some large number so don't break existing mappers
update DB.DBA.SYS_RDF_MAPPERS set RM_ID = 2000 WHERE RM_HOOK = 'DB.DBA.RDF_LOAD_TXT_META';
```

1. Paste the whole of this code into Conductor's iSQL interface and execute it to define and register the cartridge.
2. Create a simple text document with a .txt extension. For ex. with name: summary.txt
3. The .txt file must now be made Web accessible. A simple way to do this is to expose it as a WebDAV resource using Virtuoso's built-in WebDAV support:
    a. Log in to Virtuoso's ODS Briefcase application;
    b. Navigate to your Public folder;
    c. Upload your text document, ensuring that the file extension is .txt, the MIME type is set to text/plain and the file permissions are rw-r--r--.
    d. As result the file would be Web accessible via the URL http://cname/DAV/home/username/Public/summary.txt .
    e. Note: you can also check our live demo .
4. To test the mapper we just use /sparql endpoint with option 'Retrieve remote RDF data for all missing source graphs' to execute (for ex.):

```
SELECT *
FROM <http://cname/DAV/home/username/Public/summary.txt>
WHERE {?s ?p ?o}
```

5. Click the "Run Query" button.
6. As result should be shown the found triples, for ex.:

```
s                                                    p
http://cname/DAV/home/username/Public/summary.txt  http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://cname/DAV/home/username/Public/summary.txt  urn:txt:v0.0:UniqueWords
http://cname/DAV/home/username/Public/summary.txt  urn:txt:v0.0:Chars
```

Important: Setting Sponger Permissions

In order to allow the Sponger to update the local RDF quad store with triples constituting the fetched Network Resource structured data, the role "SPARQL_SPONGE" must be granted to the account "SPARQL", i.e., to the owner account of /sparql web service endpoint. This should normally be the case. If not, you must manually grant this permission. As with most Virtuoso DBA tasks, the Conductor provides the simplest means of doing this.

See Also:

- The DB.DBA.RDF_LOAD_RDFXML function to parse the content of RDF/XML text.
- The DB.DBA.TTLP_MT function to parse TTL (TURTLE or N3 resource).
- The gz_file_open function to retrieve content of a gzipped file and example for loading gzipped N3 and Turtle files.

16.9.11. Using SIMILE RDF Bank API

Virtuoso implements the HTTP-based Semantic Bank API that enables client applications to post to its RDF Triple Store. This method offers an alternative to using Virtuoso/PL functions or WebDAV uploads as the triples-insertion mechanism.

Example:

From your machine go to Firefox->Tools->PiggyBank->My Semantic Bank Accounts

Add in the shown form:

- For bank: address: http://demo.openlinksw.com/bank
- For account id: demo
- For password: demo

Go to http://demo.openlinksw.com/ods

Log in as user demo, password: demo

Go to the Weblog tab from the main ODS Navigation

Click on weblog instance name, for ex. "demo's Weblog".

When the weblog home page is loaded, click Alt + P.

As result is shown the "My PiggyBank" page with all the collected information presented in items.

For several of the items add Tags from the form "Tag" shown for each of them.

As result should be shown the message "Last updated: [here goes the date value].

You can also click "Save" and "Publish" for these items.

Go to http://demo.openlinksw.com/sparql

Enter for the "Default Graph URI" field: http://simile.org/piggybank/demo

Enter for the "Query text" text-area:

```
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix sioc: <http://rdfs.org/sioc/ns#>
SELECT *
FROM <http://simile.org/piggybank/demo>
WHERE {?s ?p  ?o}
```

Click "Run Query".

As results are shown the found results.

16.9.12. Using RDF NET

Example:

Execute the following query:

```
SQL> SELECT DB.DBA.HTTP_RDF_NET ('sparql load
"http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com"
into graph <http://www.openlinksw.com/>');
```

As result should be shown:

```
callret
VARCHAR
_______________________________________________________

<?xml version="1.0" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:vcard="http://www.w3.org/2001/vcard-rdf/3.0#"
xmlns="http://example.org/book/" xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:ns="http://example.org/ns#">
<rdf:Description>
<callret-0>Load <http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com>
into graph <http://www.openlinksw.com/> -- done</callret-0>
```

```
</rdf:Description>
</rdf:RDF>

1 Rows. -- 1982 msec.
```

16.9.13. Using the RDF Proxy (Sponger) Service

Triples can be inserted also using the Sponger Proxy URI Service. For more information and examples see here .

16.10. RDFizer Middleware (Sponger)

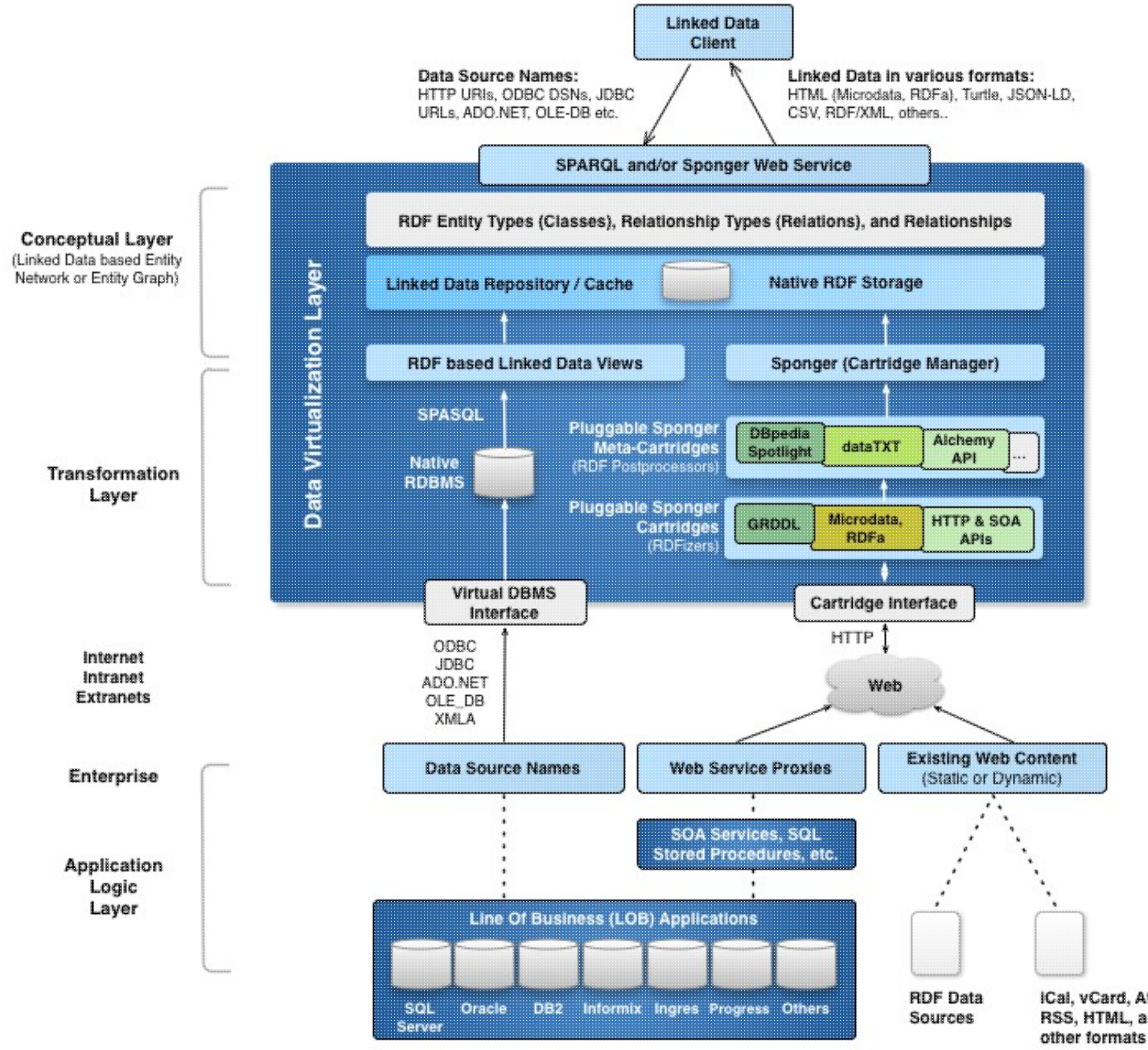
16.10.1. What Is The Sponger?

The Virtuoso Sponger is the Linked Data middleware component of Virtuoso that generates Linked Data from a variety of data sources, supporting a wide variety of data representation and serialization formats. The sponger is transparently integrated into Virtuoso's SPARQL Query Processor where it delivers URI de-referencing within SPARQL query patterns, across disparate data spaces. It also delivers configurable smart HTTP caching services. Optionally, it can be used by the Virtuoso Content Crawler to periodically populate and replenish data within the native RDF Quad Store.

The sponger is a fully fledged HTTP proxy service that is also directly accessible via SOAP or REST interfaces.

As depicted below, OpenLink's broad portfolio of Linked-Data-aware products supports a number of routes for creating or consuming Linked Data. The Sponger provides a key platform for developers to generate quality data meshes from unstructured or semi-structured data sources.

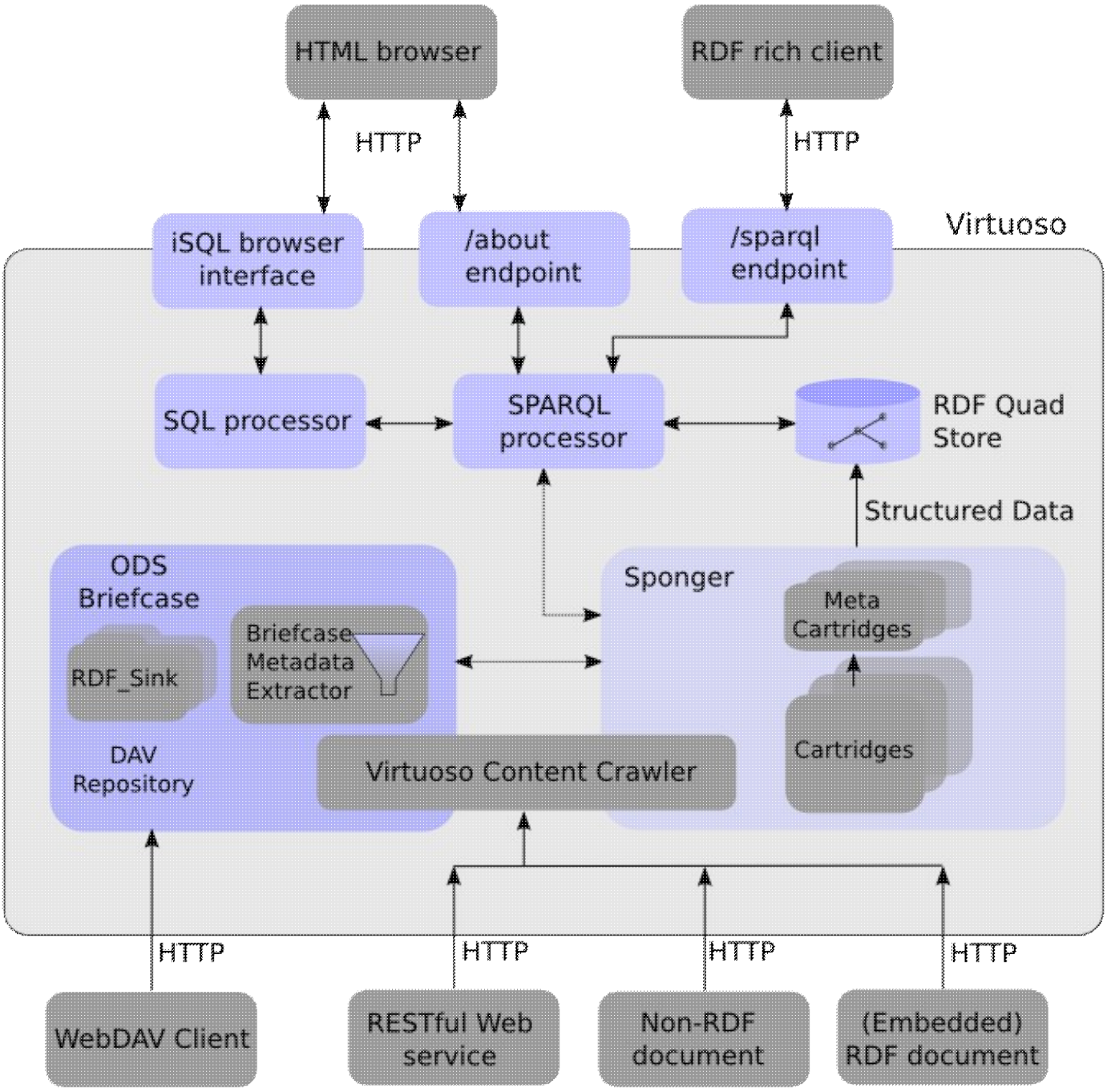
Figure 16.96. OpenLink Linked Data generation options

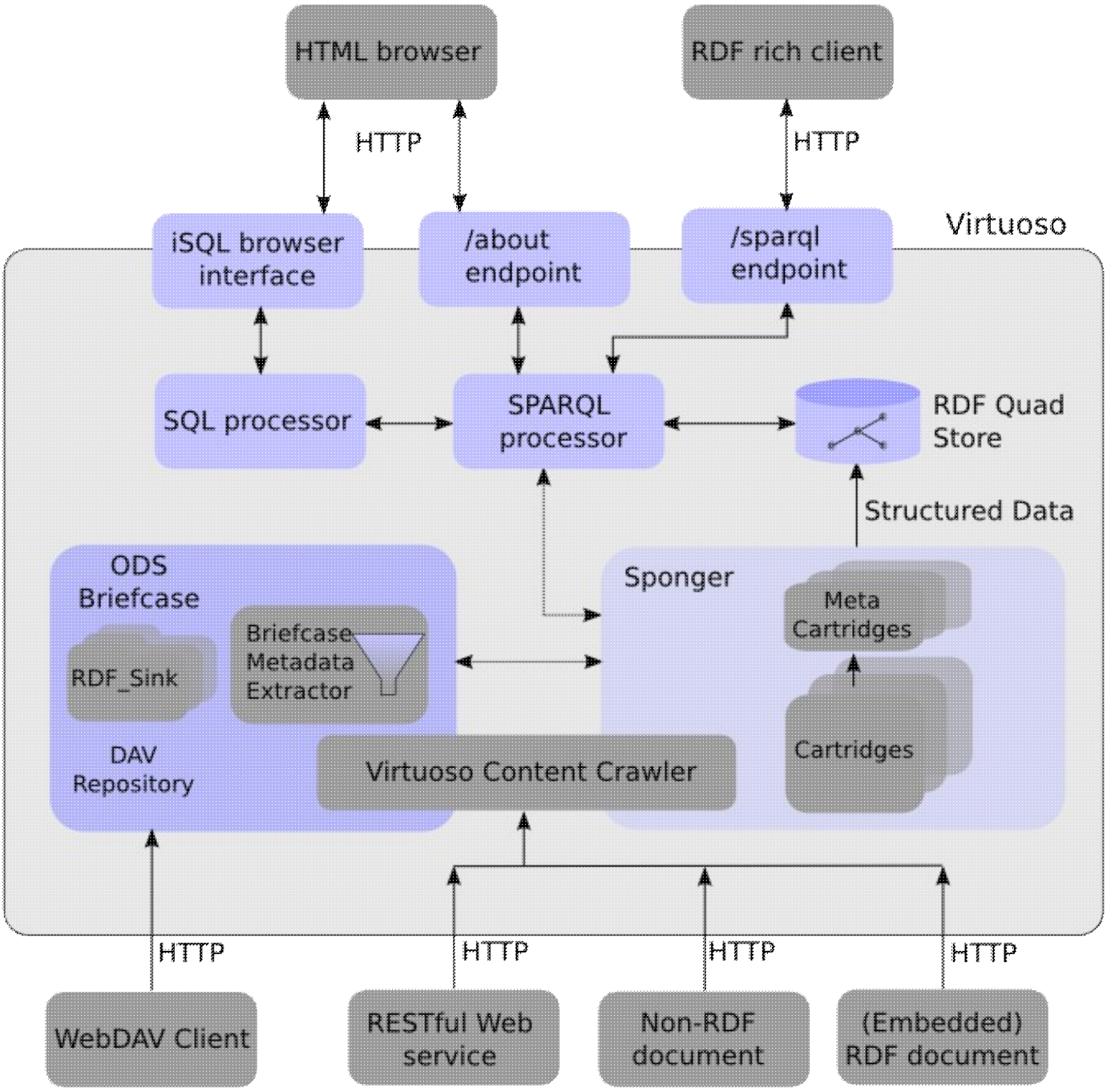


Architecturally, the Sponger is comprised of a number of Cartridges two types of cartridges: Extractor and Meta Cartridges. Extractor Cartridges focus on data extraction and transformation services while the Meta Cartridges provide lookups and joins across other linked data spaces and Web 2.0 APIs. Both cartridge types are themselves comprised of a data extractors and RDF Schema/Ontology Mapper components.

Cartridges are is highly customizable. Custom cartridges can be developed using any language supported by the Virtuoso Server Extensions API enabling structured linked data generation from resource types not available in the default Sponger Cartridge collection bundled -- as part of the Virtuoso Cartridges VAD package .

Figure 16.97. Virtuoso metadata extraction & RDF structured data generation


HTML browser
RDF rich client
HTTP
HTTP
Virtuoso
iSQL browser interface
/about endpoint
/sparql endpoint
SQL processor
SPARQL processor
RDF Quad Store
Structured Data
ODS Briefcase
RDF_Sink
Briefcase Metadata Extractor
DAV Repository
Sponger
Meta Cartridges
Cartridges
Virtuoso Content Crawler
HTTP
HTTP
HTTP
HTTP
WebDAV Client
RESTful Web service
Non-RDF document
(Embedded) RDF document



16.10.2. Why is it Important?

A majority of the worlds data naturally resides in non RDF form at the current time. The Sponger delivers middleware that accelerates the bootstrap of the Semantic Data Web by generating RDF from non RDF data sources, unobtrusively. This "Swiss army knife" for on-the-fly Linked Data generation provides a bridge between the traditional Document Web and the Linked Data Web ("Data Web").

Sponging data from non-RDF Web sources and converting it to RDF exposes the data in a canonical form for querying and inference, and enables fast and easy construction of linked data driven mesh-ups as opposed to code driven Web 2.0 mash-ups.

The RDF extraction and instance data generation products that offer functionality demonstrated by the Sponger are also commonly referred to as RDFizers.

16.10.3. How Does It Work?

When an RDF aware client requests data from a network accessible resource via the Sponger the following events occur:

- A requests in made for data in RDF form, and if RDF is returned nothing further happens
- If RDF isn't returned, then the Sponger passes the data through a Metadata Extraction Pipeline process (using Metadata Extractors)
- The extracted data is transformed to RDF via a Mapping Pipeline process (RDF is extracted by way of Ontology matching and mapping) that results in RDF Entities (instance data) generation
- RDF Entities are returned to the client

The imported data forms a local cache and its invalidation rules conform to those of traditional HTTP clients (Web Browsers). Thus, expiration time is determined based on subsequent data fetches of the same resource (note: the first data load will record the 'expires' header) with current time compared to expiration time stored in the local cache. If HTTP 'expires' header data isn't returned by the source data server, then the "Sponger" will derive it's own invalidation time frame by evaluating the 'date' header and 'last-modified' HTTP headers. Irrespective of path taken, local cache invalidation is driven by an assessment of current time relative to recorded expiration time.

To manage the cache expiration, set the *MinExpiration* parameter in your Virtuoso.ini file.

Read full description of the parameter in the [SPARQL] ini section .

Designed with a pluggable architecture, the Sponger's core functionality is provided by Cartridges. Each cartridge includes Data Extractors which extract data from one or more data sources, and Ontology Mappers which map the extracted data to one or more ontologies/schemas, and route to producing RDF Linked Data.

The Schema Mappers are typically XSLT (e.g. GRDDL and other OpenLink Mapping Schemas) or Virtuoso PL based. The Metadata Extractors may be developed in Virtuoso PL, C/C++, Java, or any other language that can be integrated into the Virtuoso via it's server extensions APIs.

The Sponger also includes a pluggable name resolution mechanism that enables the development of Custom Resolvers for naming schemes (e.g. URNs) associated with protocols beyond HTTP. Examples of custom resolvers include:

- LSID
- DOI

16.10.4. Installation Steps

1. Download the Cartridges VAD package .
2. Install the cartridges_dav.vad package by using the Conductor UI or by using iSQL:

```
SQL> DB.DBA.VAD_INSTALL('tmp/cartridges_dav.vad',0);
SQL_STATE  SQL_MESSAGE
VARCHAR  VARCHAR
_______________________________________________________________________________

00000    No errors detected
00000    Installation of "Linked Data Cartridges" is complete.
00000    Now making a final checkpoint.
00000    Final checkpoint is made.
00000    SUCCESS

6 Rows. -- 1078 msec.
```

3. Cartridge Configuration

    - Extractor Cartridges
    - Meta Cartridges

See Also: Cartridges Package content description.

16.10.5. Using The Sponger

The Sponger can be invoked via the following mechanisms:

1. Virtuoso SPARQL query processor
2. RDF Proxy Service
3. OpenLink RDF client applications
4. ODS-Briefcase (Virtuoso WebDAV)

   - a component of the OpenLink Data Spaces distributed collaborative application platform
5. Directly via Virtuoso PL

SPARQL Query Processor IRI Dereferencing

The Sponger is transparently integrated into the Virtuoso SPARQL query processor, where it supports IRI dereferencing.

Virtuoso extends the SPARQL Query Language such that it is possible to download RDF resources from a given IRI, parse, and then store the resulting triples in a graph, with all three operations performed during the SPARQL query-execution process. The IRI/URI of the graph used to store the triples is usually equal to the URL where the resources are downloaded from, consequently the feature is known as "IRI/URI dereferencing". If a SPARQL query instructs the SPARQL processor to retrieve the target graph into local storage, then the SPARQL sponger will be invoked.

The SPARQL extensions for IRI dereferencing are described below. Essentially these enable downloading and local storage of selected triples either from one or more named graphs, or based on a proximity search from a starting URI for entities matching the select criteria and also related by the specified predicates, up to a given depth. For full details see section Linked Data - IRI Dereferencing .

Note: For brevity, any reference to URI/IRIs above or in subsequent sections implies an HTTP URI/IRI, where IRI is an internationalized URI. Similarly, in the context of the Sponger, the term IRI in the Virtuoso reference documentation should be taken to mean an HTTP IRI.

SPARQL Extensions for IRI Dereferencing of FROM Clauses

In addition to the "define get:..." SPARQL extensions for IRI dereferencing in FROM clauses, Virtuoso supports dereferencing SPARQL IRIs used in the WHERE clause (graph patterns) of a SPARQL query via a set of "define input:grab-..." pragmas.

Consider an RDF resource which describes a member of a contact list, user1, and also contains statements about other users, user2 anduser3 , known to him. Resource user3 in turn contains statements about user4 and so on. If all the data relating to these users were loaded into Virtuoso's RDF database, the query to retrieve the details of all the users could be quite simple. e.g.:

```
sparql
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
select ?id ?firstname ?nick
where
  {
    graph ?g
      {
        ?id rdf:type foaf:Person .
        ?id foaf:firstName ?firstname .
        ?id foaf:knows ?fn .
        ?fn foaf:nick ?nick .
      }
   }
limit 10;
```

But what if some or all of these resources were not present in Virtuoso's quad store? The highly distributed nature of the Linked Data Web makes it highly likely that these interlinked resources would be spread across several data spaces. Virtuoso's 'input:grab-...' extensions to SPARQL enable IRI dereferencing in such a way that all appropriate Network resources are loaded, i.e. "being fetched", during query execution, even if some of the Network resources are not known beforehand. For any particular resource matched, and if necessary downloaded, by the query, it is possible to download related resources via a designated predicate path(s) to a specifiable depth i.e. number of 'hops', distance, or degrees of separation (i.e compute Transitive Closures in SPARQL).

Using Virtuoso's 'input:grab-' pragmas to enable sponging, the above query might be recast to:

```
sparql
define input:grab-var "?more"
define input:grab-depth 10
define input:grab-limit 100
define input:grab-base "http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlin
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>

select ?id ?firstname ?nick
where {
    graph ?g {
                ?id rdf:type foaf:Person .
                ?id foaf:firstName ?firstname .
                ?id foaf:knows ?fn .
                ?fn foaf:nick ?nick .
                optional { ?id rdfs:SeeAlso ?more }
              }
}
limit 10;
```

Another example showing a designated predicate traversal path via the input:grab-seealso extension is:

```
sparql
define input:grab-iri <http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlink
define input:grab-var "id"
define input:grab-depth 10
define input:grab-limit 100
define input:grab-base "http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlin
define input:grab-seealso <foaf:maker>
prefix foaf: <http://xmlns.com/foaf/0.1/>

select ?id
where
  {
    graph ?g
      {
        ?id a foaf:Person .
      }
  }
limit 10;
```

A list of the input:grab pragmas is given below:

- input:grab-var specifies the name of the SPARQL variable whose values should be used as IRIs of resources that should be downloaded.
- input:grab-iri specifies an IRI that should be retrieved before executing the rest of the query, if it is not in the quad store already. (This pragma can be included multiple times).
- input:grab-seealso (or its synonym input:grab-follow-predicate) specifies a predicate IRI to be used when traversing a graph. (This pragma can be included multiple times).
- input:grab-limit sets the maximum number of resources (graph subject or object nodes) to be retrieved from a target graph.
- input:grab-depth sets the maximum 'degrees of separation' or links (predicates) between nodes in the target graph.
- input:grab-all "yes" instructs the SPARQL processor to dereference everything related to the query. All variables and literal IRIs in the query become values for input:grab-var and input:grab-iri. The resulting performance may be very bad.
- input:grab-base specifies the base IRI to use when converting relative IRIs to absolute. (Default: empty string).
- input:grab-destination overrides the default IRI dereferencing and forces all retrieved triples to be stored in the specified graph.
- input:grab-loader identifies the procedure used to retrieve each resource via HTTP, parse and store it. (Default: DB.DBA.RDF_SPONGE_UP)
- input:grab-resolver identifies the procedure that resolves IRIs and determines the HTTP method of retrieval. (Default: DB.DBA.RDF_GRAB_RESOLVER_DEFAULT)

SPARQL Processor Usage Example

Network Resource Fetch can be performed directly from within the SPARQL processor.

After logging into Virtuoso's Conductor interface, the following query can be issued from the Interactive SQL (iSQL) panel:

```
sparql
define get:uri "http://www.ivan-herman.net/foaf.html"
define get:soft "soft"
select * from <http://mygraph> where {?s ?p ?o}
```

Here the sparql keyword invokes the SPARQL processor from the SQL interface and the RDF data fetched from page http://www.ivan-herman.net/foaf.html is loaded into the local RDF quad store as graph http://mygraph .

The new graph can then be queried using the basic SPARQL client normally available in a default Virtuoso installation at http://example.com/sparql/. e.g.:

```
select * from <http://mygraph> where {?s ?p ?o}
```

RDF Proxy Service

The Sponger's functionality is also exposed via an in-built REST style Web service. This web service takes a target URL and either returns the content "as is" or tries to transform (by sponging) to RDF. Thus, the proxy service can be used as a 'pipe' for RDF browsers to browse non-RDF sources.

When the cartridges package is installed, Virtuoso reserves the path '/about/[id|data|rdf|html]/http/' for Sponger Proxy URI Service. For example, if a Virtuoso installation on host example.com listens for HTTP requests on port 8080 then client applications should use a 'service endpoint' string equal to 'http://example.com:8080/about/[id|data|rdf|html]/http/'. If the cartridges package is not installed, then the service uses the path '/proxy/rdf/'.

Note: The old Sponger Proxy URI Service pattern '/proxy/' is now deprecated.

Example 1

The following URLs return information about musician John Cale, gleaned from the MusicBrainz music metadatabase, rendered as RDF or HTML respectively. (The Network Resource fetched data is available in the HTML rendering through the foaf:primaryTopic property.)

- http://demo.openlinksw.com/about/rdf/http://musicbrainz.org/artist/72c090b6-a68e-4cb9-b330-85278681a714.html
- http://demo.openlinksw.com/about/html/http/musicbrainz.org/artist/72c090b6-a68e-4cb9-b330-85278681a714.html

Example 2

The file http://www.ivan-herman.net/foaf.html contains a short profile of the W3C Semantic Web Activity Lead Ivan Herman. This XHTML file contains RDF embedded as RDFa. Running the file through the Sponger via Virtuoso's RDF proxy service extracts the embedded FOAF data as pure RDF, as can be seen by executing:

```
$ curl -L -H "Accept:application/rdf+xml" http://linkeddata.uriburner.com/about/id/entity/http/www.ivan-h
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:rdfs="http://www.w3.org/2000/01/rd
  <rdf:Description rdf:about="http://linkeddata.uriburner.com/about/id/http/www.ivan-herman.net/foaf.html
  <rdf:Description rdf:nodeID="b145981159"><rdf:rest rdf:nodeID="b145981158"/></rdf:Description>
  <rdf:Description rdf:about="http://linkeddata.uriburner.com/about/id/entity/http/www.mendeley.com/profi
  etc ..
  <rdf:Description rdf:nodeID="b145981130"><http-voc:elementName xmlns:http-voc="http://www.w3.org/2006/h
</rdf:RDF>
```

(linkeddata.uriburner.com hosts a public Virtuoso instance.) Though this example demonstrates the action of the /about/id/entity/ service quite transparently, it is a basic and unwieldy way to view RDF. As described earlier, the OpenLink Data Explorer uses the same proxy service to provide a more polished means to extract and view fetched RDF data.

Usage of the Sponger Middleware via REST patterns

Delegation and proxies are part of the Internet and Web's federated architecture. Thus, developers of RESTful applications benefit immensely from the ability to leverage Sponger functionality via delegation to it as a proxy.

The following table presents list of the supported URL parameters:

Table 16.9.

Parameter	Value	Description	Example
refresh	clean	*Usage* : for overwriting. The 'clean' usage explicitly clears the graph i.e. will cause the Sponger to drop cache even if it is marked to be in the fly. Thus, if fetched cache by some reason is left in some inconsistent state like shutdown during Network Resource fetching, then 'clean' is required as it doesn't check cache state. *Note* : must be used with caution as other threads may be doing fetching of network resources at same time.	Explicitly clear the graph
sponger:get	add	*Usage* : Add new triples to named graphs, progressively. This is the default value for the parameter sponger:get. May be used together with refresh=<seconds> to overwrite the expiration in the cache.	Add new triples and refresh on every 10 seconds
sponger:get	soft	*Usage* : Network Resource Fetch data subject to cache invalidation mode and associated rules of instance. May be used together with refresh=<seconds> to overwrite the expiration in the cache.	Network Resource Fetch data with option *soft* and refresh on every 10 seconds
sponger:get	replace	*Usage* : Replace subject to cache invalidation mode and rules, but coverage includes non fetched triples if such exist in a given named graph. may be used together with refresh=<seconds> to overwrite the expiration in the cache.	Replace data and refresh on every 10 seconds

OpenLink RDF Client Applications

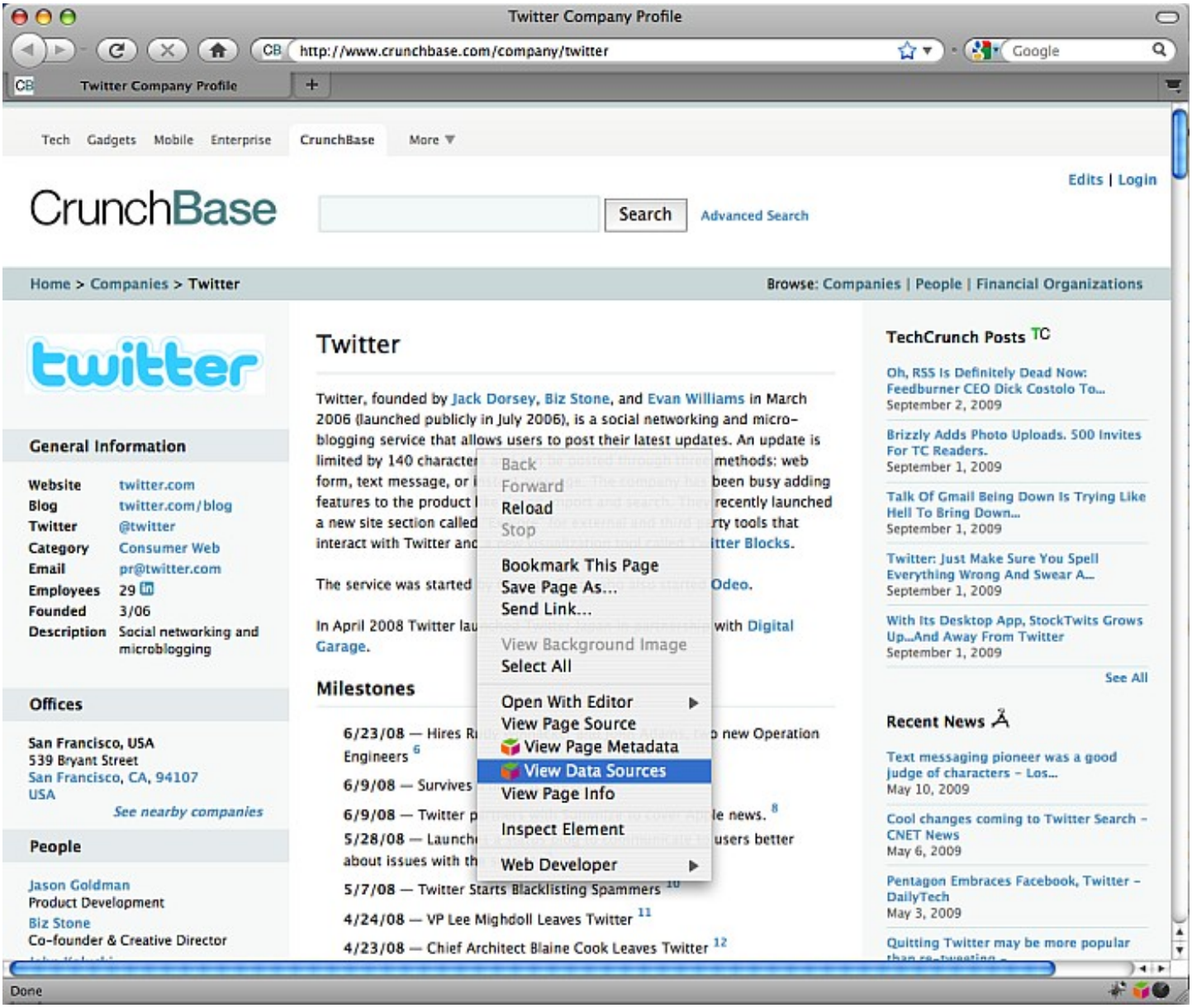
OpenLink currently provides two main RDF client applications:

- OpenLink Data Explorer (ODE)
- iSPARQL

ODE is a Linked Data explorer packaged as a Firefox plugin (support for other browsers is planned). iSPARQL is an interactive AJAX-based SPARQL query builder with support for SPARQL QBE, bundled as part of the OpenLink Ajax Toolkit (OAT). Both RIA clients utilise sponging extensively.

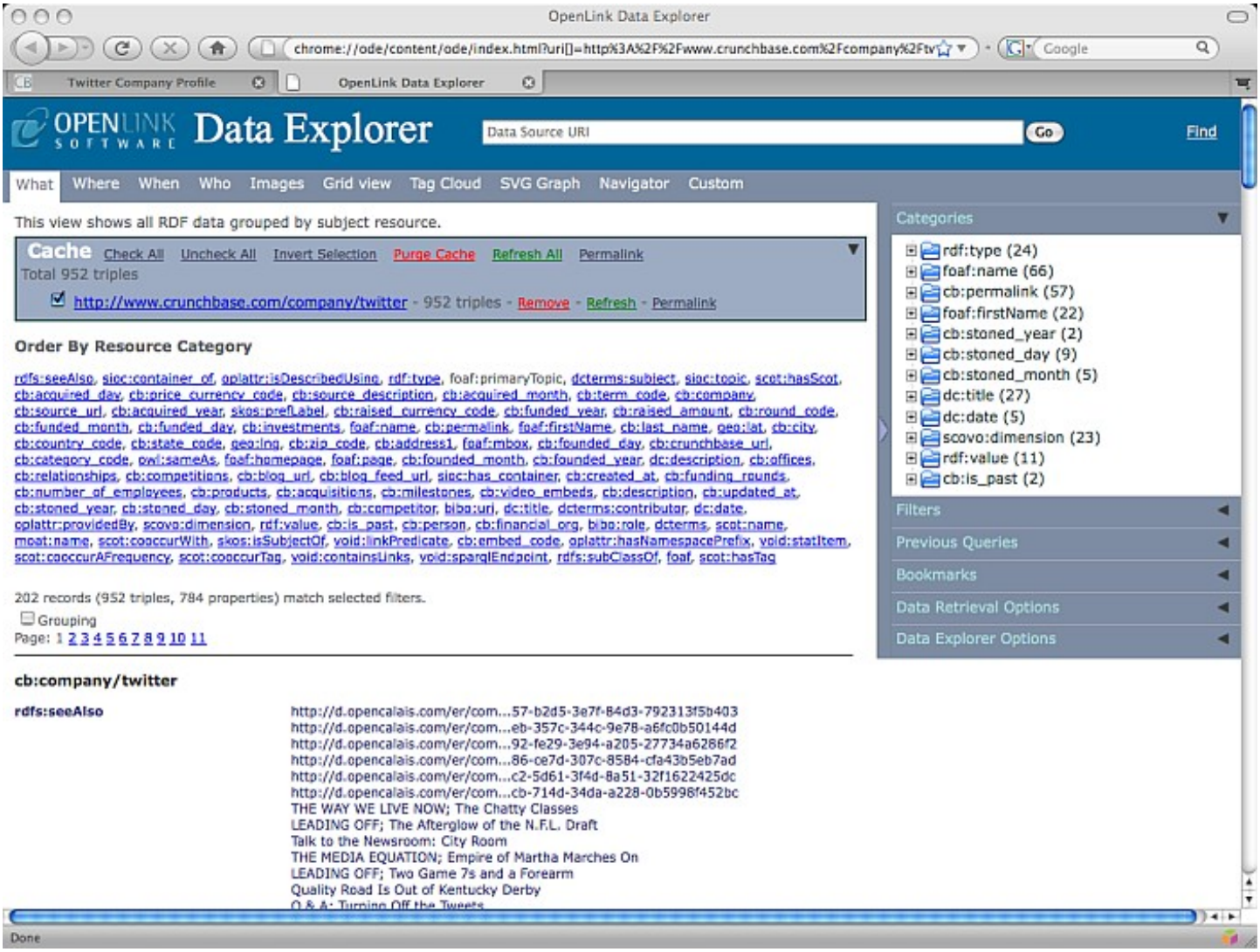
The ODE plugin is dual faceted - RDF data can be viewed and explored natively, through its integral RDF browser, or, as described above, rendered as HTML through ODE's 'View Page Metadata' option. The screenshots below show ODE's RDF browser being launched through the 'View Linked Data Sources' popup menu.

Figure 16.98. Launching ODE's RDF browser



The RDF browser then displays RDF data fetched via the Crunchbase cartridge.

Figure 16.99. ODE RDF browser displaying Crunchbase network resource fetched data



iSPARQL directs queries to the configured SPARQL endpoint. When targetting a Virtuoso /sparql service, Virtuoso specific sponging options can be enabled through the 'Preferences' dialog box.

The iSPARQL sponger settings are appended to SPARQL queries through the 'should-sponge' query parameter. These are translated to IRI dereferencing pragmas on the server as follows:

Table 16.10.

iSPARQL sponging setting	/sparql endpoint: "should-sponge" query parameter value	SPARQL processor directives
Use only local data	N/A	N/A
Retrieve remote RDF data for all missing source graphs	soft	define get:soft "soft"
Retrieve all missing remote RDF data that might be useful	grab-all	define input:grab-all "yes" define input:grab-depth 5 define input:grab-limit 100
Retrieve all missing remote RDF data that might be useful including seeAlso references	grab-seealso	define input:grab-all "yes" define input:grab-depth 5 define input:grab-limit 200 define input:grab-seealso <http://www.w3.org.2000/01/rdf-schema#seeAlso> define input:grab-seealso <http://xmlns.com/foaf/0.1/seeAlso>
Try to download all referenced resources	grab-everything	define input:grab-all "yes" define input:grab-intermediate "yes" define input:grab-depth 5 define input:grab-limit 500 define input:grab-seealso

iSPARQL sponging setting	/sparql endpoint: "should-sponge" query parameter value	SPARQL processor directives
		<http://www.w3.org/2000/01/rdf-schema#seeAlso> define input:grab-seealso <http://xmlns.com/foaf/0.1/seeAlso>

ODS-Briefcase (Virtuoso WebDAV)

ODS-Briefcase is a component of OpenLink Data Spaces (ODS), a new generation distributed collaborative application platform for creating Semantic Web presence via Data Spaces derived from weblogs, wikis, feed aggregators, photo galleries, shared bookmarks, discussion forums and more. It is also a high level interface to the Virtuoso WebDAV repository.

ODS-Briefcase offers file-sharing functionality that includes the following features:

- Web brower-based interactions
- Web Services (direct use of the HTTP based WebDAV protocol)
- SPARQL query language support - all WebDAV resources are exposed as SIOC ontology instance data (RDF data sets)

When resources or documents are put into the ODS Briefcase and are made publicly readable (via a Unix-style +r permission or ACL setting) and the resource in question is of a supported content type, metadata is automatically extracted at file upload time.

Note: ODS-Briefcase extracts metadata from a wide array of file formats, automatically.

The extracted metadata is available in two forms, pure WebDAV and RDF (with RDF/XML or N3/Turtle serialization options), that is optionally synchronized with the underlying Virtuoso Quad Store.

All public readable resources in WebDAV have their owner, creation time, update time, size and tags published, plus associated content type dependent metadata. This WebDAV metadata is also available in RDF form as a SPARQL queriable graph accessible via the SPARQL protocol endpoint using the WebDAV location as the RDF data set URI (graph or data source URI).

You can also use a special RDF_Sink folder to automate the process of uploading RDF resources files into the Virtuoso Quad Store via WebDAV or raw HTTP. The properties of the special folder control whether sponging (RDFization) occurs. Of course, by default, this feature is enabled across all Virtuoso and ODS installations (with an ODS-Briefcase Data Space instance enabled).

Raw HTTP Example for Extracting Metadata using CURL

```
Username: demo
Password: demo
Source File: wine.rdf
Destination Folder:
http://demo.openlinksw.com/DAV/home/demo/rdf_sink/
Content Type: application/rdf+xml

$ curl -v -T wine.rdf -H content-type:application/rdf+xml http://demo.openlinksw.com/DAV/home/demo/rdf_si
```

Finally, you can also get RDF data into Virtuoso's Quad Store via WebDAV using the Virtuoso Web Crawler utility (configurable via the Virtuoso Conductor UI). This feature also provides the ability to enable or disable Sponging as depicted below.

Sponger and ODS-Briefcase Structured Data Extractor Interrelationship

As the Sponger and ODS-Briefcase both extract structured data, what is the relationship between these two facilities?

The principal difference between the two is that the Sponger is anRDF data crawler & generator, whereas Briefcase's structured data extractor is a WebDAV resourcefilter. The Briefcase structured data extractor is aimed at providing RDF data from WebDAV resources. Thus, if none of the available Sponger cartridges are able to extract metadata and produce RDF structured data, the Sponger calls upon the Briefcase extractor as the last resort in the RDF structured data generation pipeline.

Figure 16.100. Conductor's content import configuration panel

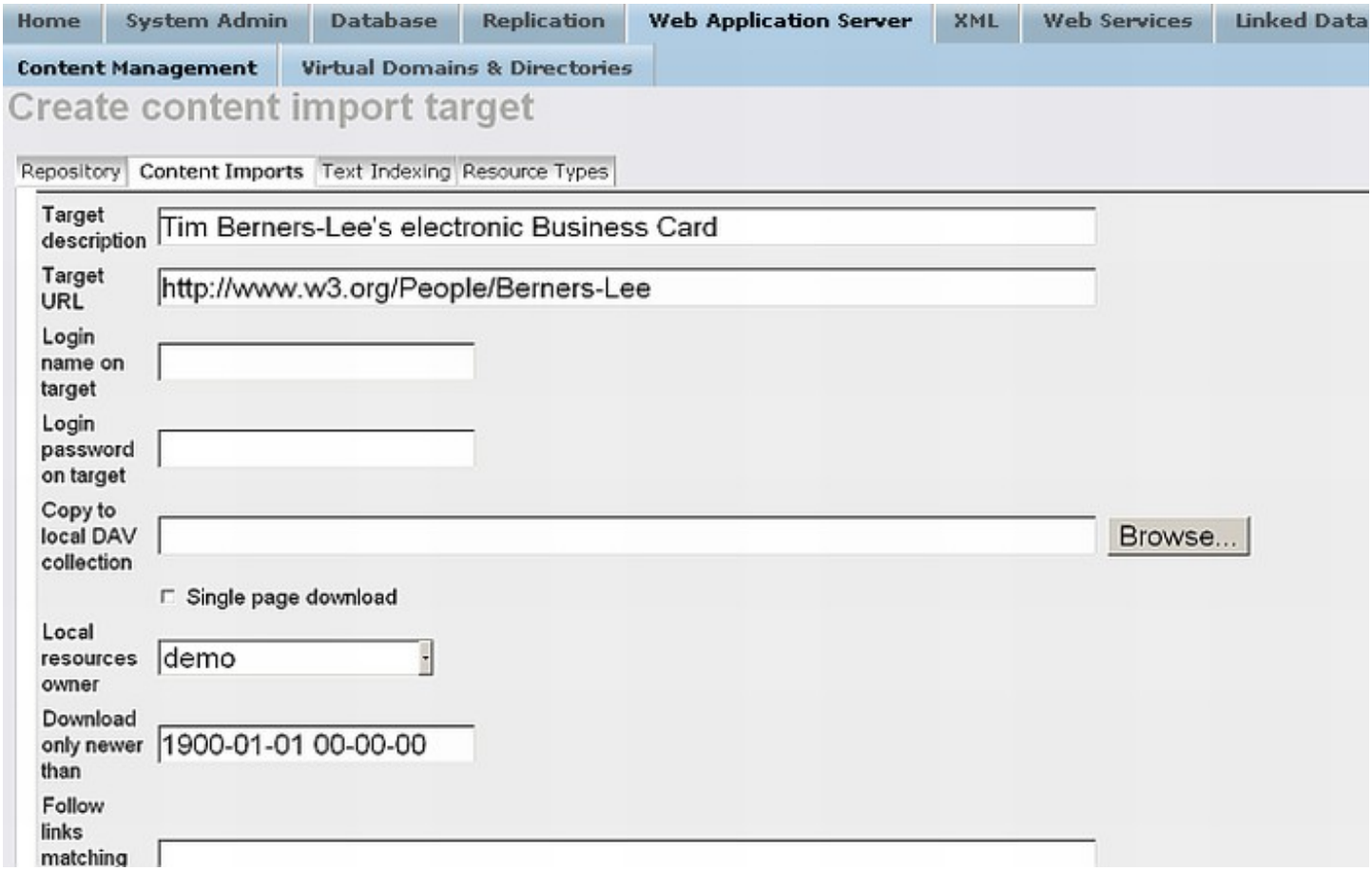


Figure 16.101. Conductor's content import configuration panel

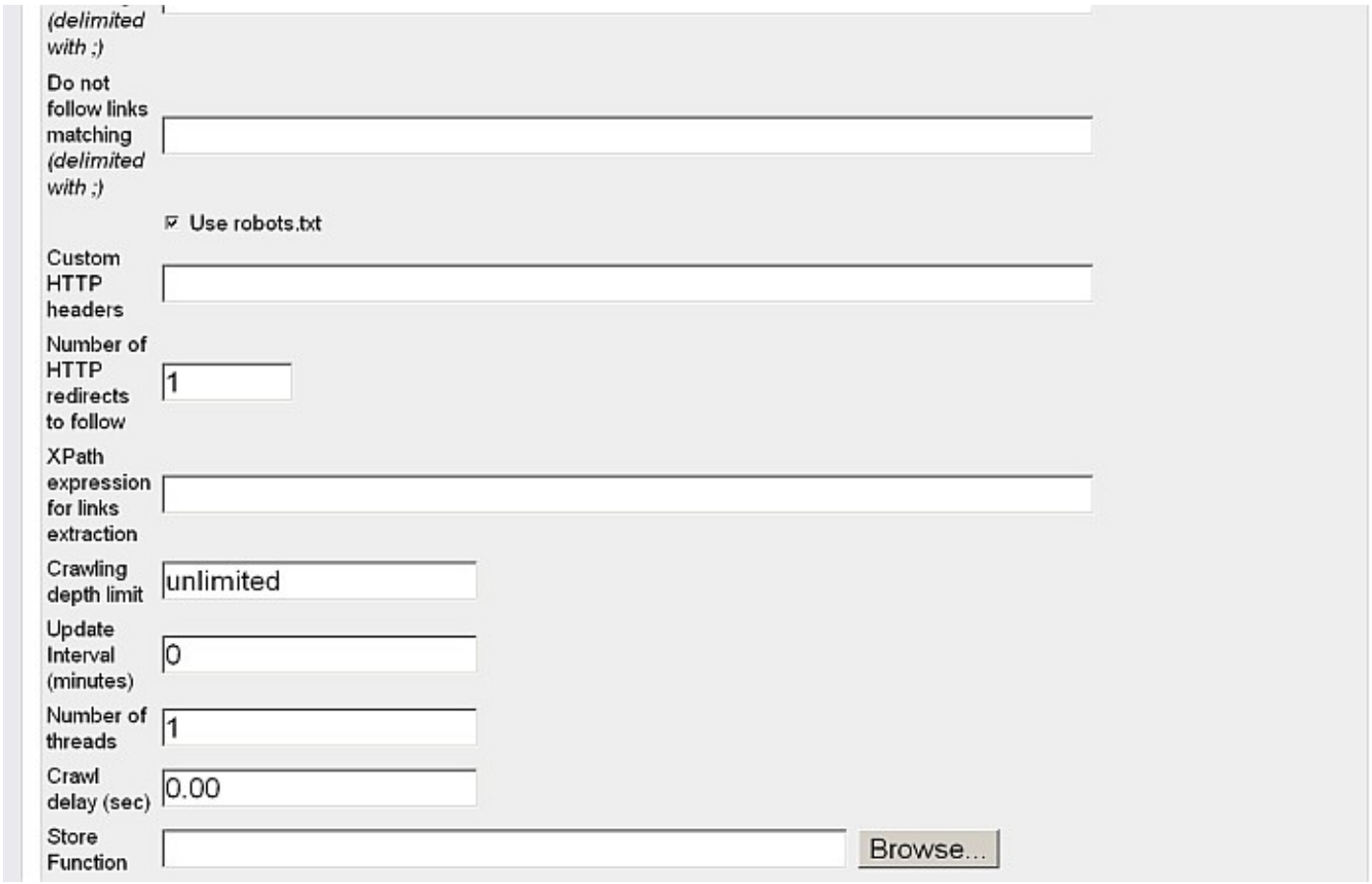


Figure 16.102. Conductor's content import configuration panel

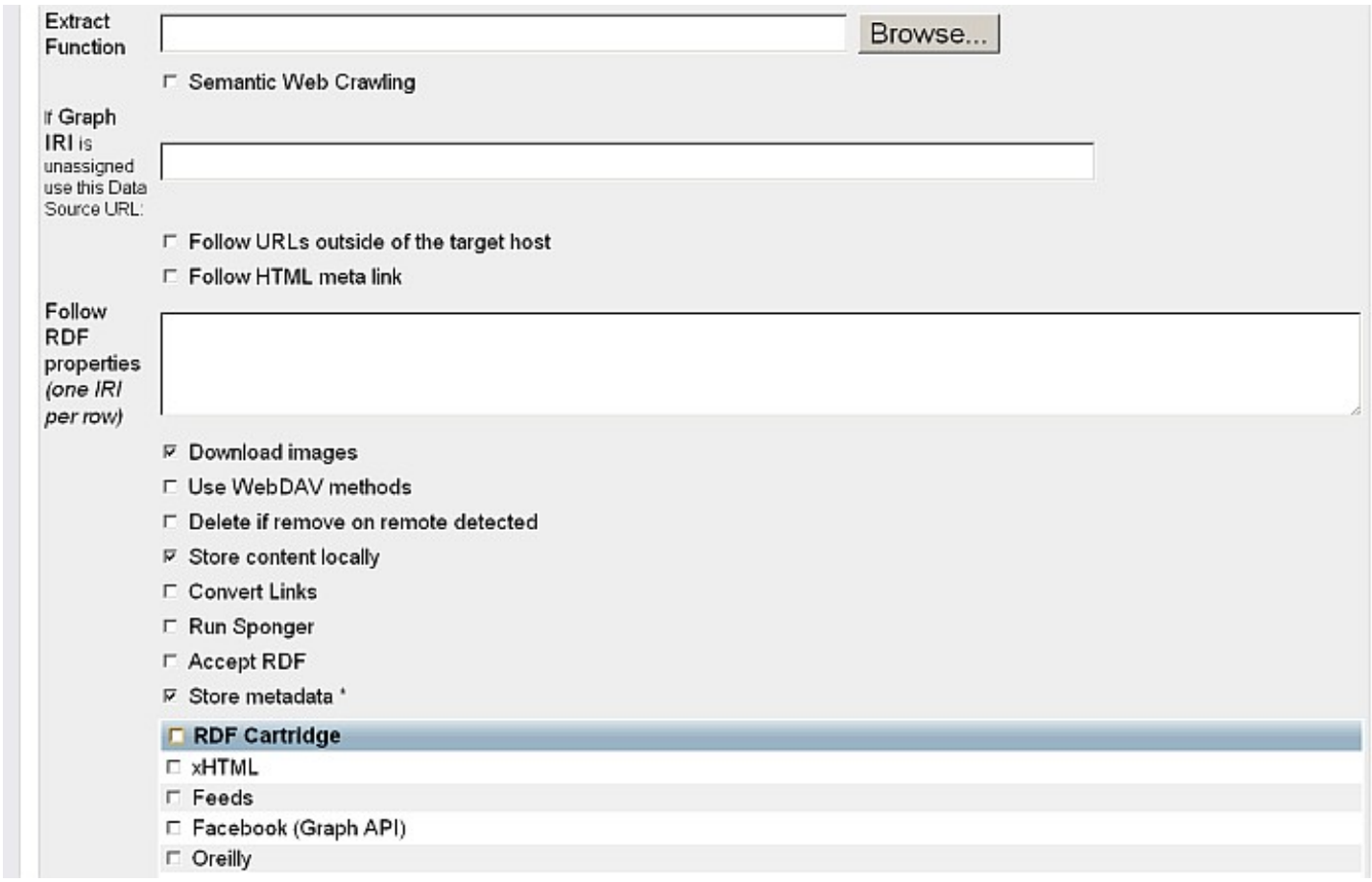
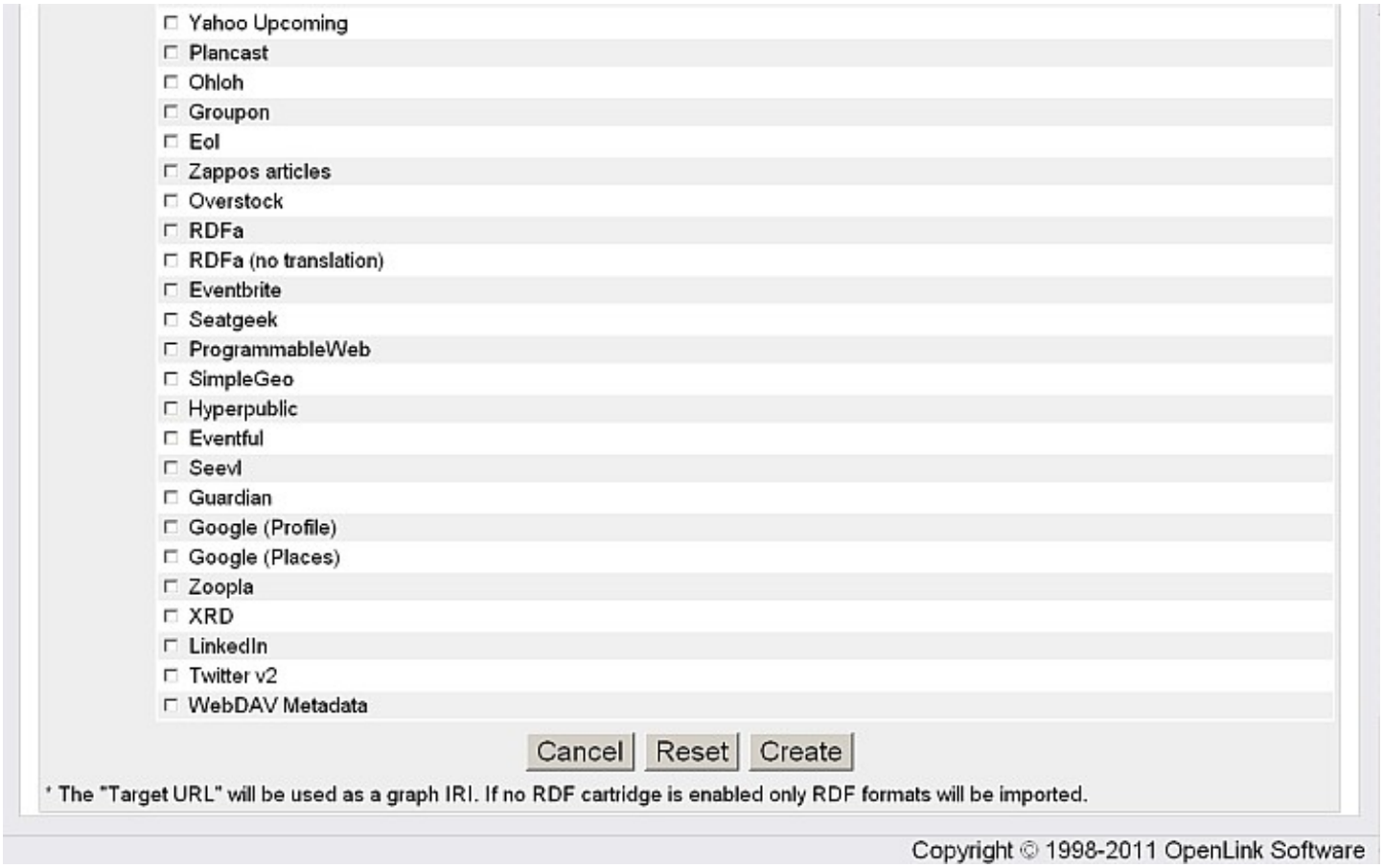


Figure 16.103. Conductor's content import configuration panel



Directly via Virtuoso PL

Sponger cartridges are invoked through a cartridge hook which provides a Virtuoso PL entry point to the packaged functionality. Should you wish to utilize the Sponger from your own Virtuoso PL procedures, you can do so by calling these hook routines directly. Full details of the hook function prototype and how to define your own cartridges are presented here .

16.10.6. Consuming the Generated RDF Structured Data

The generated RDF-based structured data (RDF) can be consumed in a number of ways, depending on whether or not the data is persisted in Virtuoso's RDF Quad Store.

If the data is persisted, it can be queried through the Virtuoso SPARQL endpoint associated with any Virtuoso instance: /sparql. The RDF is exposed in a graph typically identified using a URL matching the source resource URL from which the RDF data was generated. Naturally, any SQL query can also access this, since SPARQL can be freely intermixed with SQL via Virtuoso's SPASQL (SPARQL inside SQL) functionality. RDF data is also accessible through Virtuoso's implementation of the URIQA protocol.

If not persisted, as is the case with the RDF Proxy Service, the data can be consumed by an RDF aware Web client, e.g. an RDF browser such as the OpenLink Data Explorer (ODE).

16.10.7. RDF Cartridges Use Cases

This section contains examples of Web resources which can be transformed by RDF Cartridges. It also states where additional setup for given cartridges is needed i.e. keys account names etc.

Service based:

- amazon

```
needs: api key
example: http://www.amazon.com/gp/product/0553383043
```

- ebay

```
needs: account, api-key
example: http://cgi.ebay.com/RARE-DAY-IN-FAIRY-LAND-ELEPHANT-FOLIO-20-FULL-COLOR_W0QQitemZ14020959
```

- flickr needs: api-key example: http://farm1.static.flickr.com/212/496684670_7122c831ed.jpg

- mbz

```
example: http://musicbrainz.org/release/37e955d4-a53c-45aa-a812-1b23b88dbc13.html
```

- mql (freebase)

```
example: http://www.freebase.com/view/en/beta_ursae_majoris
```

- facebook

```
needs: api-key, secret, persistent-session-id
example: http://www.facebook.com/profile.php?id=841100003
```

- yahoo-stock

```
example: http://finance.yahoo.com/q?s=AAPL
```

- yahoo-traffic

```
example: http://local.yahooapis.com/MapsService/V1/trafficData?appid=YahooDemo&street=701+First+St
```

- Bugzilla

```
example: https://bugzilla.mozilla.org/show_bug.cgi?id=251714
```

- SVG
- OO document

```
needs: unzip plugin
```

- Wikipedia

```
needs: php plugin & dbpedia extractor
example: http://wikipedia.org/wiki/London
```

- Opencalais
- iCalendar

GRDDL

- Google Base (google)

```
example: http://www.google.com/base/feeds/snippets/17891817243016304554
```

- eRDF
- RDFa
- hCard
- hCalendar
- hReview
- relLicense
- XBRL
- HR-XML
- DC
- geoURL
- Ning
- XFN
- xFolk

URN handlers

Table 16.11. URN handlers List

URN handler	Sample URI	Resource Description	Linked Data View	Linked Data Graph	Needs
LSID	urn:lsid:ubio.org:namebank:12292	HTML Representation	Linked Data View	Data Explorer View	none
DOI	doi:10.1038/35057062	HTML Representation	Linked Data View	Data Explorer View	Needs hslookup plugin, relevant html, pdf, xml etc. mappers enabled.
OAI	oai:dcmi.ischool.washington.edu:article/8	HTML Representation	Linked Data View	Data Explorer View	none

SPARQL IRI Dereferencing

The Virtuoso SPARQL engine (called for brevity just SPARQL below) supports IRI Dereferencing, however it understands only RDF data, that is it can retrieve only files containing RDF/XML, turtle or N3 serialized RDF data, if format is unknown it will try mapping with built-in WebDAV metadata extractor. In order to extend this feature with dereferencing web or file resources which naturally don't have RDF data (like PDF, JPEG files for example) is provided a special mechanism in SPARQL engine. This mechanism is called RDF mappers for translation of non-RDF data files to RDF.

In order to instruct the SPARQL to call a RDF mapper it needs to be registered and it will be called for a given URL or MIME type pattern. In other words, when unknown for SPARQL format is received during URL dereferencing process, it will look into a special registry (a table) to match either the MIME type or IRI using a regular expression, if match is found the mapper function will be called.

Sponger Proxy service

Sponger functionality is also exposed via Virtuoso's "/proxy/rdf/" endpoint, as an in-built REST style Web service available in any Virtuoso standard installation. This web service takes a target URL and either returns the content "as is" or tries to transform (by sponging) to RDF. Thus, the proxy service can be used as a 'pipe' for RDF browsers to browse non-RDF sources.

For more information see RDF Sponger Proxy service

Cache Invalidation

To clear cache on all values of HS_LOCAL_IRI of the SYS_HTTP_SPONGE table use:

```
SPARQL clear graph <A-Named-Graph>;
```

16.10.8. Cartridge Architecture

What is a Cartridge?

See full description here

Extractor Cartridges

An Extractor Cartridge processes a Resource of a given format, extracting RDF according to rules appropriate to that format. External data does not come into play; only the content of the Resource fed to the Sponger.

Supported Standard Non-RDF Data Formats

These Cartridges handle open formats - typically community-developed, openly-documented, and freely-licensed data structures.

Table 16.12.

Cartridge	Sample URI	Resource Description	Linked Data Graph
AB Meta	example	HTML Representation	Data Explorer View
Atom	example	HTML Representation	Data Explorer View
CSV	example	HTML Representation	Data Explorer View
DC	example	HTML Representation	Data Explorer View
eRDF	example	HTML Representation	Data Explorer View
hAudio	example	HTML Representation	Data Explorer View
hCalendar	example	HTML Representation	Data Explorer View
hCard	example	HTML Representation	Data Explorer View
hListing	example	HTML Representation	Data Explorer View
hNews	example	HTML Representation	Data Explorer View
hProduct	example	HTML Representation	Data Explorer View
HR-XML	example	HTML Representation	Data Explorer View
hRecipe	example	HTML Representation	Data Explorer View
hResume	example	HTML Representation	Data Explorer View
hReview	example	HTML Representation	Data Explorer View
HTTP in RDF	example	HTML Representation	Data Explorer View
iCalendar	example	HTML Representation	Data Explorer View
Microsoft Word 2003 XML Document	example	HTML Representation	Data Explorer View
Microsoft XML Spreadsheet 2003	example	HTML Representation	Data Explorer View
Microsoft Documents	example	HTML Representation	Data Explorer View
OData	example	HTML Representation	Data Explorer View
OO document	example	HTML Representation	Data Explorer View
OPML	example	HTML Representation	Data Explorer View
PPTX	example	HTML Representation	Data Explorer View
RDFa	example	HTML Representation	Data Explorer View
RSS	example	HTML Representation	Data Explorer View
Slidy	example	HTML Representation	Data Explorer View
vCalendar	example	HTML Representation	Data Explorer View
vCard	example	HTML Representation	Data Explorer View

Cartridge	Sample URI	Resource Description	Linked Data Graph
WebDAV Metadata	example	HTML Representation	Data Explorer View
XBRL	example	HTML Representation	Data Explorer View
XFN Profile	example	HTML Representation	Data Explorer View
XFN Profile2	example	HTML Representation	Data Explorer View
xHTML	example	HTML Representation	Data Explorer View
XHTML	example	HTML Representation	Data Explorer View

Supported Vendor-specific Non-RDF Data Formats

These Cartridges handle closed formats - typically proprietary; sometimes undocumented; possibly licensed to no-one except the format originator. Sometimes data may not be parsed as desired or expected, as many of these Cartridges have required reverse-engineering of the data format in question.

Table 16.13.

Cartridge	Needs	Sample URI	Resource Description	Linked Data Graph
Amazon	API Key	example	HTML Representation	Data Explorer View
BestBuy	API Key	example	HTML Representation	Data Explorer View
Bing	none	example	HTML Representation	Data Explorer View
Bugzillas	none	example	HTML Representation	Data Explorer View
CNET	API Key	example	HTML Representation	Data Explorer View
CrunchBase	none	example	HTML Representation	Data Explorer View
Delicious	none	example	HTML Representation	Data Explorer View
Digg	none	example	HTML Representation	Data Explorer View
Discogs	php plugin, DBpedia Extractor	example	HTML Representation	Data Explorer View
Disqus	API Key, API Account	example	HTML Representation	Data Explorer View
DOI	hslookup plugin; relevant html-, pdf-, xml-, etc., -mappers enabled	example	HTML Representation	Data Explorer View
Dublin Core	none	example	HTML Representation	Data Explorer View
eBay	account, API Key	example	HTML Representation	Data Explorer View
Evri	none	example	HTML Representation	Data Explorer View
Facebook	API key and secret, OAuth token See details	example	HTML Representation	Data Explorer View
Flickr	API Key	example	HTML Representation	Data Explorer View
Freebase	none	example	HTML Representation	Data Explorer View
Geonames	none	example	HTML Representation	Data Explorer View

Cartridge	Needs	Sample URI	Resource Description	Linked Data Graph
geoURL	none	example	HTML Representation	Data Explorer View
Get Satisfaction	none	example	HTML Representation	Data Explorer View
Google+	API key See details	example	HTML Representation	Data Explorer View
Google Base	none	example	HTML Representation	Data Explorer View
Google Book	none	example	HTML Representation	Data Explorer View
Google Document	none	example	HTML Representation	Data Explorer View
Google Social Graph	none	example	HTML Representation	Data Explorer View
Google Spreadsheet	none	example	HTML Representation	Data Explorer View
Hoovers	none	example	HTML Representation	Data Explorer View
ISBN	API Key	example	HTML Representation	Data Explorer View
LastFM	API Key	example	HTML Representation	Data Explorer View
LibraryThing	API Key	example	HTML Representation	Data Explorer View
LinkedIn	API key and secret, OAuth token See details	example	HTML Representation	Data Explorer View
LSID	none	example	HTML Representation	Data Explorer View
Meetup	API Key	example	HTML Representation	Data Explorer View
MusicBrainz	none	example	HTML Representation	Data Explorer View
Ning Metadata	none	example	HTML Representation	Data Explorer View
OAI	none	example	HTML Representation	Data Explorer View
Open Social	none	example	HTML Representation	Data Explorer View
OpenLibrary	none	example	HTML Representation	Data Explorer View
OpenStreetMap	none	example	HTML Representation	Data Explorer View
oReilly	none	example	HTML Representation	Data Explorer View
Picasa	none	example	HTML Representation	Data Explorer View
Radio Pop	none	example	HTML Representation	Data Explorer View
relLicense	none	example	HTML Representation	Data Explorer View
Revyu	none	example	HTML Representation	Data Explorer View

Cartridge	Needs	Sample URI	Resource Description	Linked Data Graph
Rhapsody	none	example	HTML Representation	Data Explorer View
SalesForce.com	API Key,user login	example	HTML Representation	Data Explorer View
SlideShare	API Key, SharedSecret	example	HTML Representation	Data Explorer View
SlideSix	none	example	HTML Representation	Data Explorer View
SVG	none	example	HTML Representation	Data Explorer View
Tesco	none	example	HTML Representation	Data Explorer View
Tumblr	none	example	HTML Representation	Data Explorer View
TWFY (theyworkforyou)	API Key	example	HTML Representation	Data Explorer View
Twitter	API key and secret, OAuth token See details	example	HTML Representation	Data Explorer View
Ustream	none	example	HTML Representation	Data Explorer View
Web Resource CC (Licenses)	none	example	HTML Representation	Data Explorer View
Wikipedia	none	example	HTML Representation	Data Explorer View
xFolk	none	example	HTML Representation	Data Explorer View
Yahoo! Finance	none	example	HTML Representation	Data Explorer View
Yahoo! SearchMonkey	none	example	HTML Representation	Data Explorer View
Yahoo! Traffic Data	none	example	HTML Representation	Data Explorer View
Yahoo! Weather	none	example	HTML Representation	Data Explorer View
Yelp	none	example	HTML Representation	Data Explorer View
Youtube	none	example	HTML Representation	Data Explorer View
Zillow	none	example	HTML Representation	Data Explorer View

Meta Cartridges

A Meta Cartridge submits a Resource to a third-party Web Service for processing. Returned RDF supplements the RDF generated by Extractor and other Meta Cartridges. Locally generated RDF may also be submitted to the third-party services, instead-of or in-addition-to the original Resource itself.

Default Sponger behavior is for all installed Meta Cartridges to be brought to bear on all submitted Resources:

Table 16.14.

Cartridge	Needs	Sample URI	Resource Description	Linked Data Graph	Notes

Cartridge	Needs	Sample URI	Resource Description	Linked Data Graph	Notes
Alchemy	API Key	example	HTML Representation	Data Explorer View	
Amazon Search for products	API Key, secret	example	HTML Representation	Data Explorer View	
BBC Links		example	HTML Representation	Data Explorer View	
BestBuy Search for products	API Key	example	HTML Representation	Data Explorer View	
Bing	API Key	example	HTML Representation	Data Explorer View	
Bit.ly		example	HTML Representation	Data Explorer View	
CNET Search for products	API Key	example	HTML Representation	Data Explorer View	
Collecta		example	HTML Representation	Data Explorer View	Check rdfs:seeAlso links found for microsoft.
Crunchbase		example	HTML Representation	Data Explorer View	
Dapper Search		example	HTML Representation	Data Explorer View	
DBpedia Meta		example	HTML Representation	Data Explorer View	
Delicious Meta	User Login	example	HTML Representation	Data Explorer View	
Digg.com		example	HTML Representation	Data Explorer View	Check rdfs:seeAlso the links from Digg.com .
Discogs	API Key, php plugin, DBpedia Extractor	example	HTML Representation	Data Explorer View	
Document Links		example	HTML Representation	Data Explorer View	
eBay Search for products	account, API Key	example	HTML Representation	Data Explorer View	
Evri Meta		example	HTML Representation	Data Explorer View	
Facebook	API Key, secret, persistent-session-id.	See details	example	HTML Representation	Data Explorer View
Flickr Search for photos	API Key	example	HTML Representation	Data Explorer View	
FOAF-Search		example	HTML Representation	Data Explorer View	Check rdfs:seeAlso at: link1 , link2 , link3
Foursquare		example	HTML Representation	Data Explorer View	
Freebase NYTC	API Key	example	HTML Representation	Data Explorer View	
Freebase NYTCF	API Key	example	HTML Representation	Data Explorer View	
FriendFeed		example	HTML Representation	Data Explorer View	
Geonames Meta		example	HTML Representation	Data Explorer View	
Geopoints		example	HTML Representation	Data Explorer View	

Cartridge	Needs	Sample URI	Resource Description	Linked Data Graph	Notes
Gowalla		example	HTML Representation	Data Explorer View	
Get Glue Meta	User Login	example	HTML Representation	Data Explorer View	
Get Glue		example	HTML Representation	Data Explorer View	
Google Buzz		example	HTML Representation	Data Explorer View	
Google Book		example	HTML Representation	Data Explorer View	Check rdfs:seeAlso links like this one
Google Plus		example	HTML Representation	Data Explorer View	
Google Places		example	HTML Representation	Data Explorer View	
Google Search		example	HTML Representation	Data Explorer View	
Google Social Graph		example	HTML Representation	Data Explorer View	
Guardian	API Key	example	HTML Representation	Data Explorer View	
Hoovers	API Key	example	HTML Representation	Data Explorer View	
Jigsaw (company)		example	HTML Representation	Data Explorer View	Check the c:location link
Jigsaw (person)		example	HTML Representation	Data Explorer View	Check several Jigsaw search seeAlso link .
Journalisted		example	HTML Representation	Data Explorer View	
Last.FM	API Key	example	HTML Representation	Data Explorer View	
LinkedIn	API Key and Session Key;	See details	example	HTML Representation	Data Explorer View
Local Search		example	HTML Representation	Data Explorer View	
LOD		example	HTML Representation	Data Explorer View	
MIME Type		example	HTML Representation	Data Explorer View	
New York Times	API Key	example	HTML Representation	Data Explorer View	
NPR Meta	API Key	example	HTML Representation	Data Explorer View	Check rdfs:seeAlso links like: link1 ; link2 ; link3 .
NYT: The Article Search		example	HTML Representation	Data Explorer View	Check the rdfs:seeAlso: link .
NYT: The TimesTags		example	HTML Representation	Data Explorer View	
OpenCalais	any html page	example	HTML Representation	Data Explorer View	
Oreilly Search for products		example	HTML Representation	Data Explorer View	
Primal		example	HTML Representation	Data Explorer View	Check the set of sioc:topic and

Cartridge	Needs	Sample URI	Resource Description	Linked Data Graph	Notes
					scot:hasScot.
ProgrammableWeb		example	HTML Representation	Data Explorer View	
Provenance		example	HTML Representation	Data Explorer View	
Punkt		example	HTML Representation	Data Explorer View	
RapLeaf		example	HTML Representation	Data Explorer View	
SameAs.org		example	HTML Representation	Data Explorer View	
Sindice		example	HTML Representation	Data Explorer View	
SimpleGeo		example	HTML Representation	Data Explorer View	
Technorati	API Key	example	HTML Representation	Data Explorer View	
Tesco Product Search	User Login, DeveloperKey, ApplicationKey	example	HTML Representation	Data Explorer View	Check set of Tesco rdfs:seeAlso links like this one .
Topsy		example	HTML Representation	Data Explorer View	Check the rdfs:seeAlso from topsy.com.
TrueKnowledge		example	HTML Representation	Data Explorer View	Check set of rdfs:seeAlso links like: link1 ; link2 ; link3 .
Tweetme		example	HTML Representation	Data Explorer View	Check the rdfs:seeAlso link.
Twitter Meta	User Login	example	HTML Representation	Data Explorer View	
uClassify		example	HTML Representation	Data Explorer View	
Uclassify		example	HTML Representation	Data Explorer View	Check diff langs uc:class: link
UMBEL	min-score, max-results	example	HTML Representation	Data Explorer View	
USA Today Best-Selling Books		example	HTML Representation	Data Explorer View	
Ustream		example	HTML Representation	Data Explorer View	Check rdfs:seeAlso links like: this one .
Virtuoso Faceted Web Service		example	HTML Representation	Data Explorer View	
voID Statistics		example	HTML Representation	Data Explorer View	
whoisi?		example	HTML Representation	Data Explorer View	
World Bank	API Key	example	HTML Representation	Data Explorer View	
WorldCat Basic Search		example	HTML Representation	Data Explorer View	Check seeAlso links like this one .
xISBN	API Key	example	HTML Representation	Data Explorer View	Check set of owl:sameAs links: link1 ; link2 .
XRD		example			

Cartridge	Needs	Sample URI	Resource Description	Linked Data Graph	Notes
			HTML Representation	Data Explorer View	
Yahoo BOSS	API Key	example	HTML Representation	Data Explorer View	
Yahoo Geocode	API Key	example	HTML Representation	Data Explorer View	
Yelp Search for business	API Key	example	HTML Representation	Data Explorer View	
Zemanta	API Key, min-score, max-results	example	HTML Representation	Data Explorer View	
Zillow	API Key	example	HTML Representation	Data Explorer View	

Meta Cartridge Usage via REST Request

Description.vsp underlies the /about/html/ page, and accepts the parameters described below.

Table 16.15.

Parameter	Value	Description	Example
@Lookup@		The type of lookup	
	No Value	When value is not given (i.e., *@Lookup@=*), all will work as if the parameter were not present. %BR% The "Lookup" name is chosen to distinguish between parameters belonging to the URL being processed, and parameters for the Sponger.	Refresh the graph with all current cartridges, either type
	0	NLP meta only	Execute only NLP meta extraction
	-2	Keywords-based only	Execute only keywords-based meta extraction
	x,y...	A list of meta cartridges to be executed, by their unique IDs. The ID column can be found in *Conductor -> Linked Data -> Sponger -> Meta Cartridges*	Execute only CNET (ID=19) and NYT: The TimesTags (ID=22) meta cartridges
refresh=0,1,2 etc.		*Usage* : for cache invalidation. When used 1 or larger number (n), adds *get:refresh "N"* (explicit refresh interval in seconds) as a directive to Sponger. A refresh of zero ("0") seconds will make a new graph on the next lookup with the '*@Lookup@* ' parameter value.	Refresh the graph with all current cartridges
refresh=clean		*Usage* : for overwriting. The 'clean' usage explicitly clears the graph i.e. will cause the Sponger to drop cache even if it is marked to be in the fly. Thus, if network resource fetched cache by some reason is left in some inconsistent state like shutdown during the fetching, then 'clean' is required as it doesn't check cache state. *Note* : must be used with caution as other threads may be doing Network Resource Fetch at same time.	

Meta Cartridges Parametrized Examples

All examples in the table below start from the same Resource, http://www.news.com, and submit it to the Sponger for processing with the single listed Meta Cartridge.

It can be informative to start by seeing what the results would be with no Meta Cartridges at all .

If you have a lot of time to spare, you may want to see what the results would be with all Meta Cartridges combined . As may be obvious, this must wait for each of the above services to respond, so it may take quite some time to return.

Table 16.16.

Cartridge	URL Pattern	Example
Alchemy	@Lookup@=8&refresh=0	cURL example
Amazon Search for products	@Lookup@=13&refresh=0	cURL example
BBC	@Lookup@=1665&refresh=0	cURL example
BestBuy Search for products	@Lookup@=14&refresh=0	cURL example
Bing	@Lookup@=11&refresh=0	cURL example
Bit.ly	@Lookup@=915&refresh=0	cURL example
CNET	@Lookup@=19&refresh=0	cURL example
Crunchbase	@Lookup@=839&refresh=0	cURL example
Dapper	@Lookup@=243&refresh=0	cURL example
DBpedia	@Lookup@=26&refresh=0	cURL example
Delicious Meta	@Lookup@=23&refresh=0	cURL example
Discogs	@Lookup@=840&refresh=0	cURL example
Document Links	@Lookup@=34&refresh=0	cURL example
eBay	@Lookup@=18&refresh=0	cURL example
Evri Meta	@Lookup@=3966&refresh=0	cURL example
Flickr Search for photos	@Lookup@=16&refresh=0	cURL example
Freebase NYTC	@Lookup@=5&refresh=0	cURL example
Freebase NYTCF	@Lookup@=4&refresh=0	cURL example
Geonames Meta	@Lookup@=24&refresh=0	cURL example
Geopoints	@Lookup@=3731&refresh=0	cURL example
Get Glue Meta	@Lookup@=25&refresh=0	cURL example
Google Search	@Lookup@=1382&refresh=0	cURL example
Google Social Graph	@Lookup@=30&refresh=0	cURL example
Guardian	@Lookup@=28&refresh=0	cURL example
Hoovers	@Lookup@=2&refresh=0	cURL example
Journalisted	@Lookup@=3174&refresh=0	cURL example
Local Search	@Lookup@=15&refresh=0	cURL example
LOD	@Lookup@=21&refresh=0	cURL example
MIME Type	@Lookup@=1029&refresh=0	cURL example
New York Times	@Lookup@=22&refresh=0	cURL example
NPR Meta	@Lookup@=29&refresh=0	cURL example
NYT: The Article Search	@Lookup@=9&refresh=0	cURL example
NYT: The TimesTags	@Lookup@=22&refresh=0	cURL example
OpenCalais	@Lookup@=1&refresh=0	cURL example
Oreilly Search for products	@Lookup@=17&refresh=0	cURL example
RapLeaf	@Lookup@=2745&refresh=0	cURL example
SameAs.org	@Lookup@=3257&refresh=0	cURL example
Sindice	@Lookup@=12&refresh=0	cURL example
Technorati	@Lookup@=27&refresh=0	cURL example
Tesco	@Lookup@=31&refresh=0	cURL example
TrueKnowledge	@Lookup@=3967&refresh=0	cURL example
Twitter	@Lookup@=4020&refresh=0	cURL example
uClassify	@Lookup@=3086&refresh=0	cURL example
UMBEL	@Lookup@=6&refresh=0	cURL example
Ustream	@Lookup@=3902&refresh=0	cURL example
Virtuoso Faceted Web Service	@Lookup@=21&refresh=0	cURL example

Cartridge	URL Pattern	Example
voID Statistics	@Lookup@=35&refresh=0	cURL example
whoisi?	@Lookup@=3052&refresh=0	cURL example
World Bank	@Lookup@=3&refresh=0	cURL example
XRD	@Lookup@=3650&refresh=0	cURL example
Yahoo BOSS	@Lookup@=10&refresh=0	cURL example
Yahoo Geocode	@Lookup@=2855&refresh=0	cURL example
Yelp Search for business	@Lookup@=20&refresh=0	cURL example
Zemanta	@Lookup@=7&refresh=0	cURL example
Zillow	@Lookup@=32&refresh=0	cURL example

16.10.9. Sponger Programmers Guide

The Sponger forms part of the extensible RDF framework built into Virtuoso Universal Server. A key component of the Sponger's pluggable architecture is its support for Sponger Cartridges, which themselves are comprised of an Entity Extractor and an Ontology Mapper. Virtuoso bundles numerous pre-written cartridges for RDF data extraction from a wide range of data sources. However, developers are free to develop their own custom cartridges. This programmer's guide describes how.

The guide is a companion to the Virtuoso Sponger whitepaper. The latter describes the Sponger in depth, its architecture, configuration, use and integration with other Virtuoso facilities such as the Open Data Services (ODS) application framework. This guide focuses solely on custom cartridge development.

Configuration of CURIEs used by the Sponger

For configuring CURIEs used by the Sponger which is exposed via Sponger clients such as "description.vsp" - the VSP based information resource description utility, you can use the `xml_set_ns_decl` function.

Here is sample example to add curie pattern:

```
-- Example link: http://linkeddata.uriburner.com/about/rdf/http://twitter.com/guykawasaki/status/11449455
XML_SET_NS_DECL ('uriburner',
                 'http://linkeddata.uriburner.com/about/rdf/http://',
                 2);
```

Cartridge Architecture

The Sponger is comprised of cartridges which are themselves comprised of an entity extractor and an ontology mapper. Entities extracted from non-RDF resources are used as the basis for generating structured data by mapping them to a suitable ontology. A cartridge is invoked through its cartridge hook, a Virtuoso/PL procedure entry point and binding to the cartridge's entity extractor and ontology mapper.

Entity Extractor

When an RDF aware client requests data from a network accessible resource via the Sponger the following events occur:

- A request is made for data in RDF form (explicitly via HTTP Accept Headers), and if RDF is returned nothing further happens.
- If RDF isn't returned, the Sponger passes the data through a

  Entity Extraction Pipeline

  (using Entity Extractors).
- The extracted data is transformed into RDF via a

  Mapping Pipeline

  . RDF instance data is generated by way of ontology matching and mapping.
- RDF instance data (aka. RDF Structured Linked Data) are returned to the client.

Extraction Pipeline

Depending on the file or format type detected at ingest, the Sponger applies the appropriate entity extractor. Detection occurs at the time of content negotiation instigated by the retrieval user agent. The normal extraction pipeline processing is as follows:

- The Sponger tries to get RDF data (including N3 or Turtle) directly from the dereferenced URL. If it finds some, it returns it, otherwise, it continues.
- If the URL refers to a HTML file, the Sponger tries to find "link" elements referring to RDF documents. If it finds one or more of them, it adds their triples into a temporary RDF graph and continues its processing.
- The Sponger then scans for microformats or GRDDL. If either is found, RDF triples are generated and added to a temporary RDF graph before continuing.
- If the Sponger finds eRDF or RDFa data in the HTML file, it extracts it from the HTML file and inserts it into the RDF graph before continuing.
- If the Sponger finds it is talking with a web service such as Google Base, it maps the API of the web service with an ontology, creates triples from that mapping and includes the triples into the temporary RDF graph.
- The next fallback is scanning of the HTML header for different Web 2.0 types or RSS 1.1, RSS 2.0, Atom, etc.
- Failing those tests, the scan then uses standard Web 1.0 rules to search in the header tags for metadata (typically Dublin Core) and transform them to RDF and again add them to the temporary graph. Other HTTP response header data may also be transformed to RDF.
- If nothing has been retrieved at this point, the ODS-Briefcase metadata extractor is tried.
- Finally, if nothing is found, the Sponger will return an empty graph.

Ontology Mapper

Sponger ontology mappers peform the the task of generating RDF instance data from extracted entities (non-RDF) using ontologies associated with a given data source type. They are typically XSLT (using GRDDL or an in-built Virtuoso mapping scheme) or Virtuoso/PL based. Virtuoso comes preconfigured with a large range of ontology mappers contained in one or more Sponger cartridges.

Cartridge Registry

To be recognized by the SPARQL engine, a Sponger cartridge must be registered in the Cartridge Registry by adding a record to the table DB.DBA.SYS_RDF_MAPPERS, either manually via DML, or more easily through Conductor, Virtuoso's browser-based administration console, which provides a UI for adding your own cartridges. (Sponger configuration using Conductor is described in detail later.) The SYS_RDF_MAPPERS table definition is as follows:

```
create table "DB"."DBA"."SYS_RDF_MAPPERS"
(
"RM_ID" INTEGER IDENTITY,  -- cartridge ID. Determines the order of the cartridge's invocation in the Spo
"RM_PATTERN" VARCHAR,  -- a REGEX pattern to match the resource URL or MIME type
"RM_TYPE" VARCHAR,  -- which property of the current resource to match: "MIME" or "URL"
"RM_HOOK" VARCHAR,  -- fully qualified Virtuoso/PL function name
"RM_KEY" LONG VARCHAR,  -- API specific key to use
"RM_DESCRIPTION" LONG VARCHAR,  -- cartridge description (free text)
"RM_ENABLED" INTEGER,  -- a 0 or 1 integer flag to exclude or include the cartridge from the Sponger proc
"RM_OPTIONS" ANY,  -- cartridge specific options
"RM_PID" INTEGER IDENTITY,
PRIMARY KEY ("RM_PATTERN", "RM_TYPE")
);
```

Cartridge Invocation

The Virtuoso SPARQL processor supports IRI dereferencing via the Sponger. If a SPARQL query references non-default graph URIs, the Sponger goes out (via HTTP) to Fetch the Network Resource data source URIs and inserts the extracted RDF data into the local RDF quad store. The Sponger invokes the appropriate cartridge for the data source type to produce RDF instance data. If none of the registered cartridges are capable of handling the received content type, the Sponger will attempt to obtain RDF instance data via the in-built WebDAV metadata extractor.

Sponger cartridges are invoked as follows:

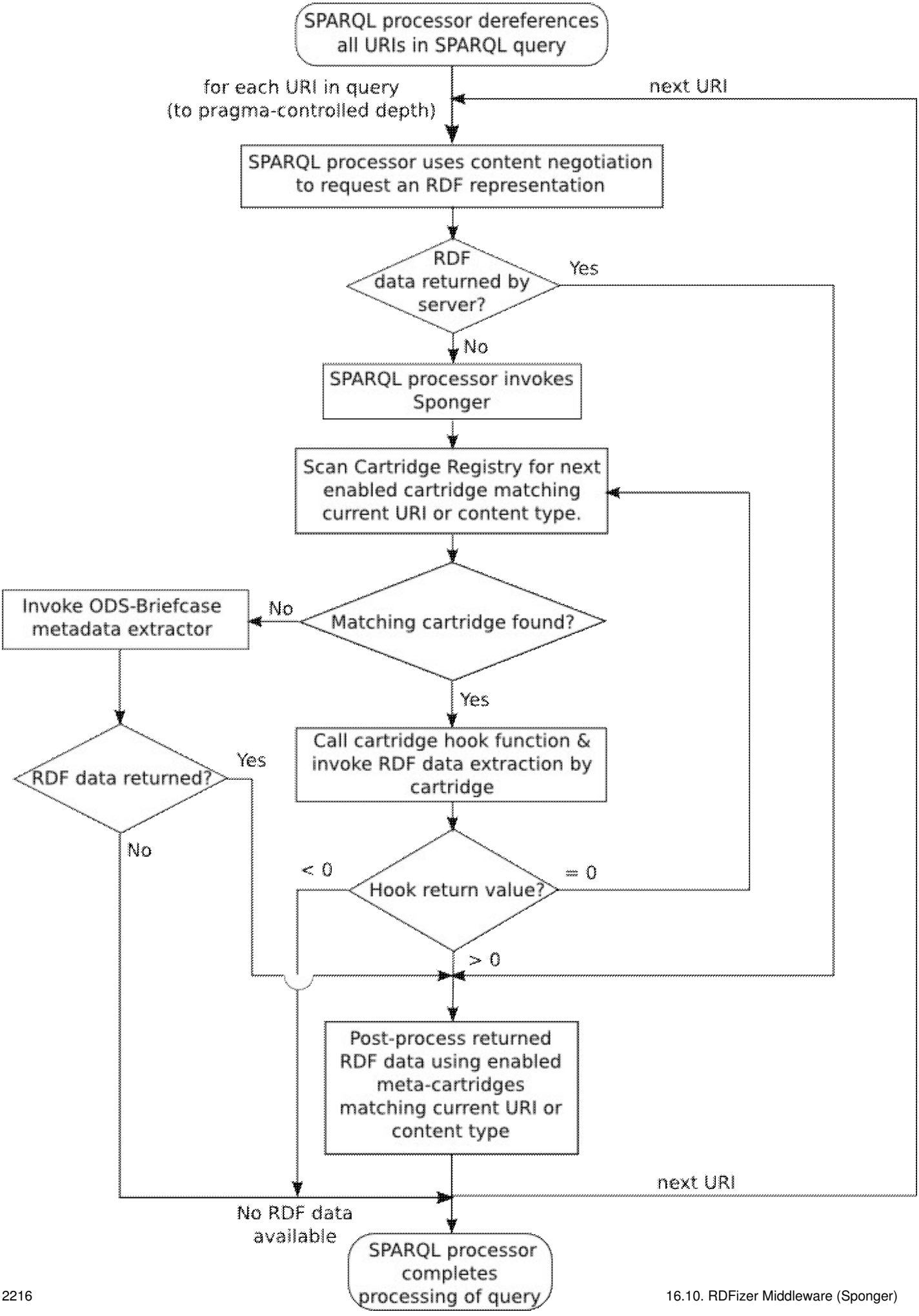
When the SPARQL processor dereferences a URI, it plays the role of an HTTP user agent (client) that makes a content type specific request to an HTTP server via the HTTP request's Accept headers. The following then occurs:

- If the content type returned is RDF then no further transformation is needed and the process stops. For instance, when consuming an (X)HTML document with a GRDDL profile, the profile URI points to a data provider that simply returns RDF instance data.
- If the content type is not RDF (i.e. application/rdf+xml or text/rdf+n3), for instance 'text/plain', the Sponger looks in the Cartridge Registry iterating over every record for which the RM_ENABLED flag is true, with the look-up sequence ordered on the RM_ID column values. For each record, the processor tries matching the content type or URL against the RM_PATTERN value and, if there is match, the function specified in RM_HOOK column is called. If the function doesn't exist, or signals an error, the SPARQL processor looks at next record.
    - If the hook returns zero, the next cartridge is tried. (A cartridge function can return zero if it believes a subsequent cartridge in the chain is capable of extracting more RDF data.)
    - If the result returned by the hook is negative, the Sponger is instructed that no RDF was generated and the process stops.
    - If the hook result is positive, the Sponger is informed that structured data was retrieved and the process stops.
- If none of the cartridges match the source data signature (content type or URL), the ODS-Briefcase WebDAV metadata extractor and RDF generator is called.

Meta-Cartridges

The above describes the RDF generation process for 'primary' Sponger cartridges. Virtuoso also supports another cartridge type - a 'meta-cartridge'. Meta-cartridges act as post-processors in the cartridge pipeline, augmenting entity descriptions in an RDF graph with additional information gleaned from 'lookup' data sources and web services. Meta-cartridges are described in more detail in a later section.

Figure 16.104. Meta-Cartridges


SPARQL processor dereferences all URIs in SPARQL query
for each URI in query (to pragma-controlled depth)
next URI
SPARQL processor uses content negotiation to request an RDF representation
RDF data returned by server?
Yes
No
SPARQL processor invokes Sponger
Scan Cartridge Registry for next enabled cartridge matching current URI or content type.
Matching cartridge found?
No
Invoke ODS-Briefcase metadata extractor
Yes
Call cartridge hook function & invoke RDF data extraction by cartridge
RDF data returned?
Yes
No
Hook return value?
< 0
= 0
> 0
Post-process returned RDF data using enabled meta-cartridges matching current URI or content type
next URI
No RDF data available
SPARQL processor completes processing of query

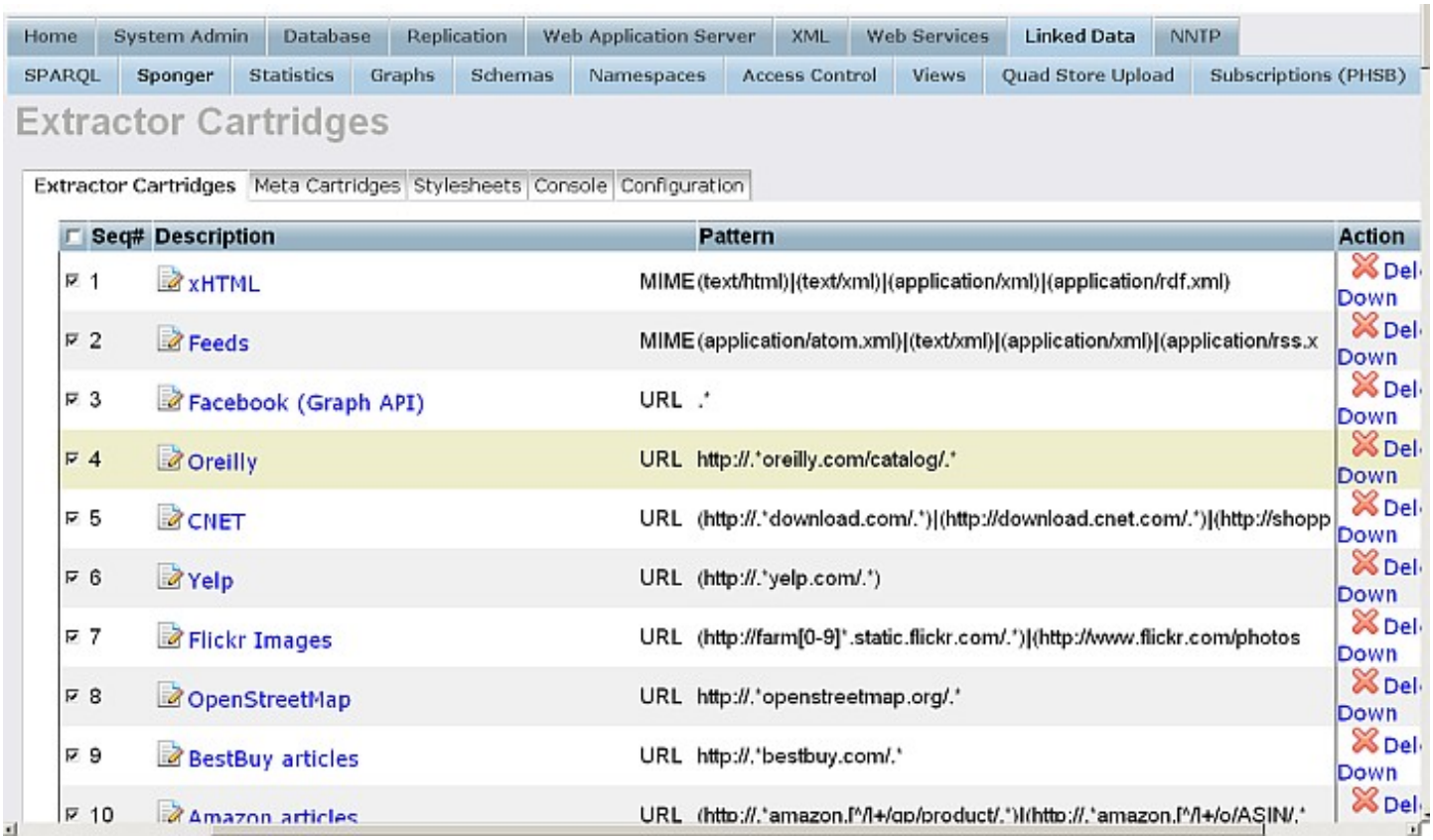
Cartridges Bundled with Virtuoso

Cartridges VAD

Virtuoso supplies a number of prewritten cartridges for extracting RDF data from a variety of popular Web resources and file types. The cartridges are bundled as part of the cartridges_dav VAD (Virtuoso Application Distribution).

To see which cartridges are available, look at the 'Linked Data' screen in Conductor. This can be reached through the Linked Data -> Sponger -> Extractor Cartridges and Meta Cartridges menu items.

Figure 16.105. RDF Cartridges



To check which version of the cartridges VAD is installed, or to upgrade it, refer to Conductor's 'VAD Packages' screen, reachable through the 'System Admin' > 'Packages' menu items.

The latest VADs for the closed source releases of Virtuoso can be downloaded from the downloads area on the OpenLink website. Select either the 'DBMS (WebDAV) Hosted' or 'File System Hosted' product format from the 'Distributed Collaborative Applications' section, depending on whether you want the Virtuoso application to use WebDAV or native filesystem storage. VADs for Virtuoso Open Source edition (VOS) are available for download from the VOS Wiki.

Example Source Code

For developers wanting example cartridge code, the most authoritative reference is the cartridges VAD source code itself. This is included as part of the VOS distribution. After downloading and unpacking the sources, the script used to create the cartridges, and the associated stylesheets can be found in:

- <vos root>/binsrc/rdf_mappers/rdf_mappers.sql
- <vos root>/binsrc/rdf_mappers/xslt/*.xsl

Alternatively, you can look at the actual cartridge implementations installed in your Virtuoso instance by inspecting the cartridge hook function used by a particular cartridge. This is easily identified from the 'Cartridge name' field of Conductor's 'RDF Cartridges' screen, after selecting the cartridge of interest. The hook function code can be viewed from the 'Schema Objects' screen under the 'Database' menu, by locating the function in the 'DB' > 'Procedures' folder. Stylesheets used by the cartridges are

installed in the WebDAV folder DAV/VAD/cartridges/xslt. This can be explored using Conductor's WebDAV interface. The actual rdf_mappers.sql file installed with your system can also be found in the DAV/VAD/cartridges folder.

Custom Cartridge

Virtuoso comes well supplied with a variety of Sponger cartridges and GRDDL filters. When then is it necessary to write your own cartridge?

In the main, writing a new cartridge should only be necessary to generate RDF from a REST-style Web service not supported by an existing cartridge, or to customize the output from an existing cartridge to your own requirements. Apart from these circumstances, the existing Sponger infrastructure should meet most of your needs. This is particularly the case for document resources.

Document Resources

We use the term document resource to identify content which is not being returned from a Web service. Normally it can broadly be conceived as some form of document, be it a text based entity or some form of file, for instance an image file.

In these cases, the document either contains RDF, which can be extracted directly, or it holds metadata in a supported format which can be transformed to RDF using an existing filter.

The following cases should all be covered by the existing Sponger cartridges:

- embedded or linked RDF
- RDFa, eRDF and other popular microformats extractable directly or via GRDDL
- popular syndication formats (RSS 2.0 , Atom, OPML , OCS , XBEL)

GRDDL

GRDDL (Gleaning Resource Descriptions from Dialects of Languages) is mechanism for deriving RDF data from XML documents and in particular XHTML pages. Document authors may associate transformation algorithms, typically expressed in XSLT, with their documents to transform embedded metadata into RDF.

The cartridges VAD installs a number of GRDDL filters for transforming popular microformats (such as RDFa, eRDF or hCalendar) into RDF. The available filters can be viewed, or configured, in Conductor's 'GRDDL Filters for XHTML' screen. Navigate to the 'RDF Cartridges' screen using the 'RDF' > 'RDF Cartridges' menu items, then SELECT the 'GRDDL Mappings' tab to display the 'GRDDL Filters for XHTML' screen. GRDDL filters are held in the WebDAV folder /DAV/VAD/rdf_cartridges/xslt/ alongside other XSLT templates. The Conductor interface allows you to add new GRDDL filters should you so wish.

For an introduction to GRDDL, try the GRDDL Primer . To underline GRDDL's utility, the primer includes an example of transforming Excel spreadsheet data, saved as XML, into RDF.

A comprehensive list of stylesheets for transforming HTML and non-HTML XML dialects is maintained on the ESW Wiki. The list covers a range of microformats, syndication formats and feedlists.

To see which Web Services are already catered for, view the list of cartridges in Conductor's 'RDF Cartridges' screen.

Creating Custom Cartridges

The Sponger is fully extensible by virtue of its pluggable cartridge architecture. New data formats can be fetched by creating new cartridges. While OpenLink is active in adding cartridges for new data sources, you are free to develop your own custom cartridges. Entity extractors can be built using Virtuoso PL, C/C++, Java or any other external language supported by Virtuoso's Server Extension API. Of course, Virtuoso's own entity extractors are written in Virtuoso PL.

The Anatomy of a Cartridge

Cartridge Hook Prototype

Every Virtuoso PL hook function used to plug a custom Sponger cartridge into the Virtuoso SPARQL engine must have a parameter list with the following parameters (the names of the parameters are not important, but their order and presence are):

- *in graph_iri varchar*

  : the IRI of the graph being retrieved/crawled
- *in new_origin_uri varchar*

  : the URL of the document being retrieved
- *in dest varchar*

  : the destination/target graph IRI
- *inout content any*

  : the content of the retrieved document
- *inout async_queue any*

  : if the PingService initialization parameter has been configured in the [SPARQL] section of the virtuoso.ini file, this is a pre-allocated asynchronous queue to be used to call the ping service
- *inout ping_service any*

  : the URL of a ping service, as assigned to the PingService parameter in the [SPARQL] section of the virtuoso.ini configuration file. PingTheSemanticWeb is an example of a such a service.
- *inout api_key any*

  : a string value specific to a given cartridge, contained in the RC_KEY column of the DB.DBA.SYS_RDF_CARTRIDGES table. The value can be a single string or a serialized array of strings providing cartridge specific data.
- *inout opts any*

  : cartridge specific options held in a Virtuoso/PL vector which acts as an array of key-value pairs.

Return Value

If the hook procedure returns zero the next cartridge will be tried. If the result is negative the sponging process stops, instructing the SPARQL engine that nothing was retrieved. If the result is positive the process stops, this time instructing the SPARQL engine that RDF data was successfully retrieved.

If your cartridge should need to test whether other cartridges are configured to handle a particular data source, the following extract taken from the RDF_LOAD_CALAIS hook procedure illustrates how you might do this:

```
if (xd is not null)
{
  -- Sponging successful. Load network resource data being fetched in the Virtuoso Quad Store:
  DB.DBA.RM_RDF_LOAD_RDFXML (xd, new_origin_uri, coalesce (dest, graph_iri));
  flag := 1;
}

declare ord any;
ord := (SELECT RM_ID FROM DB.DBA.SYS_RDF_MAPPERS WHERE
         RM_HOOK = 'DB.DBA.RDF_LOAD_CALAIS');
for SELECT RM_PATTERN FROM DB.DBA.SYS_RDF_MAPPERS WHERE
  RM_ID > ord and RM_TYPE = 'URL' and RM_ENABLED = 1 ORDER BY RM_ID do
{
  if (regexp_match (RM_PATTERN, new_origin_uri) is not null)
    -- try next candidate cartridge
    flag := 0;
}
return flag;
```

Specifying the Target Graph

Two cartridge hook function parameters contain graph IRIs, graph_iri and dest. graph_iri identifies an input graph being crawled. dest holds the IRI specified in any input:grab-destination pragma defined to control the SPARQL processor's IRI dereferencing. The pragma overrides the default behaviour and forces all retrieved triples to be stored in a single graph, irrespective of their graph of origin.

So, under some circumstances depending on how the Sponger has been invoked and whether it is being used to crawl an existing RDF graph, or derive RDF data from a non-RDF data source, dest may be null.

Consequently, when loading network resource being fetched as RDF data into the quad store, cartridges typically specify the graph to receive the data using the coalesce function which returns the first non-null parameter. e.g.

```
DB.DBA.RDF_LOAD_RDFXML (xd, new_origin_uri, coalesce (dest, graph_iri));
```

Here xd is an RDF/XML string holding the fetched RDF.

Specifying & Retrieving Cartridge Specific Options

The hook function prototype allows cartridge specific data to be passed to a cartridge through the RM_OPTIONS parameter, a Virtuoso/PL vector which acts as a heterogeneous array.

In the following example, two options are passed, 'add-html-meta' and 'get-feeds' with both values set to 'no'.

```
insert soft DB.DBA.SYS_RDF_MAPPERS (
  RM_PATTERN, RM_TYPE, RM_HOOK, RM_KEY, RM_DESCRIPTION, RM_OPTIONS
)
values (
  '(text/html)|(text/xml)|(application/xml)|(application/rdf.xml)',
  'MIME', 'DB.DBA.RDF_LOAD_HTML_RESPONSE', null, 'xHTML',
  vector ('add-html-meta', 'no', 'get-feeds', 'no')
);
```

The RM_OPTIONS vector can be handled as an array of key-value pairs using the get_keyword function. get_keyword performs a case sensitive search for the given keyword at every even index of the given array. It returns the element following the keyword, i.e. the keyword value.

Using get_keyword, any options passed to the cartridge can be retrieved using an approach similar to that below:

```
create procedure DB.DBA.RDF_LOAD_HTML_RESPONSE (
  in graph_iri varchar, in new_origin_uri varchar, in dest varchar,
  inout ret_body any, inout aq any, inout ps any, inout _key any,
  inout opts any )
{
  declare get_feeds, add_html_meta;
  ...
  get_feeds := add_html_meta := 0;
  if (isarray (opts) and 0 = mod (length(opts), 2))
  {
    if (get_keyword ('get-feeds', opts) = 'yes')
      get_feeds := 1;
    if (get_keyword ('add-html-meta', opts) = 'yes')
      add_html_meta := 1;
  }
  ...
```

XSLT - The Fulchrum

XSLT is the fulchrum of all OpenLink supplied cartridges. It provides the most convenient means of converting structured data extracted from web content by a cartridge's Entity Extractor into RDF.

Virtuoso's XML Infrastructure & Tools

Virtuoso's XML support and XSLT support are covered in detail in the on-line documentation. Virtuoso includes a highly capable XML parser and supports XPath, XQuery, XSLT and XML Schema validation.

Virtuoso supports extraction of XML documents from SQL datasets. A SQL long varchar, long xml or xmltype column in a database table can contain XML data as text or in a binary serialized format. A string representing a well-formed XML entity can be converted into an entity object representing the root node.

While Sponger cartridges will not normally concern themselves with handling XML extracted from SQL data, the ability to convert a string into an in-memory XML document is used extensively. The function xtree_doc(string) converts a string into such

a document and returns a reference to the document's root. This document together with an appropriate stylesheet forms the input for the transformation of the extracted entities to RDF using XSLT. The input string to xtree_doc generally contains structured content derived from a web service.

Virtuoso XSLT Support

Virtuoso implements XSLT 1.0 transformations as SQL callable functions. The xslt() Virtuoso/PL function applies a given stylesheet to a given source XML document and returns the transformed document. Virtuoso provides a way to extend the abilities of the XSLT processor by creating user defined XPath functions. The functions xpf_extension() and xpf_extension_remove() allow addition and removal of XPath extension functions.

General Cartridge Pipeline

The broad pipeline outlined here reflects the steps common to most cartridges:

- Redirect from the requested URL to a Web service which returns XML
- Stream the content into an in-memory XML document
- Convert it to the required RDF/XML, expressed in the chosen ontology, using XSLT
- Encode the RDF/XML as UTF-8
- Load the RDF/XML into the quad store

The MusicBrainz cartridge typifies this approach. MusicBrainz is a community music metadatabase which captures information about artists, their recorded works, and the relationships between them. Artists always have a unique ID, so the URL http://musicbrainz.org/artist/4d5447d7-c61c-4120-ba1b-d7f471d385b9.html takes you directly to entries for John Lennon.

If you were to look at this page in your browser, you would see that the information about the artist contains no RDF data. However, the cartridge is configured to intercept requests to URLs of the form http://musicbrainz.org/([^/]*)/([^.]*) and redirect to the cartridge to Fetch all the available information on the given artist, release, track or label.

The cartridge extracts entities by redirecting to the MusicBrainz XML Web Service using as the basis for the initial query the item ID, e.g. an artist or label ID, extracted from the original URL. Stripped to its essentials, the core of the cartridge is:

```
webservice_uri := sprintf ('http://musicbrainz.org/ws/1/%s/%s?type=xml&inc=%U',
                                    kind, id, inc);
content := RDF_HTTP_URL_GET (webservice_uri, '', hdr, 'GET', 'Accept: */*');
xt := xtree_doc (content);
...
xd := DB.DBA.RDF_MAPPER_XSLT (registry_get ('_cartridges_path_') || 'xslt/mbz2rdf.xsl', xt);
...
xd := serialize_to_UTF8_xml (xd);
DB.DBA.RM_RDF_LOAD_RDFXML (xd, new_origin_uri, coalesce (dest, graph_iri));
```

In the above outline, RDF_HTTP_URL_GET sends a query to the MusicBrainz web service, using query parameters appropriate for the original request, and retrieves the response using Network Resource Fetch.

The returned XML is parsed into an in-memory parse tree by xtree_doc. Virtuoso/PL function RDF_MAPPER_XSLT is a simple wrapper around the function xslt which sets the current user to dba before returning an XML document transformed by an XSLT stylesheet, in this case mbz2rdf.xsl. Function serialize_to_UTF8_xml changes the character set of the in-memory XML document to UTF8. Finally, RM_RDF_LOAD_RDFXML is a wrapper around RDF_LOAD_RDFXML which parses the content of an RDF/XML string into a sequence of RDF triples and loads them into the quad store. XSLT stylesheets are usually held in the DAV/VAD/cartridges/xslt folder of Virtuoso's WebDAV store. registry_get('cartridges_path') returns the Cartridges VAD path, 'DAV/VAD/cartridges', from the Virtuoso registry.

Error Handling with Exit Handlers

Virtuoso condition handlers determine the behaviour of a Virtuoso/PL procedure when a condition occurs. You can declare one or more condition handlers in a Virtuoso/PL procedure for general SQL conditions or specific SQLSTATE values. If a statement in your procedure raises an SQLEXCEPTION condition and you declared a handler for the specific SQLSTATE or SQLEXCEPTION condition the server passes control to that handler. If a statement in your Virtuoso/PL procedure raises an SQLEXCEPTION condition, and you have not declared a handler for the specific SQLSTATE or the SQLEXCEPTION condition, the server passes the exception to the calling procedure (if any). If the procedure call is at the top-level, then the exception is signaled to the calling client.

A number of different condition handler types can be declared (see the Virtuoso reference documentation for more details.) Of these, exit handlers are probably all you will need. An example is shown below which handles any SQLSTATE. Commented out is a debug statement which outputs the message describing the SQLSTATE.

```
create procedure DB.DBA.RDF_LOAD_SOCIALGRAPH (in graph_iri varchar, ...)
{
  declare qr, path, hdr any;
  ...
  declare exit handler for sqlstate '*'
  {
    -- dbg_printf ('%s', __SQL_MESSAGE);
    return 0;
  };
  ...
  -- data extraction and mapping successful
  return 1;
}
```

Exit handlers are used extensively in the Virtuoso supplied cartridges. They are useful for ensuring graceful failure when trying to convert content which may not conform to your expectations. The RDF_LOAD_FEED_SIOC procedure (which is used internally by several cartridges) shown below uses this approach:

```
-- /* convert the feed in rss 1.0 format to sioc */
create procedure DB.DBA.RDF_LOAD_FEED_SIOC (in content any, in iri varchar, in graph_iri varchar, in is_d
{
  declare xt, xd any;
  declare exit handler for sqlstate '*'
    {
      goto no_sioc;
    };
  xt := xtree_doc (content);
  xd := DB.DBA.RDF_MAPPER_XSLT (
      registry_get ('_cartridges_path_') || 'xslt/feed2sioc.xsl', xt,
      vector ('base', graph_iri, 'isDiscussion', is_disc));
  xd := serialize_to_UTF8_xml (xd);
  DB.DBA.RM_RDF_LOAD_RDFXML (xd, iri, graph_iri);
  return 1;
no_sioc:
  return 0;
}
```

Loading RDF into the Quad Store

RDF_LOAD_RDFXML & TTLP

The two main Virtuoso/PL functions used by the cartridges for loading RDF data into the Virtuoso quad store are DB.DBA.TTLP and DB.DBA.RDF_LOAD_RDFXML. Multithreaded versions of these functions, DB.DBA.TTLP_MT and DB.DBA.RDF_LOAD_RDFXML_MT, are also available.

RDF_LOAD_RDFXML parses the content of an RDF/XML string as a sequence of RDF triples and loads then into the quad store. TTLP parses TTL (Turtle or N3) and places its triples into quad storage. Ordinarily, cartridges use RDF_LOAD_RDFXML. However there may be occasions where you want to insert statements written as TTL, rather than RDF/XML, in which case you should use TTLP.

See Also:

- Loading RDF using API functions

Attribution

Many of the OpenLink supplied cartridges actually use RM_RDF_LOAD_RDFXML to load data into the quad store. This is a thin wrapper around RDF_LOAD_RDFXML which includes in the generated graph an indication of the external ontologies being used. The attribution takes the form:

```
<ontologyURI> a opl:DataSource .
<spongedResourceURI> rdfs:isDefinedBy <ontologyURI> .
```

```
<ontologyURI> opl:hasNamespacePrefix "<ontologyPrefix>" .
```

where prefix opl: denotes the ontology http://www.openlinksw.com/schema/attribution#.

Deleting Existing Graphs

Before loading network resource fetched RDF data into a graph, you may want to delete any existing graph with the same URI. To do so, select the 'RDF' > 'List of Graphs' menu commands in Conductor, then use the 'Delete' command for the appropriate graph. Alternatively, you can use one of the following SQL commands:

```
SPARQL CLEAR GRAPH
-- or
DELETE FROM DB.DBA.RDF_QUAD WHERE G = DB.DBA.RDF_MAKE_IID_OF_QNAME (graph_iri)
```

Proxy Service Data Expiration

When the Proxy Service is invoked by a user agent, the Sponger records the expiry date of the imported data in the table DB.DBA.SYS_HTTP_SPONGE. The data invalidation rules conform to those of traditional HTTP clients (Web browsers). The data expiration time is determined based on subsequent data fetches of the same resource. The first data retrieval records the 'expires' header. On subsequent fetches, the current time is compared to the expiration time stored in the local cache. If HTTP 'expires' header data isn't returned by the source data server, the Sponger will derive its own expiration time by evaluating the 'date' header and 'last-modified' HTTP headers.

Ontology Mapping

After extracting entities from a web resource and converting them to an in-memory XML document, the entities must be transformed to the target ontology using XSLT and an appropriate stylesheet. A typical call sequence would be:

```
xt := xtree_doc (content);
...
xd := DB.DBA.RDF_MAPPER_XSLT (registry_get ('_cartridges_path_') || 'xslt/mbz2rdf.xsl', xt);
```

Because of the wide variation in the data mapped by cartridges, it is not possible to present a typical XSL stylesheet outline. The Examples section presented later includes detailed extracts from the MusicBrainz? cartridge's stylesheet which provide a good example of how to map to an ontology. Rather than attempting to be an XSLT tutorial, the material which follows offers some general guidelines.

Passing Parameters to the XSLT Processor

Virtuoso's XSLT processor will accept default values for global parameters from the optional third argument of the xslt() function. This argument, if specified, must be a vector of parameter names and values of the form vector(name1, value1,... nameN, valueN), where name1 ... nameN must be of type varchar, and value1 ... valueN may be of any Virtuoso datatype, but may not be null.

This extract from the Crunchbase cartridge shows how parameters may be passed to the XSLT processor. The function RDF_MAPPER_XSLT (in xslt varchar, inout xt any, in params any := null) passes the parameters vector directly to xslt().

```
xt := DB.DBA.RDF_MAPPER_XSLT (
registry_get ('_cartridges_path_') || 'xslt/crunchbase2rdf.xsl', xt,
vector ('baseUri', coalesce (dest, graph_iri), 'base', base, 'suffix', suffix)
);
```

The corresponding stylesheet crunchbase2rdf.xsl retrieves the parameters baseUri, base and suffix as follows:

```
...
<xsl:output method="xml" indent="yes" />
  <xsl:variable name="ns">http://www.crunchbase.com/</xsl:variable>
  <xsl:param name="baseUri" />
  <xsl:param name="base"/>
  <xsl:param name="suffix"/>

  <xsl:template name="space-name">
...
```

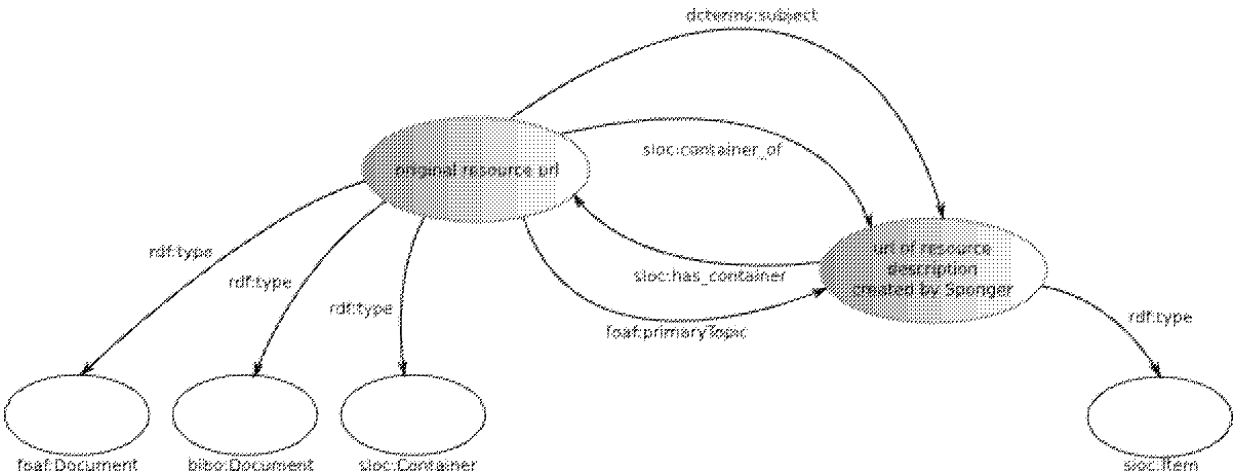
An RDF Description Template

Defining A Generic Resource Description Wrapper

Many of the OpenLink cartridges create a resource description formed to a common "wrapper" template which describes the relationship between the (usually) non-RDF source network resource being fetched and the RDF description generated by the Sponger. The wrapper is appropriate for resources which can broadly be conceived as documents. It provides a generic minimal description of the source document, but also links to the much more detailed description provided by the Sponger. So, instead of just emitting a resource description, the Sponger factors the container into the generated graph constituting the RDF description.

The template is depicted below:

Figure 16.106. Template



To generate an RDF description corresponding to the wrapper template, a stylesheet containing the following block of instructions is used. This extract is taken from the eBay cartridge's stylesheet, ebay2rdf.xsl. Many of the OpenLink cartridges follow a similar pattern.

```
  <xsl:param name="baseUri"/>
  ...
  <xsl:variable name="resourceURL">
      <xsl:value-of select="$baseUri"/>
  </xsl:variable>
  ...

<xsl:template match="/">
      <rdf:RDF>
          <rdf:Description rdf:about="{$resourceURL}">
              <rdf:type rdf:resource="Document"/>
              <rdf:type rdf:resource="Document"/>
              <rdf:type rdf:resource="Container"/>
              <sioc:container_of rdf:resource="{vi:proxyIRI ($resourceURL)}"/>
              <foaf:primaryTopic rdf:resource="{vi:proxyIRI ($resourceURL)}"/>
              <dcterms:subject rdf:resource="{vi:proxyIRI ($resourceURL)}"/>
          </rdf:Description>
          <rdf:Description rdf:about="{vi:proxyIRI ($resourceURL)}">
              <rdf:type rdf:resource="Item"/>
              <sioc:has_container rdf:resource="{$resourceURL}"/>
              <xsl:apply-templates/>
          </rdf:Description>
      </rdf:RDF>
  </xsl:template>
  ...
```

Using SIOC as a Generic Container Model

The generic resource description wrapper just described uses SIOC to establish the container/contained relationship between the source resource and the generated graph. Although the most important classes for the generic wrapper are obviously Container and Item, SIOC provides a generic data model of containers, items, item types, and associations between items which can be combined with other vocabularies such as FOAF and Dublin Core.

SIOC defines a number of other classes, such as User, UserGroup, Role, Site, Forum and Post. A separate SIOC types module (T-SIOC) extends the SIOC Core ontology by defining subclasses and subproperties of SIOC terms. Subclasses include: AddressBook, BookmarkFolder, Briefcase, EventCalendar, ImageGallery, Wiki, Weblog, BlogPost, Wiki plus many others.

OpenLink Data Spaces (ODS) uses SIOC extensively as a data space "glue" ontology to describe the base data and containment hierarchy of all the items managed by ODS applications (Data Spaces). For example, ODS-Weblog is an application of type sioc:Forum. Each ODS-Weblog application instance contains blogs of type sioct:Weblog. Each blog is a sioc:container_of posts of type sioc:Post.

Generally, when deciding how to describe resources handled by your own custom cartridge, SIOC provides a useful framework for the description which complements the SIOC-based container model adopted throughout the ODS framework.

Naming Conventions for Sponger Generated Descriptions

As can be seen from the stylesheet extract just shown, the URI of the resource description generated by the Sponger to describe the network resource being fetched, is given by the function {vi:proxyIRI ($resourceURL)} where resourceURL is the URL of the original network resource being fetched. proxyIRI is an XPath extension function defined in rdf_mappers.sql as

```
xpf_extension ('http://www.openlinksw.com/virtuoso/xslt/:proxyIRI', 'DB.DBA.RDF_SPONGE_PROXY_IRI');
```

which maps to the Virtuoso/PL procedure DB.DBA.RDF_SPONGE_PROXY_IRI. This procedure in turn generates a resource description URI which typically takes the form: http://<hostName:port>/about/html/http/<resourceURL>#this

Registering & Configuring Cartridges

Once you have developed a cartridge, you must register it in the Cartridge Registry to have the SPARQL processor recognize and use it. You should have compiled your cartridge hook function first by issuing a "create procedure DB.DBA.RDF_LOAD_xxx ..." command through one of Virtuoso's SQL interfaces. You can create the required Cartridge Registry entry either by adding a row to the SYS_REF_MAPPERS table directly using SQL, or by using the Conductor UI.

Using SQLs

If you choose register your cartridge using SQL, possibly as part of a Virtuoso/PL script, the required SQL will typically mirror one of the following INSERT commands.

Below, a cartridge for OpenCalais is being installed which will be tried when the MIME type of the network resource data being fetched is one of text/plain, text/xml or text/html. (The definition of the SYS_RDF_MAPPERS table was introduced earlier in section 'Cartridge Registry'.)

```
insert soft DB.DBA.SYS_RDF_MAPPERS (
  RM_PATTERN, RM_TYPE, RM_HOOK, RM_KEY, RM_DESCRIPTION, RM_ENABLED)
values (
  '(text/plain)|(text/xml)|(text/html)', 'MIME', 'DB.DBA.RDF_LOAD_CALAIS',
  null, 'Opencalais', 1);
```

As an alternative to matching on the content's MIME type, candidate cartridges to be tried in the conversion pipeline can be identified by matching the data source URL against a URL pattern stored in the cartridge's entry in the Cartridge Registry.

```
insert soft DB.DBA.SYS_RDF_MAPPERS (
  RM_PATTERN, RM_TYPE, RM_HOOK, RM_KEY, RM_DESCRIPTION, RM_OPTIONS)
values (
  '(http://api.crunchbase.com/v/1/.*)|(http://www.crunchbase.com/.*)', 'URL',
  'DB.DBA.RDF_LOAD_CRUNCHBASE', null, 'CrunchBase', null);
```

The value of RM_ID to set depends on where in the cartridge invocation order you want to position a particular cartridge. RM_ID should be set lower than 10028 to ensure the cartridge is tried before the ODS-Briefcase (WebDAV) metadata extractor, which is always the last mapper to be tried if no preceding cartridge has been successful.

```
UPDATE DB.DBA.SYS_RDF_MAPPERS
SET RM_ID = 1000
WHERE RM_HOOK = 'DB.DBA.RDF_LOAD_BIN_DOCUMENT';
```

Using Conductor

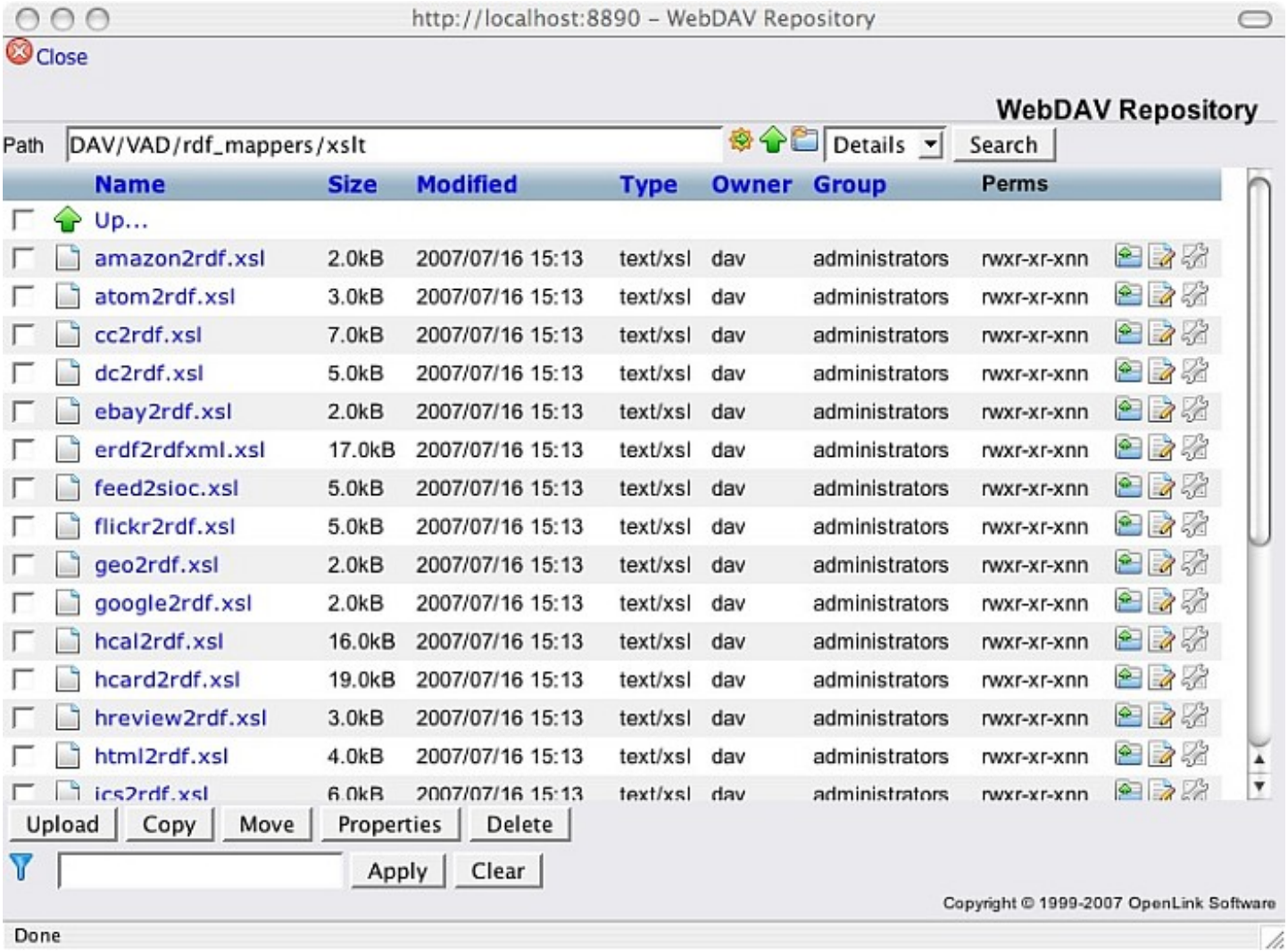
Cartridges can be added manually using the 'Add' panel of the 'RDF Cartridges' screen.

Figure 16.107. RDF Cartridges

Installing Stylesheets

Although you could place your cartridge stylesheet in any folder configured to be accessible by Virtuoso, the simplest option is to upload them to the DAV/VAD/cartridges/xslt folder using the WebDAV browser accessible from the Conductor UI.

Figure 16.108. WebDAV browser



Should you wish to locate your stylesheets elsewhere, ensure that the DirsAllowed setting in the virtuoso.ini file is configured appropriately.

Setting API Key

Some Cartridges require and API account and/or API Key to be provided for accessing the required service. This can be done from the Linked Data -> Sponger tab of the Conductor by selecting the cartridge from the list provided, entering the API Account and API Key in the dialog at the bottom of the page and click update to save, as indicated in the screenshot below:

Figure 16.109. Registering API Key

For example, for the service Flickr developers must register to obtain a key. See http://developer.yahoo.com/flickr/. In order to cater for services which require an application key, the Cartridge Registry SYS_RDF_MAPPERS table includes an RM_KEY column to store any key required for a particular service. This value is passed to the service's cartridge through the _key parameter of the cartridge hook function.

Alternatively a cartridge can store a key value in the virtuoso.ini configuration file and retrieve it in the hook function.

Flickr Cartridge

This example shows an extract from the Flickr cartridge hook function DB.DBA.RDF_LOAD_FLICKR_IMG and the use of an API key. Also, commented out, is a call to cfg_item_value() which illustrates how the API key could instead be stored and retrieved from the SPARQL section of the virtuoso.ini file.

```
create procedure DB.DBA.RDF_LOAD_FLICKR_IMG (
in graph_iri varchar, in new_origin_uri varchar, in dest varchar,
inout _ret_body any, inout aq any, inout ps any, inout _key any,
inout opts any )
{
declare xd, xt, url, tmp, api_key, img_id, hdr, exif any;
declare exit handler for sqlstate '*'
{
 return 0;
};
tmp := sprintf_inverse (new_origin_uri,
  'http://farm%s.static.flickr.com/%s/%s_%s.%s', 0);
img_id := tmp[2];
api_key := _key;
--cfg_item_value (virtuoso_ini_path (), 'SPARQL', 'FlickrAPIkey');
if (tmp is null or length (tmp) <> 5 or not isstring (api_key))
  return 0;
url :=  sprintf('http://api.flickr.com/services/rest/?method=flickr.photos.getInfo&photo_id=%s&api_key=%s
tmp := http_get (url, hdr);
```

MusicBrainz Example: A Music Metadatabase

To illustrate some of the material presented so far, we'll delve deeper into the MusicBrainz cartridge mentioned earlier.

MusicBrainz XML Web Service

The cartridge extracts data through the MusicBrainz XML Web Service using, as the basis for the initial query, an item type and MBID (MusicBrainz ID) extracted from the original URI submitted to the RDF proxy. A range of item types are supported including artist, release and track.

Using the album "Imagine" by John Lennon as an example, a standard HTML description of the album (which has an MBID of f237e6a0-4b0e-4722-8172-66f4930198bc) can be retrieved direct from MusicBrainz using the URL:

```
http://musicbrainz.org/release/f237e6a0-4b0e-4722-8172-66f4930198bc.html
```

Alternatively, information can be extracted in XML form through the web service. A description of the tracks on the album can be obtained with the query:

```
http://musicbrainz.org/ws/1/release/f237e6a0-4b0e-4722-8172-66f4930198bc?type=xml&inc=tracks
```

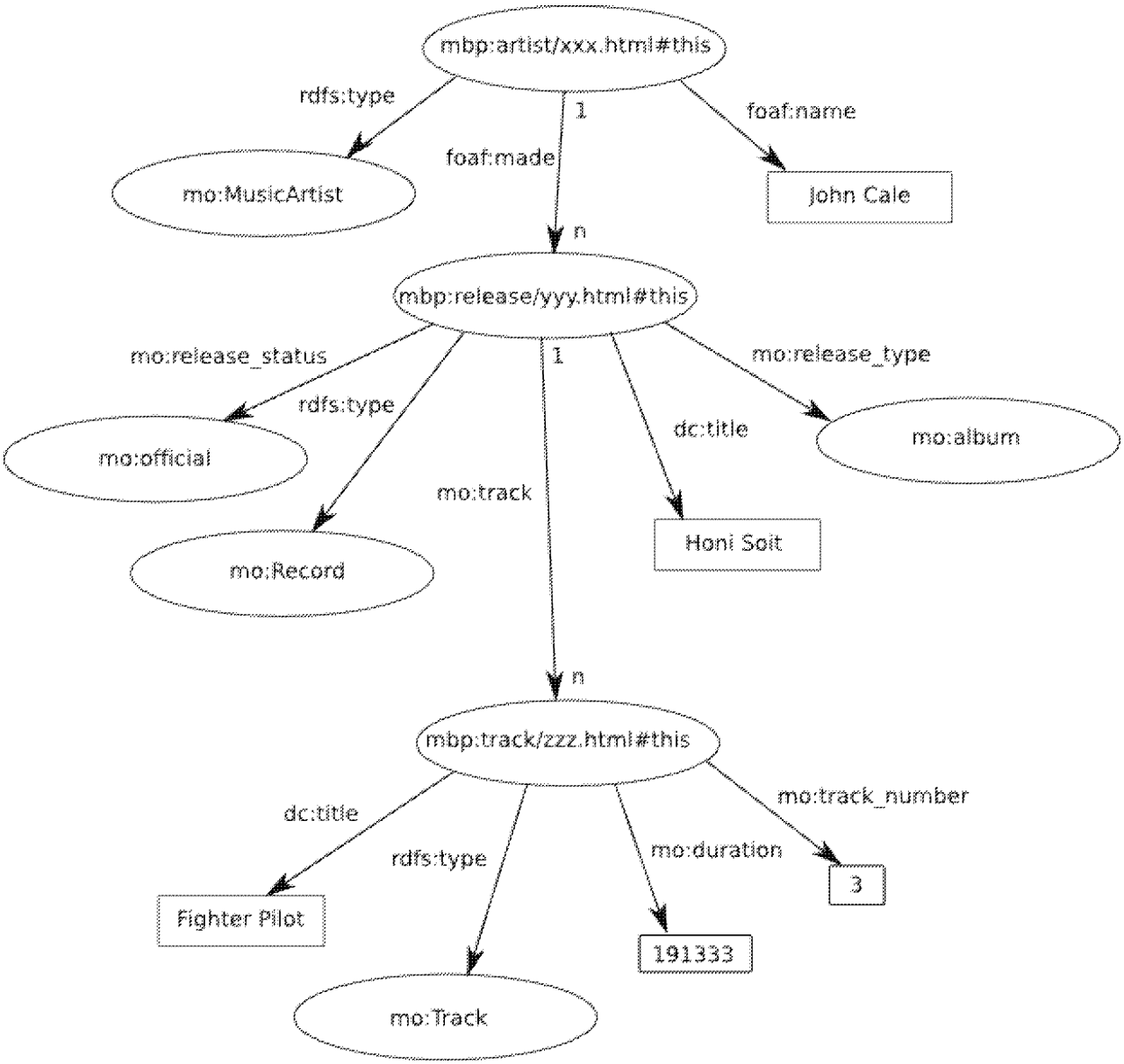
The XML returned by the web service is shown below (only the first two tracks are shown for brevity):

```
<?xml version="1.0" encoding="UTF-8"?>
  <metadata xmlns="http://musicbrainz.org/ns/mmd-1.0#"
   xmlns:ext="http://musicbrainz.org/ns/ext-1.0#">
    <release
xml:id="f237e6a0-4b0e-4722-8172-66f4930198bc" type="Album Official" >
      <title>Imagine</title>
        <text-representation language="ENG" script="Latn"/>
        <asin>B0000457L2</asin>
        <track-list>
          <track
xml:id="b88bdafd-e675-4c6a-9681-5ea85ab99446">
            <title>Imagine</title>
```

```
              <duration>182933</duration>
            </track>
            <track
xml:id="b38ce90d-3c47-4ccd-bea2-4718c4d34b0d">
              <title>Crippled Inside</title>
              <duration>227906</duration>
            </track>
            . . .
          </track-list>
        </release>
    </metadata>
```

Although, as shown above, MusicBrainz defines its own XML Metadata Format to represent music metadata, the MusicBrainz sponger converts the raw data to a subset of the Music Ontology , an RDF vocabulary which aims to provide a set of core classes and properties for describing music on the Semantic Web. Part of the subset used is depicted in the following RDF graph (representing in this case a John Cale album).

Figure 16.110. RDF graph



mbp: http://localhost:8890/proxy/rdf/http://musicbrainz.org/
mo: http://purl.org/ontology/mo/
dc: http://purl.org/dc/elements/1.1/
rdfs: http://www.w3.org/2000/01/rdf-schema#
foaf: http://xmlns.com/foaf/0.1/

Artist ID: xxx = 72c090b6-a68e-4cb9-b330-85278681a714b330b
Release ID: yyy = 9d0cadc4-69cd-4692-b6c4-9458522733e1
Track ID: zzz = 94cd2383-c828-4835-9c83-645148379136

With the prefix mo: denoting the Music Ontology at http://purl.org/ontology/mo/, it can be seen that artists are represented by instances of class mo:Artist, their albums, records etc. by instances of class mo:Release and tracks on these releases by class mo:Track. The property foaf:made links an artist and his/her releases. Property mo:track links a release with the tracks it contains

RDF Output

An RDF description of the album can be obtained by sponging the same URL, i.e. by submitting it to the Sponger's proxy interface using the URL:

```
http://demo.openlinksw.com/about/rdf/http://musicbrainz.org/release/f237e6a0-4b0e-4722-8172-66f4930198bc.
```

The extract below shows part of the (reorganized) RDF output returned by the Sponger for "Imagine". Only the album's title track is included.

```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

<rdf:Description
 rdf:about="http://musicbrainz.org/release/f237e6a0-4b0e-4722-8172-66f4930198bc.html">
  <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Document"/>
</rdf:Description>

<rdf:Description
 rdf:about="http://musicbrainz.org/release/f237e6a0-4b0e-4722-8172-66f4930198bc.html">
  <foaf:primaryTopic xmlns:foaf="http://xmlns.com/foaf/0.1/"
   rdf:resource="http://demo.openlinksw.com/about/rdf/http://musicbrainz.org/release/f237e6a0-4b0e-4722-8
</rdf:Description>

<rdf:Description rdf:about="http://purl.org/ontology/mo/">
  <rdf:type rdf:resource="http://www.openlinksw.com/schema/attribution#DataSource"/>
</rdf:Description>
...
<rdf:Description
 rdf:about="http://musicbrainz.org/release/f237e6a0-4b0e-4722-8172-66f4930198bc.html">
  <rdfs:isDefinedBy rdf:resource="http://purl.org/ontology/mo/"/>

</rdf:Description>
...

<!-- Record description -->

<rdf:Description
 rdf:about="http://demo.openlinksw.com/about/rdf/http://musicbrainz.org/release/f237e6a0-4b0e-4722-8172-6
  <rdf:type rdf:resource="http://purl.org/ontology/mo/Record"/>
</rdf:Description>

<rdf:Description
 rdf:about="http://demo.openlinksw.com/about/rdf/http://musicbrainz.org/release/f237e6a0-4b0e-4722-8172-6
  <dc:title xmlns:dc="http://purl.org/dc/elements/1.1/">Imagine</dc:title>
</rdf:Description>

<rdf:Description
 rdf:about="http://demo.openlinksw.com/about/rdf/http://musicbrainz.org/release/f237e6a0-4b0e-4722-8172-6
  <mo:release_status xmlns:mo="http://purl.org/ontology/mo/" rdf:resource="http://purl.org/ontology/mo/of
</rdf:Description>

<rdf:Description
 rdf:about="http://demo.openlinksw.com/about/rdf/http://musicbrainz.org/release/f237e6a0-4b0e-4722-8172-6
  <mo:release_type xmlns:mo="http://purl.org/ontology/mo/"
   rdf:resource="http://purl.org/ontology/mo/album"/>
</rdf:Description>

<!-- Title track description -->

<rdf:Description
 rdf:about="http://demo.openlinksw.com/about/rdf/http://musicbrainz.org/release/f237e6a0-4b0e-4722-8172-6
  <mo:track xmlns:mo="http://purl.org/ontology/mo/"
   rdf:resource="http://demo.openlinksw.com/about/rdf/http://musicbrainz.org/track/b88bdafd-e675-4c6a-968
</rdf:Description>
<rdf:Description
 rdf:about="http://demo.openlinksw.com/about/rdf/http://musicbrainz.org/track/b88bdafd-e675-4c6a-9681-5ea
  <rdf:type rdf:resource="http://purl.org/ontology/mo/Track"/>
</rdf:Description>

<rdf:Description
 rdf:about="http://demo.openlinksw.com/about/rdf/http://musicbrainz.org/track/b88bdafd-e675-4c6a-9681-5ea
  <dc:title xmlns:dc="http://purl.org/dc/elements/1.1/">Imagine</dc:title>
</rdf:Description>

<rdf:Description
 rdf:about="http://demo.openlinksw.com/about/rdf/http://musicbrainz.org/track/b88bdafd-e675-4c6a-9681-5ea
  <mo:track_number xmlns:mo="http://purl.org/ontology/mo/">1</mo:track_number>
</rdf:Description>
```

```
<rdf:Description
 rdf:about="http://demo.openlinksw.com/about/rdf/http://musicbrainz.org/track/b88bdafd-e675-4c6a-9681-5ea
  <mo:duration xmlns:mo="http://purl.org/ontology/mo/" rdf:datatype="http://www.w3.org/2001/XMLSchema#int
</rdf:Description>
</rdf:RDF>
```

Cartridge Hook Function

The cartridge's hook function is listed below. It is important to note that MusicBrainz supports a variety of query types, each of which returns a different set of information, depending on the item type being queried. Full details can be found on the MusicBrainz? site. The sponger cartridge is capable of handling all the query types supported by MusicBrainz? and is intended to be used in a drill-down scenario, as would be the case when using an RDF browser such as the OpenLink Data Explorer (ODE) . This example focuses primarily on the types release and track.

```
create procedure DB.DBA.RDF_LOAD_MBZ (
  in graph_iri varchar, in new_origin_uri varchar, in dest varchar,
  inout _ret_body any, inout aq any, inout ps any, inout _key any,
  inout opts any)
{
  declare kind, id varchar;
  declare tmp, incs any;
  declare uri, cnt, hdr, inc, xd, xt varchar;
  tmp := regexp_parse ('http://musicbrainz.org/([^/]*)/([^\.]+)', new_origin_uri, 0);
  declare exit handler for sqlstate '*'
  {
    -- dbg_printf ('%s', __SQL_MESSAGE);
    return 0;
  };
  if (length (tmp) < 6)
    return 0;

  kind := subseq (new_origin_uri, tmp[2], tmp[3]);
  id :=   subseq (new_origin_uri, tmp[4], tmp[5]);
  incs := vector ();
  if (kind = 'artist')
    {
      inc := 'aliases artist-rels label-rels release-rels track-rels url-rels';
      incs :=
        vector (
        'sa-Album', 'sa-Single', 'sa-EP', 'sa-Compilation', 'sa-Soundtrack',
        'sa-Spokenword', 'sa-Interview', 'sa-Audiobook', 'sa-Live', 'sa-Remix', 'sa-Other'
        , 'va-Album', 'va-Single', 'va-EP', 'va-Compilation', 'va-Soundtrack',

        'va-Spokenword', 'va-Interview', 'va-Audiobook', 'va-Live', 'va-Remix', 'va-Other'
        );
    }
  else if (kind = 'release')
    inc := 'artist counts release-events discs tracks artist-rels label-rels release-rels track-rels url-
  else if (kind = 'track')
    inc := 'artist releases puids artist-rels label-rels release-rels track-rels url-rels';
  else if (kind = 'label')
    inc := 'aliases artist-rels label-rels release-rels track-rels url-rels';
  else
    return 0;
  if (dest is null)
    DELETE FROM DB.DBA.RDF_QUAD WHERE G = DB.DBA.RDF_MAKE_IID_OF_QNAME (graph_iri);
  DB.DBA.RDF_LOAD_MBZ_1 (graph_iri, new_origin_uri, dest, kind, id, inc);
  DB.DBA.TTLP (sprintf ('<%S> <http://xmlns.com/foaf/0.1/primaryTopic> <%S> .\n<%S> a <http://xmlns.com/f
        new_origin_uri, DB.DBA.RDF_SPONGE_PROXY_IRI (new_origin_uri), new_origin_uri),
        '', graph_iri);
  foreach (any inc1 in incs) do
    {
      DB.DBA.RDF_LOAD_MBZ_1 (graph_iri, new_origin_uri, dest, kind, id, inc1);
    }
  return 1;
};
```

The hook function uses a subordinate procedure RDF_LOAD_MBZ_1:

```
create procedure DB.DBA.RDF_LOAD_MBZ_1 (in graph_iri varchar, in new_origin_uri varchar,
```

```
   in dest varchar, in kind varchar, in id varchar, in inc varchar)
{
  declare uri, cnt, xt, xd, hdr any;
  uri := sprintf ('http://musicbrainz.org/ws/1/%s/%s?type=xml&inc=%U', kind, id, inc);
  cnt := RDF_HTTP_URL_GET (uri, '', hdr, 'GET', 'Accept: */*');
  xt := xtree_doc (cnt);
  xd := DB.DBA.RDF_MAPPER_XSLT (registry_get ('_cartridges_path_') || 'xslt/mbz2rdf.xsl', xt,
        vector ('baseUri', new_origin_uri));
  xd := serialize_to_UTF8_xml (xd);
  DB.DBA.RM_RDF_LOAD_RDFXML (xd, new_origin_uri, coalesce (dest, graph_iri));
};
```

XSLT Stylesheet

The key sections of the MusicBrainz XSLT template relevant to this example are listed below. Only the sections relating to an artist, his releases, or the tracks on those releases, are shown.

```
<!DOCTYPE xsl:stylesheet [
<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
<!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
<!ENTITY mo "http://purl.org/ontology/mo/">
<!ENTITY foaf "http://xmlns.com/foaf/0.1/">
<!ENTITY mmd "http://musicbrainz.org/ns/mmd-1.0#">
<!ENTITY dc "http://purl.org/dc/elements/1.1/">
]>

<xsl:stylesheet
    version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:vi="http://www.openlinksw.com/virtuoso/xslt/"
    xmlns:rdf=""
    xmlns:rdfs=""
    xmlns:foaf=""
    xmlns:mo=""
    xmlns:mmd=""
    xmlns:dc=""
    >

    <xsl:output method="xml" indent="yes" />
    <xsl:variable name="base" select="'http://musicbrainz.org/'"/>
    <xsl:variable name="uc">ABCDEFGHIJKLMNOPQRSTUVWXYZ</xsl:variable>
    <xsl:variable name="lc">abcdefghijklmnopqrstuvwxyz</xsl:variable>

  <xsl:template match="/mmd:metadata">
        <rdf:RDF>
            <xsl:apply-templates />
        </rdf:RDF>
    </xsl:template>

    ...

  <xsl:template match="mmd:artist[@type='Person']">
        <mo:MusicArtist rdf:about="{vi:proxyIRI (concat($base,'artist/',@id,'.html'))}">
            <foaf:name><xsl:value-of select="mmd:name"/></foaf:name>
            <xsl:for-each select="mmd:release-list/mmd:release|mmd:relation-list[@target-type='Release']/
                <foaf:made rdf:resource="{vi:proxyIRI (concat($base,'release/',@id,'.html'))}"/>
            </xsl:for-each>
        </mo:MusicArtist>
        <xsl:apply-templates />
    </xsl:template>

  <xsl:template match="mmd:release">
        <mo:Record rdf:about="{vi:proxyIRI (concat($base,'release/',@id,'.html'))}">
            <dc:title><xsl:value-of select="mmd:title"/></dc:title>
            <mo:release_type rdf:resource="{translate (substring-before (@type, ' '),
                                                       $uc, $lc)}"/>
            <mo:release_status rdf:resource="{translate (substring-after (@type, ' '), $uc,
                                                $lc)}"/>
            <xsl:for-each select="mmd:track-list/mmd:track">
                <mo:track rdf:resource="{vi:proxyIRI (concat($base,'track/',@id,'.html'))}"/>
```

```
            </xsl:for-each>
        </mo:Record>
        <xsl:apply-templates select="mmd:track-list/mmd:track"/>
    </xsl:template>

  <xsl:template match="mmd:track">
        <mo:Track rdf:about="{vi:proxyIRI (concat($base,'track/',@id,'.html'))}">
            <dc:title><xsl:value-of select="mmd:title"/></dc:title>
            <mo:track_number><xsl:value-of select="position()"/></mo:track_number>
            <mo:duration rdf:datatype="integer">
             <xsl:value-of select="mmd:duration"/>
           </mo:duration>
            <xsl:if test="artist[@id]">
                <foaf:maker rdf:resource="{vi:proxyIRI (concat ($base, 'artist/',
                                    artist/@id, '.html'))}"/>
            </xsl:if>
            <mo:musicbrainz rdf:resource="{vi:proxyIRI (concat ($base, 'track/', @id, '.html'))}"/>
        </mo:Track>
    </xsl:template>

    ...

  <xsl:template match="text()"/>
</xsl:stylesheet>
```

Entity Extractor & Mapper Component

Used to extract RDF from a Web Data Source the Virtuoso Sponger Cartridge RDF Extractor consumes services from: Virtuoso PL, C/C++, Java based RDF Extractors

The RDF mappers provide a way to extract metadata from non-RDF documents such as HTML pages, images Office documents etc. and pass to SPARQL sponger (crawler which retrieve missing source graphs). For brevity further in this article the "RDF mapper" we simply will call "mapper".

The mappers consist of PL procedure (hook) and extractor, where extractor itself can be built using PL, C or any external language supported by Virtuoso server.

Once the mapper is developed it must be plugged into the SPARQL engine by adding a record in the table DB.DBA.SYS_RDF_MAPPERS.

If a SPARQL query instructs the SPARQL processor to retrieve target graph into local storage, then the SPARQL sponger will be invoked. If the target graph IRI represents a dereferenceable URL then content will be retrieved using content negotiation. The next step is the content type to be detected:

- If RDF and no further transformation such as GRDDL is needed, then the process would stop.
- If such as 'text/plain' and is not known to have metadata, then the SPARQL sponger will look in the DB.DBA.SYS_RDF_MAPPERS table by order of RM_ID and for every matching URL or MIME type pattern (depends on column RM_TYPE) will call the mapper hook.
    - If hook returns zero the next mapper will be tried;
    - If result is negative the process would stop instructing the SPARQL nothing was retrieved;
    - If result is positive the process would stop instructing the SPARQL that metadata was retrieved.

Virtuoso/PL based Extractors

PL hook requirements:

Every PL function used to plug a mapper into SPARQL engine must have following parameters in the same order:

- in graph_iri varchar: the graph IRI which is currently retrieved
- in new_origin_uri varchar: the URL of the document retrieved
- in destination varchar: the destination graph IRI
- inout content any: the content of the document retrieved by SPARQL sponger
- inout async_queue any: an asynchronous queue, can be used to push something to execute on background if needed.
- inout ping_service any: the value of [SPARQL] - PingService INI parameter, could be used to configure a service

notification such as pingthesemanticweb.com
- inout api_key any: a plain text id single key value or serialized vector of key structure, basically the value of RM_KEY column of the DB.DBA.SYS_RDF_MAPPERS table.

Note: the names of the parameters are not important, but their order and presence are!

Example Implementation:

In the example script below we implement a basic mapper, which maps a text/plain mime type to an imaginary ontology, which extends the class Document from FOAF with properties 'txt:UniqueWords' and 'txt:Chars', where the prefix 'txt:' we specify as 'urn:txt:v0.0:'.

```
use DB;

create procedure DB.DBA.RDF_LOAD_TXT_META
 (
  in graph_iri varchar,
  in new_origin_uri varchar,
  in dest varchar,
  inout ret_body any,
  inout aq any,
  inout ps any,
  inout ser_key any
  )
{
  declare words, chars int;
  declare vtb, arr, subj, ses, str any;
  declare ses any;
  -- if any error we just say nothing can be done
  declare exit handler for sqlstate '*'
    {
      return 0;
    };
  subj := coalesce (dest, new_origin_uri);
  vtb := vt_batch ();
  chars := length (ret_body);

  -- using the text index procedures we get a list of words
  vt_batch_feed (vtb, ret_body, 1);
  arr := vt_batch_strings_array (vtb);

  -- the list has 'word' and positions array, so we must divide by 2
  words := length (arr) / 2;
  ses := string_output ();

  -- we compose a N3 literal
  http (sprintf ('<%s> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/Docum
  http (sprintf ('<%s> <urn:txt:v0.0:UniqueWords> "%d" .\n', subj, words), ses);
  http (sprintf ('<%s> <urn:txt:v0.0:Chars> "%d" .\n', subj, chars), ses);
  str := string_output_string (ses);

  -- we push the N3 text into the local store
  DB.DBA.TTLP (str, new_origin_uri, subj);
  return 1;
};

DELETE FROM DB.DBA.SYS_RDF_MAPPERS WHERE RM_HOOK = 'DB.DBA.RDF_LOAD_TXT_META';

INSERT SOFT DB.DBA.SYS_RDF_MAPPERS (RM_PATTERN, RM_TYPE, RM_HOOK, RM_KEY, RM_DESCRIPTION)
VALUES ('(text/plain)', 'MIME', 'DB.DBA.RDF_LOAD_TXT_META', null, 'Text Files (demo)');

-- here we set order to some large number so don't break existing mappers
update DB.DBA.SYS_RDF_MAPPERS
SET RM_ID = 2000
WHERE RM_HOOK = 'DB.DBA.RDF_LOAD_TXT_META';
```

To test the mapper we just use /sparql endpoint with option 'Retrieve remote RDF data for all missing source graphs' to execute:

```
SELECT *
FROM <URL-of-a-txt-file>
```

```
WHERE { ?s ?p ?o }
```

It is important that the SPARQL_UPDATE role to be granted to "SPARQL" account in order to allow local repository update via Network Resource Fetch feature.

Authentication in Sponger

To enable usage of user defined authentication, there are added more parameters to the /proxy/rdf and /sparql endpoints. So to use it, the RDF browser and iSPARQL should send following url parameters:

- for /proxy/rdf endpoint:

```
'login=<account name>'
```

- for /sparql endpoint:

```
get-login=<account name>
```

Registry

The table DB.DBA.SYS_RDF_MAPPERS is used as registry for registering RDF mappers.

```
create table DB.DBA.SYS_RDF_MAPPERS (
    RM_ID integer identity,         -- mapper ID, designate order of execution
    RM_PATTERN varchar,             -- a REGEX pattern to match URL or MIME type
    RM_TYPE varchar default 'MIME', -- what property of the current resource to match: MIME or URL are su
    RM_HOOK varchar,                -- fully qualified PL function name e.q. DB.DBA.MY_MAPPER_FUNCTION
    RM_KEY  long varchar,           -- API specific key to use
    RM_DESCRIPTION long varchar,    -- Mapper description, free text
    RM_ENABLED integer default 1,   -- a flag 0 or 1 integer to include or exclude the given mapper from
    primary key (RM_TYPE, RM_PATTERN))
;
```

The current way to register/update/unregister a mapper is just a DML statement e.g. NSERT/UPDATE/DELETE.

Execution order and processing

When SPARQL retrieves a resource with unknown content it will look in the mappers registry and will loop over every record having RM_ENABLED flag true. The sequence of look-up is based on ordering by RM_ID column. For every record it will either try matching the MIME type or URL against RM_PATTERN value and if there is match the function specified in RM_HOOK column will be called. If the function doesn't exists or signal an error the SPARQL will look at next record.

When it stops looking? It will stop if value returned by mapper function is positive or negative number, if the return is negative processing stops with meaning no RDF was supplied, if return is positive the meaning is that RDF data was extracted, if zero integer is returned then SPARQL will look for next mapper. The mapper function also can return zero if it is expected next mapper in the chain to get more RDF data.

If none of the mappers matches the signature (MIME type nor URL) the built-in WebDAV metadata extractor will be called.

Extension function

The mapper function is a PL stored procedure with following signature:

```
THE_MAPPER_FUNCTION_NAME (
        in graph_iri varchar,
        in origin_uri varchar,
        in destination_uri varchar,
        inout content varchar,
        inout async_notification_queue any,
        inout ping_service any,
        inout keys any
        )
{
   -- do processing here
   -- return -1, 0 or 1 (as explained above in Execution order and processing section)
}
;
```

Parameters

- graph_iri - the target graph IRI
- origin_uri - the current URI of processing
- destination_uri - get:destination value
- content - the resource content
- async_notification_queue - if INI parameter PingService is specified in SPARQL section in the INI file, this is a pre-allocated asynchronous queue to be used to call ping service
- ping_service - the URL of the ping service configured in SPARQL section in the INI in PingService parameter
- keys - a string value contained in the RM_KEY column for given mapper, can be single string or serialized array, generally can be used as mapper specific data.

Return value

- 0 - no data was retrieved or some next matching mapper must extract more data
- 1 - data is retrieved, stop looking for other mappers
- -1 - no data is retrieved, stop looking for more data

Cartridges package content

The Virtuoso supply as a cartridges_dav.vad VAD package a cartridge for extracting RDF data from certain popular Web resources and file types. It can be installed (if not already) using VAD_INSTALL function, see the VAD chapter in documentation on how to do that.

HTTP-in-RDF

Maps the HTTP request response to HTTP Vocabulary in RDF, see http://www.w3.org/2006/http#.

This mapper is disabled by default. If it's enabled , it must be first in order of execution.

Also it always will return 0, which means any other mapper should push more data.

HTML

This mapper is composite, it looking for metadata which can specified in a HTML pages as follows:

- Embedded/linked RDF
  - scan for meta in RDF

    ```
    <link rel="meta" type="application/rdf+xml"
    ```
  - RDF embedded in xHTML (as markup or inside XML comments)
- Micro-formats
  - GRDDL - GRDDL Data Views: RDF expressed in XHTML and XML: http://www.w3.org/2003/g/data-view#
  - eLinked Data - http://purl.org/NET/erdf/profile
  - RDFa
  - hCard - http://www.w3.org/2006/03/hcard
  - hCalendar - http://dannyayers.com/microformats/hcalendar-profile
  - hReview - http://dannyayers.com/micromodels/profiles/hreview
  - relLicense - CC license: http://web.resource.org/cc/schema.rdf
  - Dublin Core (DCMI) - http://purl.org/dc/elements/1.1/
  - geoURL - http://www.w3.org/2003/01/geo/wgs84_pos#
  - Google Base - OpenLink Virtuoso specific mapping
  - Ning Metadata
- Feeds extraction
  - RSS/Linked Data - SIOC & AtomOWL
  - RSS 1.0 - RSS/RDF, SIOC & AtomOWL
  - Atom 1.0 - RSS/RDF, SIOC & AtomOWL
- xHTML metadata transformation using FOAF (foaf:Document) and Dublin Core properties (dc:title, dc:subject etc.)

The HTML page mapper will look for RDF data in order as listed above, it will try to extract metadata on each step and will return positive flag if any of the above step give a RDF data. In case where page URL matches some of other RDF mappers listed in registry it will return 0 so next mapper to extract more data. In order to function properly, this mapper must be executed before any other specific mappers.

Flickr URLs

This mapper extracts metadata of the Flickr images, using Flickr REST API. To function properly it must have configured key. The Flickr mapper extracts metadata using: CC license, Dublin Core, Dublin Core Metadata Terms, GeoURL, FOAF, EXIF: http://www.w3.org/2003/12/exif/ns/ ontology.

Amazon URLs

This mapper extracts metadata for Amazon articles, using Amazon REST API. It needs a Amazon API key in order to be functional.

eBay URLs

Implements eBay REST API for extracting metadata of eBay articles, it needs a key and user name to be configured in order to work.

Open Office (OO) documents

The OO documents contains metadata which can be extracted using UNZIP, so this extractor needs Virtuoso unzip plugin to be configured on the server.

Yahoo traffic data URLs

Implements transformation of the result of Yahoo traffic data to RDF.

iCal files

Transform iCal files to RDF as per http://www.w3.org/2002/12/cal/ical# .

Binary content, PDF, PowerPoint

The unknown binary content, PDF and MS PowerPoint files can be transformed to RDF using Aperture framework (http://aperture.sourceforge.net/). This mapper needs Virtuoso with Java hosting support, Aperture framework and MetaExtractor.class installed on the host system in order to work.

The Aperture framework & MetaExtractor.class must be installed on the system before to install the Cartridges VAD package . If the package is already installed, then to activate this mapper you can just re-install the VAD.

Setting-up Virtuoso with Java hosting to run Aperture framework

- Install a Virtuoso binary which includes built-in Java hosting support (The executable name will indicate whether the required hosting support is built in - a suitably enabled executable will include javavm in the name, for example virtuoso-javavm-t, rather than virtuoso-t).
- Download the Aperture framework from http://aperture.sourceforge.net.
- Unpack the contents of the framework's lib directory into an 'aperture' subdirectory of the Virtuoso working directory, i.e. of the directory containing the database and virtuoso.ini files.
- Ensure the Virtuoso working directory includes a 'lib' subdirectory containing the file MetaExtractor.class. (At the current time MetaExtractor.class in not included in the cartridges VAD. Please contact OpenLink Technical Support to obtain a copy.)
- In the [Parameters] section of the virtuoso.ini configuration file:
    - Add the line (linebreaks have been inserted for clarity):

```
JavaClasspath = lib:aperture/DFKIUtils2.jar:aperture/JempBox-0.2.0.jar:aperture/activation-
aperture/ant-compression-utils-1.7.1.jar:aperture/aperture-1.2.0.jar:aperture/aperture-exam
aperture/applewrapper-0.2.jar:aperture/bcmail-jdk14-132.jar:aperture/bcprov-jdk14-132.jar:a
aperture/commons-lang-2.3.jar:aperture/demork-2.1.jar:aperture/flickrapi-1.0.jar:aperture/f
```

```
aperture/ical4j-1.0-beta4.jar:aperture/infsail-0.1.jar:aperture/jacob-1.10.jar:aperture/jai
aperture/jcl104-over-slf4j-1.5.0.jar:aperture/jpim-0.1-aperture-1.jar:aperture/junit-3.8.1.
aperture/metadata-extractor-2.4.0-beta-1.jar:aperture/mstor-0.9.11.jar:aperture/nrlvalidato
aperture/osgi.core-4.0.jar:aperture/pdfbox-0.7.4-dev-20071030.jar:aperture/poi-3.0.2-FINAL-
aperture/rdf2go.api-4.6.2.jar:aperture/rdf2go.impl.base-4.6.2.jar:aperture/rdf2go.impl.sesa
aperture/slf4j-api-1.5.0.jar:aperture/slf4j-jdk14-1.5.0.jar:aperture/unionsail-0.1.jar:aper
```

    - Ensure DirsAllowed includes directories /tmp, (or the temporary directory for the host operating system), lib and aperture.
- Start the Virtuoso server with java hosting support
- Configure the cartridge either by installing the cartridges VAD or, if the VAD is already installed, by executing procedure DB.DBA.RDF_APERTURE_INIT.
- During the VAD installation process, RDF_APERTURE_INIT() configures the Aperture cartridge. If you look in the list of available cartridges under the RDF > Sponger tab in Conductor, you should see an entry for 'Binary Files'.

To check the cartridge has been configured, connect with Virtuoso's ISQL tool:

- Issue the command:

```
SQL> SELECT udt_is_available('APERTURE.DBA.MetaExtractor');
```

- Copy a test PDF document to the Virtuoso working directory, then execute:

```
SQL> SELECT APERTURE.DBA."MetaExtractor"().getMetaFromFile ('some_pdf_in_server_working_dir.pdf',

... some RDF data should be returned ...
```

You should now be able to Fetch all Network Resource document types supported by the Aperture framework, (using one of the standard Sponger invocation mechanisms, for instance with a URL of the form http://example.com/about/rdf/http://targethost/targetfile.pdf), subject to the MIME type pattern filters configured for the cartridge in the Conductor UI. By default the Aperture cartridge is registered to match MIME types (application/octet-stream)|(application/pdf)|(application/mspowerpoint). To Fetch all the Network Resource MIME types Aperture is capable of handling, changed the MIME type pattern to 'application/.*'.

Important: The installation guidelines presented above have been verified on Mac OS X with Aperture 1.2.0. Some adjustment may be needed for different operating systems or versions of Aperture.

Examples & tutorials

How to write own RDF mapper? Look at Virtuoso tutorial on this subject http://demo.openlinksw.com/tutorial/rdf/rd_s_1/rd_s_1.vsp .

Meta-Cartridges

So far the discussion has centered on 'primary' cartridges. However, Virtuoso supports an alternative type of cartridge, a 'meta-cartridge'. The way a meta-cartridge operates is essentially the same as a primary cartridge, that is it has a cartridge hook function with the same signature and its inserts data into the quad store through entity extraction and ontology mapping as before. Where meta-cartridges differ from primary cartridges is in their intent and their position in the cartridge invocation pipeline.

The purpose of meta-cartridges is to enrich graphs produced by other (primary) cartridges. They serve as general post-processors to add additional information about selected entities in an RDF graph. For instance, a particular meta-cartridge might be designed to search for entities of type 'umbel:Country' in a given graph, and then add additional statements about each country it finds, where the information contained in these statements is retrieved from the web service targeted by the meta-cartridge. One such example might be a 'World Bank' meta-cartridge which adds information relating to a country's GDP, its exports of goods and services as a percentage of GDP etc; retrieved using the World Bank web service API . In order to benefit from the World Bank meta-cartridge, any primary cartridge which might generate instance data relating to countries should ensure that each country instance it handles is also described as being of rdf:type 'umbel:Country'. Here, the UMBEL (Upper Mapping and Binding Exchange Layer) ontology is used as a data-source-agnostic classification system. It provides a core set of 20,000+ subject concepts which act as "a fixed set of reference points in a global knowledge space". The use of UMBEL in this way serves to decouple meta-cartridges from primary cartridges and data source specific ontologies.

Virtuoso includes two default meta-cartridges which use UMBEL and OpenCalais to augment source graphs.

Registration

Meta-cartridges must be registered in the RDF_META_CARTRIDGES table, which fulfills a role similar to the SYS_RDF_MAPPERS table used by primary cartridges. The structure of the table, and the meaning and use of its columns, are similar to SYS_RDF_MAPPERS. The meta-cartridge hook function signature is identical to that for primary cartridges.

The RDF_META_CARTRIDGES table definition is as follows:

```
create table DB.DBA.RDF_META_CARTRIDGES (
MC_ID INTEGER IDENTITY,          -- meta-cartridge ID. Determines the order of the
                                    meta-cartridge's invocation in the Sponger
                                  processing chain
MC_SEQ INTEGER IDENTITY,
MC_HOOK VARCHAR,                        -- fully qualified Virtuoso/PL function name
MC_TYPE VARCHAR,
MC_PATTERN VARCHAR,              -- a REGEX pattern to match resource URL or
                                            MIME type
MC_KEY VARCHAR,                  -- API specific key to use
MC_OPTIONS ANY,                  -- meta-cartridge specific options
MC_DESC LONG VARCHAR,            -- meta-cartridge description (free text)
MC_ENABLED INTEGER               -- a 0 or 1 integer flag to exclude or include
                                            meta-cartridge from Sponger processing chain
);
```

(At the time of writing there is no Conductor UI for registering meta-cartridges, they must be registered using SQL. A Conductor interface for this task will be added in due course.)

Invocation

Meta-cartridges are invoked through the post-processing hook procedure RDF_LOAD_POST_PROCESS which is called, for every document retrieved, after RDF_LOAD_RDFXML loads fetched data into the Quad Store.

Cartridges in the meta-cartridge registry (RDF_META_CARTRIDGES) are configured to match a given MIME type or URI pattern. Matching meta-cartridges are invoked in order of their MC_SEQ value. Ordinarily a meta-cartridge should return 0, in which case the next meta-cartridge in the post-processing chain will be invoked. If it returns 1 or -1, the post-processing stops and no further meta-cartridges are invoked.

The order of processing by the Sponger cartridge pipeline is thus:

1. Try to get RDF in the form of TTL or RDF/XML. If RDF is retrieved if go to step 3
2. Try generating RDF through the Sponger primary cartridges as before
3. Post-process the RDF using meta-cartridges in order of their MC_SEQ value. If a meta-cartridge returns 1 or -1, stop the post-processing chain.

Notice that meta-cartridges may be invoked even if primary cartridges are not.

Example - A Campaign Finance Meta-Cartridge for Freebase

Note

The example which follows builds on a Freebase Sponger cartridge developed prior to the announcement of Freebase's support for generating Linked Data through the endpoint http://rdf.freebase.com/ . The OpenLink cartridge has since evolved to reflect these changes. A snapshot of the Freebase cartridge and stylesheet compatible with this example can be found here .

Freebase is an open community database of the world's information which serves facts and statistics rather than articles. Its designers see this difference in emphasis from article-oriented databases as beneficial for developers wanting to use Freebase facts in other websites and applications.

Virtuoso includes a Freebase cartridge in the cartridges VAD. The aim of the example cartridge presented here is to provide a lightweight meta-cartridge that is used to conditionally add triples to graphs generated by the Freebase cartridge, if Freebase is describing a U.S. senator.

New York Times Campaign Finance (NYTCF) API

The New York Times Campaign Finance (NYTCF) API allows you to retrieve contribution and expenditure data based on United States Federal Election Commission filings. You can retrieve totals for a particular presidential candidate, see aggregates by ZIP

code or state, or get details on a particular donor.

The API supports a number of query types. To keep this example from being overly long, the meta-cartridge supports just one of these - a query for the candidate details. An example query and the resulting output follow:

Query:

```
http://api.nytimes.com/svc/elections/us/v2/president/2008/finances/candidates/obama,barack.xml?api-key=xx
```

Result:

```
<result_set>
 <status>OK</status>
 <copyright>
  Copyright (c) 2008 The New York Times Company. All Rights Reserved.
 </copyright>
 <results>
  <candidate>
    <candidate_name>Obama, Barack</candidate_name>
    <committee_id>C00431445</committee_id>
    <party>D</party>
    <total_receipts>468841844</total_receipts>
    <total_disbursements>391437723.5</total_disbursements>
    <cash_on_hand>77404120</cash_on_hand>
    <net_individual_contributions>426902994</net_individual_contributions>
    <net_party_contributions>150</net_party_contributions>
    <net_pac_contributions>450</net_pac_contributions>
    <net_candidate_contributions>0</net_candidate_contributions>
    <federal_funds>0</federal_funds>
    <total_contributions_less_than_200>222694981.5</total_contributions_less_than_200>
    <total_contributions_2300>76623262</total_contributions_2300>
    <net_primary_contributions>46444638.81</net_primary_contributions>
    <net_general_contributions>30959481.19</net_general_contributions>
    <total_refunds>2058240.92</total_refunds>
    <date_coverage_from>2007-01-01</date_coverage_from>
    <date_coverage_to>2008-08-31</date_coverage_to>
  </candidate>
 </results>
</result_set>
```

Sponging Freebase

Using OpenLink Data Explorer

The following instructions assume you have the OpenLink Data Explorer (ODE) browser extension installed in your browser.

An HTML description of Barack Obama can be obtained directly from Freebase by pasting the following URL into your browser: http://www.freebase.com/view/en/barack_obama

To view RDF data fetched from this page, select 'Linked Data Sources' from the browser's 'View' menu. An OpenLink Data Explorer interface will load in a new tab.

Clicking on the 'Barack Obama' link under the 'Person' category displayed by ODE fetches RDF data using the Freebase cartridge. Click the 'down arrow' adjacent to the 'Barack Obama' link to explore the retrieved data.

Assuming your Virtuoso instance is running on port 8890 on localhost, the list of data caches displayed by ODE should include: http://example.com/about/html/http/www.freebase.com/view/en/barack_obama#this

The information displayed in the rest of the page relates to the entity instance identified by this URI. The prefix http://example.com/about/html/http/ prepended to the original URI indicates that the Sponger Proxy Service has been invoked. The Sponger creates an associated entity instance (identified by the above URI with the #this suffix) which holds network resource information being fetched about the original entity.

Using the Command Line

As an alternative to ODE, you can perform Network Resource Fetch from the command line with the command:

```
curl -H "Accept: text/xml" "http://example.com/about/html/http/www.freebase.com/view/en/barack_obama"
```

To view the results, you can use Conductor's browser-based SPARQL interface (e.g. http://example.com/sparql) to query the resulting graph generated by the Sponger, http://www.freebase.com/view/en/barack_obama.

Installing the Meta-Cartridge

To register the meta-cartridge, a procedure similar to the following can be used:

```
create procedure INSTALL_RDF_LOAD_NYTCF ()
{
  -- delete any previous NYTCF cartridge installed as a primary cartridge
  DELETE FROM SYS_RDF_MAPPERS WHERE RM_HOOK = 'DB.DBA.RDF_LOAD_NYTCF';
  -- register in the meta-cartridge post-processing chain
  INSERT SOFT DB.DBA.RDF_META_CARTRIDGES (MC_PATTERN, MC_TYPE, MC_HOOK,
    MC_KEY, MC_DESC, MC_OPTIONS)
    VALUES (
    'http://www.freebase.com/view/.*',
    'URL', 'DB.DBA.RDF_LOAD_NYTCF', '2c1d95a62e5fxxxxx', 'Freebase NYTCF',
    vector ());
};
```

Looking at the list of cartridges in Conductor's 'RDF Cartridges' screen, you will see that the Freebase cartridge is configured by default to perform Network Resource Fetch of URIs which match the pattern "http://www.freebase.com/view/.*" The meta-cartridge is configured to match on the same URI pattern.

To use the Campaign Finance API, you must register and request an API key. The script above shows an invalid key. Replace it with your own key before executing the procedure.

NYTCF Meta-Cartridge Functions

The meta-cartridge function definitions are listed below. They can be executed by pasting them into Conductor's iSQL interface.

```
-- New York Times: Campaign Finance Web Service
-- See http://developer.nytimes.com/docs/campaign_finance_api

-- DB.DBA.RDF_NYTCF_LOOKUP is in effect a lightweight lookup cartridge that is used
-- to conditionally add triples to graphs generated by the Wikipedia and
-- Freebase cartridges. These cartridges call on RDF_NYTCF_LOOKUP when
-- handling an entity of rdf:type yago:Congressman109955781. The NYTCF lookup
-- cartridge (aka a metacartridge) is used to return campaign finance data
-- for the candidate in question retrieved from the New York Times Campaign
-- Finance web service.
create procedure DB.DBA.RDF_NYTCF_LOOKUP(
  in candidate_id any,          -- id of candidate
  in graph_iri varchar,         -- graph into which the additional campaign finance triples should be loa
  in api_key varchar            -- NYT finance API key
)
{
  declare version, campaign_type, year any;
  declare nyt_url, hdr, tmp any;
  declare xt, xd any;

  -- Common parameters - The NYT API only supports the following values at present:
  version := 'v2';
  campaign_type := 'president';
  year := '2008';

  -- Candidate summaries
  -- nyt_url := sprintf('http://api.nytimes.com/svc/elections/us/%s/%s/%s/finances/totals.xml?api-key=%s'
  --    version, campaign_type, year, api_key);

  -- Candidate details
  nyt_url := sprintf('http://api.nytimes.com/svc/elections/us/%s/%s/%s/finances/candidates/%s.xml?api-key
        version, campaign_type, year, candidate_id, api_key);

  tmp := http_client_ext (nyt_url, headers=>hdr, proxy=>connection_get ('sparql-get:proxy'));
  if (hdr[0] not like 'HTTP/1._ 200 %')
    signal ('22023', trim(hdr[0], '\r\n'), 'DB.DBA.RDF_LOAD_NYTCF_LOOKUP');
```

```
  xd := xtree_doc (tmp);

  -- baseUri specifies what the generated RDF description is about
  -- <rdf:Description rdf:about="{baseUri}">
  -- Example baseUri's:
  -- http://example.com/about/rdf/http://www.freebase.com/view/en/barack_obama#this
  -- http://example.com/about/rdf/http://www.freebase.com/view/en/hillary_rodham_clinton#this
  declare path any;
  declare lang, k, base_uri varchar;

  if (graph_iri like 'http://rdf.freebase.com/ns/%.%')
    base_uri := graph_iri;
  else
    {
      path := split_and_decode (graph_iri, 0, '%\0/');
      k := path [length(path) - 1];
      lang := path [length(path) - 2];

      base_uri := sprintf ('http://rdf.freebase.com/ns/%U.%U', lang, k);
    }

  xt := DB.DBA.RDF_MAPPER_XSLT (registry_get ('_cartridges_path_') || 'xslt/nytcf2rdf.xsl', xd,
        vector ('baseUri', base_uri));
  xd := serialize_to_UTF8_xml (xt);
  DB.DBA.RDF_LOAD_RDFXML (xd, '', graph_iri);
}
;

create procedure DB.DBA.RDF_MQL_RESOURCE_IS_SENATOR (
  in fb_graph_uri varchar       -- URI of graph containing Freebase resource
)
{
  -- Check if the resource described by Freebase is a U.S. senator. Only then does it make sense to query
  -- data from the NYT data space.
  --
  -- To test for senators, we start by looking for two statements in the Freebase cartridge output, simil
  --
  -- <rdf:Description rdf:about="http://example.com/about/rdf/http://www.freebase.com/view/en/hillary_rod
  --   <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
  --   <rdfs:seeAlso rdf:resource="http://en.wikipedia.org/wiki/Hillary_Rodham_Clinton"/>
  --   ...
  -- where the graph generated by the Sponger will be <http://www.freebase.com/view/en/hillary_rodham_cli
  --
  -- To test whether a resource is a senator:
  -- 1) Check whether the Freebase resource is of rdf:type foaf:Person
  -- 2) Extract the person_name from the Wikipedia URI referenced by rdfs:seeAlso
  -- 3) Use the extracted person_name to build a URI to DBpedia's description of the person.
  -- 4) Query the DBpedia description to see if the person is of rdf:type yago:Senator110578471
  declare xp, xt, tmp any;
  declare qry varchar;                  -- SPARQL query
  declare qry_uri varchar;              -- query URI
  declare qry_res varchar;              -- query result
  declare dbp_resource_name varchar;    -- Equivalent resource name in DBpedia
  declare fb_resource_uri varchar;      -- Freebase resource URI
  declare path any;
  declare lang, k varchar;

  declare exit handler for sqlstate '*' {
    return 0;
  };

  if (fb_graph_uri like 'http://rdf.freebase.com/ns/%.%')
    fb_resource_uri := fb_graph_uri;
  else
    {
      path := split_and_decode (fb_graph_uri, 0, '%\0/');
      if (length (path) < 2)
        return 0;

      k := path [length(path) - 1];
      lang := path [length(path) - 2];

      fb_resource_uri := sprintf ('http://rdf.freebase.com/ns/%U.%U', lang, k);
```

```
    }

  -- 1) Check whether the Freebase resource is a politician from united_states
  {
    declare stat, msg varchar;
    declare mdata, rset any;

    qry := sprintf ('sparql ask from <%s> where { <%s> <http://rdf.freebase.com/ns/people.person.professi
    exec (qry, stat, msg, vector(), 1, mdata, rset);
    if (length(rset) = 0 or rset[0][0] <> 1)
      return 0;
  }

  return 1;
}
;

create procedure DB.DBA.RDF_LOAD_NYTCF_META (in graph_iri varchar, in new_origin_uri varchar,  in dest va
    inout _ret_body any, inout aq any, inout ps any, inout _key any, inout opts any)
{
  declare candidate_id, candidate_name any;
  declare api_key any;
  declare indx, tmp any;
  declare ord int;

  declare exit handler for sqlstate '*'
  {
    return 0;
  };

  if (not DB.DBA.RDF_MQL_RESOURCE_IS_SENATOR (new_origin_uri))
    return 0;

  -- TO DO: hardcoded for now
  -- Need a mechanism to specify API key for meta-cartridges
  -- Could retrieve from virtuoso.ini?
  api_key := _key;

  -- NYT API supports a candidate_id in one of two forms:
  -- candidate_id ::= {candidate_ID} | {last_name [,first_name]}
  -- first_name is optional. If included, there should be no space after the comma.
  --
  -- However, because this meta cartridge supplies additional triples for the
  -- Wikipedia or Freebase cartridges, only the second form of candidate_id is
  -- supported. i.e. We extract the candidate name, rather than a numeric
  -- candidate_ID (FEC committee ID) from the Wikipedia or Freebase URL.
  --
  -- It's assumed that the source URI includes the candidate's first name.
  -- If it is omitted, the NYT API will return information about *all* candidates
  -- with that last name - something we don't want.

  indx := strstr(graph_iri, 'www.freebase.com/view/en/');
  if (indx is not null)
  {
    -- extract candidate_id from Freebase URI
    tmp := sprintf_inverse(subseq(graph_iri, indx), 'www.freebase.com/view/en/%s', 0);
    if (length(tmp) <> 1)
      return 0;
    candidate_name := tmp[0];
  }
  else
  {
    indx := strstr(graph_iri, 'wikipedia.org/wiki/');
    if (indx is not null)
    {
      -- extract candidate_id from Wikipedia URI
      tmp := sprintf_inverse(subseq(graph_iri, indx), 'wikipedia.org/%s', 0);
      if (length(tmp) <> 1)
        return 0;
      candidate_name := tmp[0];
    }
    else
      {
```

```
        tmp := sprintf_inverse(graph_iri, 'http://%s.freebase.com/ns/%s/%s', 0);
        if (length (tmp) <> 3)
          tmp := sprintf_inverse(graph_iri, 'http://%s.freebase.com/ns/%s.%s', 0);
        if (length (tmp) <> 3)
          return 0;
        candidate_name := tmp[2];
      }
  }

  -- split candidate_name into its component parts
  --   candidate_name is assumed to be firstname_[middlename_]*lastname
  --   e.g. hillary_rodham_clinton (Freebase), Hillary_clinton (Wikipedia)
  {
    declare i, _end, len int;
    declare names, tmp_name varchar;

    names := vector ();
    tmp_name := candidate_name;
    len := length (tmp_name);
    while (1)
    {
      _end := strchr(tmp_name, '_');
      if (_end is not null)
      {
        names := vector_concat (names, vector(subseq(tmp_name, 0, _end)));
        tmp_name := subseq(tmp_name, _end + 1);
      }
      else
      {
        names := vector_concat(names, vector(tmp_name));
        goto done;
      }
    }
done:
    if (length(names) < 2)
      return 0;
    -- candidate_id ::= lastname,firstname
    candidate_id := sprintf('%s,%s', names[length(names)-1], names[0]);
  }

  DB.DBA.RDF_NYTCF_LOOKUP(candidate_id, coalesce (dest, graph_iri), api_key);
  return 0;
}
;
```

NYTCF Meta-Cartridge Stylesheet

The XSLT stylesheet, nyctf2rdf.xsl, used by the meta-cartridge to transform the base Campaign Finance web service output to RDF is shown below. RDF_NYCTF_LOOKUP() assumes the stylesheet is located alongside the other stylesheets provided by the cartridges VAD in the Virtuoso WebDAV folder DAV/VAD/cartridges/xslt. You should create nyctf2rdf.xsl here from the following listing. The WebDAV Browser interface in Conductor provides the easiest means to upload the stylesheet.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xsl:stylesheet [
<!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<!ENTITY nyt "http://www.nytimes.com/">
]>

<xsl:stylesheet
    version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:vi="http://www.openlinksw.com/virtuoso/xslt/"
    xmlns:rdf=""
    xmlns:nyt=""
    >
    <xsl:output method="xml" indent="yes" />

  <xsl:template match="/result_set/status">
      <xsl:if test="text() = 'OK'">
        <xsl:apply-templates mode="ok" select="/result_set/results/candidate"/>
      </xsl:if>
    </xsl:template>
```

```
  <xsl:template match="candidate" mode="ok">
      <rdf:Description rdf:about="{vi:proxyIRI($baseUri)}">
          <nyt:candidate_name><xsl:value-of select="candidate_name"/></nyt:candidate_name>
          <nyt:committee_id><xsl:value-of select="committee_id"/></nyt:committee_id>
          <nyt:party><xsl:value-of select="party"/></nyt:party>
          <nyt:total_receipts><xsl:value-of select="total_receipts"/></nyt:total_receipts>
          <nyt:total_disbursements>
            <xsl:value-of select="total_disbursements"/>
          </nyt:total_disbursements>
          <nyt:cash_on_hand><xsl:value-of select="cash_on_hand"/></nyt:cash_on_hand>
          <nyt:net_individual_contributions>
            <xsl:value-of select="net_individual_contributions"/>
         </nyt:net_individual_contributions>
          <nyt:net_party_contributions>
            <xsl:value-of select="net_party_contributions"/>
          </nyt:net_party_contributions>
          <nyt:net_pac_contributions>
            <xsl:value-of select="net_pac_contributions"/>
          </nyt:net_pac_contributions>
          <nyt:net_candidate_contributions>
            <xsl:value-of select="net_candidate_contributions"/>
          </nyt:net_candidate_contributions>
          <nyt:federal_funds><xsl:value-of select="federal_funds"/></nyt:federal_funds>
          <nyt:total_contributions_less_than_200>
            <xsl:value-of select="total_contributions_less_than_200"/>
          </nyt:total_contributions_less_than_200>
          <nyt:total_contributions_2300>
            <xsl:value-of select="total_contributions_2300"/>
          </nyt:total_contributions_2300>
          <nyt:net_primary_contributions>
            <xsl:value-of select="net_primary_contributions"/>
          </nyt:net_primary_contributions>
          <nyt:net_general_contributions>
            <xsl:value-of select="net_general_contributions"/>
          </nyt:net_general_contributions>
          <nyt:total_refunds><xsl:value-of select="total_refunds"/></nyt:total_refunds>
          <nyt:date_coverage_from rdf:datatype="date">
            <xsl:value-of select="date_coverage_from"/>
          </nyt:date_coverage_from>
          <nyt:date_coverage_to rdf:datatype="date">
           <xsl:value-of select="date_coverage_to"/>
          </nyt:date_coverage_to>
      </rdf:Description>
    </xsl:template>

  <xsl:template match="text()|@*"/>
</xsl:stylesheet>
```

The stylesheet uses the prefix nyt: (http://www.nytimes.com) for the predicates of the augmenting triples. This has been used purely for illustration - you may prefer to define your own ontology for RDF data derived from New York Times APIs.

Testing the Meta-Cartridge

After creating the required Virtuoso/PL functions and installing the stylesheet, you should be able to test the meta-cartridge by sponging a Freebase page as described earlier using ODE or the command line. For instance:

- http://www.freebase.com/view/en/barack_obama , or
- http://www.freebase.com/view/en/hillary_rodham_clinton

You should see campaign finance data added to the graph created by the Sponger in the form of triples with predicates starting http://www.nytimes.com/xxx, e.g. http://www.nytimes.com/net_primary_contribution.

How The Meta-Cartridge Works

The comments in the meta-cartridge code detail how the cartridge works. In brief:

Given the URI of the graph being created by the Freebase cartridge, RDF_MQL_RESOURCE_IS_SENATOR checks if the resource described by Freebase is a U.S. senator. Only then does it make sense to query for campaign finance data from the

NYTCF data space.

To test for senators, the procedure starts by looking for two statements in the Freebase cartridge output similar to:

```
<rdf:Description rdf:about="http://example.com/about/rdf/http://www.freebase.com/view/en/barack_obama#thi
  <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
  <rdfs:seeAlso rdf:resource="http://en.wikipedia.org/wiki/Barack_Obama"/>
   ...
```

where the graph generated by the Sponger will be

```
<http://www.freebase.com/view/en/barack_obama>
```

To test whether a resource is a senator, RDF_MQL_RESOURCE_IS_SENATOR

- Checks whether the Freebase resource is of rdf:type foaf:Person
- Extracts the person's name from the Wikipedia URI referenced by rdfs:seeAlso
- Uses the extracted name to build a URI to DBpedia's description of the person.
- Queries the DBpedia description to see if the person is of rdf:type yago:Senator110578471 (YAGO is a semantic knowledge base which provides a core set of concepts which in turn are used by DBpedia.)

Only if this is the case is the RDF_NYTCF_LOOKUP routine called to query for and return campaign finance data for the candidate. The form of the query and the resulting XML output from the Campaign Finance service were presented earlier.

Sponger Queue API

Functions

- *DB.DBA.RDF_SPONGER_QUEUE_ADD*

  : This function is available when rdf cartridges vad is installed.

  ```
  DB.DBA.RDF_SPONGER_QUEUE_ADD  (url, options);
  ```

  - *url*

    : the URI to perform Network Resource Fetch
  - *options*

    : an array usually typical sponger pragmas, for ex:

    ```
    vector ('get:soft',  'soft',  'refresh_free_text',  1);
    ```

REST Web service

The Sponger REST Web service has the following characteristics:

- endpoint: http://cname/about/service
- parameters:
  - op=add: type of operation, for now addition to the queue is supported
  - uris=[json array]: an array of URIs to be added to the sponger queue, the format is JSON array, for example:

    ```
    { "uris":["http://www.amazon.co.uk/Hama-Stylus-Input-Apple-iPad/dp/B003O0OM0C", "http://ww
    ```

The service will return a json encoded result of the number of items added, for example:

```
{ "result":2 }
```

In case of error a JSON with error text will be returned and http status 500.

cURL example

1. Assume file.txt which contains URL encoded JSON string:

```
uris=%7B%20%22uris%22%3A%5B%22http%3A%2F%2Fwww.amazon.co.uk%2FHama-Stylus-Input-Apple-iPad%2Fdp%2F
```

2. Execute the following command:

```
curl -i -d@file.txt http://cname/about/service?op=add
HTTP/1.1 200 OK
Server: Virtuoso/06.02.3129 (Darwin) i686-apple-darwin10.0.0  VDB
Connection: Keep-Alive
Date: Thu, 05 May 2011 12:06:24 GMT
Accept-Ranges: bytes
Content-Type: applcation/json; charset="UTF-8"
Content-Length: 14

{ "result":2 }
```

Virtuoso functions usage examples

String Functions

sprintf_inverse

```
tmp := sprintf_inverse (new_origin_uri, 'http://farm%s.static.flickr.com/%s/%s_%s.%s', 0);
img_id := tmp[2];
```

split_and_decode

```
request_hdr := headers[0];
response_hdr := headers[1];
host := http_request_header (request, 'Host');
tmp := split_and_decode (request_hdr[0], 0, '\0\0 ');

http_method := tmp[0];
url := tmp[1];
protocol_version := substring (tmp[2], 6, 8);
tmp := rtrim (response_hdr[0], '\r\n');
tmp := split_and_decode (response_hdr[0], 0, '\0\0 ');
```

Retrieving URLs

http_get

```
url := sprintf('http://api.flickr.com/services/rest/?i"??
        method=flickr.photos.getInfo&photo_id=%s&api_key=%s', img_id, api_key);
tmp := http_get (url, hdr);
if (hdr[0] not like 'HTTP/1._ 200 %')
  signal ('22023', trim(hdr[0], '\r\n'), 'RDFXX');
xd := xtree_doc (tmp);
```

DB.DBA.RDF_HTTP_URL_GET

A wrapper around http_get. Retrieves a URL using the specified HTTP method (defaults to GET). The function can handle proxies, redirects (up to fifteen) and HTTPS.

```
uri := sprintf ('http://musicbrainz.org/ws/1/%s/%s?type=xml&inc=%U',
        kind, id, inc);
cnt := RDF_HTTP_URL_GET (uri, '', hdr, 'GET', 'Accept: */*');
xt := xtree_doc (cnt);
xd := DB.DBA.RDF_MAPPER_XSLT (registry_get ('_cartridges_path_') || 'xslt/mbz2rdf.xsl', xt, vector ('base
```

http_request_header

```
content := RDF_HTTP_URL_GET (rdf_url, new_origin_uri, hdr, 'GET',
                'Accept: application/rdf+xml, text/rdf+n3, */*');
ret_content_type := http_request_header (hdr, 'Content-Type', null, null);
```

Handling Non-XML Response Content

json_parse : Parses JSON content into a tree.

```
url := sprintf ('http://www.freebase.com/api/service/mqlread?queries=%U', qr);
  content := http_get (url, hdr);
  tree := json_parse (content);
  tree := get_keyword ('ROOT', tree);
  tree := get_keyword ('result', tree);
```

Writing Arbitrarily Long Text

http

```
-- Writing N3 to a string output stream using function http(), parsing the N3 into a graph, then loading
ses := string_output ();
http ('@prefix opl: <http://www.openlinksw.com/schema/attribution#> .\n', ses);
http ('@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .\n', ses);
...
DB.DBA.TTLP (ses, base, graph);
DB.DBA.RDF_LOAD_RDFXML (strg, base, graph);
```

string_output

```
ses := string_output ();
cnt := http_get (sprintf ('http://download.finance.yahoo.com/d/quotes.csv?s=%U&f=nsbavophg&e=.csv',
    symbol));
arr := rdfm_yq_parse_csv (cnt);
http ('<quote stock="NASDAQ">', ses);
foreach (any q in arr) do
  {
    http_value (q[0], 'company', ses);
    http_value (q[1], 'symbol', ses);
    ...
  }
  http ('</quote>', ses);
  content := string_output_string (ses);
  xt := xtree_doc (content);
```

string_output_string

XML & XSLT

xtree_doc

```
content := RDF_HTTP_URL_GET (uri, '', hdr, 'GET', 'Accept: */*');
xt := xtree_doc (content);
```

xpath_eval

```
profile := cast (xpath_eval ('/html/head/@profile', xt) as varchar);
```

DB.DBA.RDF_MAPPER_XSLT

```
tmp := http_get (url);
xd := xtree_doc (tmp);
xt := DB.DBA.RDF_MAPPER_XSLT (
        registry_get ('_cartridges_path_') || 'xslt/atom2rdf.xsl',
        xd, vector ('baseUri', coalesce (dest, graph_iri)));
```

Character Set Conversion

serialize_to_UTF8_xml

```
xt := DB.DBA.RDF_MAPPER_XSLT (
        registry_get ('_cartridges_path_') || 'xslt/crunchbase2rdf.xsl',
        xt, vector ('baseUri', coalesce (dest, graph_iri), 'base', base,
        'suffix', suffix));
```

```
xd := serialize_to_UTF8_xml (xt);
DB.DBA.RM_RDF_LOAD_RDFXML (xd, new_origin_uri, coalesce (dest, graph_iri));
```

Loading Data Into the Quad Store

DB.DBA.RDF_LOAD_RDFXML

```
content := RDF_HTTP_URL_GET (uri, '', hdr, 'GET', 'Accept: */*');
xt := xtree_doc (content);
xd := DB.DBA.RDF_MAPPER_XSLT (
        registry_get ('_cartridges_path_') || 'xslt/mbz2rdf.xsl',
        xt, vector ('baseUri', new_origin_uri));
xd := serialize_to_UTF8_xml (xd);
DB.DBA.RM_RDF_LOAD_RDFXML (xd, new_origin_uri, coalesce (dest, graph_iri));
```

DB.DBA.TTLP

```
sess := string_output ();
...
http (sprintf ('<http://dbpedia.org/resource/%s>
        <http://xbrlontology.com/ontology/finance/stock_market#hasCompetitor>
        <http://dbpedia.org/resource/%s> .\n',
        symbol, x), sess);
http (sprintf ('<http://dbpedia.org/resource/%s>
        <http://www.w3.org/2000/01/rdf-schema#isDefinedBy>
        <http://finance.yahoo.com/q?s=%s> .\n',
         x, x), sess);
content := string_output_string (sess);
DB.DBA.TTLP (content, new_origin_uri, coalesce (dest, graph_iri));
```

See Also:

- Loading RDF using API functions

Debug Output

dbg_obj_print

```
dbg_obj_print ('try all grddl mappings here');
```

References

- RDF Primer: http://www.w3.org/TR/2004/REC-rdf-primer-20040210/
- RDF/XML Syntax Specification: http://www.w3.org/TR/rdf-syntax-grammar/
- GRDDL Primer: http://www.w3.org/TR/grddl-primer/

PingTheSemanticWeb RDF Notification Service

PingtheSemanticWeb (PTSW) is a repository for RDF documents. The PTSW web service archives the location of recently created or updated RDF documents on the Web. It is intended for use by crawlers or other types of software agents which need to know when and where the latest updated RDF documents can be found. They can request a list of recently updated documents as a starting location to crawl the Semantic Web.

You may find this service useful for publicizing your own RDF content. Content authors can notify PTSW that an RDF document has been created or updated by pinging the service with the URL of the document. The Sponger supports this facility through the async_queue and ping_service parameters of the cartridge hook function, where the ping_service parameter contains the ping service URL as configured in the SPARQL section of the virtuoso.ini file:

```
[SPARQL]
...
PingService = http://rpc.pingthesemanticweb.com/
...
```

The configured ping service can be called using an asynchronous request and the RDF_SW_PING procedure as illustrated below.

```
create procedure DB.DBA.RDF_LOAD_HTML_RESPONSE (
```

```
  in graph_iri varchar, in new_origin_uri varchar, in dest varchar,
  inout ret_body any, inout async_queue any, inout ping_service any,
  inout _key any, inout opts any )
{
  ...
  if ( ... and async_queue is not null)
    aq_request (async_queue, 'DB.DBA.RDF_SW_PING',
                vector (ping_service, new_origin_uri));
```

For more details refer to section Asynchronous Execution and Multithreading in Virtuoso/PL

Main Namespaces used by OpenLink Cartridges

A list of the main namespaces / ontologies used by OpenLink-provided Sponger cartridges is given below. Some of these ontologies may prove useful when creating your own cartridges.

- - http://www.openlinksw.com/virtuoso/xslt/
- - http://example.com/schemas/XHTML#
- rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
- rdfs: http://www.w3.org/2000/01/rdf-schema#
- dc: http://purl.org/dc/elements/1.1/
- dcterms: http://purl.org/dc/terms/
- foaf: http://xmlns.com/foaf/0.1/
- sioc: http://rdfs.org/sioc/ns#
- sioct: http://rdfs.org/sioc/types#
- skos: http://www.w3.org/2004/02/skos/core#
- bibo: http://purl.org/ontology/bibo/

Freebase Cartridge & Stylesheet

Snapshots of the Freebase cartridge and stylesheet compatible with the meta-cartridge example presented earlier in this document can be found below.

DB.DBA.RDF_LOAD_MQL:

```
--no_c_escapes-
create procedure DB.DBA.RDF_LOAD_MQL (in graph_iri varchar, in new_origin_uri varchar,  in dest varchar,
    inout _ret_body any, inout aq any, inout ps any, inout _key any, inout opts any)
{
  declare qr, path, hdr any;
  declare tree, xt, xd, types any;
  declare k, cnt, url, sa varchar;

  hdr := null;
  sa := '';
  declare exit handler for sqlstate '*'
    {
      --dbg_printf ('%s', __SQL_MESSAGE);
      return 0;
    };

  path := split_and_decode (new_origin_uri, 0, '%\0/');
  if (length (path) < 1)
    return 0;
  k := path [length(path) - 1];
  if (path [length(path) - 2] = 'guid')
    k := sprintf ('"id":"/guid/%s"', k);
  else
  {
    if (k like '#%')
        k := sprintf ('"id":"%s"', k);
    else
      {
        sa := DB.DBA.RDF_MQL_GET_WIKI_URI (k);
    k := sprintf ('"key":"%s"', k);
  }
  }
  qr := sprintf ('{"ROOT":{"query":[{%s, "type":[]}]}}', k);
```

```
  url := sprintf ('http://www.freebase.com/api/service/mqlread?queries=%U', qr);
  cnt := http_get (url, hdr);
  tree := json_parse (cnt);
  xt := get_keyword ('ROOT', tree);
  if (not isarray (xt))
    return 0;
  xt := get_keyword ('result', xt);
  types := vector ();
  foreach (any tp in xt) do
    {
      declare tmp any;
      tmp := get_keyword ('type', tp);
      types := vector_concat (types, tmp);
    }
  --types := get_keyword ('type', xt);
  DELETE FROM DB.DBA.RDF_QUAD WHERE g =  iri_to_id(new_origin_uri);
  foreach (any tp in types) do
    {
      qr := sprintf ('{"ROOT":{"query":{%s, "type":"%s", "*":[]}}}', k, tp);
      url := sprintf ('http://www.freebase.com/api/service/mqlread?queries=%U', qr);
      cnt := http_get (url, hdr);
      --dbg_printf ('%s', cnt);
      tree := json_parse (cnt);
      xt := get_keyword ('ROOT', tree);
      xt := DB.DBA.MQL_TREE_TO_XML (tree);
      --dbg_obj_print (xt);
      xt := DB.DBA.RDF_MAPPER_XSLT (registry_get ('_cartridges_path_') || 'xslt/mql2rdf.xsl', xt,
        vector ('baseUri', coalesce (dest, graph_iri), 'wpUri', sa));
      sa := '';
      xd := serialize_to_UTF8_xml (xt);
--      dbg_printf ('%s', xd);
      DB.DBA.RM_RDF_LOAD_RDFXML (xd, new_origin_uri, coalesce (dest, graph_iri));
    }
  return 1;
}
```

mql2rdf.xsl:

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
 -
 -  $Id$
 -
 -  This file is part of the OpenLink Software Virtuoso Open-Source (VOS)
 -  project.
 -
 -  Copyright (C) 1998-2024 OpenLink Software
 -
 -  This project is free software; you can redistribute it and/or modify it
 -  under the terms of the GNU General Public License as published by the
 -  Free Software Foundation; only version 2 of the License, dated June 1991.
 -
 -  This program is distributed in the hope that it will be useful, but
 -  WITHOUT ANY WARRANTY; without even the implied warranty of
 -  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 -  General Public License for more details.
 -
 -  You should have received a copy of the GNU General Public License along
 -  with this program; if not, write to the Free Software Foundation, Inc.,
 -  51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
-->

<!DOCTYPE xsl:stylesheet [
<!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<!ENTITY bibo "http://purl.org/ontology/bibo/">
<!ENTITY xsd  "http://www.w3.org/2001/XMLSchema#">
<!ENTITY foaf "http://xmlns.com/foaf/0.1/">
<!ENTITY sioc "http://rdfs.org/sioc/ns#">
]>

<xsl:stylesheet
    version="1.0"
```

```
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:vi="http://www.openlinksw.com/virtuoso/xslt/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:sioc=""
  xmlns:bibo=""
  xmlns:foaf=""
  xmlns:skos="http://www.w3.org/2004/02/skos/core#"
  xmlns:dcterms= "http://purl.org/dc/terms/"
  xmlns:mql="http://www.freebase.com/">

  <xsl:output method="xml" indent="yes" />

  <xsl:param name="baseUri" />
  <xsl:param name="wpUri" />

  <xsl:variable name="ns">http://www.freebase.com/</xsl:variable>

<xsl:template match="/">
      <rdf:RDF>
          <xsl:if test="/results/ROOT/result/*">
              <rdf:Description rdf:about="{$baseUri}">
                  <rdf:type rdf:resource="Document"/>
                  <rdf:type rdf:resource="Document"/>
                  <rdf:type rdf:resource="Container"/>
                  <sioc:container_of rdf:resource="{vi:proxyIRI($baseUri)}"/>
                  <foaf:primaryTopic rdf:resource="{vi:proxyIRI($baseUri)}"/>
                  <dcterms:subject rdf:resource="{vi:proxyIRI($baseUri)}"/>
              </rdf:Description>
              <rdf:Description rdf:about="{vi:proxyIRI($baseUri)}">
                  <rdf:type rdf:resource="Item"/>
                  <sioc:has_container rdf:resource="{$baseUri}"/>
                  <xsl:apply-templates select="/results/ROOT/result/*"/>
                  <xsl:if test="$wpUri != ''">
                      <rdfs:seeAlso rdf:resource="{$wpUri}"/>
                  </xsl:if>
              </rdf:Description>
          </xsl:if>
      </rdf:RDF>
  </xsl:template>

<xsl:template match="*[starts-with(.,'http://') or starts-with(.,'urn:')]">
      <xsl:element namespace="{$ns}" name="{name()}">
          <xsl:attribute name="rdf:resource">
              <xsl:value-of select="vi:proxyIRI (.)"/>
          </xsl:attribute>
      </xsl:element>
  </xsl:template>

<xsl:template match="*[starts-with(.,'/')]">
      <xsl:if test="local-name () = 'type' and . like '%/person'">
          <rdf:type rdf:resource="Person"/>
      </xsl:if>
      <xsl:if test="local-name () = 'type'">
          <sioc:topic>
              <skos:Concept rdf:about="{vi:proxyIRI (concat ($ns, 'view', .))}"/>
          </sioc:topic>
      </xsl:if>

      <xsl:element namespace="{$ns}" name="{name()}">
          <xsl:attribute name="rdf:resource">
              <xsl:value-of select="vi:proxyIRI(concat ($ns, 'view', .))"/>
          </xsl:attribute>
      </xsl:element>
  </xsl:template>

<xsl:template match="*[* and ../../*]">
      <xsl:element namespace="{$ns}" name="{name()}">
          <xsl:attribute name="rdf:parseType">Resource</xsl:attribute>
          <xsl:apply-templates select="@*|node()"/>
      </xsl:element>
  </xsl:template>
```

```
  <xsl:template match="*">
        <xsl:if test="* or . != ''">
                <xsl:choose>
                    <xsl:when test="name()='image'">
                        <foaf:depiction rdf:resource="{vi:mql-image-by-name (.)}"/>
                    </xsl:when>
                    <xsl:otherwise>
                        <xsl:element namespace="{$ns}" name="{name()}">
                            <xsl:if test="name() like 'date_%'">
                                <xsl:attribute name="rdf:datatype">dateTime</xsl:attribute>
                            </xsl:if>
                            <xsl:apply-templates select="@*|node()"/>
                        </xsl:element>
                    </xsl:otherwise>
                </xsl:choose>
        </xsl:if>
    </xsl:template>
</xsl:stylesheet>
```

Using Python to perform Virtuoso Sponging

This section contains the generic steps to use Python language to extend the Virtuoso Sponger.

1. Build the latest Python hosting module. It will introduce a new function `python_exec ()` The parameters of python_exec are :

   - string containing a python code, it should define one or more functions, see remarks below
   - string containing name of function to be called
   - list of parameters for the function

   For Example:

   ```
   python_exec (file_to_string ('spoonge.py'), 'rdf4uri', 'http://url..', 'http://base...');
   ```

   The above means call the rdf4uri ('http://url..', 'http://base...') function from spoonge.py file. It is importnat to know that python_exec is restricted to DBA only group and that the python source should not have __main__ or this to be restricted in python code to not be called . Any print etc. for stdout/stderr will go on server console if server is on foreground. Can be used for debug for example but not for real work.

   The function is supposed to return just single string, don't try to return multiple results, this will not work in this revision.
2. Setup the Virtuoso server INI to include python module:

   ```
   ...
   [Plugins]
   LoadPath = ../lib
   Load1    = Hosting, hosting_python.so
   ...
   ```
3. Download and install the rdflib package from http://www.rdflib.net/ Note before to build, disable Zope interface in rdflib as this not work with C-API correctly. Or make sure Python has no Zope interfaces installed. To disable the zope in rdflib, just comment out following in <rdflibhome>/rdflib/__init__.py:

   ```
    36 #from rdflib.interfaces import IIdentifier, classImplements
    37 #classImplements(URIRef, IIdentifier)
    38 #classImplements(BNode, IIdentifier)
    39 #classImplements(Literal, IIdentifier)
   ```

   Then do:

   ```
   perl setup.py build
   perl setup.py --user install
   ```
4. Get an example of python code for sponger like: http://www.ebusiness-unibw.org/wiki/Python4Spongers and make sure you disable the last lines which not suitable for calling inside Sponger:

   ```
   ...
   #if __name__ == '__main__':
   #       rdf_xml = rdf4uri(uri='http://www.amazon.com/Apple-touch-Generation-NEWEST-MODEL/dp/B002M3
   #       print rdf_xml
   ```

Store the python code in sponge.py in server working directory. Make sure this directory is allowed to read in DirsAllowed INI setting.

5. Create a procedure and register with Sponger:

```
-- THIS IS FOR DEMO PURPOSE ONLY

-- for demo purposes we delete all other cartridges registrations to see effect from only this car
delete from DB.DBA.SYS_RDF_MAPPERS;
delete from DB.DBA.RDF_META_CARTRIDGES;

-- register cartridge
insert soft DB.DBA.SYS_RDF_MAPPERS (RM_PATTERN, RM_TYPE, RM_HOOK, RM_KEY, RM_DESCRIPTION)
        values ('(http://.*amazon.[^/]+/[^/]+/dp/[^/]+(/.*)?)', 'URL', 'DB.DBA.RDF_LOAD_PYTHON_AMA

-- the cartridge stored procedure itself
create procedure DB.DBA.RDF_LOAD_PYTHON_AMAZON_ARTICLE (in graph_iri varchar, in new_origin_uri va
    inout _ret_body any, inout aq any, inout ps any, inout _key any, inout opts any)
{
  declare result any;
  -- we check first python hosting is capable to run code
  if (__proc_exists ('python_exec', 2) is null)
    return 0;
  -- handle any error
  declare exit handler for sqlstate '*'
    {
      -- log the error
      DB.DBA.RM_RDF_SPONGE_ERROR (current_proc_name (), graph_iri, dest, __SQL_MESSAGE);
      return 0;
    };
  -- call the python code
  result := python_exec (file_to_string ('sponge.py'), 'rdf4uri', new_origin_uri);
  -- in case of python error we will get integer zero, so we check
  if (not isstring (result))
    return 0;
  -- for demo purpose we delete all from this graph
  delete from DB.DBA.RDF_QUAD where G = DB.DBA.RDF_MAKE_IID_OF_QNAME (graph_iri);
  -- load the results
  DB.DBA.RDF_LOAD_RDFXML (result, new_origin_uri, coalesce (dest, graph_iri), 0);
  return 1;
}
;
```

6. Test the Sponger code like this:

```
sparql define get:soft "soft" select * from <http://www.amazon.com/Apple-touch-Generation-NEWEST-M
```

16.10.10. Sponger and Nanotations

Situation Analysis

Since the advent of blogging, it has been clear to everyone that posts require augmentation in order to truly function as rapid-fire meme vectors. For instance, could you imagine early blog posts without tagging?

Today, we've evolved from early literal tagging (which didn't scale an iota) to wide use of `@handles` and `#hastags` . Basically, `@handles` are social network specific HTTP URIs that denote Agents (People, Organizations, and Bots) while `#hashtags` are HTTP URIs that denote Topics.

Problem: Looking at the picture above, in regards to productively encoding and decoding information via the World Wide Web medium, it should be obvious that `@handles` and `#hashtags` are basically the digital equivalents of nouns. And as a consequence, we are basically trying to replicate the power of natural language sentences without critical components such as verbs (connectors) and adjectives (classifiers).

A Solution: Leverage the power of an existing language, based on open standards, that already delivers the power of natural language without being limited by the physical constraints of paper (as a mechanism for sentence persistence and exchange).

This is where RDF comes into play. It is an open standards based language for constructing digital sentences that pack the same (or even more power) its natural language equivalents. Through the power of RDF it is possible to create micro-annotations (aka. Nanotations) that are embeddable in any kind of text based documents. Naturally, the aforementioned claim doesn't apply to every

RDF notation, which is why RDF-Turtle is the vehicle we've chosen to unleash the full power of RDF and the Semantic Web it enables when digital sentences take the form of Linked Open Data.

What is Nanotation?

Nanotation is a mechanism for using embedded digital sentences to enhance blog posts, forum discussion posts, tweets (and other micro-blogging posts), HTML, and plain text document. In addition, it turns each of the aforementioned document types into end-user oriented conduits for contributing data to public and/or private Linked Open Data clouds, on a piecemeal basis -- i.e., you turn curating and publishing Linked Open Data Cloud into a productive crowd-sourced jigsaw puzzle game.

Why is it important?

Being able to say anything, about anything, whenever, and from wherever, in a manner that's both machine and human comprehensible has always sat at the very foundations of the Semantic Web Project's value proposition. Unfortunately, confusion about RDF -- the powerful language that drives the notion of a global Semantic Web -- lead to a general bottom-up misconception whereby most perceive it as a document content format rather that an abstract language (system of signs, syntax, and semantics) exploitable using a wide variety of notations.

How do I use it?

Due to the compact nature of RDF-Turtle notation, it is possible to embed RDF statements into any text based content. The only requirements are as follows:

- Use the following as a marker for embedded RDF-Turtle based RDF statements:

```
## Turtle Start ##
## {Trutle-based-RDF-statements}
## Turtle End ##
```

- Honor the use of <> to indicate reference identifiers (absolute or relative)
- Optionally treat `@handle` as an HTTP URIs that denote Agents -- for a given data space (e.g., Twitter, LinkedIn, Facebook, Google+ etc..)
- Optionally treat `#hashtag` as an HTTP URI that denotes a Topic -- ditto .

Basic Rules (authors and processing engines)

As per natural language sentences we have the following parts:

1. *Subject*

   -- statement focal point
2. *Predicate*

   -- connection, association, link
3. *Object*

   -- value of the connection, association, link.

Each sentence subject, predicate, object is denoted (named or referred to) using an identifier (word, phrase, or term). If you want to generate Linked Data that flows across data spaces your best bet is to denote (refer to) sentence subject, predicate, and object (optionally) using identifiers that function like terms -- by using HTTP URIs.

Use prefixes to shorten RDF-Turtle statements:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
```

Enables statements such as:

```
<>
a foaf:Document .
```

Examples

1. Most basic Nanotation:

```
<> a <#Document> .
<> <#topic> <#Nanotation>.
```

2. More sophisticated Nanotation that leverages terms from existing vocabularies:

```
<> a foaf:Document .
<> foaf:topic <#Nanotation>  .
```

3. More sophisticated Nanotation that leverages commas and semicolons for statement brevity e.g. when multiple sentences have a common Subject:

```
<>
a foaf:Document;
foaf:topic <#Nanotation>.
```

4. When multiple sentences have a common Subject and Predicate but varying list of Objects:

```
<>
a foaf:Document;
foaf:topic <#Nanotation>, <#SemanticWeb>, <#LinkedData>.
```

5. Incorporation of Pronouns into RDF sentence:

```
<> a foaf:Document .
<> foaf:maker [a foaf:Person;
               foaf:name "Kingsley Idehen"  ] .
```

6. Processor (parser) hint markers that help Nanotation processors negate seriously mangled HTML content:

```
## Turtle Start ##

<>
a foaf:Document;
foaf:topic <#Nanotation>, <#SemanticWeb>, <#LinkedData>.

## Turtle End ##
```

Nanotation Processor Usage

A nanotation processer is an application or service that's capable of consuming text content enhanced with RDF-Turtle based nanotations. Virtuoso's in-built Linked Data Transformation middleware (aka "Sponger") is an example of an application that supports nanotation. Likewise, our URIBurner service which is a free public service driven by an instance of Virtuoso with the Sponger module enabled:

- When using your own instance of Virtuoso, the Sponger service is invoked via the URL pattern:

```
http://{cname-of-virtuoso-host-machine}/sponger
```

- When using the public URIBurner service , the Sponger services is invoked via the URL:

```
http://linkeddata.uriburner.com/
```

- Note: Either approach outlined above will lead you to an HTML page that contains an input field into which you can type or paste an HTTP-accessible document URL. Alternatively you can use the following URI patterns:

```
http://{cname-of-virtuoso-host-machine}/about/html/{document-http-uri}
```

In all cases, you will end up with an HTML document that includes RDF statements that describe the processed document in a manner that also reveals all the embedded nanotations.

Virtuoso Sponger Implementation Notes

The Sponger treats resources transferred over HTTP as a duality of both a container document and a primary entity. When a resource is deemed to be an HTML document, the document is treated as the primary entity. Otherwise, where the domain is well known, a custom extractor cartridge populates the primary entity with data arising from API calls and the HTML content is regarded as secondary, relegated to the container document. For example, G+ posts are recognized and the Sponger concentrates on presenting the timestamp, body, tags and links and other features of a post.

When sponging an HTTP resource, multiple extractor cartridges might be brought to bear. Consequently, there may be multiple triples containing the entity's content.

The Turtle Sniffer is implemented as a Metacartridge, i.e it runs after all the extractor cartridges have run, augmenting data in the graph. It uses SPARQL inference to collate predicates that constitute "content" for this purpose, along with the HTTP request content (if any), flattening each to plain text.

Currently, the list of potential content predicates is:

- `bibo:content` (e.g. arising from the HTML+Variants extractor cartridge)
- `bibo:abstract`
- `oplgplus:annotation` (used by Google+ for text when sharing items)

For each of these contents, it checks if it matches the patterns:

```
## Nanotation Start ## .... ## Nannotation End ##

## Turtle Start ## .... ## Turtle Stop ##

{.... } (note: only applies to tweets on Twitter)
```

If a content contains one or more nanotation blocks, each block is parsed in turn as Turtle; if not, it attempts to parse the content item as Turtle in entirety.

The HTTP document content is only inspected in case of no triples having been extracted by prior means.

Optionally (enabled by default) each triple may be reified, i.e an rdf:Statement entity created to describe its subject, predicate and object, so you can identify triples arising from nanotations as entities labelled 'Embedded Turtle Statement' and a number in the graph.

Domain-Aware Tag and User Expansion

The Turtle Sniffer expands the patterns `#word` and `@word` when they appear in URI (<>) or double quotes (""), in the context of the domain of the URI being sponged.

For example, a Tweet containing the nanotation:

```
## Nannotation Start ##
<@kidehen> foaf:name "Kingsley Idehen" ;
           foaf:knows <@openlink> ;
           scot:has_tag <#Data> .
## Nannotation End ##
```

will be expanded to a Turtle string:

```
<https://twitter.com/kidehen> foaf:name "Kingsley Idehen" ;
  foaf:knows <https://twitter.com/openlink> ;
  scot:has_tag <https://twitter.com/hashtag/Data#this> .
```

We recognize custom URI formats for users and tags in the contexts of Facebook, Twitter, G+, LinkedIn and Delicious.

Note that the word must appear within quotes -- this is to avoid confusion with Turtle's @prefix directive (which is not a user!) and problems that would be caused by performing similar expansions within a quoted sentence.

Live Examples

- RDF statement about Privacy;
- RDFs usage re. property/predicate description;
- owl:sameAs relation inserted into a Google+ Post;
- Heavy duty RDF statements in a Facebook post;
- Another Facebook post that puts Ted Nelsons talk on documents and hypertext into contemporary context;
- A tweet reply demonstrating multiple embedded nanotation blocks and tags.

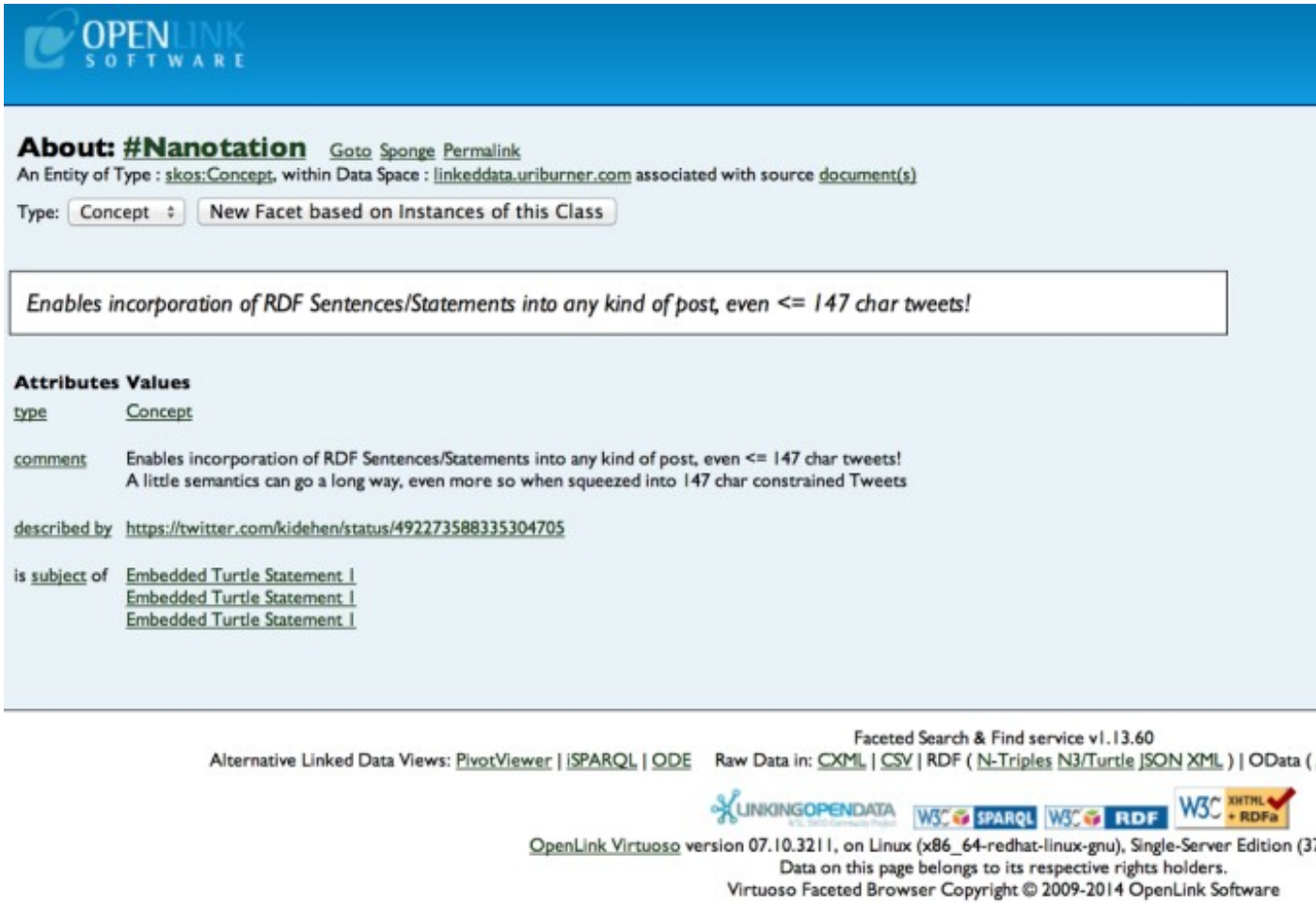
Nanotation generated 5-Star Linked Open Data (via URIBurner - a Nanotation Processor) Examples

- Various Nanotations from 28th July 2014;
- Various Nanotations from 27th July 2014;
- Various Nanotations from 25th July 2014;
- A tweet reply demonstrating multiple embedded nanotation blocks and tags.

Faceted Browser Nanotation Examples

- Nanotation based RDF statement that describes Nannotation:

Figure 16.111. Faceted Browser Nanotation RDF

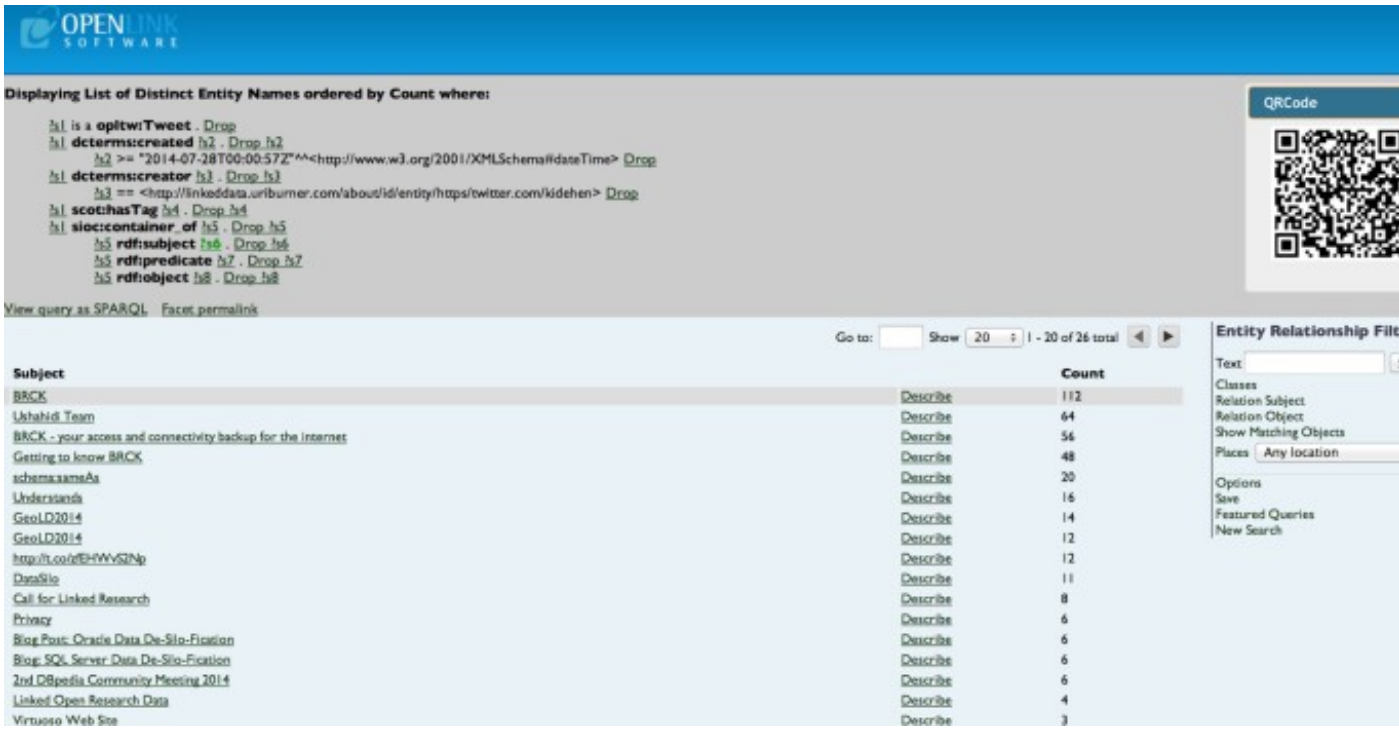


- Nanotation that represents a "Hat Tip" relationship type:

Figure 16.112. Faceted Browser Nanotation Hat Tip Type

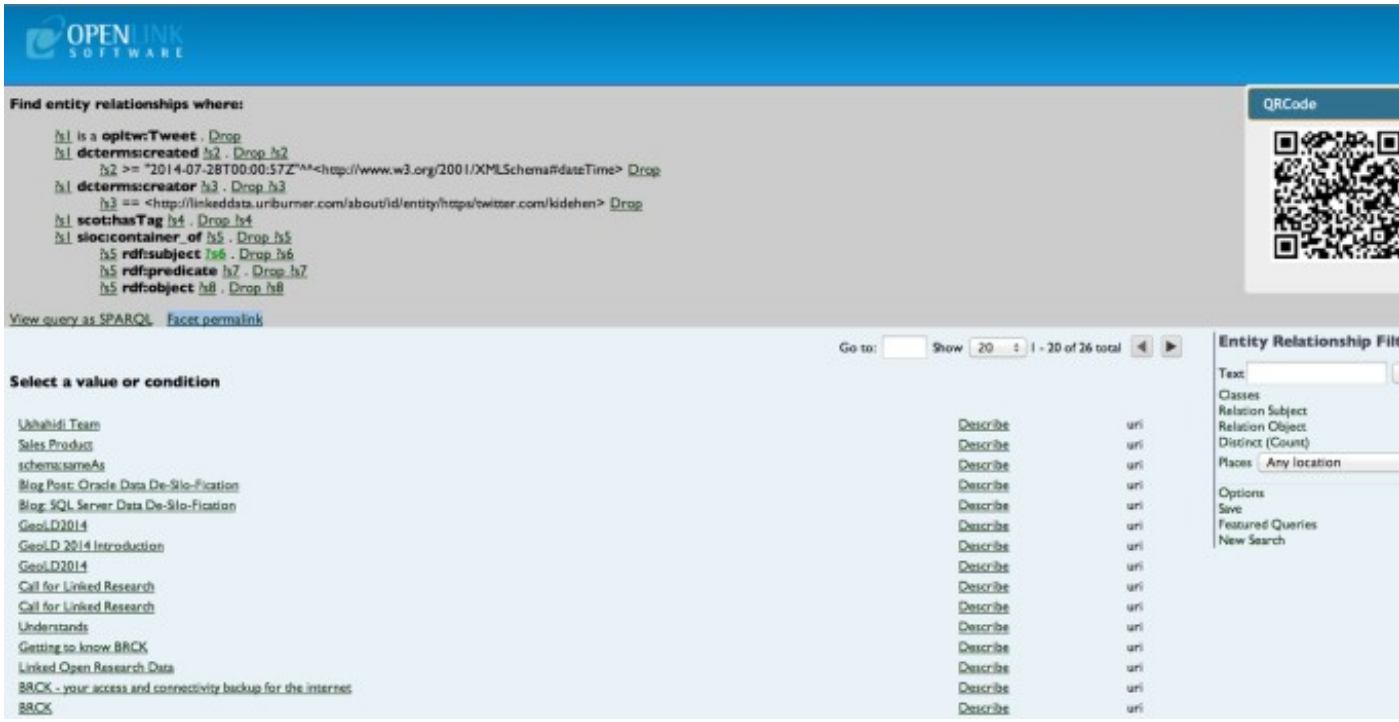
- Nanotation generated RDF statements aggregated by Subject:

Figure 16.113. Faceted Browser Nanotation RDF Aggregation by Subject



- Nanotation generated RDF statements by Subject:

Figure 16.114. Faceted Browser Nanotation Generation by Subject



16.10.11. Sponger Usage Examples

- SPARQL Processor Usage Example
- RDF Proxy Service Example
- Browsing & Exploring Linked Data View Example Using ODE
- Browsing & Exploring Linked Data View Example Using iSPARQL
- Basic Sponger Cartridge Example
- HTTP Example for Extracting Metadata using CURL
- RESTFul Interaction Examples
- Flickr Cartridge Example
- MusicBrainz Metadatabase Example

16.11. Virtuoso Faceted Browser Installation and configuration

16.11.1. Prerequisites

Requires Virtuoso 6.0 TP1 or higher for use.

16.11.2. Pre Installation

Note : This step is not required for Virtuoso Release 6.1 and above builds

If you have an existing Virtuoso 6.x installation, and your Quad Store has greater than 10K worth of triples, please perform the following steps:

1. Run the following commands using the Virtuoso isql program before installing the Faceted Browser VAD:

```
drop index RDF_QUAD_OPGS;
drop index RDF_QUAD_POGS;
drop index RDF_QUAD_GPOS;
drop index RDF_QUAD_OGPS;

checkpoint;

create table R2 (G iri_id_8, S iri_id_8, P iri_id_8, O any, primary key (S, P, O, G));
```

```
alter index R2 on R2 partition (S int (0hexffff00));

log_enable (2);

INSERT INTO R2 (G, S, P, O) select G, S, P, O FROM rdf_quad;

DROP TABLE RDF_QUAD;
ALTER TABLE r2 rename RDF_QUAD;

checkpoint;

create bitmap index RDF_QUAD_OPGS on RDF_QUAD (O, P, G, S) partition (O varchar (-1, 0hexffff));
create bitmap index RDF_QUAD_POGS on RDF_QUAD (P, O, G, S) partition (O varchar (-1, 0hexffff));
create bitmap index RDF_QUAD_GPOS on RDF_QUAD (G, P, O, S) partition (O varchar (-1, 0hexffff));

checkpoint;
```

Note this step may take sometime depending on how many triples are already in your Quad Store.

16.11.3. VAD Package Installation

1. Download and install the Virtuoso Faceted Browser VAD package using the Conductor System Admin - > Packages tab.

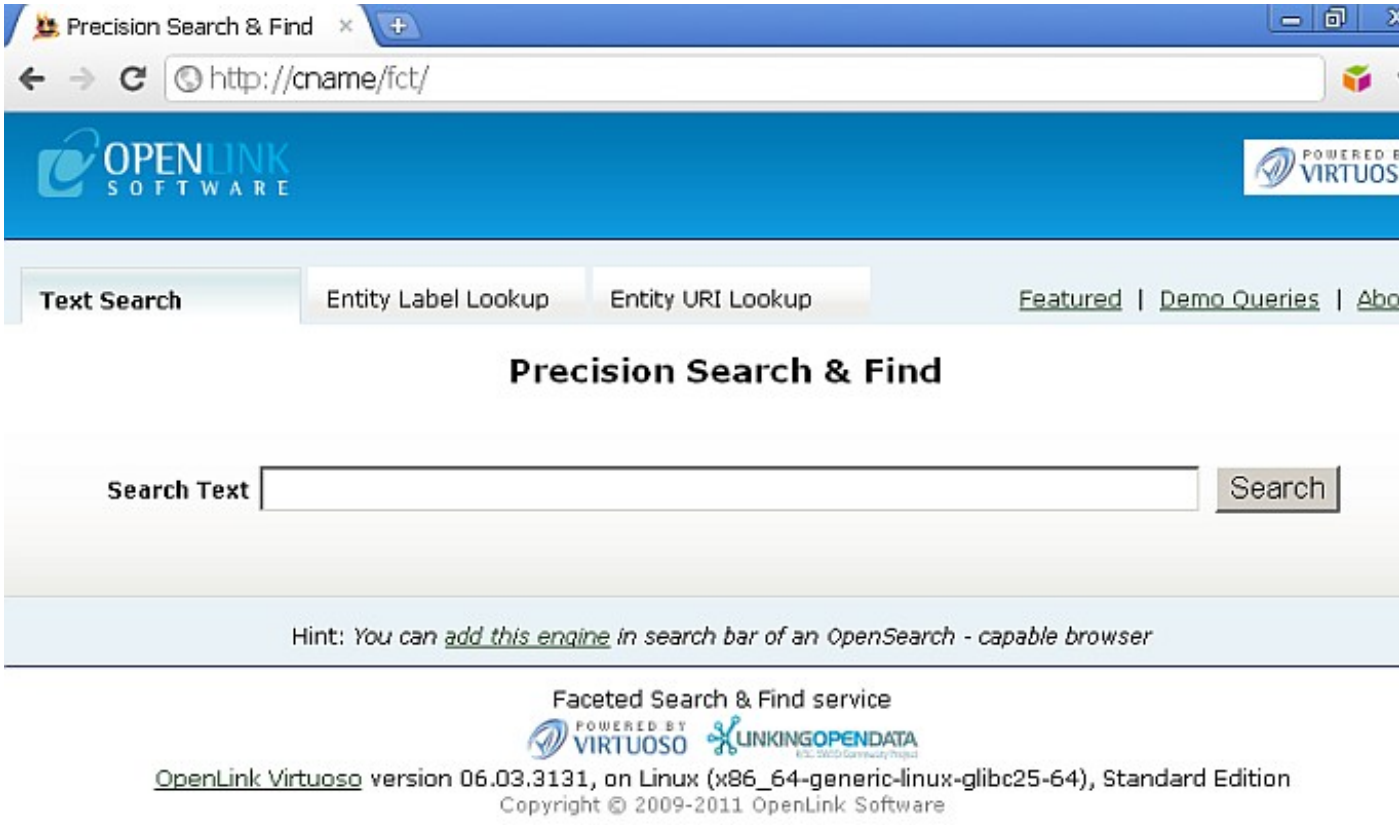
   Figure 16.115. Install the FCT package

2. The HTML interface of the Faceted Browser Engine is exposed at:

   http://<cname>/fct

   , where "cname" is the hostname:portno your Virtuoso instance is running on.

   Figure 16.116. FCT HTML interface



3. The Faceted Browser Engine exposes a REST API at the endpoint:

   http://<cname>/fct/service

   .

   See Also:
   - Virtuoso APIs for Faceted REST services
   - Faceted Web Service and Linked Data

16.11.4. Post Installation

1. Build Full Text Indexes by running the following commands using the Virtuoso

   isql

   program:

   ```
   RDF_OBJ_FT_RULE_ADD (null, null, 'All');
   VT_INC_INDEX_DB_DBA_RDF_OBJ ();
   ```

2. Run the following procedure using the Virtuoso

   isql

   program to populate label lookup tables periodically and activate the

   Label

   text box of the

   Entity Label Lookup

tab:

```
urilbl_ac_init_db()
```

3. Run the following procedure using the Virtuoso

   isql

   program to calculate the IRI ranks. Note this should be run periodically as the data grows to re-rank the IRIs.

```
s_rank()
```

4. Perform Network Resource Fetch of some data to load some RDF triples in the quad store. This can easily be done using the Virtuoso

   description.vsp

   page which provides a hypertext description of RDF Linked Data, by describing the following page for example (or one of your choice):

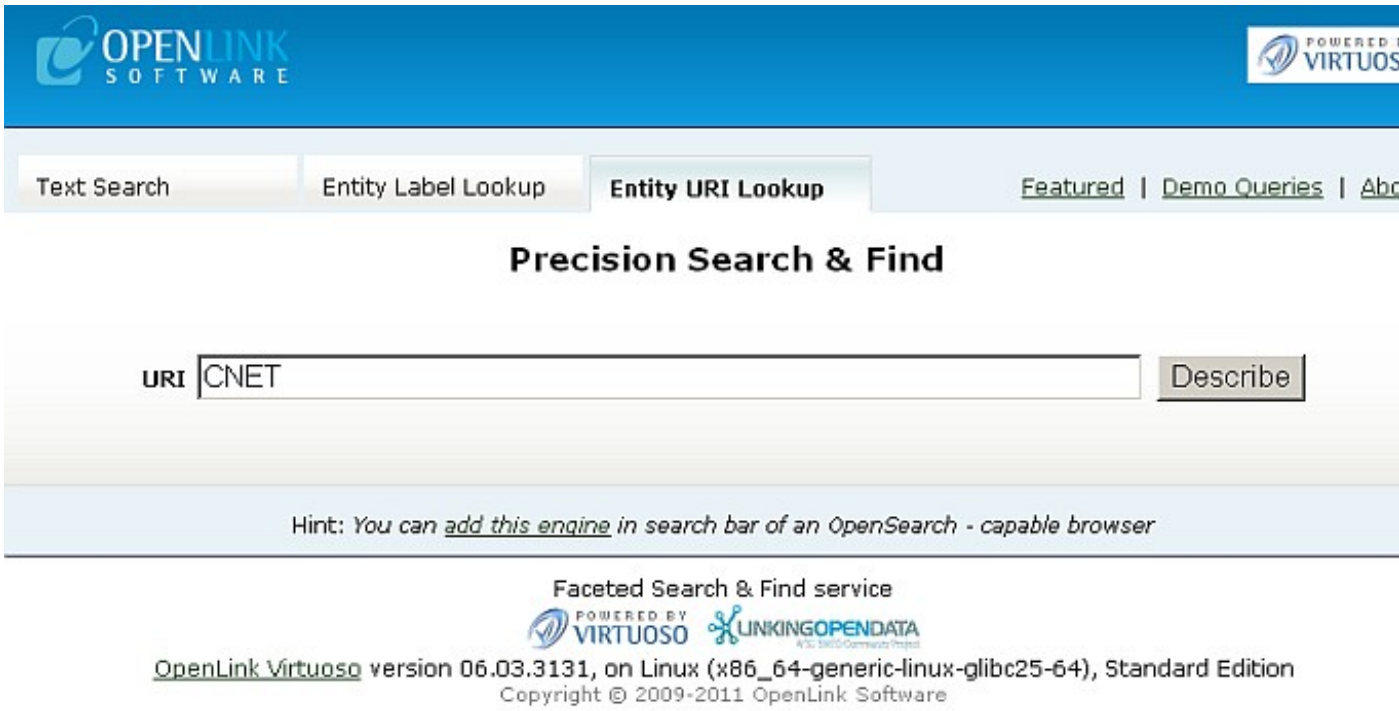
```
http://cname/about/html/http/news.cnet.com
```

Figure 16.117. Network Resource Fetch data

Figure 16.118. Network Resource Fetch data

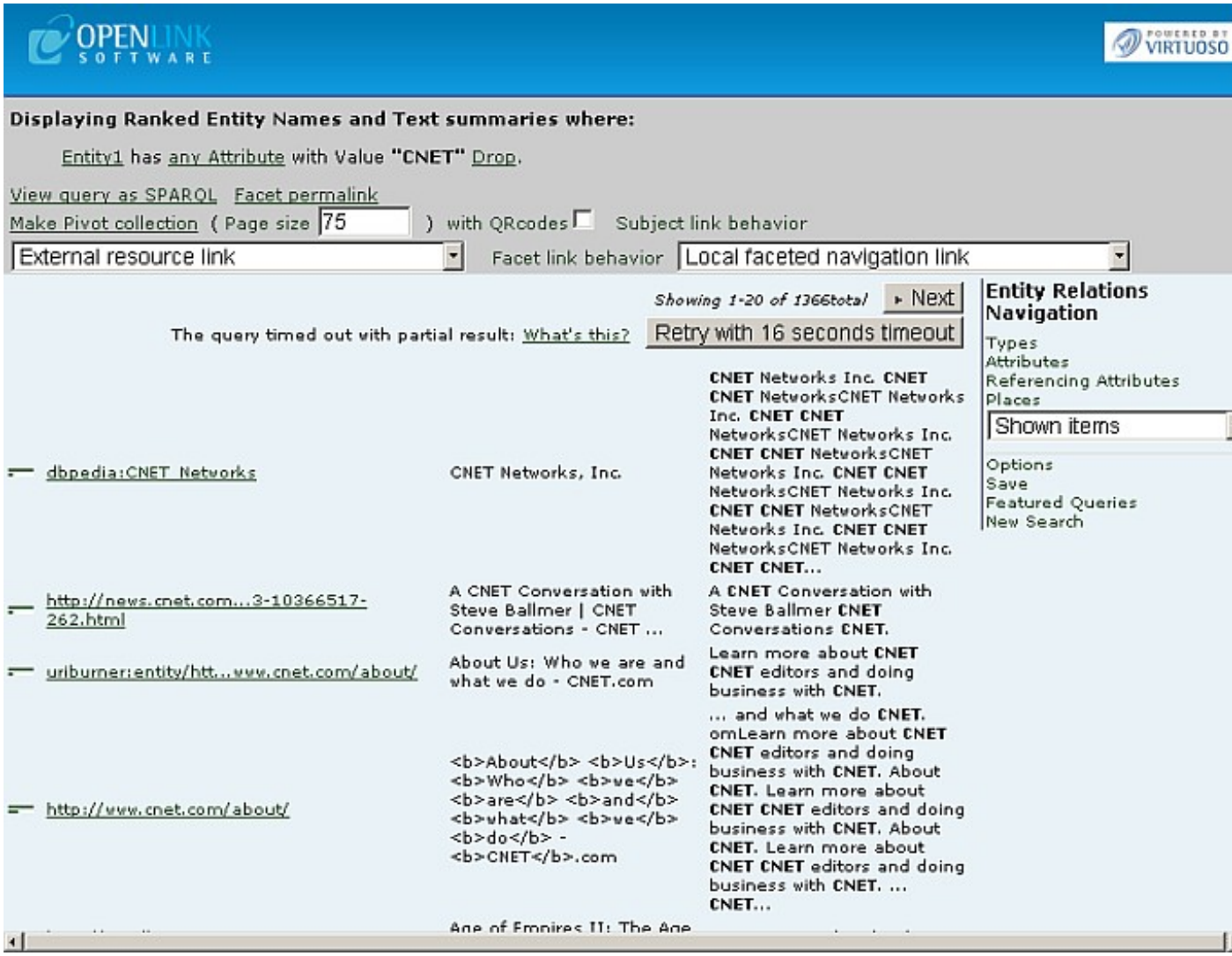
5. Use the Faceted Browser Search and Find User Interface to search for information on "CNET":

Figure 16.119. Faceted Browser Search



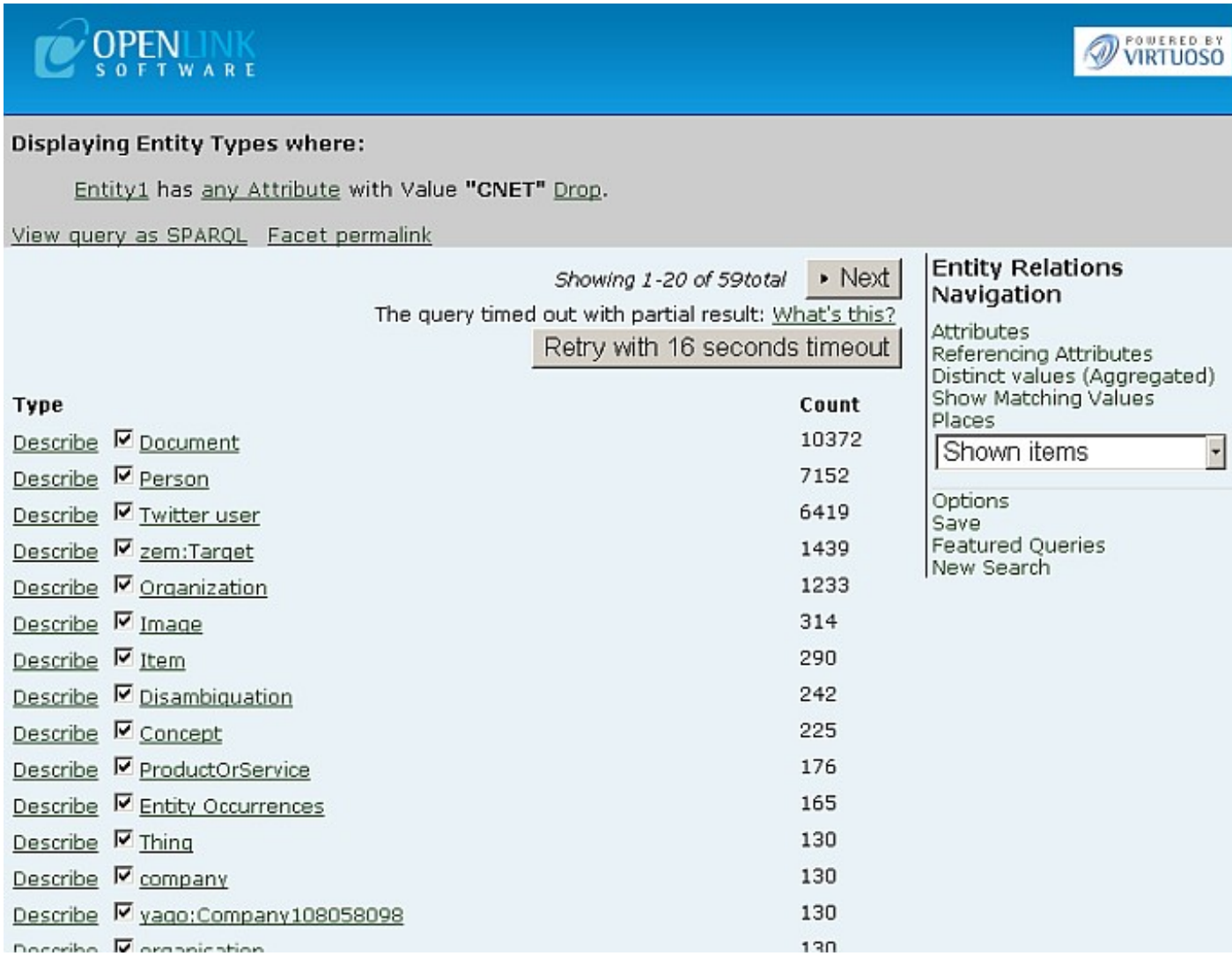
6. Results of the following form should be returned for the network resource data being fetched.

Figure 16.120. Faceted Browser Search Results



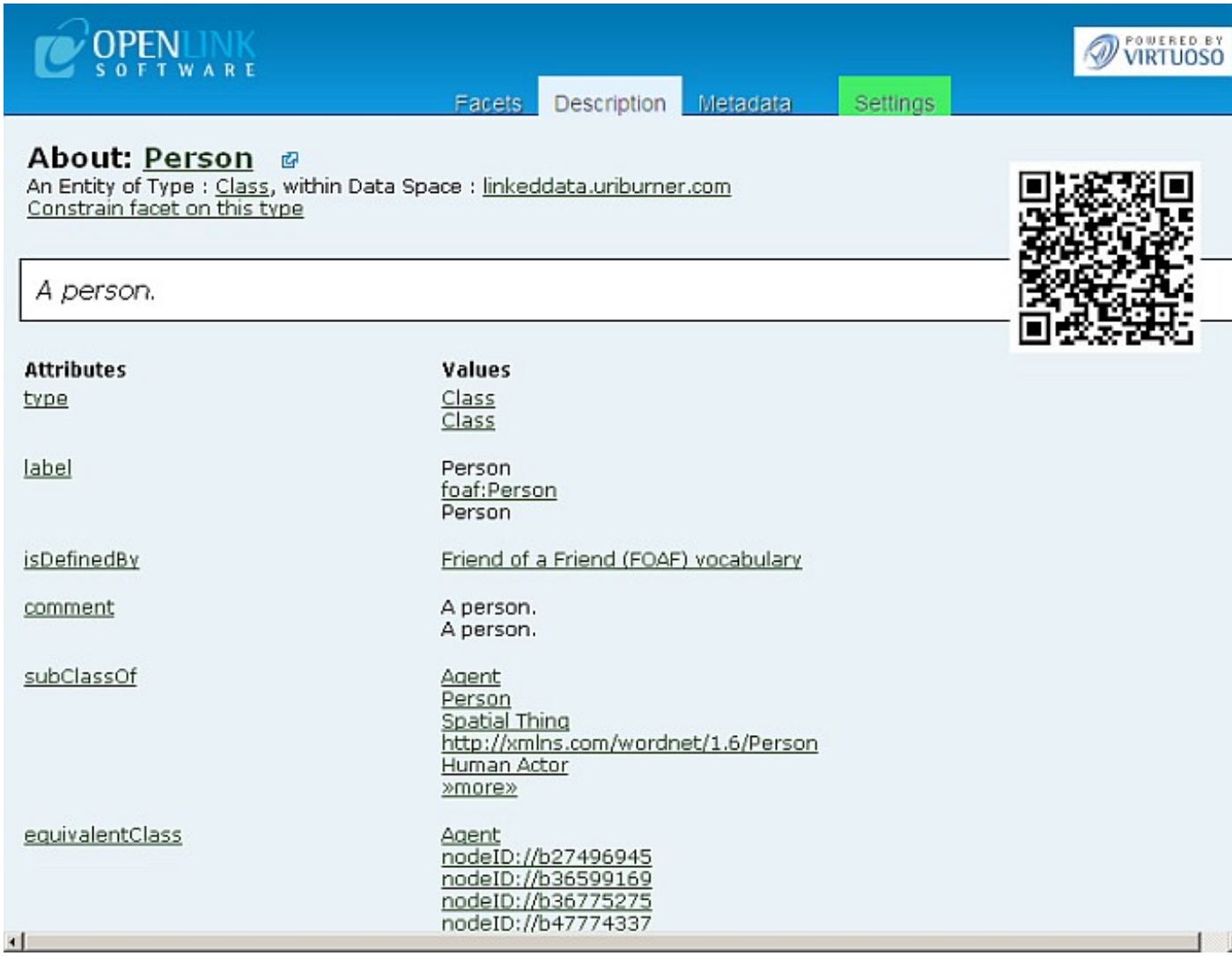
7. Click "Types" link shown at the right vertical Navigation
8. Results of the classes/properties should be returned:

Figure 16.121. Results of the classes/properties



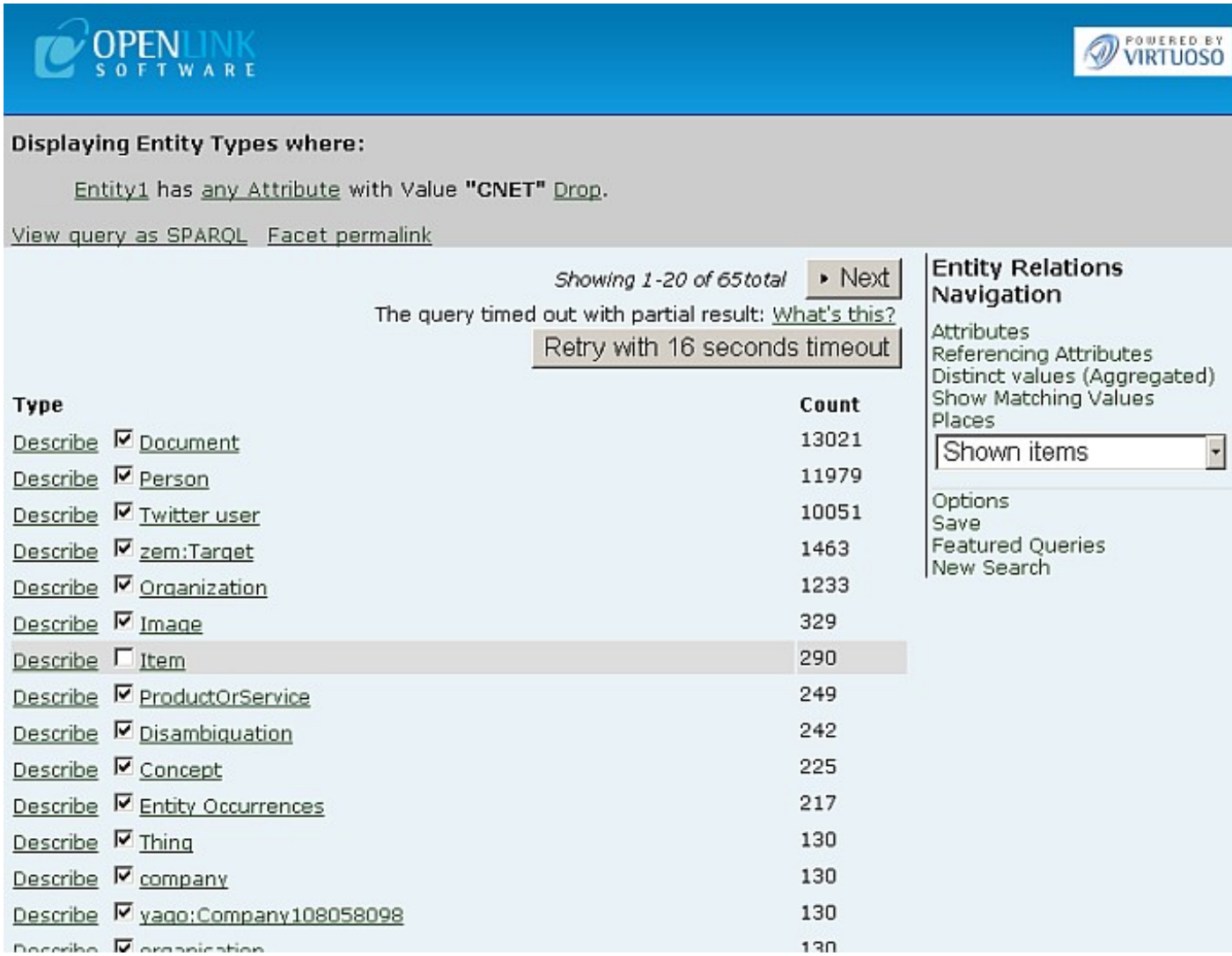
9. To get Type description, click "Describe" link for a given type, for ex. "Person".
10. A list of attributes and values should be presented for the given resource. Note that automatically is generated QRCode image for the described entity.

Figure 16.122. Results of the classes/properties



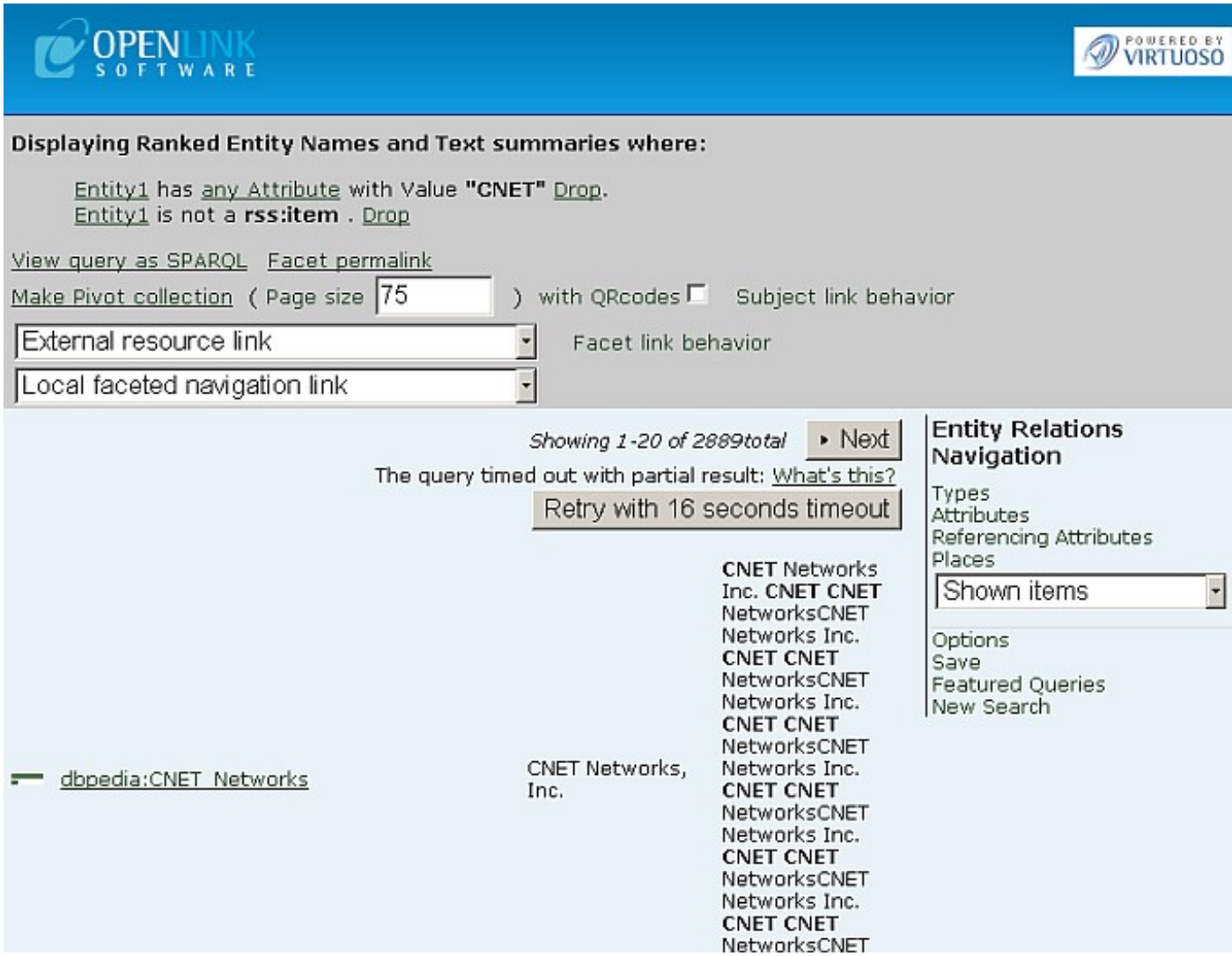
11. Return back to the Attributes list from above by going to the "Facets" tab.
12. To exclude a type, un-tick the checkbox associated with the type:

Figure 16.123. Exclude Type(s)



13. Click the Type URI link
14. Results of excluding the Type(s) should be shown:

Figure 16.124. Results of Excluded Type(s)



15. The Faceted Browser Web service endpoint can also be queried to obtain the same results:

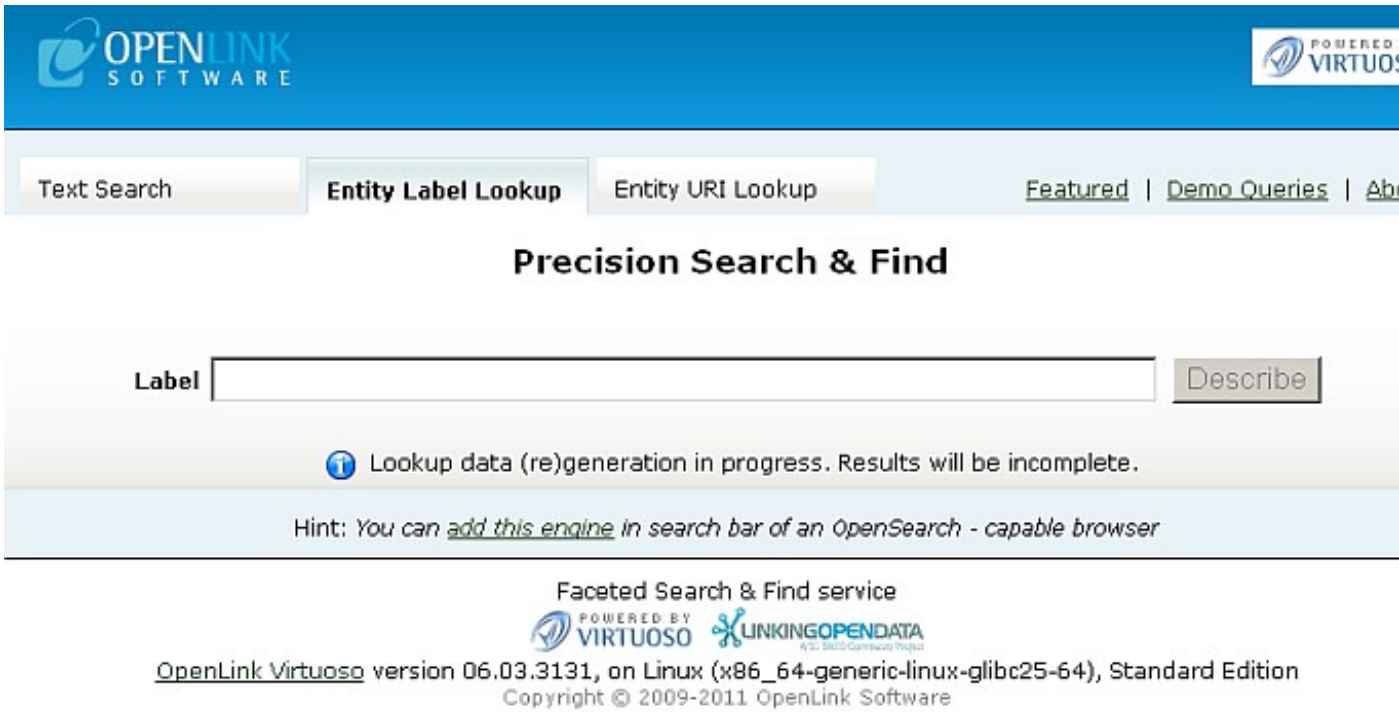
```
$ more cnet.xml
<?xml version="1.0"?>
<query xmlns="http://openlinksw.com/services/facets/1.0" inference="" same-as="">
  <text>CNET</text>
  <view type="text" limit="20" offset=""/>
</query>

$ curl -H "Content-Type: text/xml" -d @cnet.xml  http://cname/fct/service
<fct:facets xmlns:fct="http://openlinksw.com/services/facets/1.0/">
<fct:sparql>    SELECT ?s1 as ?c1, (bif:search_excerpt (bif:vector ('CNET'), ?o1)) as ?c2, ?sc, ?r
<fct:time>16</fct:time>
<fct:complete>yes</fct:complete>
<fct:timeout>0</fct:timeout>
<fct:db-activity>   131R rnd     36R seq      0P disk      0B /      0 messages</fct:db-activity>
 <fct:result type="text">
  <fct:row>
    <fct:column datatype="trank">4.5</fct:column>
    <fct:column datatype="erank">5.881291583872905e-014</fct:column>
    <fct:column datatype="url" shortform="http://news.com">http://news.com</fct:column>
    <fct:column>Technology News - CNET News</fct:column>
    <fct:column><span class="srch_xerpt"><b>CNET</b> News.</span></fct:column>
  </fct:row>
  <fct:row>
    <fct:column datatype="trank">4.5</fct:column>
    <fct:column datatype="erank">5.881291583872905e-014</fct:column>
    <fct:column datatype="url" shortform="http://news.cnet.com/2547-1_3-0-20.xml">http://news.cnet
    <fct:column>CNET News.com</fct:column>
    <fct:column><span class="srch_xerpt"><b>CNET</b> News.</span></fct:column>
  </fct:row>
```

```
<fct:row>
  <fct:column datatype="trank">4.5</fct:column>
  <fct:column datatype="erank">5.881291583872905e-014</fct:column>
  <fct:column datatype="url" shortform="http://news.cnet.com">http://news.cnet.com</fct:column>
  <fct:column>Technology News - CNET News</fct:column>
  <fct:column><span class="srch_xerpt"><b>CNET</b> News.</span></fct:column>
</fct:row>
<fct:row>
  <fct:column datatype="trank">3.9</fct:column>
  <fct:column datatype="erank">5.881291583872905e-014</fct:column>
  <fct:column datatype="url" shortform="http://news.com">http://news.com</fct:column>
  <fct:column>Technology News - CNET News</fct:column>
  <fct:column><span class="srch_xerpt">Technology News <b>CNET</b> News.</span></fct:column>
</fct:row>
<fct:row>
  <fct:column datatype="trank">3.9</fct:column>
  <fct:column datatype="erank">5.881291583872905e-014</fct:column>
  <fct:column datatype="url" shortform="http://news.cnet.com">http://news.cnet.com</fct:column>
  <fct:column>Technology News - CNET News</fct:column>
  <fct:column><span class="srch_xerpt">Technology News <b>CNET</b> News.</span></fct:column>
</fct:row>
<fct:row>
  <fct:column datatype="trank">3</fct:column>
  <fct:column datatype="erank">5.881291583872905e-014</fct:column>
  <fct:column datatype="url" shortform="http://news.com">http://news.com</fct:column>
  <fct:column>Technology News - CNET News</fct:column>
  <fct:column><span class="srch_xerpt">Tech news and business reports by <b>CNET</b> News.</span
</fct:row>
<fct:row>
  <fct:column datatype="trank">3</fct:column>
  <fct:column datatype="erank">5.881291583872905e-014</fct:column>
  <fct:column datatype="url" shortform="http://news.cnet.com/2547-1_3-0-20.xml">http://news.cnet
  <fct:column>CNET News.com</fct:column>
  <fct:column><span class="srch_xerpt">Tech news and business reports by <b>CNET</b> News.</span
</fct:row>
<fct:row>
  <fct:column datatype="trank">3</fct:column>
  <fct:column datatype="erank">5.881291583872905e-014</fct:column>
  <fct:column datatype="url" shortform="http://news.cnet.com">http://news.cnet.com</fct:column>
  <fct:column>Technology News - CNET News</fct:column>
  <fct:column><span class="srch_xerpt">Tech news and business reports by <b>CNET</b> News.</span
</fct:row>
<fct:row>
  <fct:column datatype="trank">3</fct:column>
  <fct:column datatype="erank">5.881291583872905e-014</fct:column>
  <fct:column datatype="url" shortform="http://news.com#6">http://news.com#6</fct:column>
  <fct:column>There's an electric car in your future</fct:column>
  <fct:column><span class="srch_xerpt">... <b>CNET</b> Car Tech posts photos of electric cars ex
</fct:row>
<fct:row>
  <fct:column datatype="trank">3</fct:column>
  <fct:column datatype="erank">5.881291583872905e-014</fct:column>
  <fct:column datatype="url" shortform="http://news.cnet.com/2547-1_3-0-20.xml#9">http://news.cn
  <fct:column>There's an electric car in your future</fct:column>
  <fct:column><span class="srch_xerpt">... <b>CNET</b> Car Tech posts photos of electric cars ex
</fct:row>
<fct:row>
  <fct:column datatype="trank">3</fct:column>
  <fct:column datatype="erank">5.881291583872905e-014</fct:column>
  <fct:column datatype="url" shortform="http://news.cnet.com#9">http://news.cnet.com#9</fct:colu
  <fct:column>There's an electric car in your future</fct:column>
  <fct:column><span class="srch_xerpt">... <b>CNET</b> Car Tech posts photos of electric cars ex
</fct:row>
<fct:row>
  <fct:column datatype="trank">3</fct:column>
  <fct:column datatype="erank">5.881291583872905e-014</fct:column>
  <fct:column datatype="url" shortform="http://news.com#6">http://news.com#6</fct:column>
  <fct:column>There's an electric car in your future</fct:column>
  <fct:column><span class="srch_xerpt">... <b>CNET</b> Car Tech posts photos of electric cars ex
</fct:row>
<fct:row>
  <fct:column datatype="trank">3</fct:column>
  <fct:column datatype="erank">5.881291583872905e-014</fct:column>
```

```
      <fct:column datatype="url" shortform="http://news.cnet.com/2547-1_3-0-20.xml#9">http://news.cn
      <fct:column>There's an electric car in your future</fct:column>
      <fct:column><span class="srch_xerpt">... <b>CNET</b> Car Tech posts photos of electric cars ex
    </fct:row>
    <fct:row>
      <fct:column datatype="trank">3</fct:column>
      <fct:column datatype="erank">5.881291583872905e-014</fct:column>
      <fct:column datatype="url" shortform="http://news.cnet.com#9">http://news.cnet.com#9</fct:colu
      <fct:column>There's an electric car in your future</fct:column>
      <fct:column><span class="srch_xerpt">... <b>CNET</b> Car Tech posts photos of electric cars ex
    </fct:row>
  </fct:result>
</fct:facets>
```

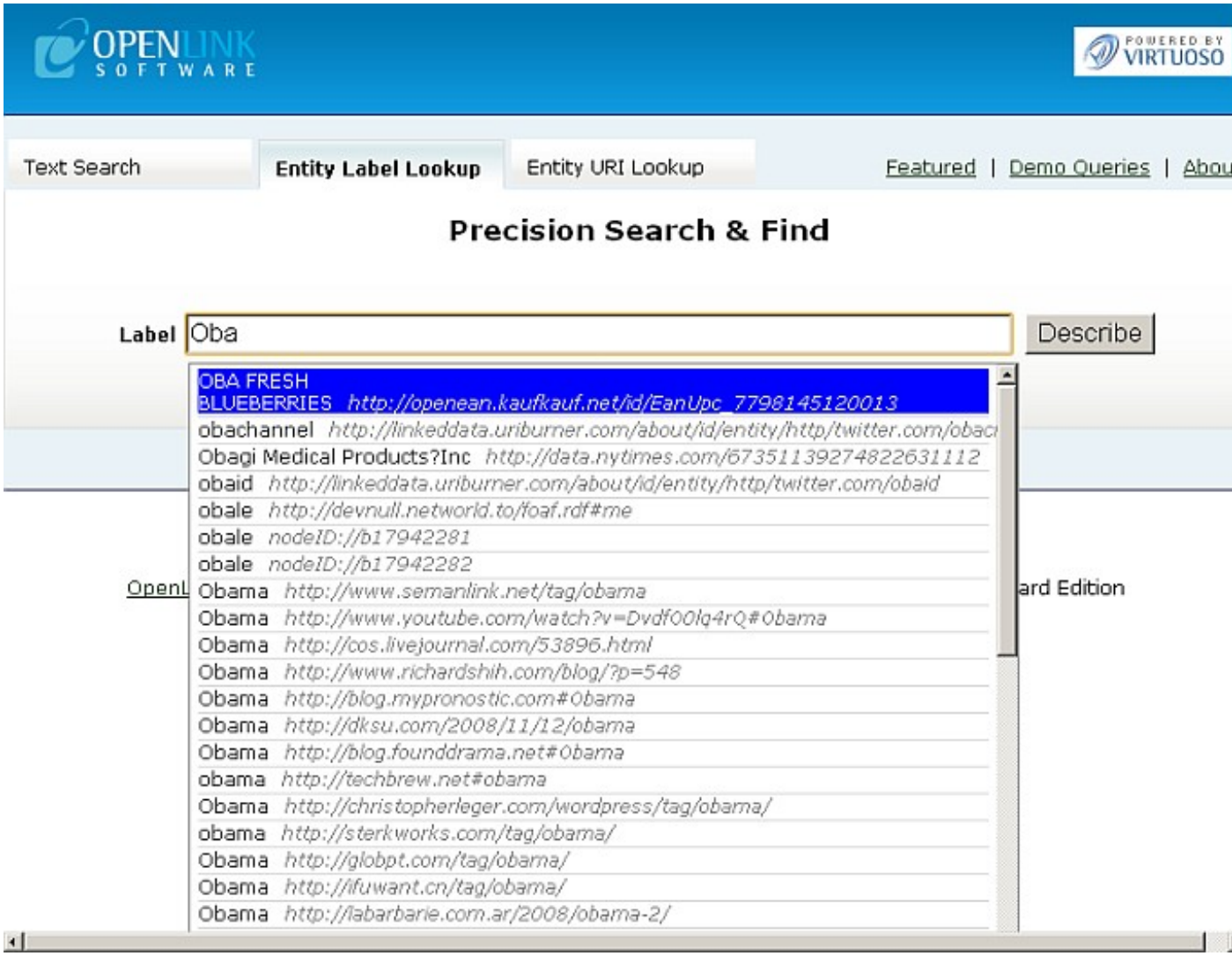
16. Click "New search" from the Entity Relations Navigation and go to "Entity Label Lookup" tab:

Figure 16.125. Query Faceted Browser Web service endpoint



17. In the Label auto-complete text box of the Entity Label Lookup tab, enter the name of an rdfs label to be Described:

Figure 16.126. Select a URI from the list of available Labels



18. Select a URI from the list of available Labels to obtain a description of the URI:

Figure 16.127. Select a URI from the list of available Labels

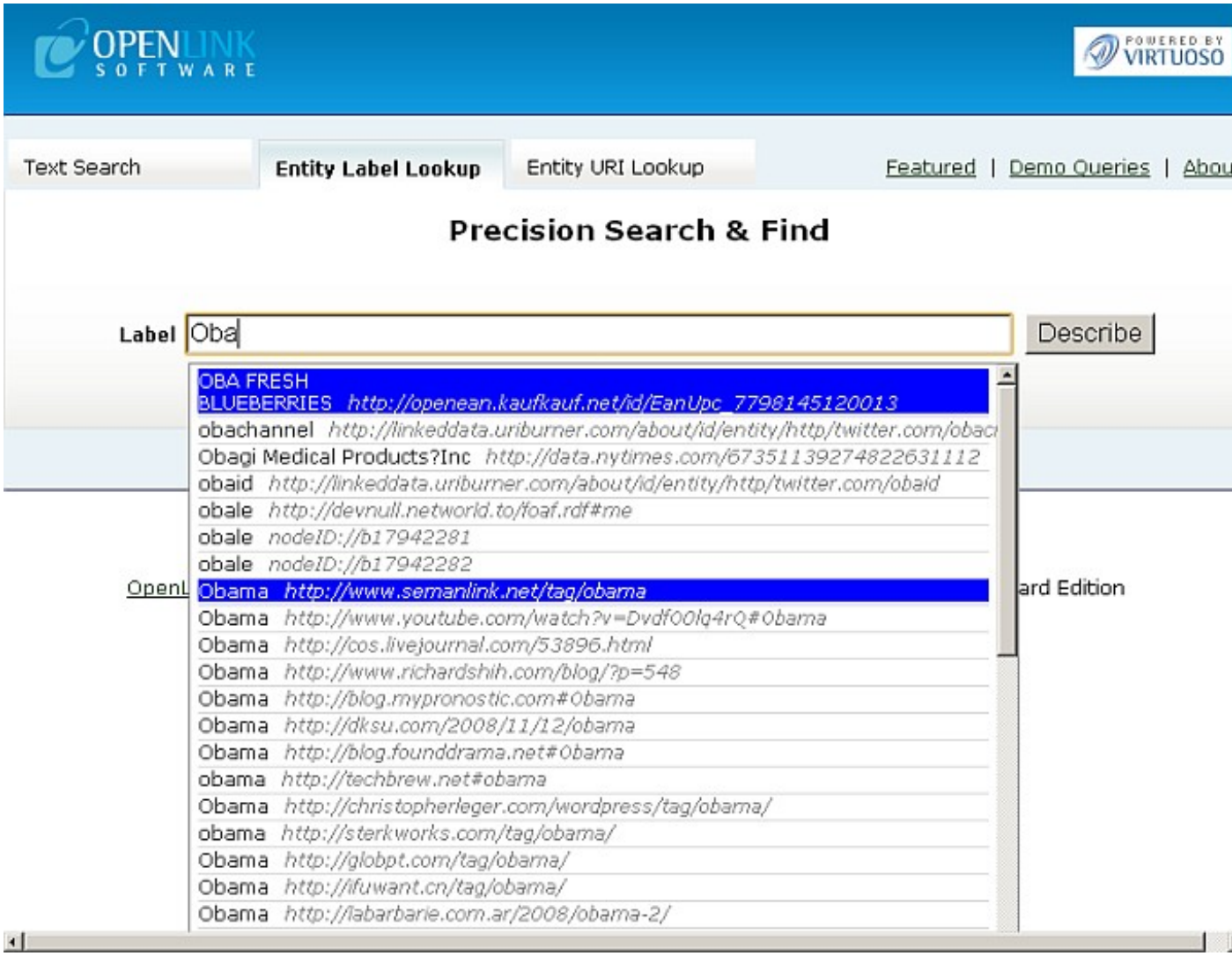
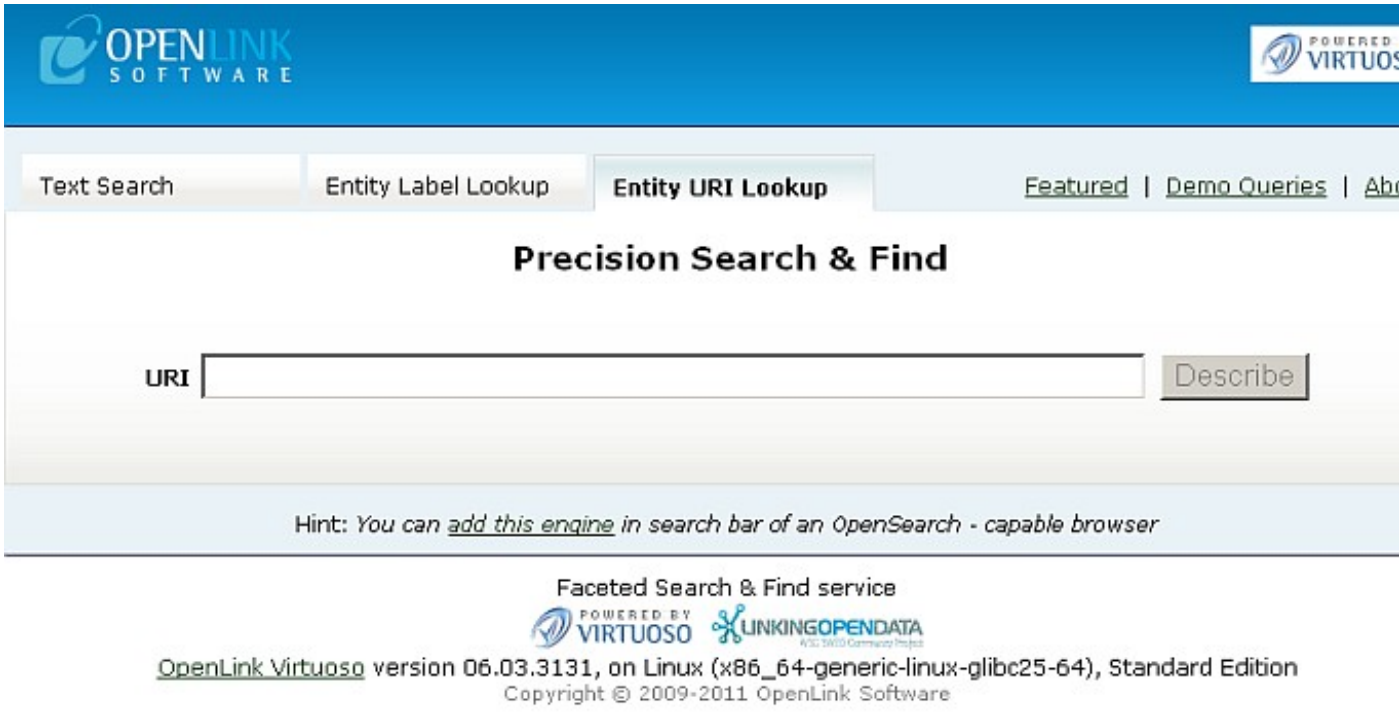


Figure 16.128. Select a URI from the list of available Labels

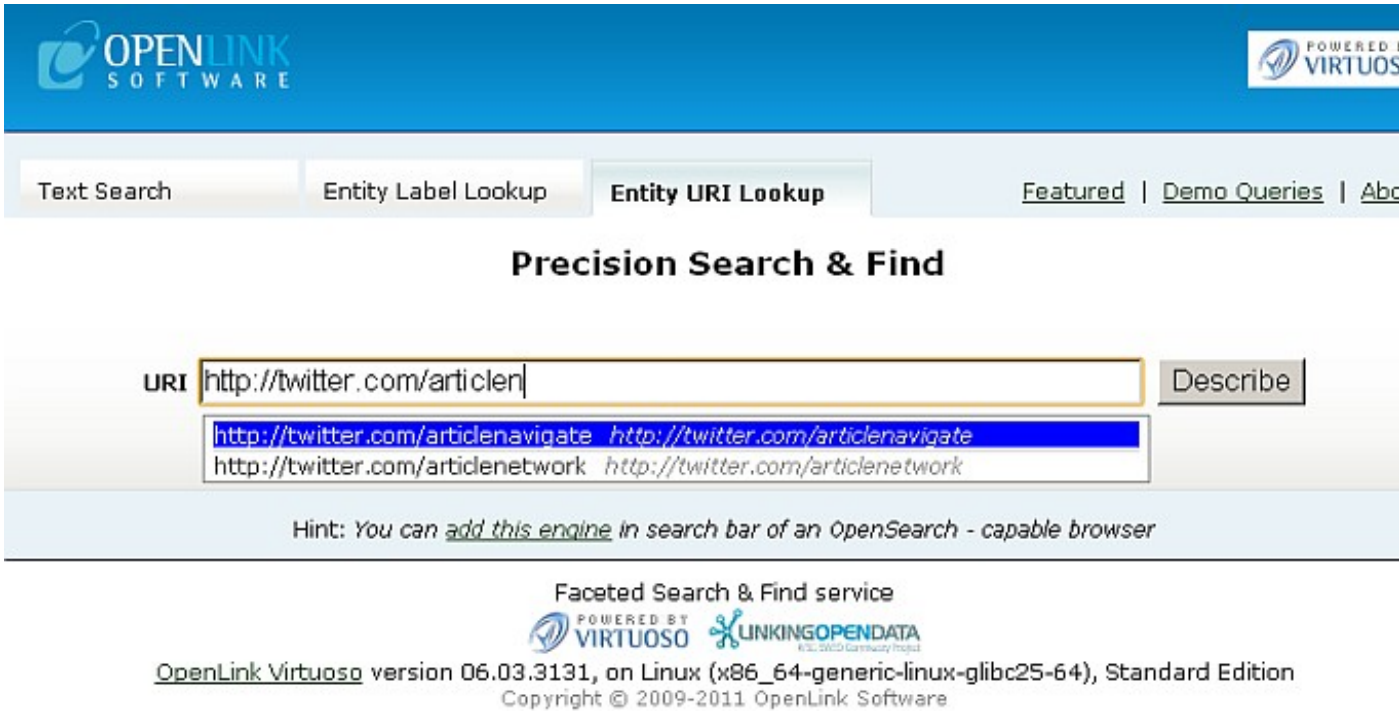
19. Click "Facets" and go to "Entity URI Lookup" tab:

Figure 16.129. Enter URI



20. In the URI auto-complete text box of the Entity URI Lookup tab enter the name URI to be Described:

Figure 16.130. Enter URI



21. Select a URI from the list of available Labels to obtain a description of the URI:

Figure 16.131. Obtain a description of the URI

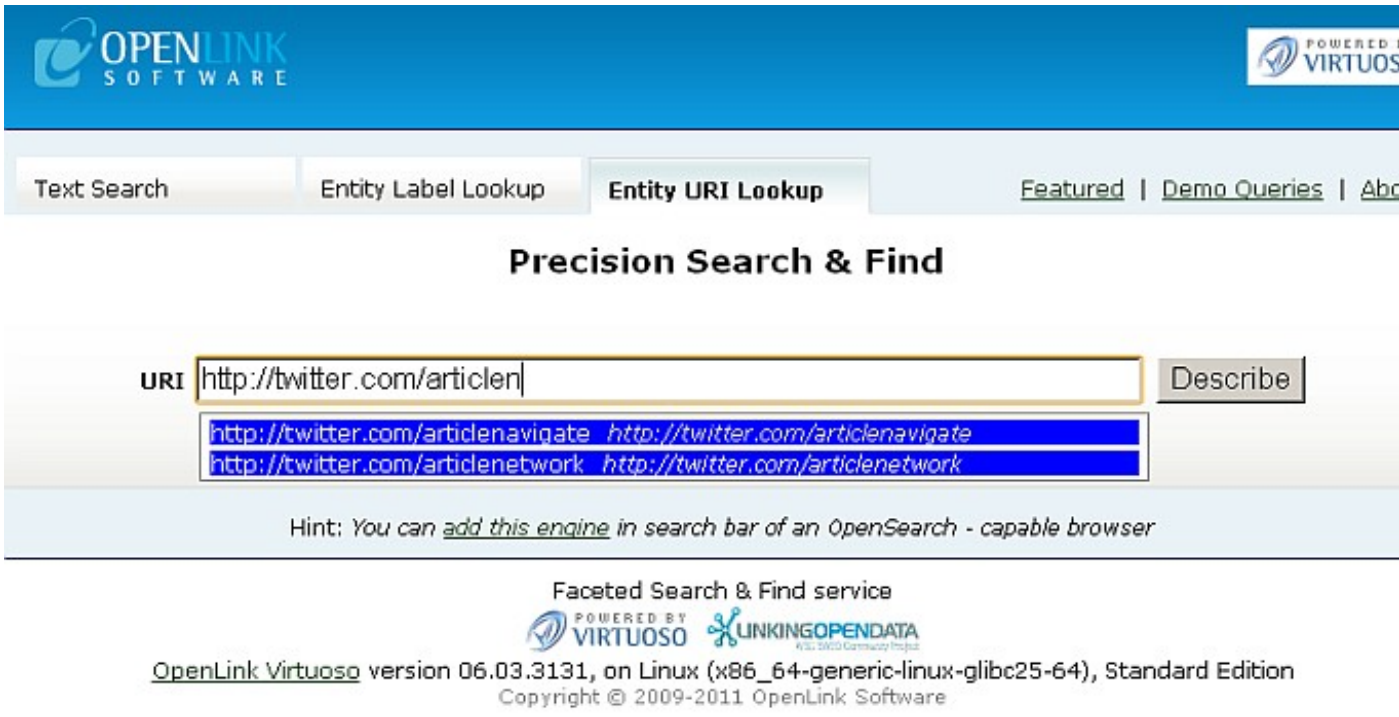


Figure 16.132. Obtain a description of the URI

22. If data is loaded into the quad store via DML functions (TTLP, RDF_LOAD_RDFXML etc.) the following procedure needs to run from

    isql

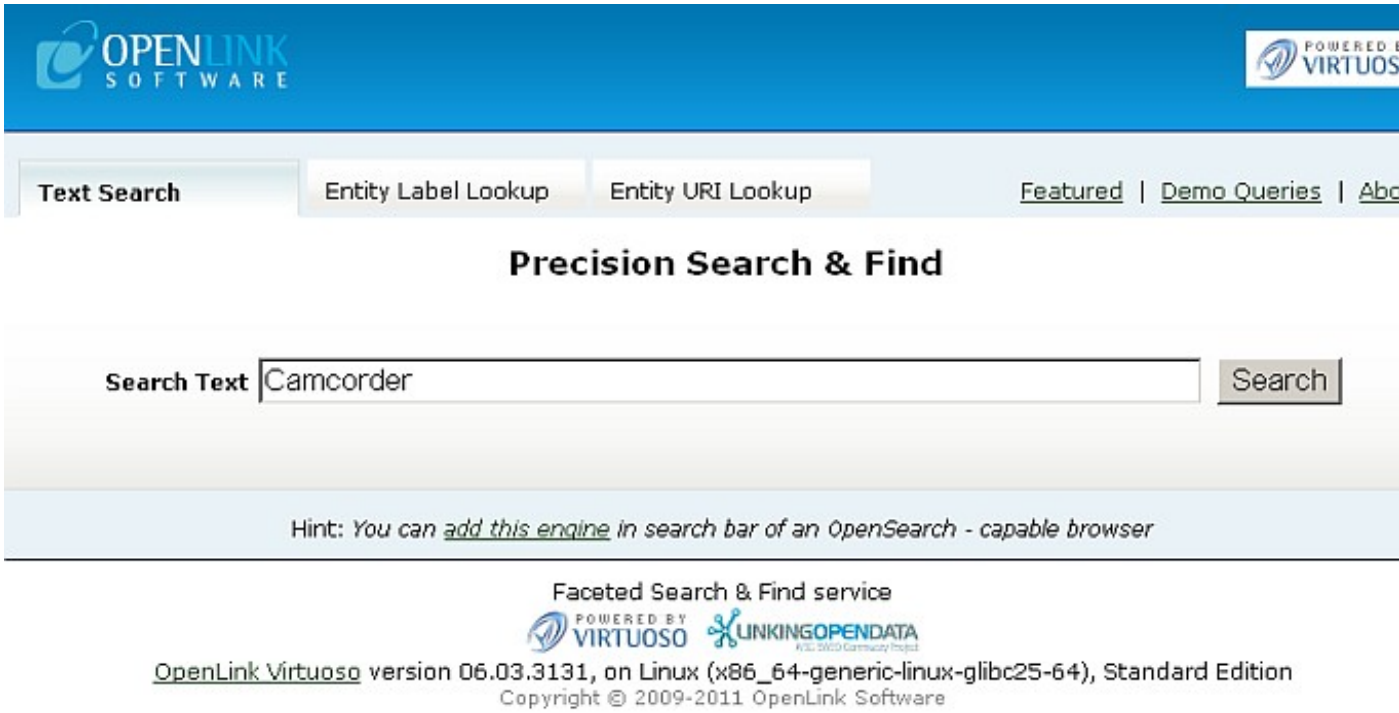
    to build the free text indexes required each time:

    ```
    VT_INC_INDEX_DB_DBA_RDF_OBJ ()
    ```

16.11.5. URI Labels

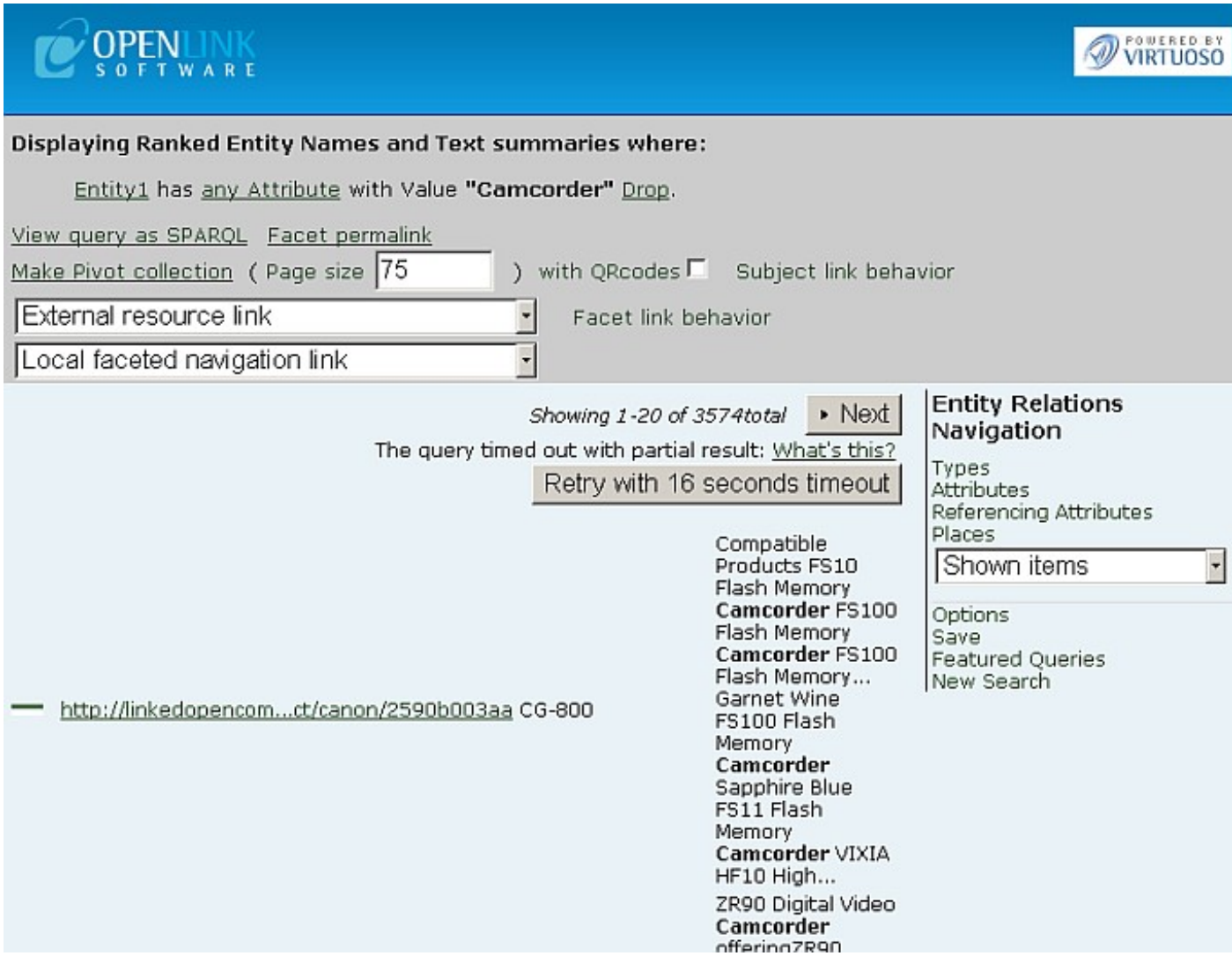
1. Go to http://cname/fct
2. Enter a free text search pattern (for example, "Camcorder" as consumer product), and click Search:

Figure 16.133. URI Labels



3. Your initial query results page will display a list of literal value snippets where for each URL will be displayed a label:

Figure 16.134. URI Labels



4. Click for ex. on the URL link of the first row result.
5. The product description page should be shown and a list of Attributes and Values will be presented. An URL label of the product also will be shown: "Charges Lithium Ion 800 series batteries":

Figure 16.135. URI Labels

Figure 16.136. URI Labels

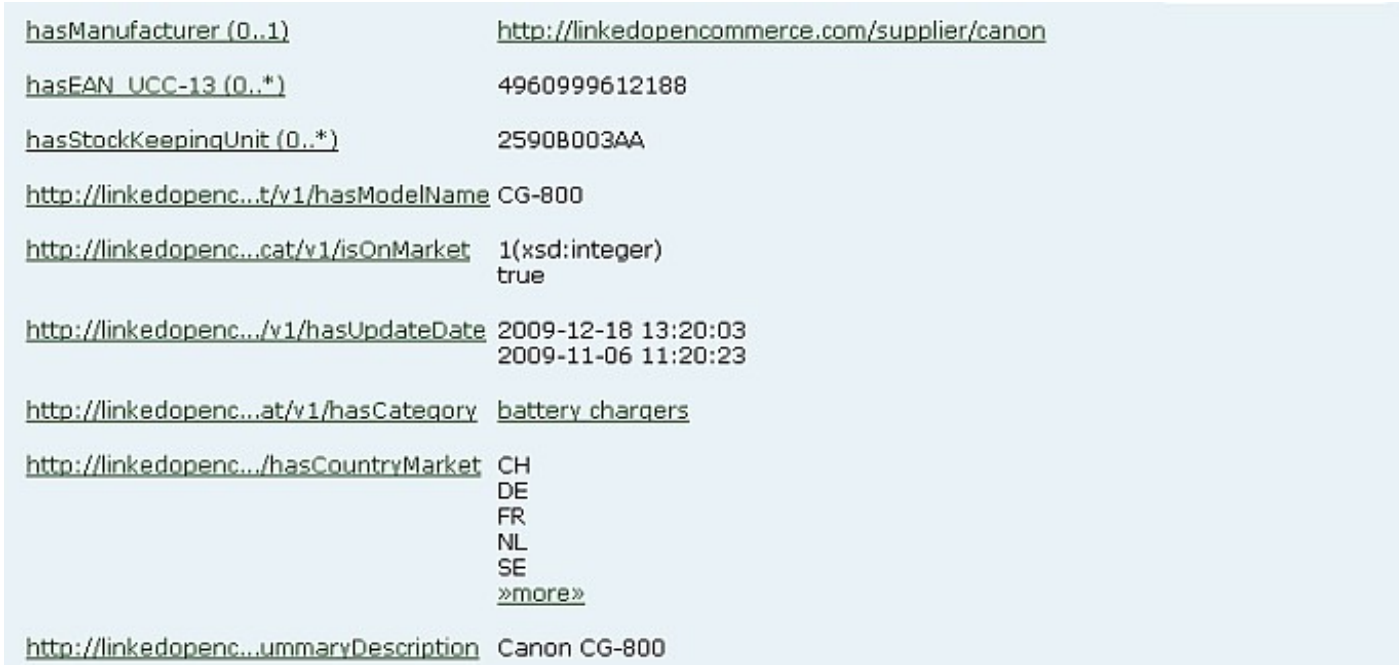
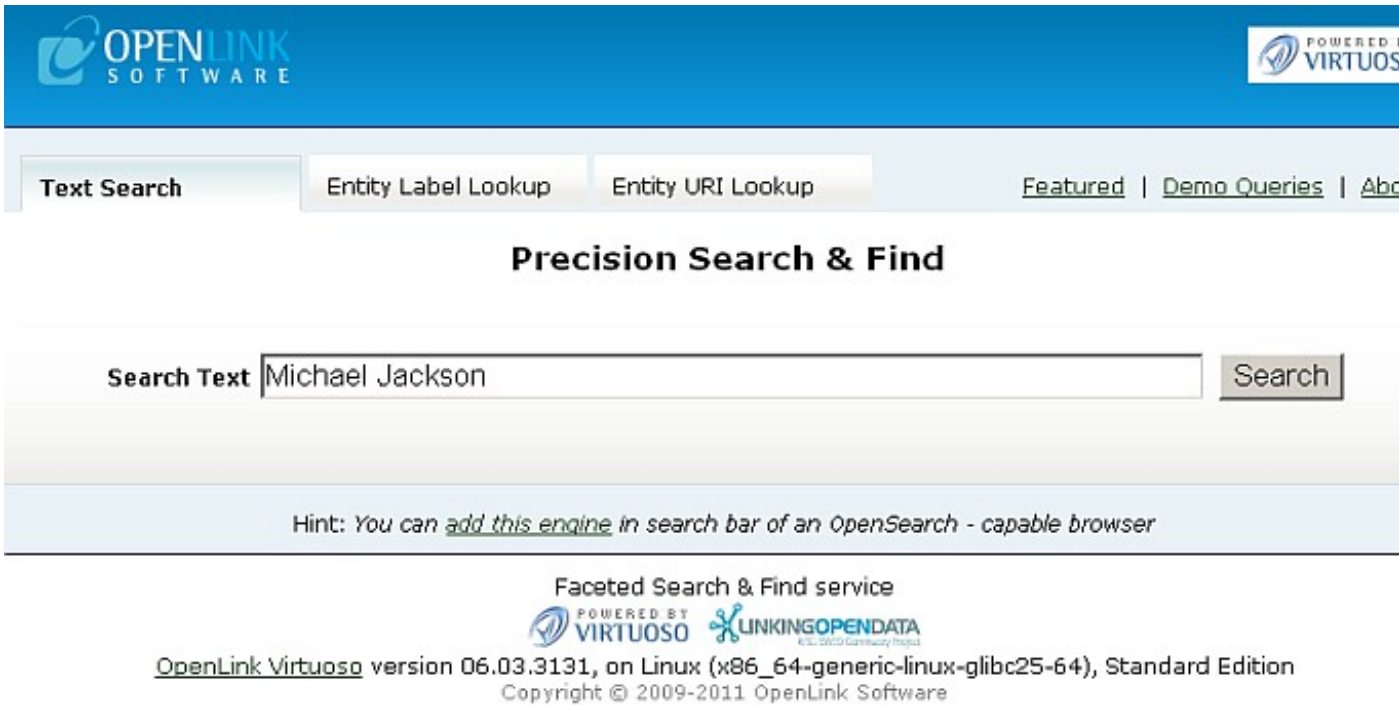


Figure 16.137. URI Labels

16.11.6. Usage Statistics

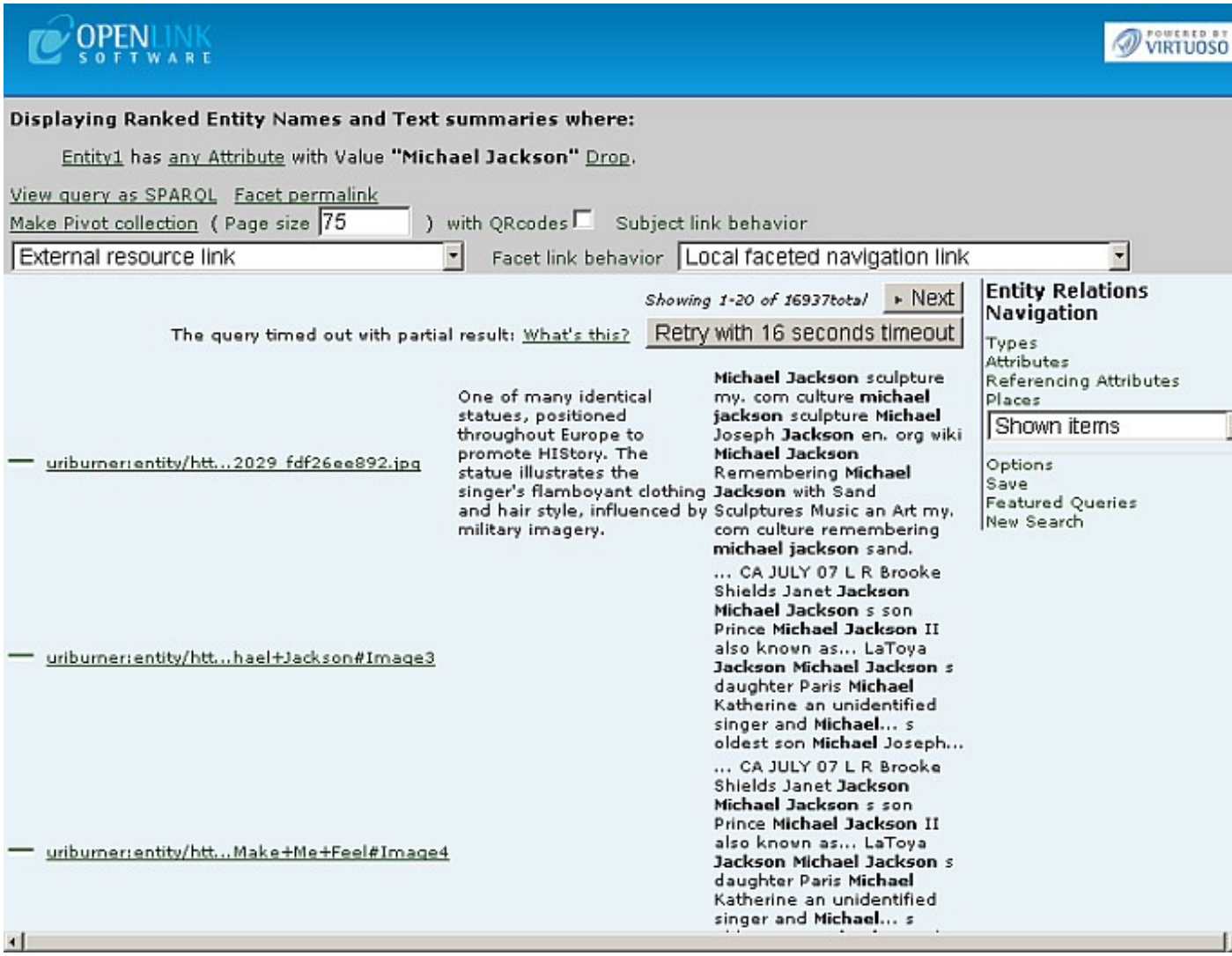
1. Use the Faceted Browser Search and Find User Interface to search for information on "Michael Jackson":

Figure 16.138. Usage Statistics



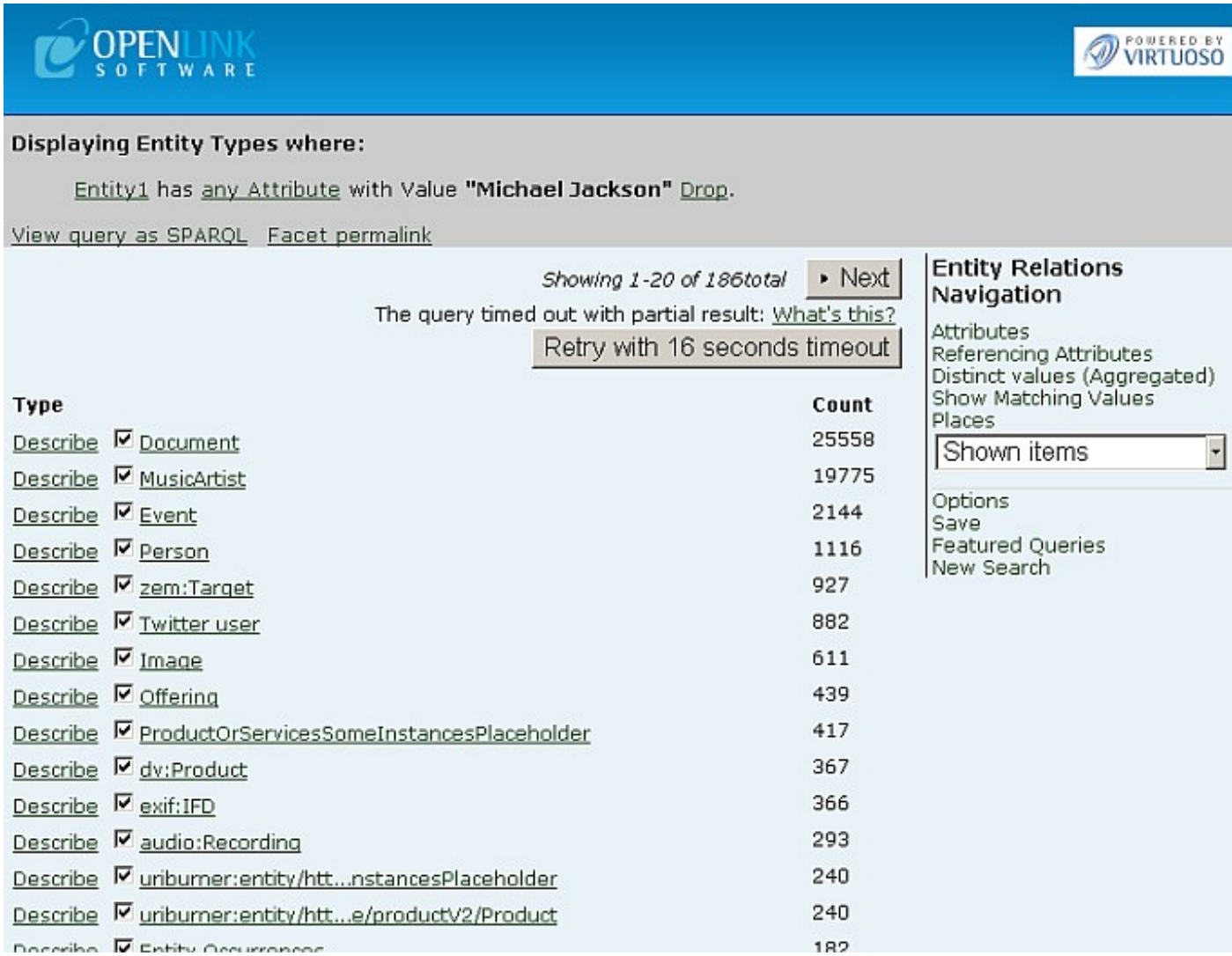
2. Results of the following form should be returned for the network resource data being fetched.

Figure 16.139. Usage Statistics



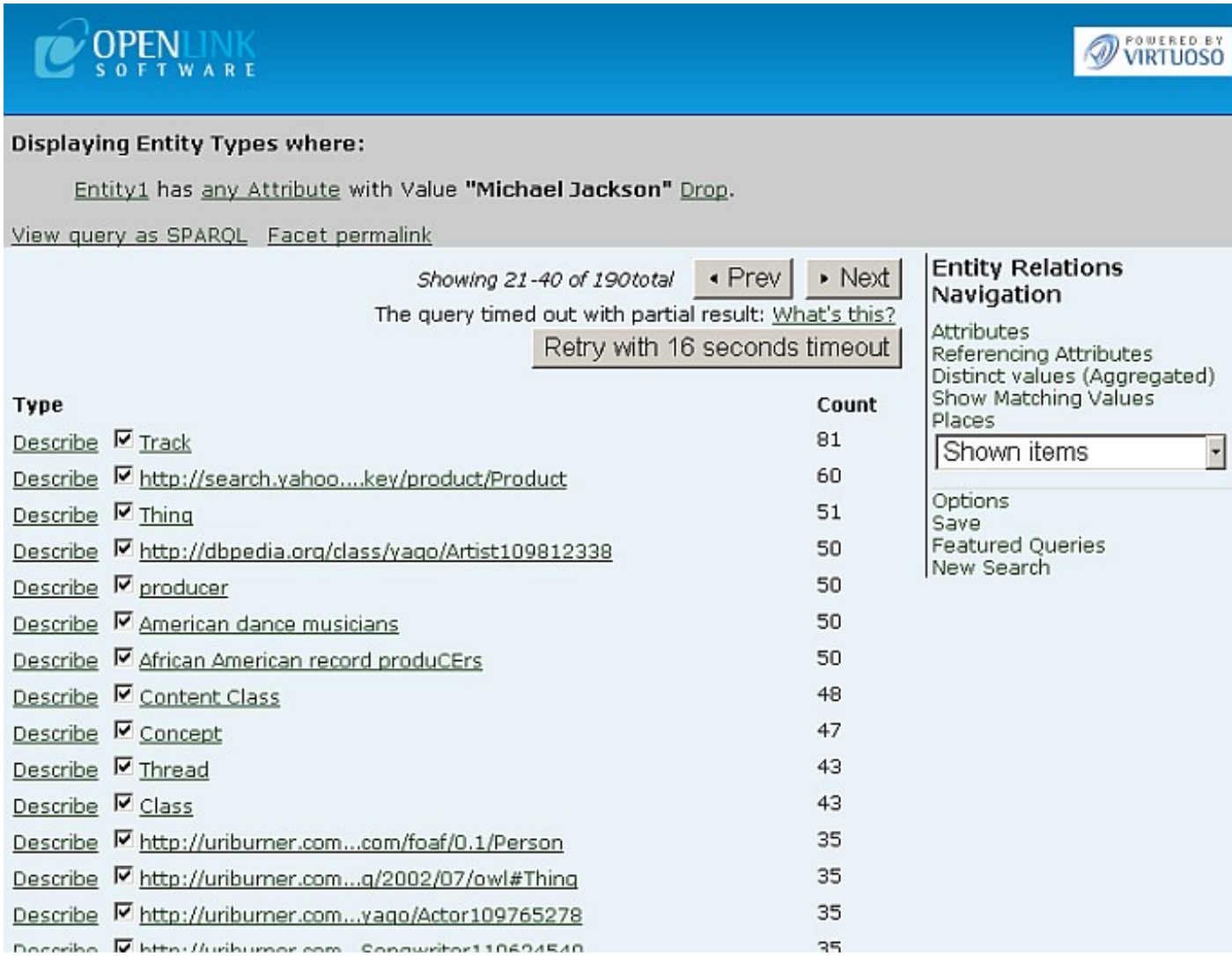
3. Click the "Types" link under "Entity Relations Navigation".
4. Results about "Michael Jackson" as Type/Label/Count list should be displayed:

Figure 16.140. Usage Statistics



5. You can navigate amongst the search results pages by using the "Prev" and "Next" buttons. Click for ex. "Next":

Figure 16.141. Usage Statistics



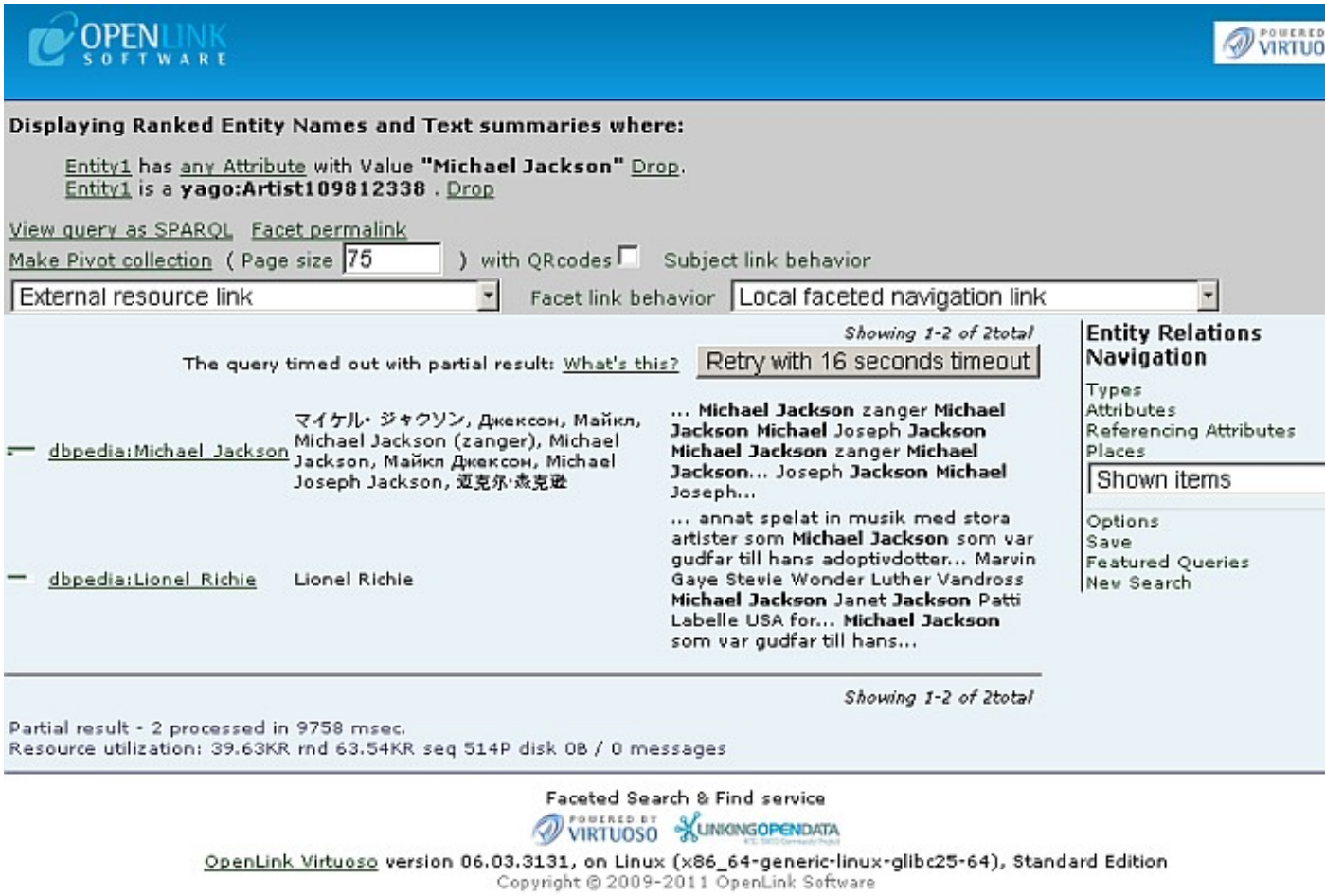
6. Click a type link, for ex.:

```
http://dbpedia.org/class/yago/Artist109812338
```

7. Should be shown type results and:

```
Displaying Ranked Entity Names and Text summaries where:

Entity1 has any Attribute with Value "Michael Jackson" Drop.
Entity1 is a yago:Artist109812338 . Drop
```

Figure 16.142. Usage Statistics



8. Click the link:

   ```
   dbpedia:Michael_Jackson
   ```

9. Results about "Michael Jackson" as Attribute/Value list should be presented:

Figure 16.143. Usage Statistics

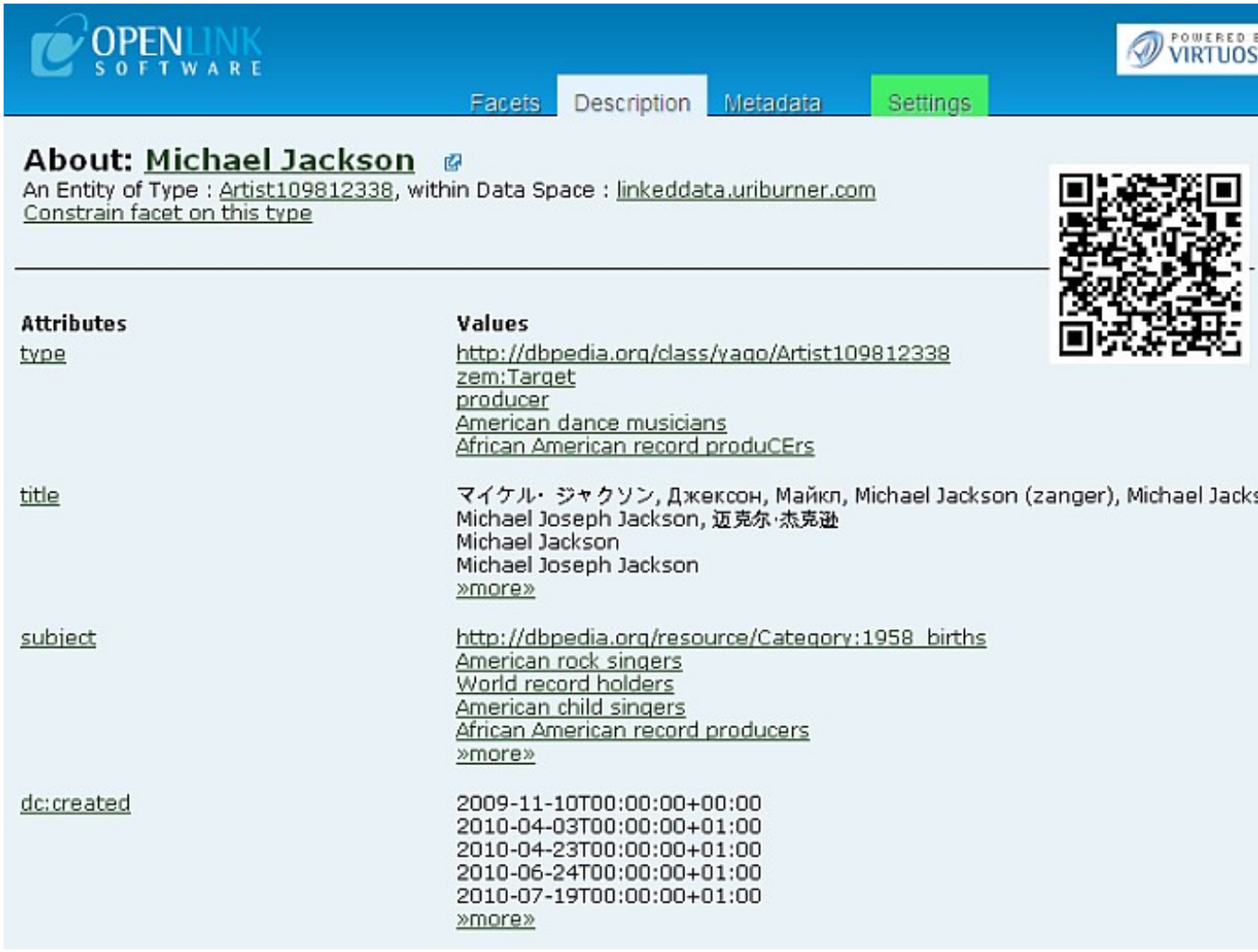


Figure 16.144. Usage Statistics

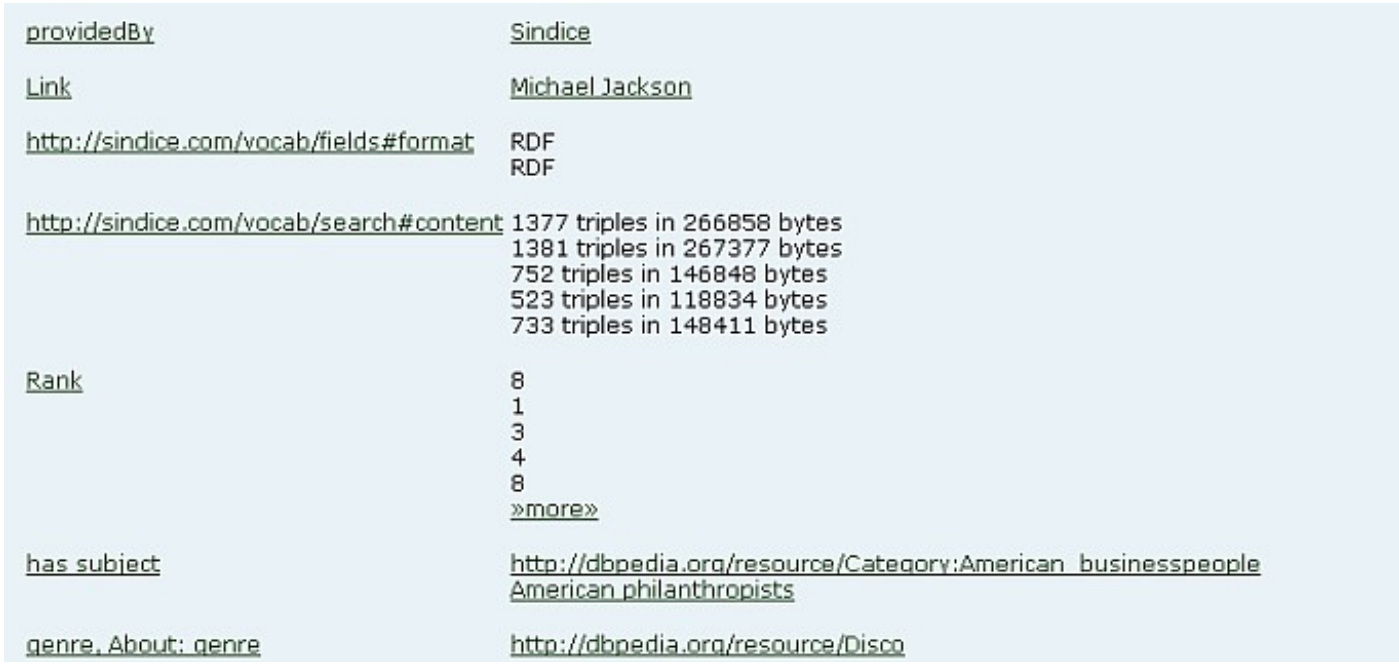


Figure 16.145. Usage Statistics

Figure 16.146. Usage Statistics

10. You can navigate amongst the search results pages by using the "First", "Prev", "Next" and "Last" buttons. Click for ex. "Last":

Figure 16.147. Usage Statistics

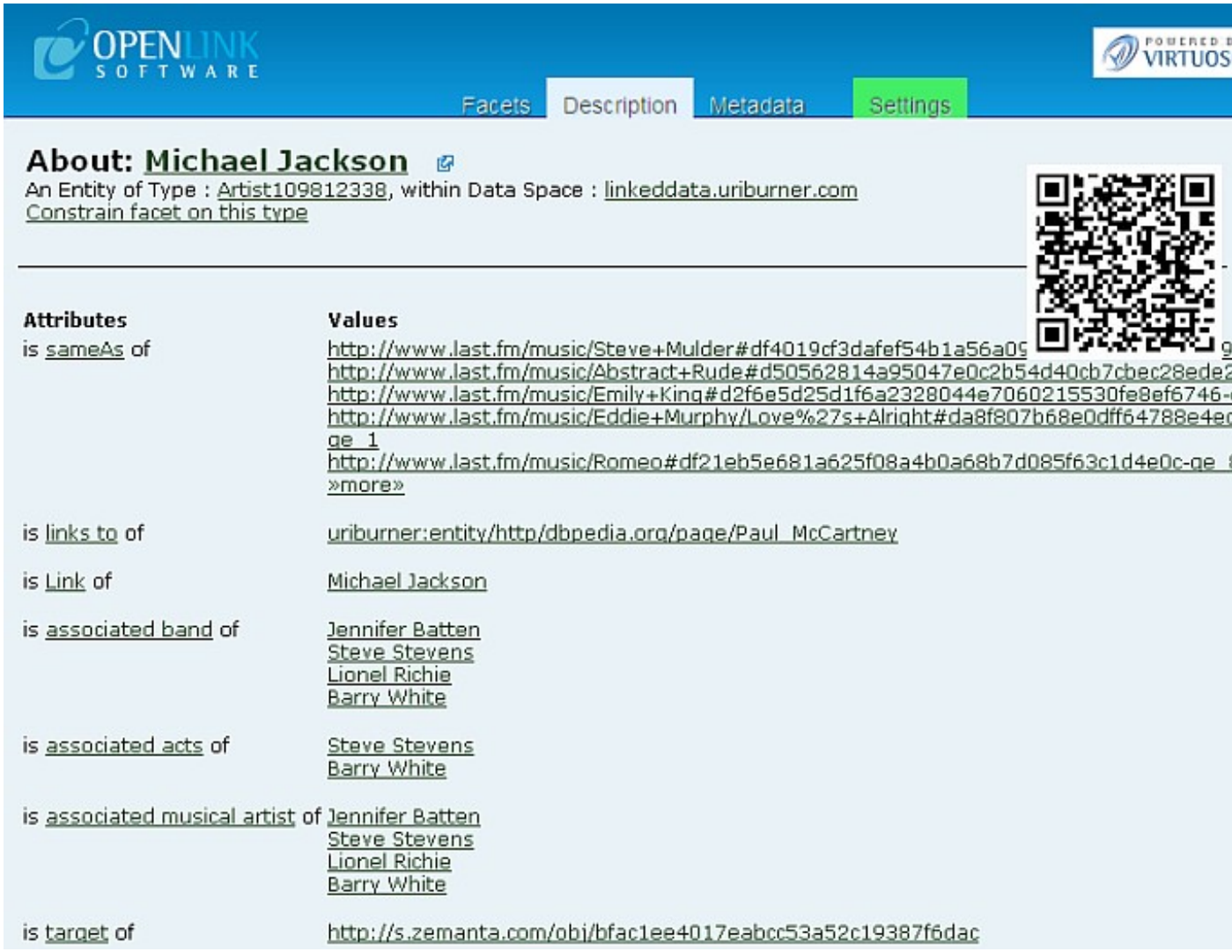
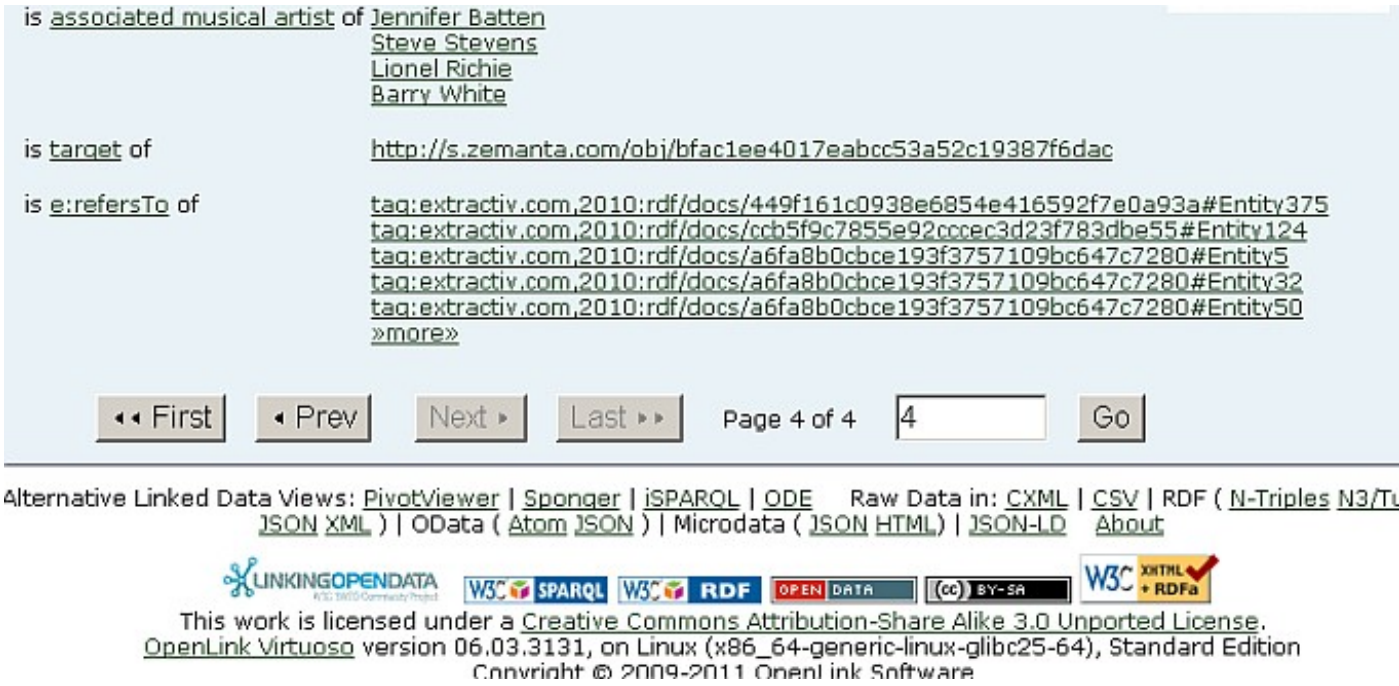


Figure 16.148. Usage Statistics

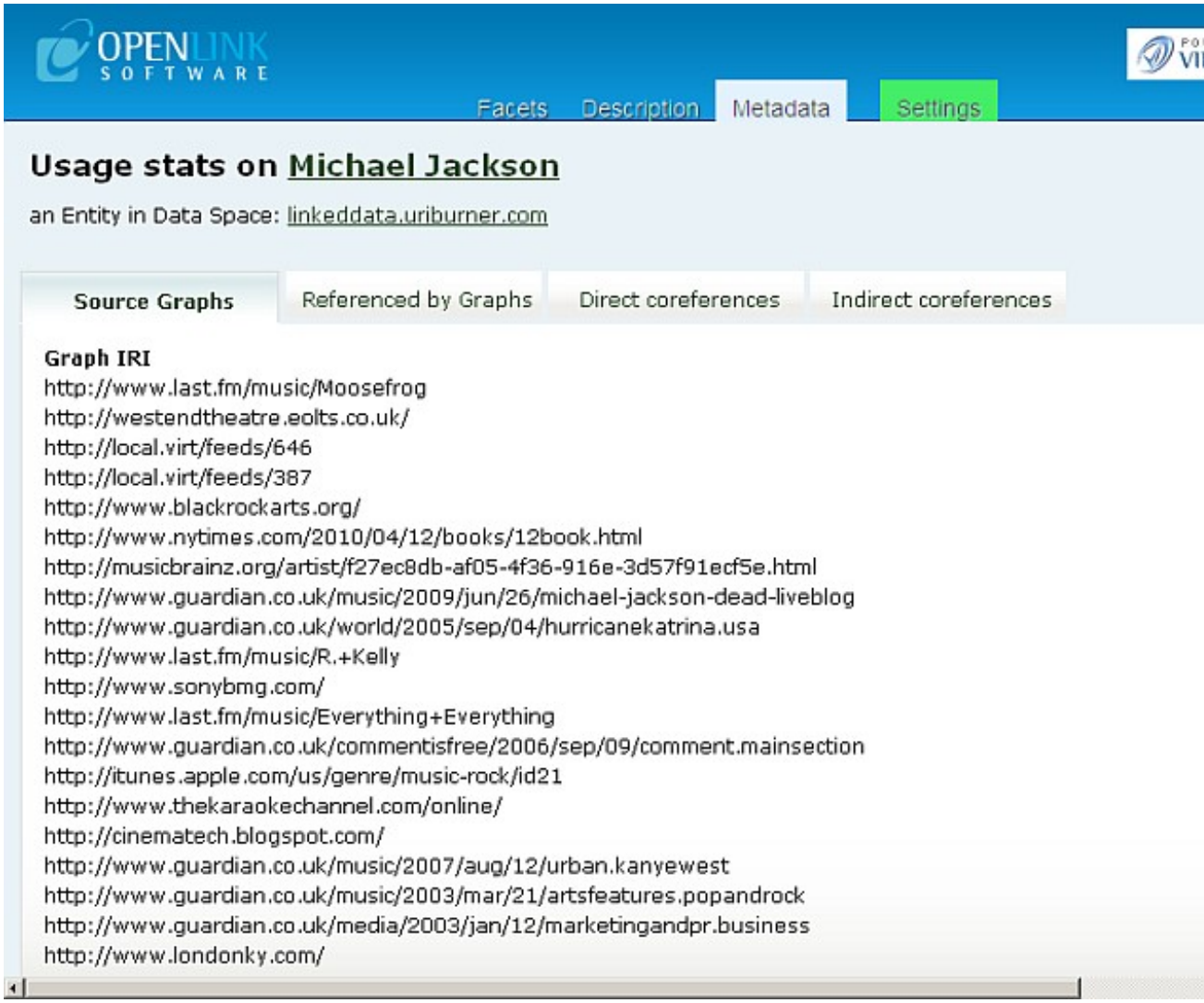


11. "Metadata" tab.
12. Results of usage statistics for "Michael Jackson" grouped in 4 tabs should be shown:

a. Referenced by Graphs: shows how many times the URI is found as subject in the relevant graph(s):

```
SPARQL
SELECT ?g count (*)
where
  {
    graph ?g { <URI> ?p ?o }
  }
group by ?g
order by desc 2
limit 20
```

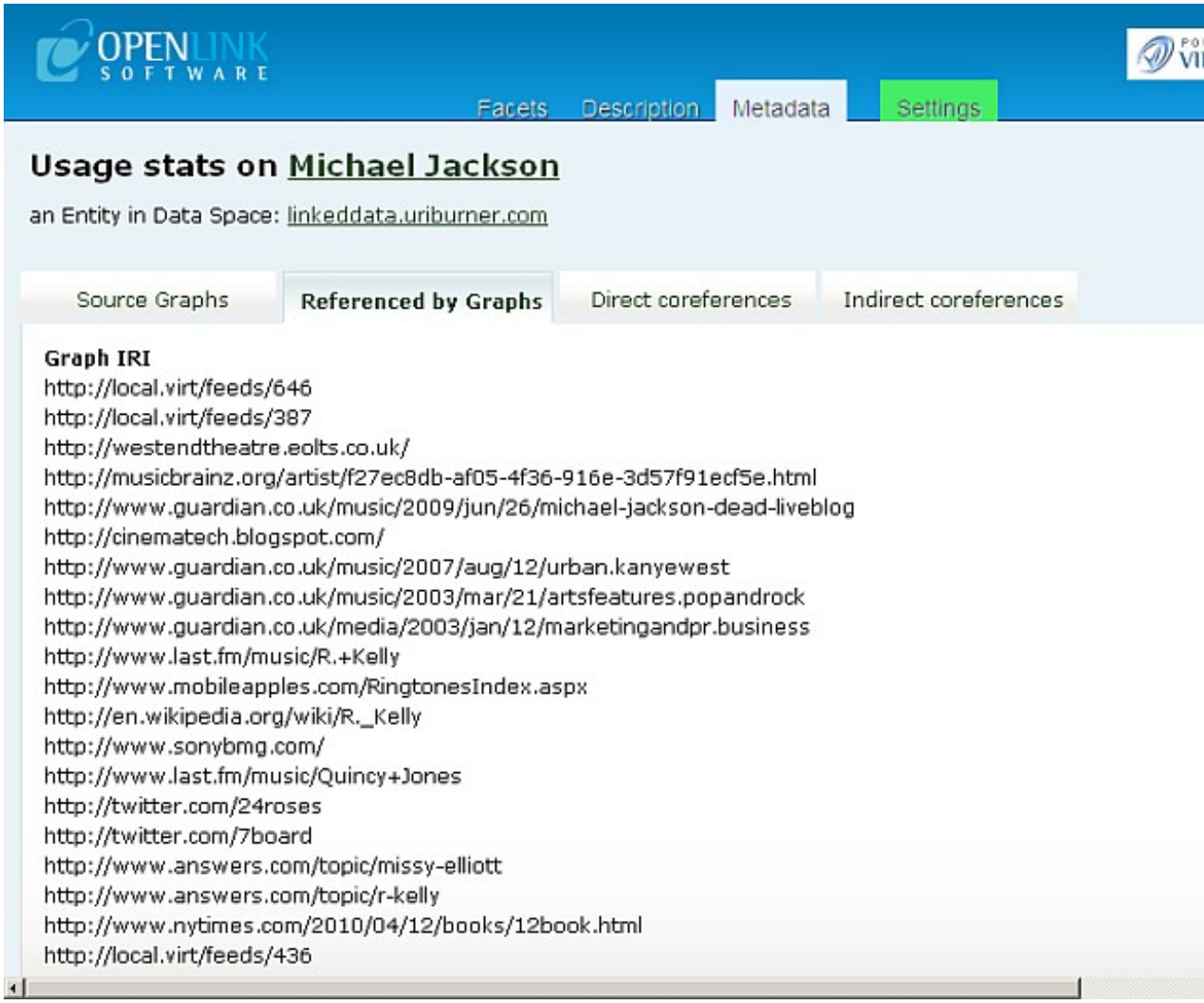
Figure 16.149. Usage Statistics



b. Source Graphs: shows how many times the URI is found as object in the relevant graph(s):

```
SPARQL
SELECT ?g count (*)
where
  {
    graph ?g { ?s ?p <URI>  }
  }
group by ?g
order by desc 2
limit 20
```

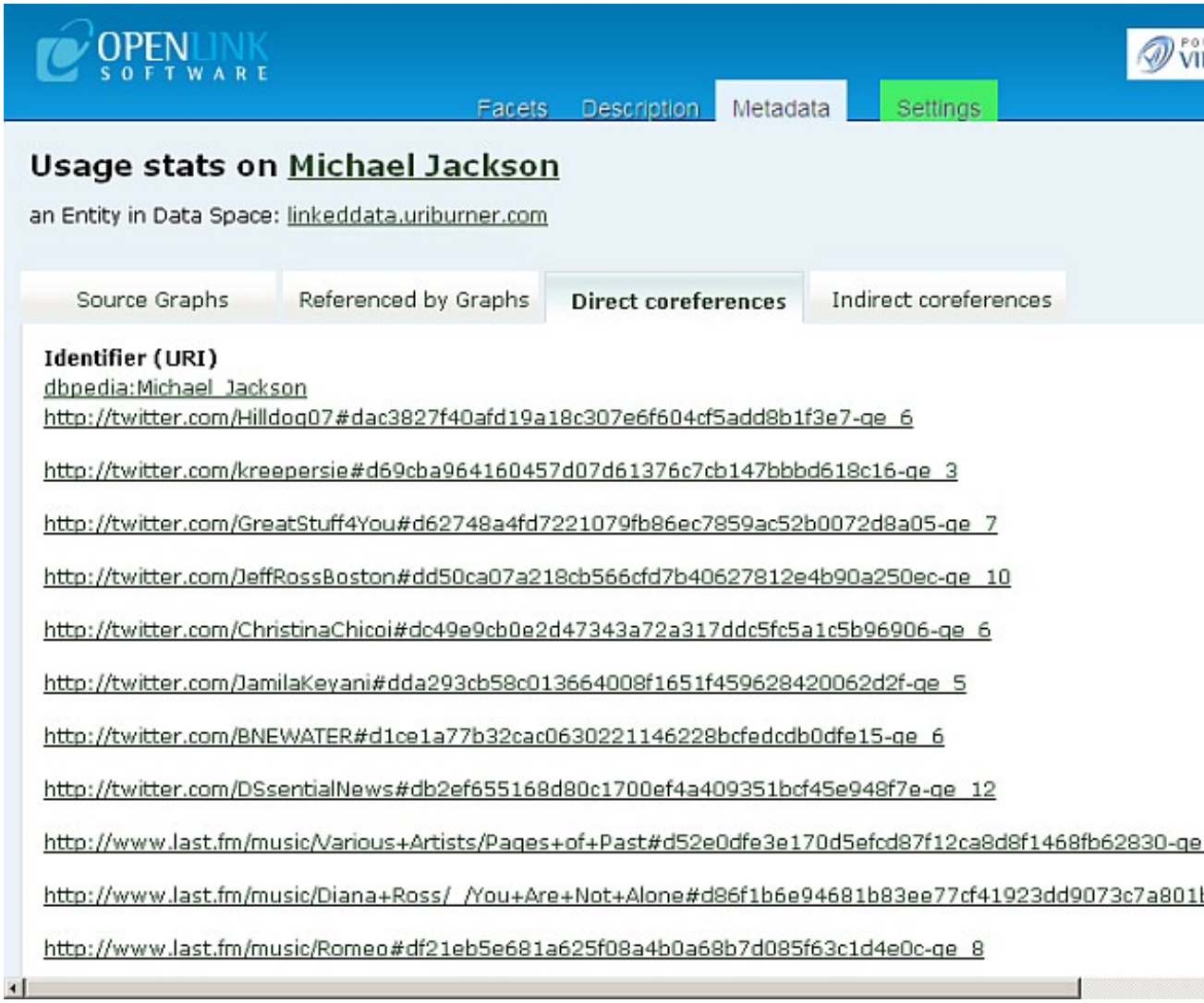
Figure 16.150. Usage Statistics



c. Direct co-references: shows results as subject and calculated rank, based on running transitive closure over owl:sameAs of the URI in subject or object:

```
SPARQL
SELECT ?syn ( sql:rnk_scale (<LONG::IRI_RANK> (?syn)))
where
  {
    { SELECT ?s ?syn
      where
       {
         {?syn owl:sameAs ?s } union {?s owl:sameAs ?syn}
       }
    }
    option (transitive, t_distinct, t_min (0), T_in (?s), t_out (?syn)) . filter (!isliter
  }
order by desc 2
limit 20
```

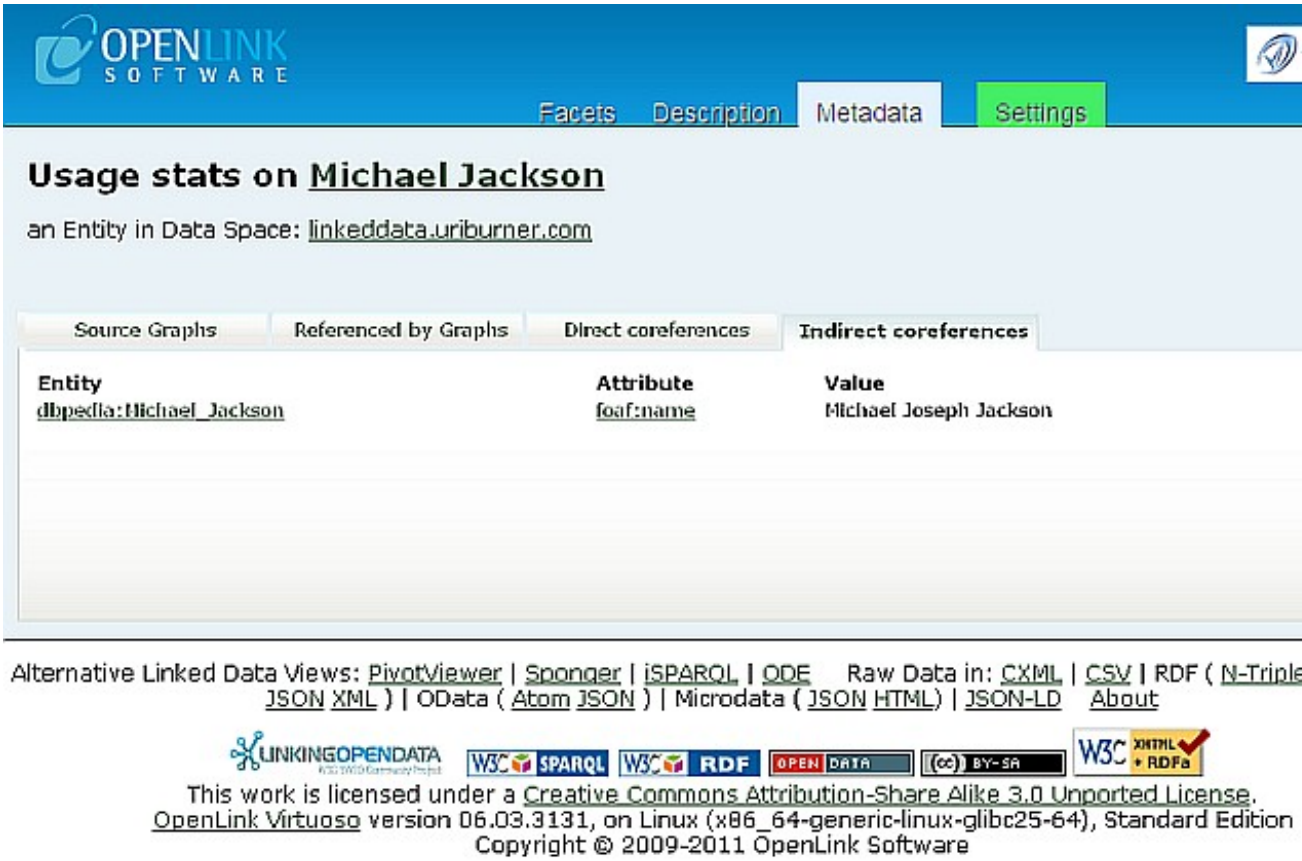
Figure 16.151. Usage Statistics



d. Indirect co-references: shows expanded results for objects concur with the URI by IFP:

```
SPARQL
SELECT distinct ?syn ?p ?o (sql:rnk_scale (<LONG::IRI_RANK> (?syn)))
where
  { <URI> ?p ?o .  filter (0 != (<LONG::bif:rdf_is_sub> ("b3sifp", ?p, lod:ifp_like, 3))) .
    ?syn ?p ?o .
}
order by desc 4
limit 20
```

Figure 16.152. Usage Statistics



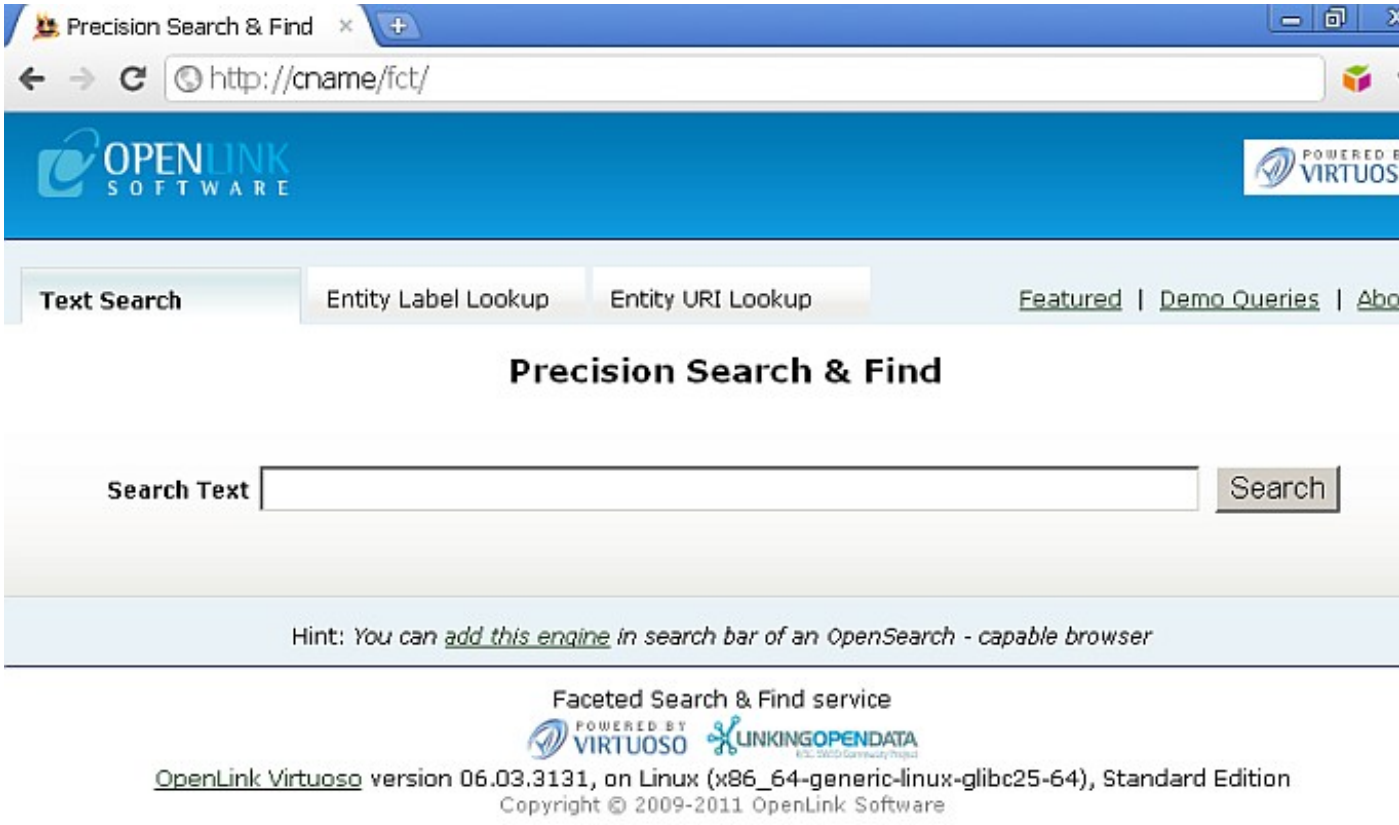
16.11.7. Examples

Faceted Browsing Sample using LOD Cloud Cache data space

The following example demonstrates a simple scenario of tracking Kingsley Idehen's conversations across the Web, using the Virtuoso Faceted Browser hosted on LOD.

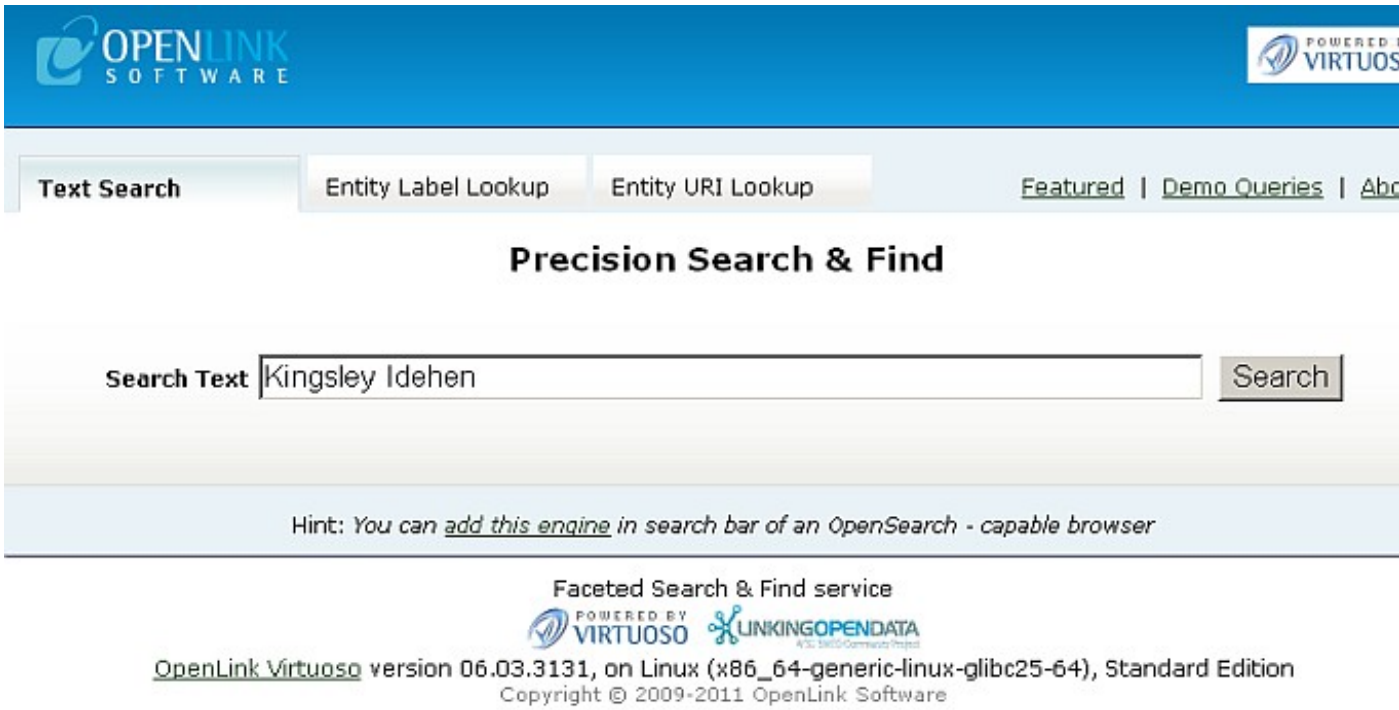
1. Go to http://lod.openlinksw.com/fct/

Figure 16.153. Faceted Navigation Example



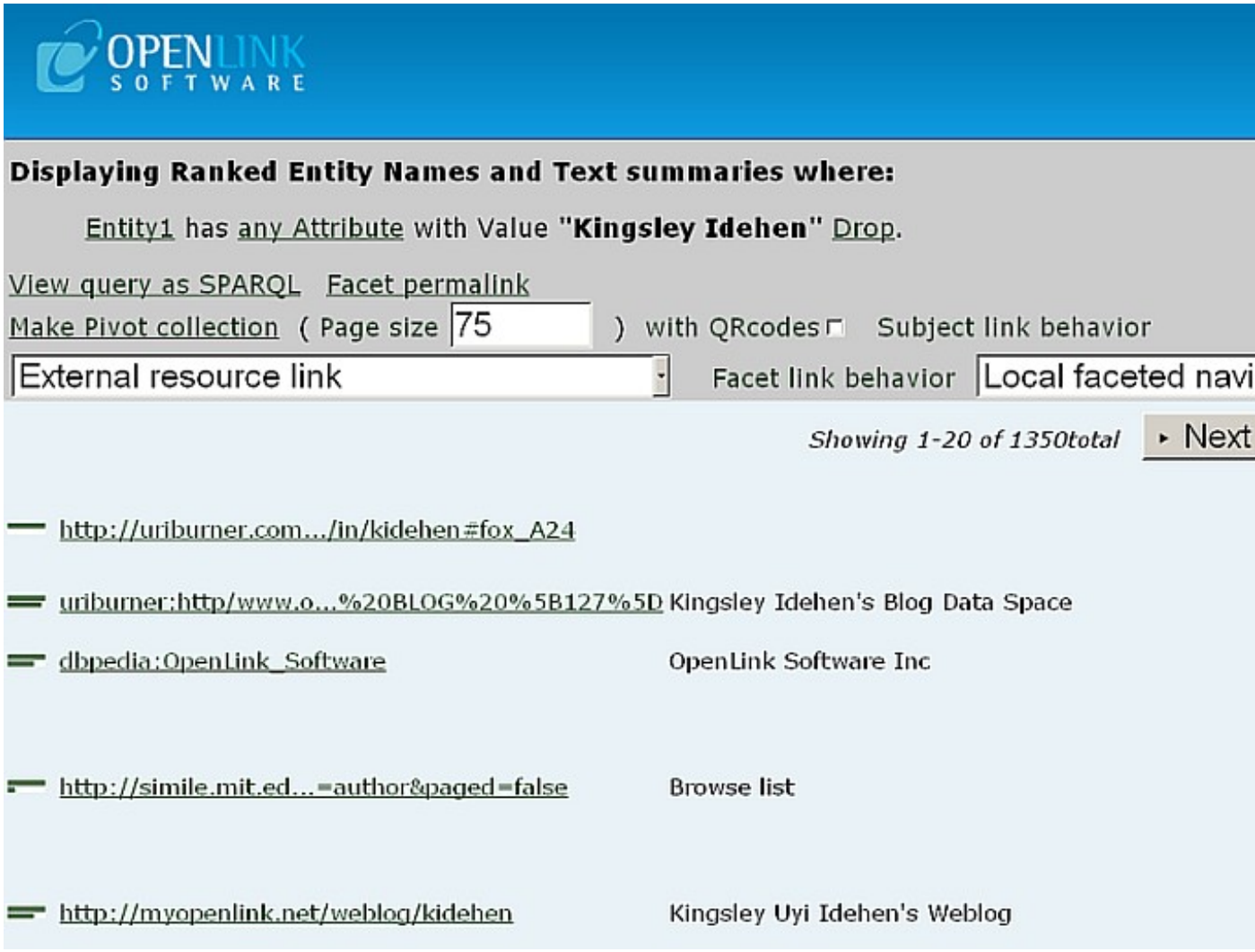
2. Enter a free text search pattern (for example, "Kingsley Idehen"), and click Search

Figure 16.154. Faceted Navigation Example



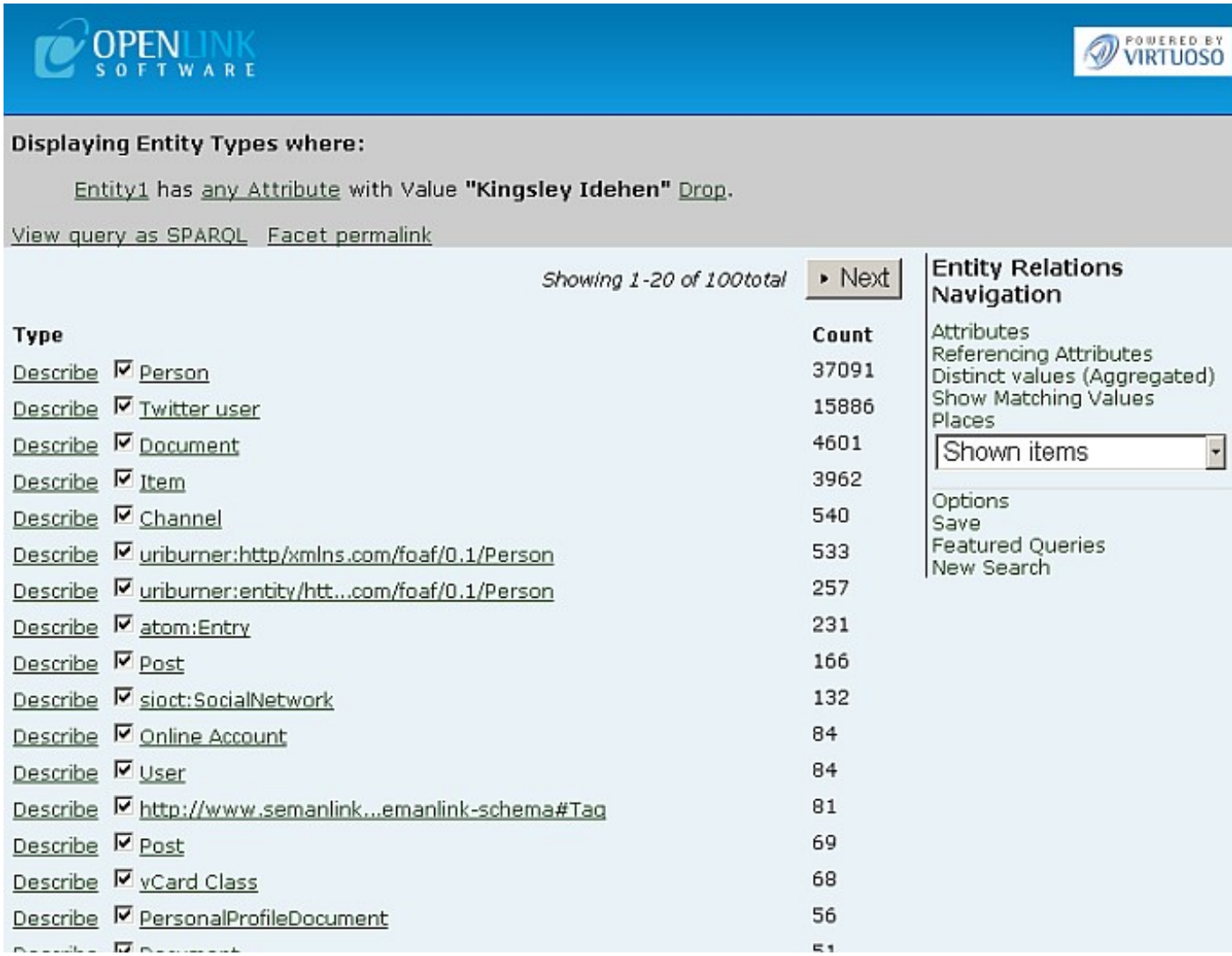
3. Your initial query results page will display a list of literal value snippets from property values associated with the query text pattern

Figure 16.155. Faceted Navigation Example



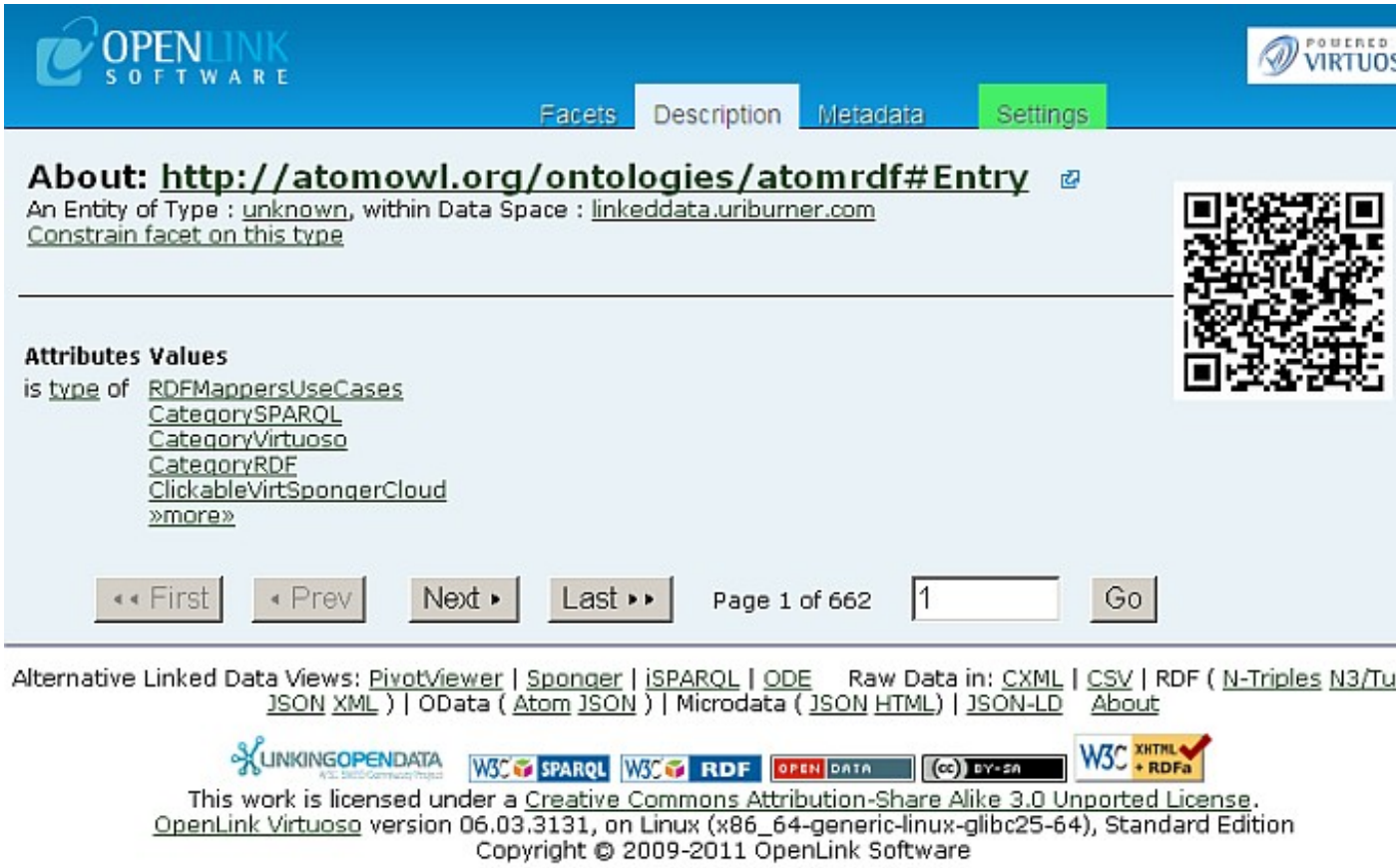
4. Using the Navigation section on the right, click on "Types", which alters the contents of the query results area by presenting CURIE based hyperlinks for each of the Entity Types associated with Property values that contains the query text pattern

Figure 16.156. Faceted Navigation Example



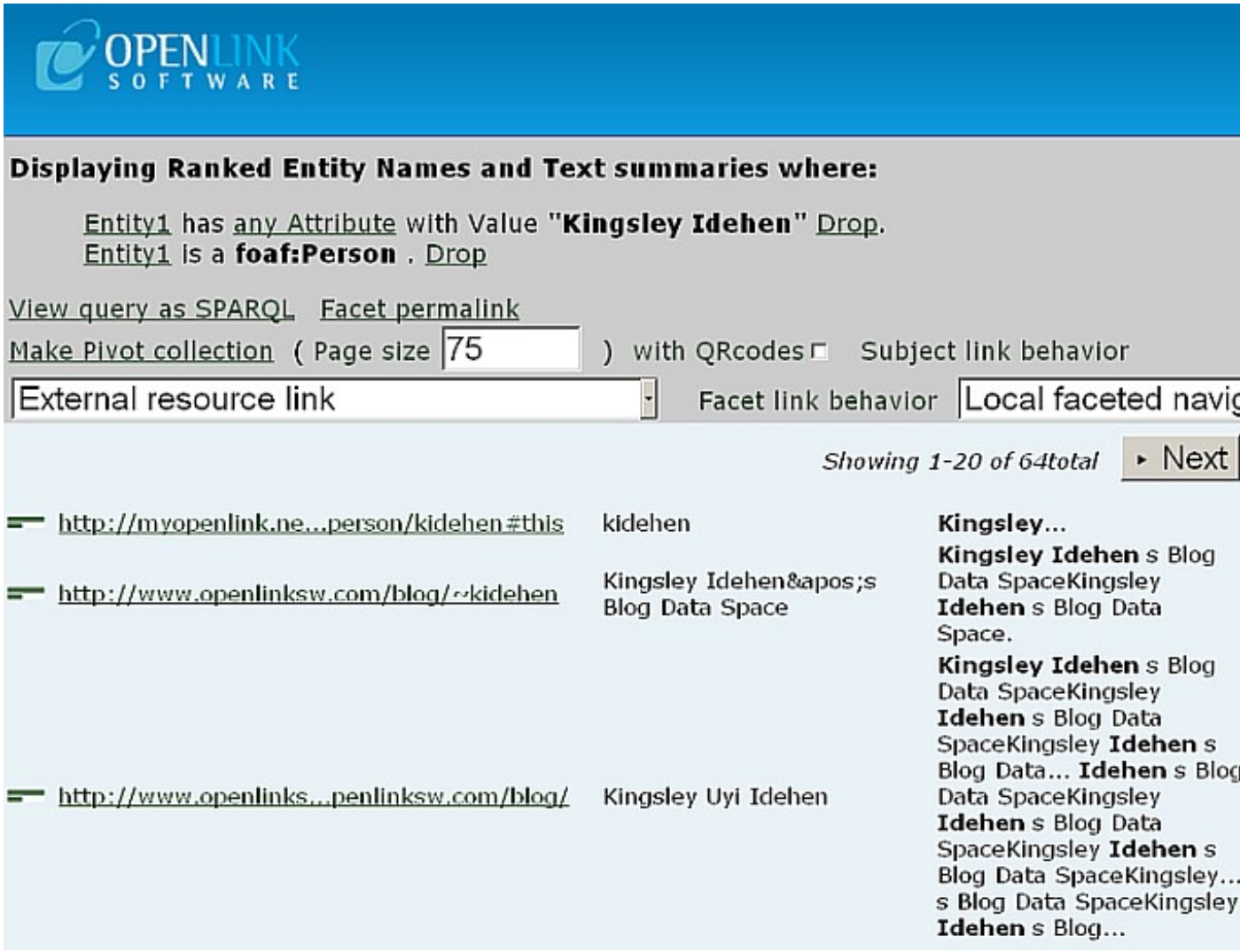
5. You can perform *Describe* for a given found type, by clicking the "Describe" link in the "Type" column. For ex, for "atom:Entry" the produced describe type page would show a list of Attributes and Values + automatically generated QRCode image:

Figure 16.157. Faceted Navigation Example



6. Click "Facets" tab to return to the Types content page from the previous step.
7. Click on the "foaf:Person" link to narrow the result set down to Entities of this Type, un-hatch the checkbox beside this link for Negation (filtering out) based on this Entity Type

Figure 16.158. Faceted Navigation Example



- For Negation (filtering out) based on this Entity Type un-hatch the check-box shown besides the link:

Figure 16.159. Faceted Navigation Example

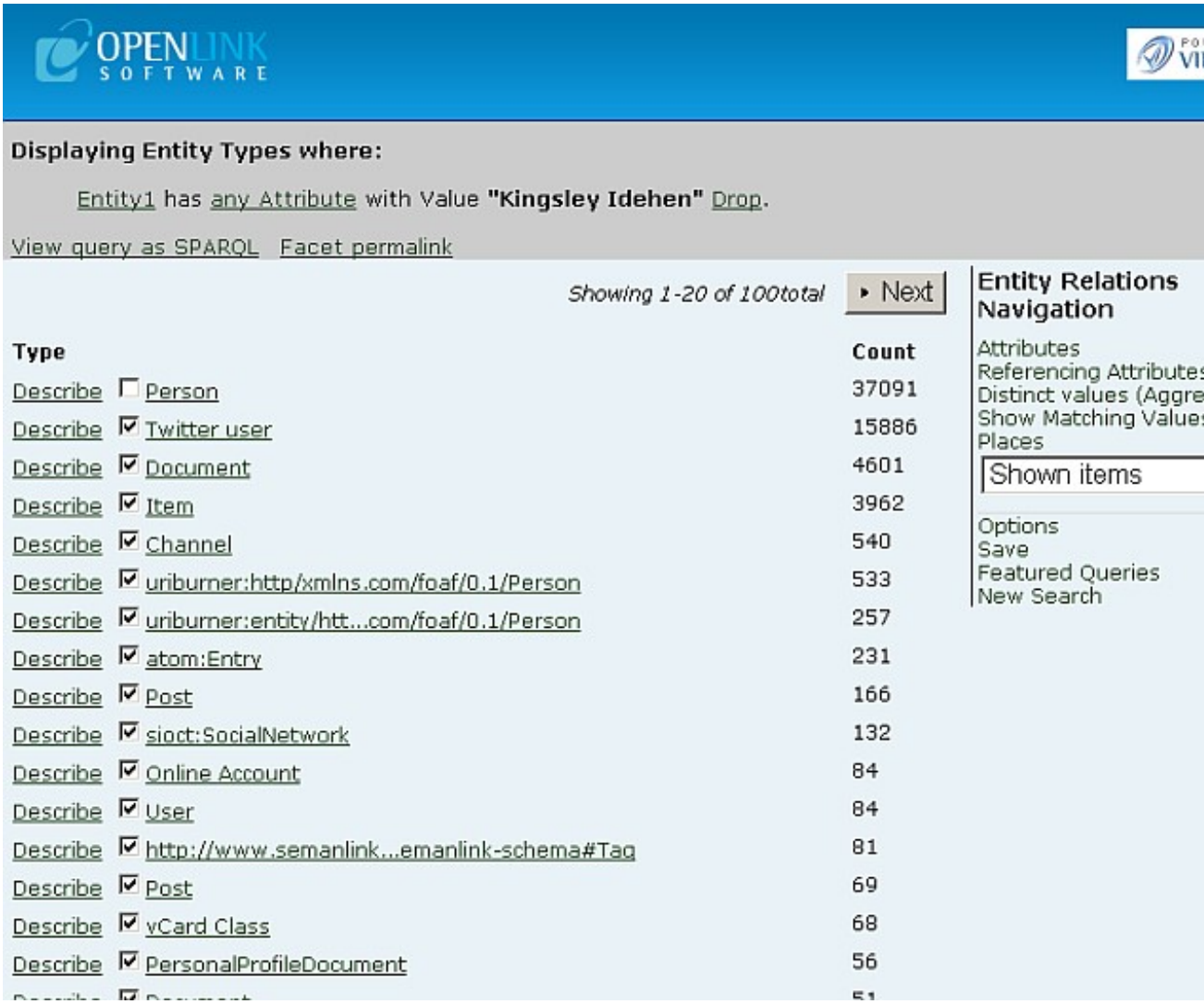
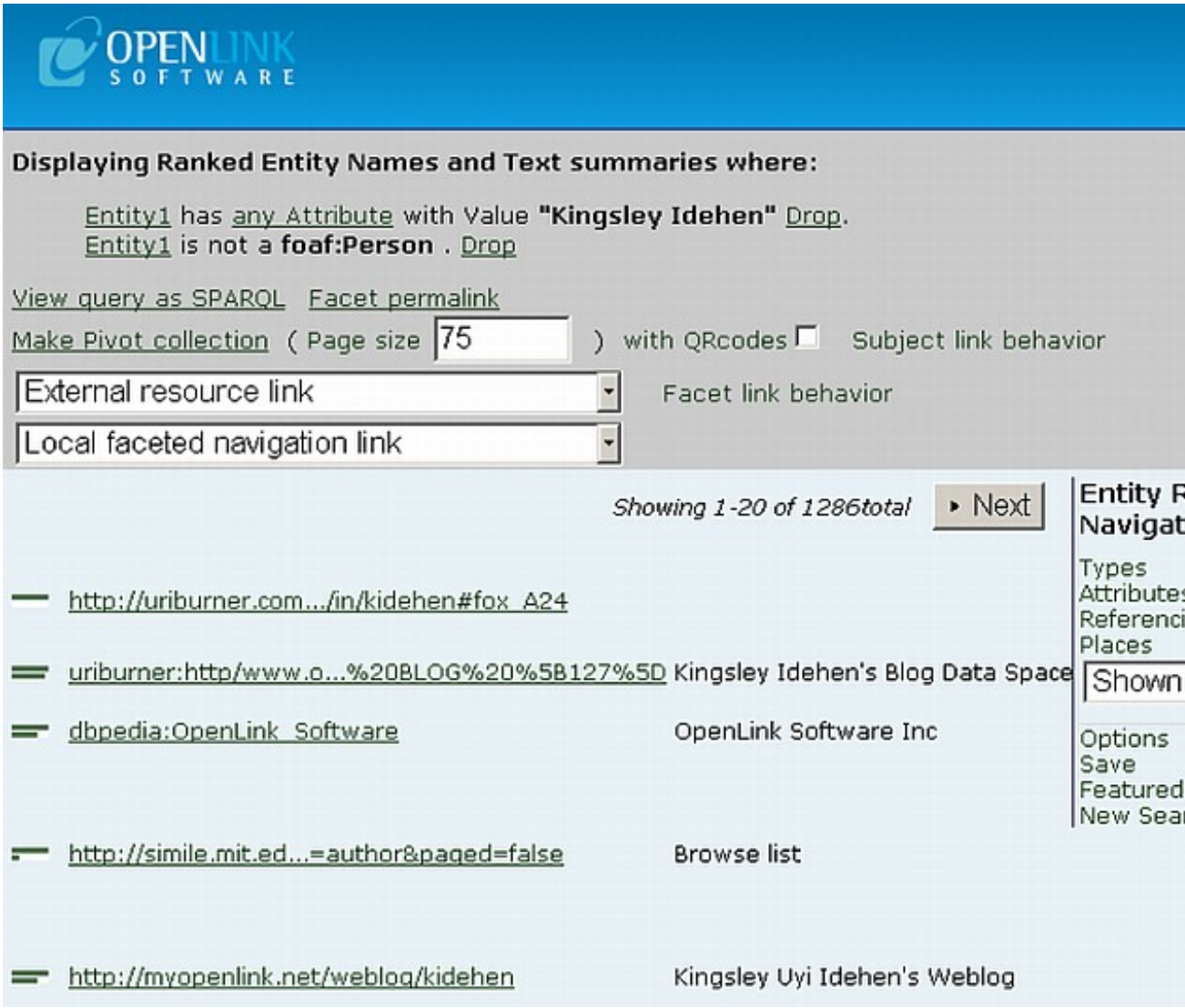
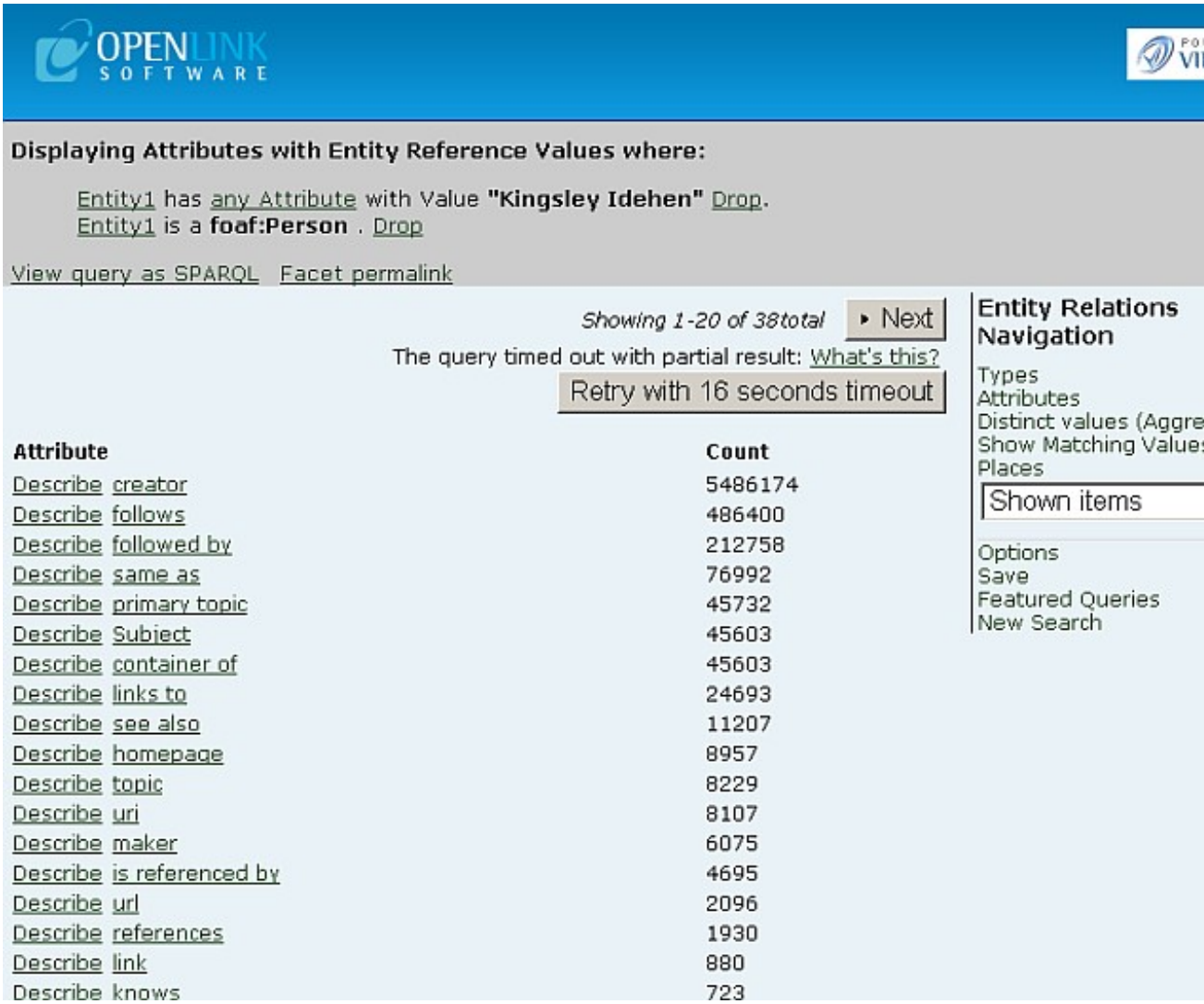


Figure 16.160. Faceted Navigation Example



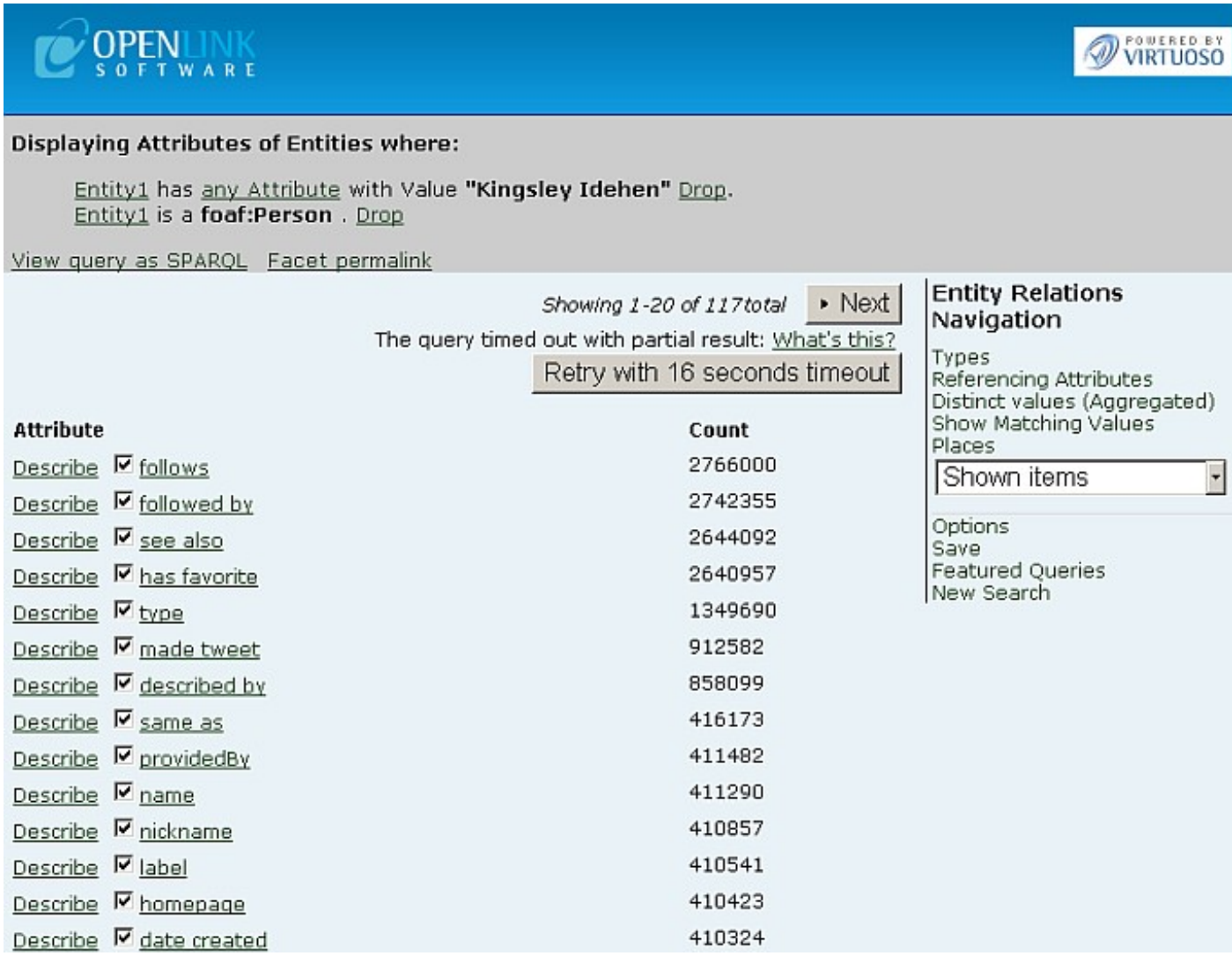
- You can filter further, by switching (pivoting) to the a Property based view, by returning to the Navigation section and then clicking on "Properties" or "Referencing Properties" links; in either case, you have further filtering of based on the combination of Properties and Entities where Entities in the result-set contain values matching the query text pattern

Figure 16.161. Faceted Navigation Example



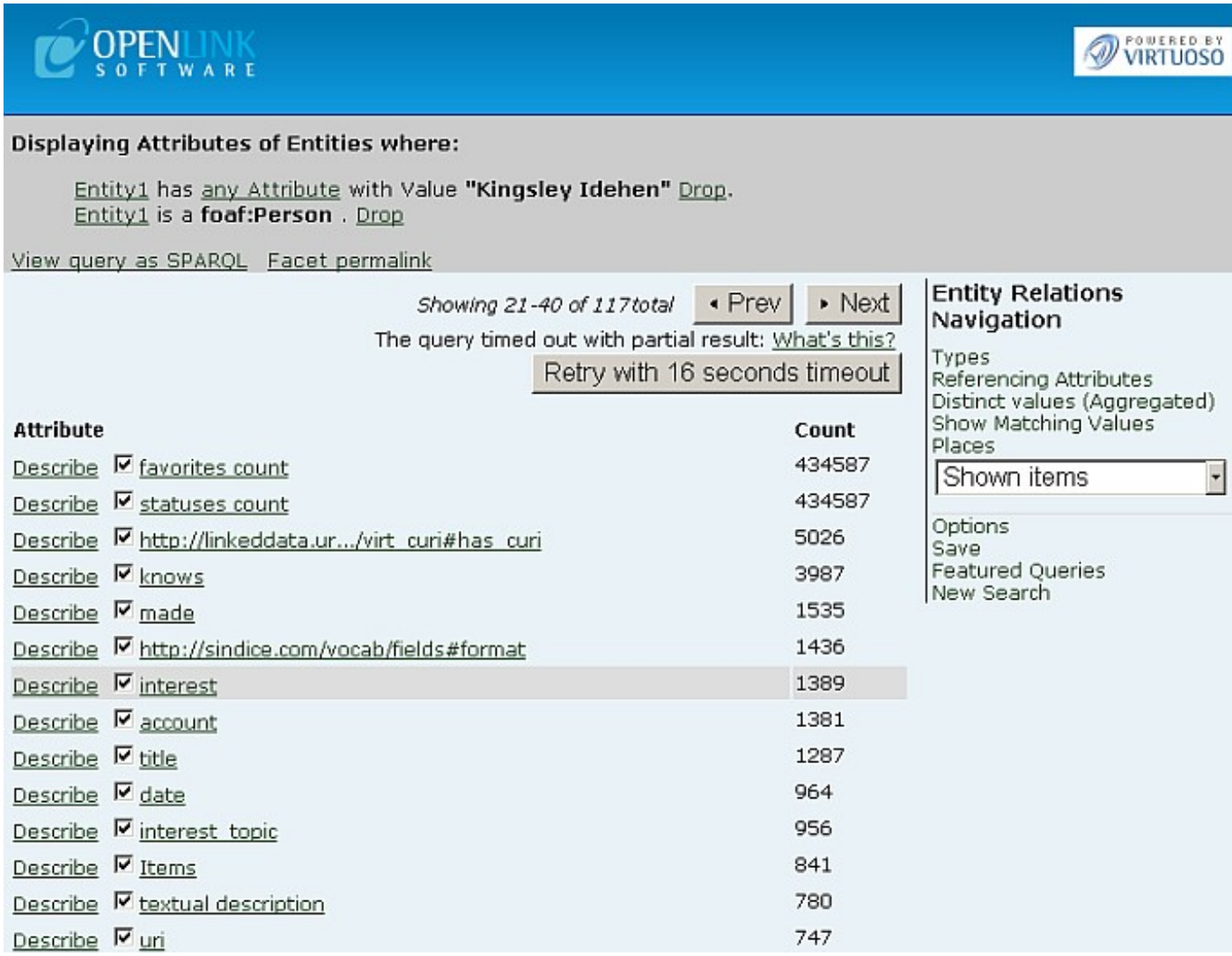
8. From "Entity Relations Navigation" click "Attributes".

Figure 16.162. Faceted Navigation Example



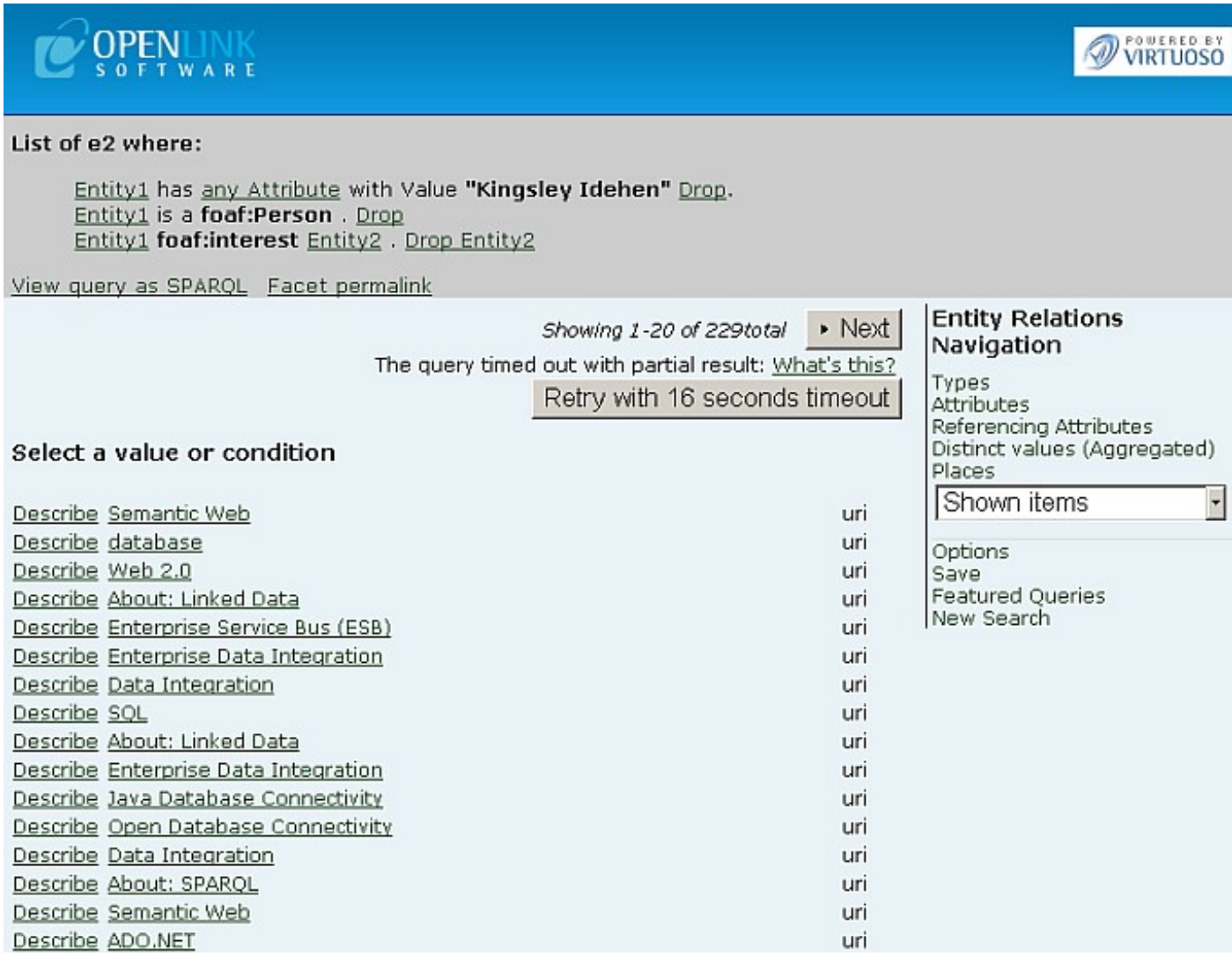
9. You can navigate amongst the search results pages by using the "Prev" and "Next" buttons. Click for ex. "Next":

Figure 16.163. Faceted Navigation Example



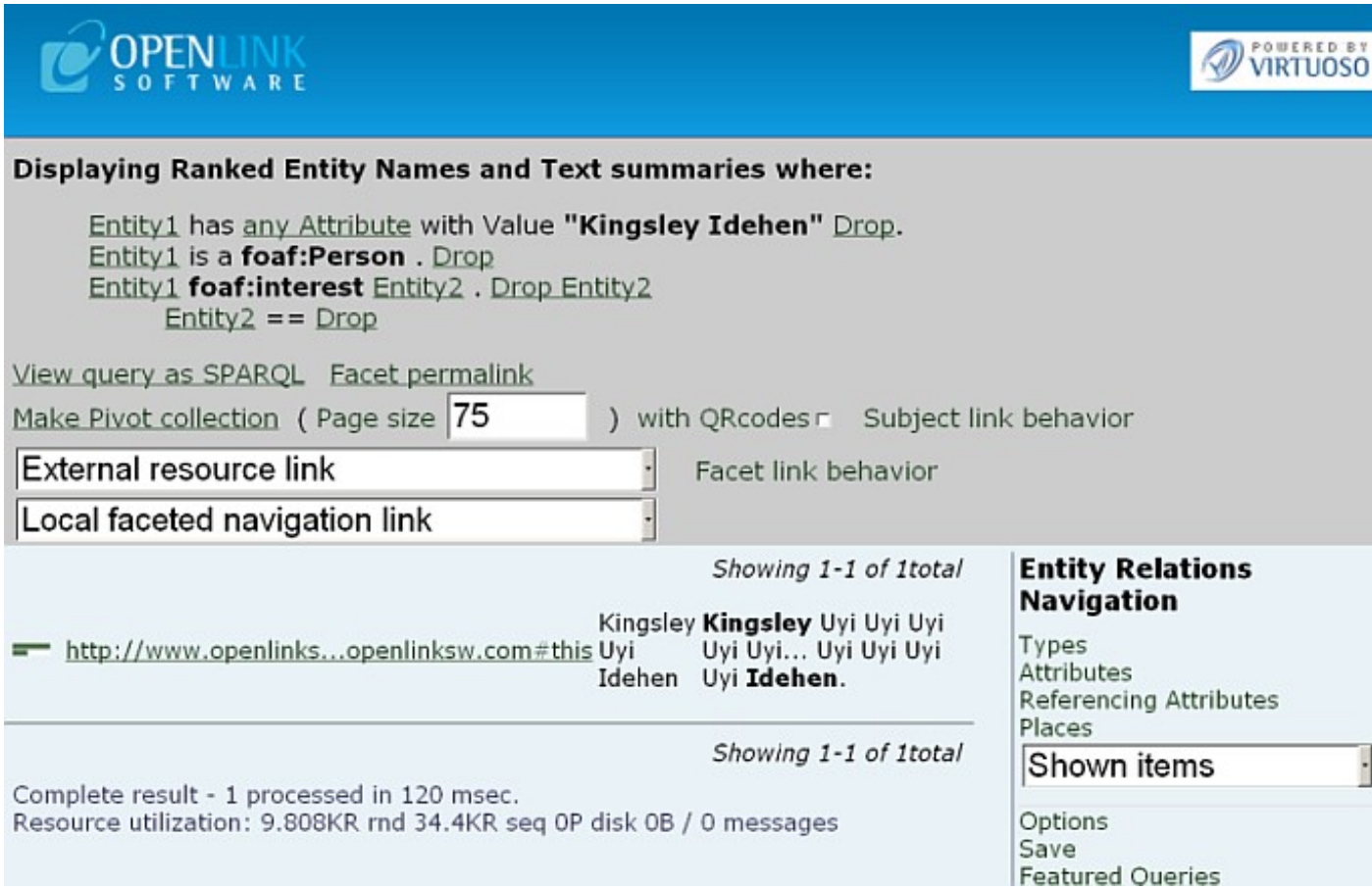
10. From the list of Property Types, click on the "foaf:interest" link to filter further, based on the values of this property:

Figure 16.164. Faceted Navigation Example



11. From the list of "foaf:interest" Values, click on "About:Linked Data", which filters the result-set further to display reveal Entity Identifier Links (Generic HTTP URIs) and Labels for each "foaf:Person" associated with the property "foaf:interest", in the LOD data space:

Figure 16.165. Faceted Navigation Example



12. Click on one of the HTTP URIs in the filtered results-set to obtain a detailed structured description of a given Entity i.e. about the person Kingsley Uyi Idehen. Each listed Property is a

    Link

    ; thus, each Property is a link to other structured Entity descriptions. Additionally, a QRCode image will be produced automatically for the given entity:

    Figure 16.166. Faceted Navigation Example

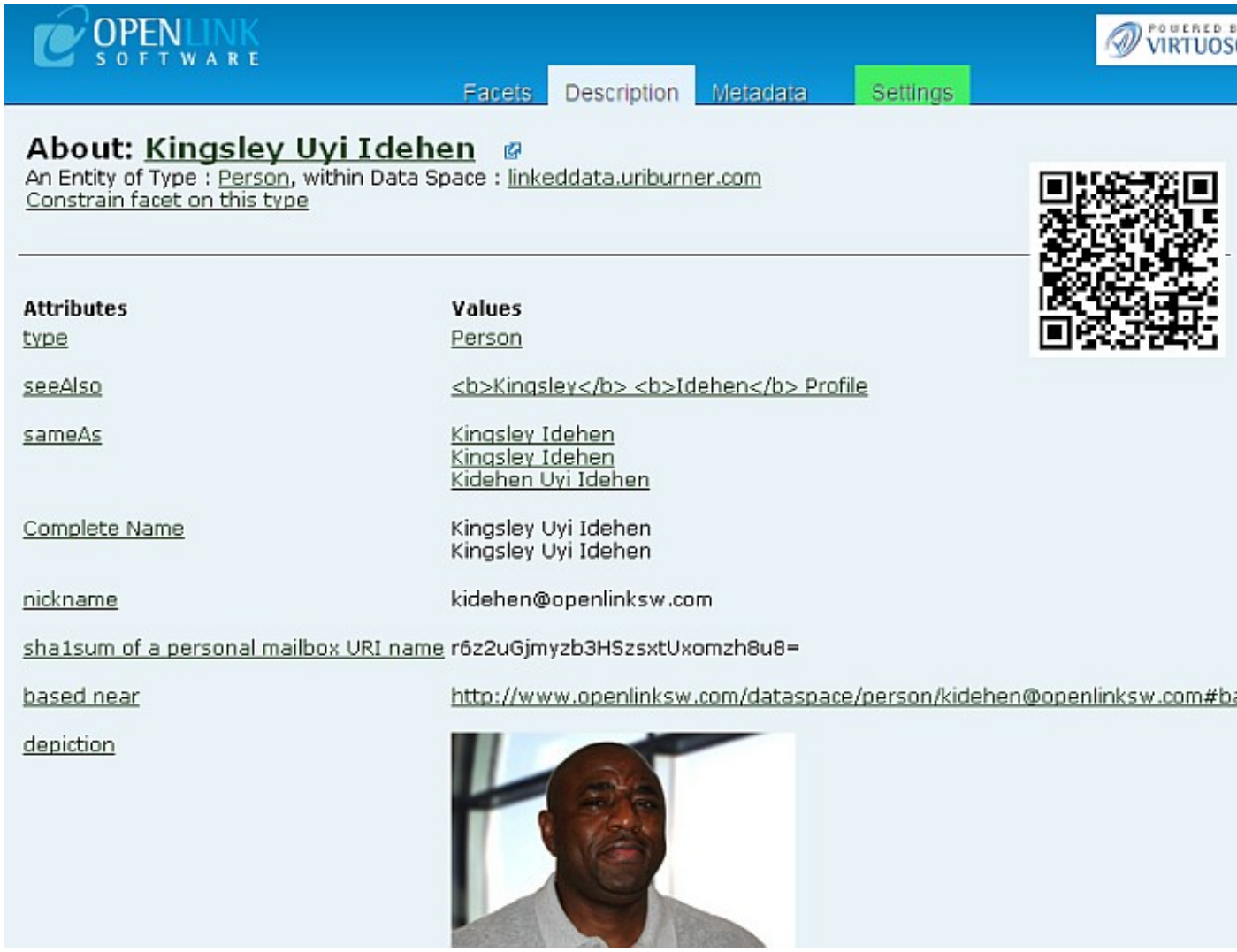


Figure 16.167. Faceted Navigation Example

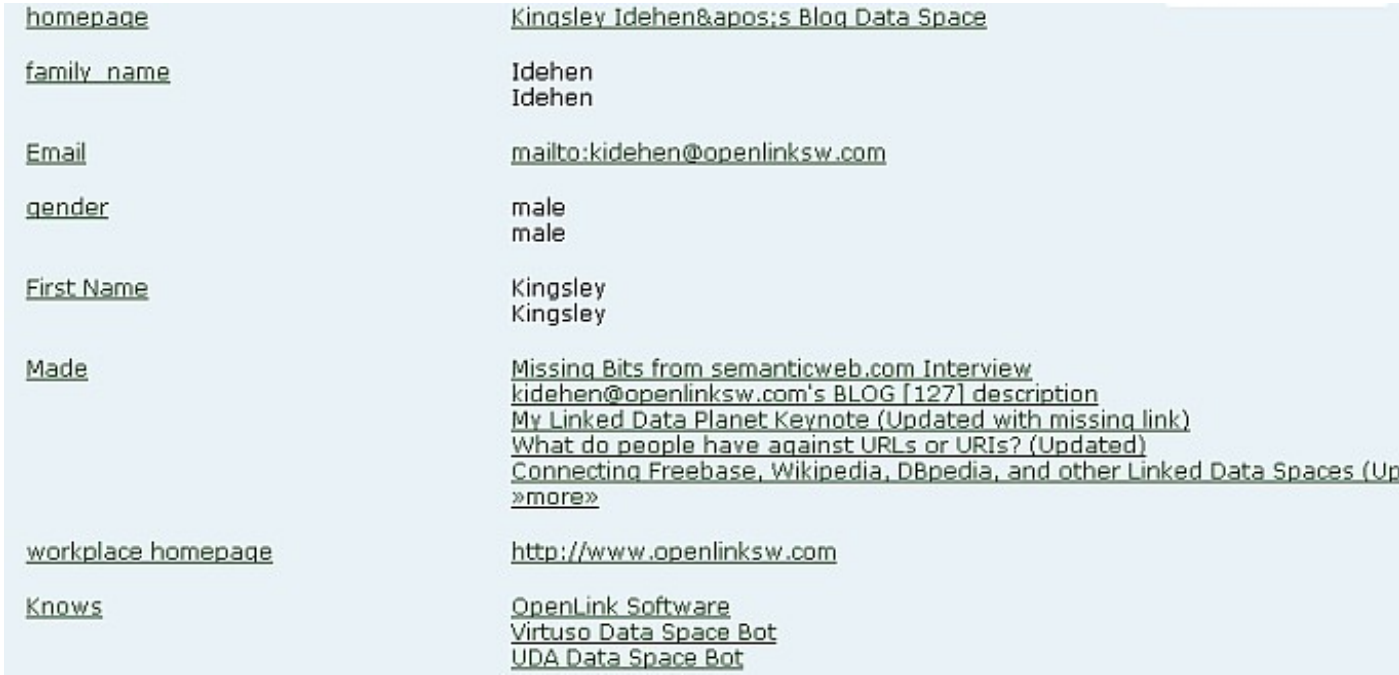
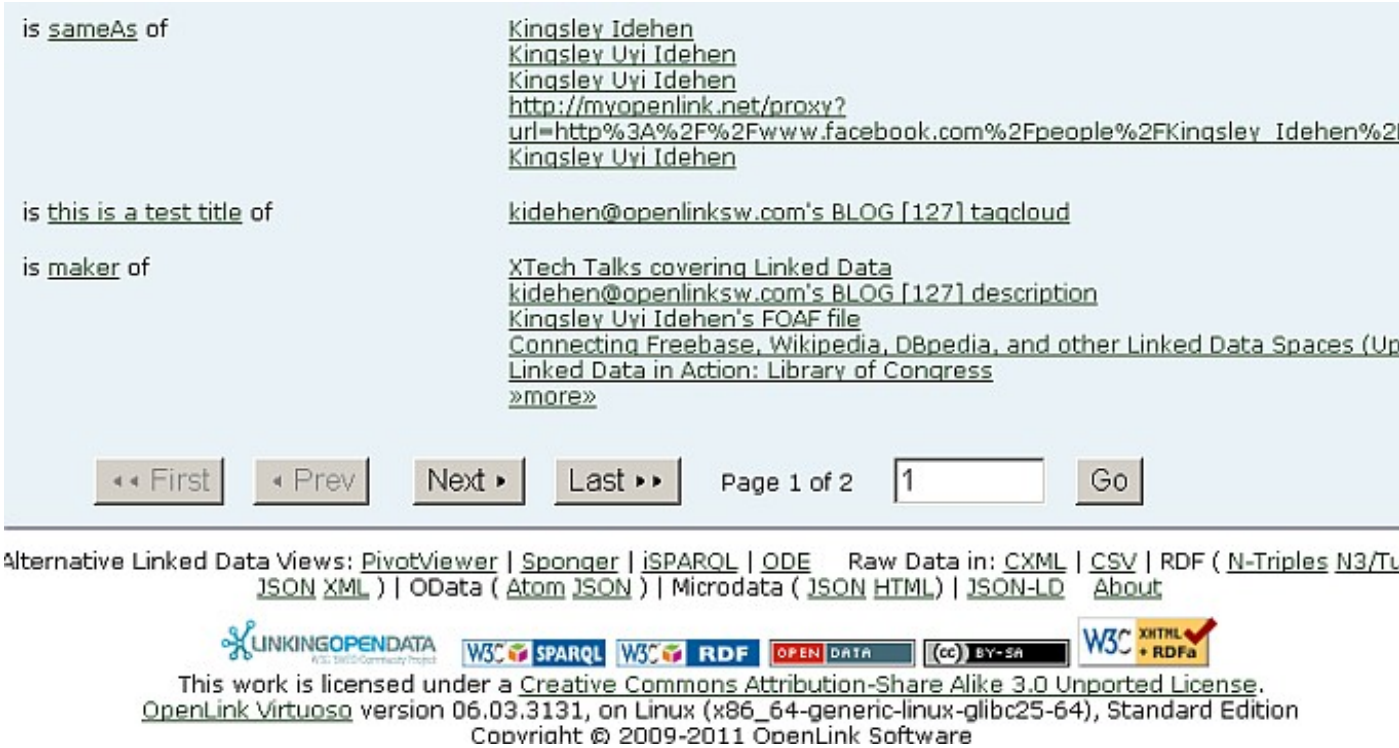


Figure 16.168. Faceted Navigation Example

Figure 16.169. Faceted Navigation Example



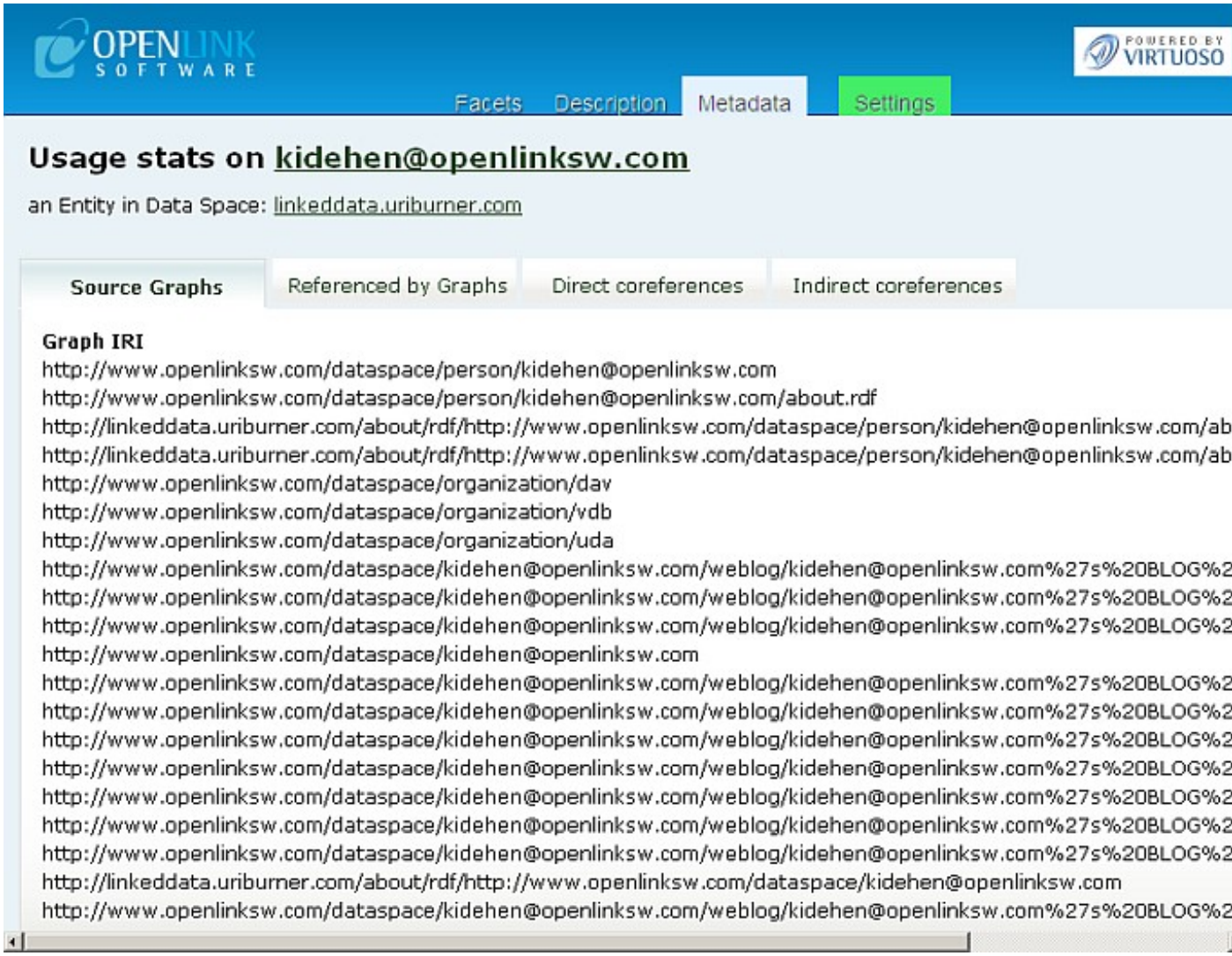
13. You can navigate amongst the search results pages by using the "First", "Prev", "Next" and "Last" buttons. Click for ex. "Last":

Figure 16.170. Faceted Navigation Example

Figure 16.171. Faceted Navigation Example

14. Click on " *Metadata* " link to get a summary view of this Linked Data Space, "Source" and "Reference" graphs are akin to saying "Table X" and "Table Y" where each table is the container of Records re. RDBMS or Worksheet re. Spreadsheet.:

Figure 16.172. Faceted Navigation Example



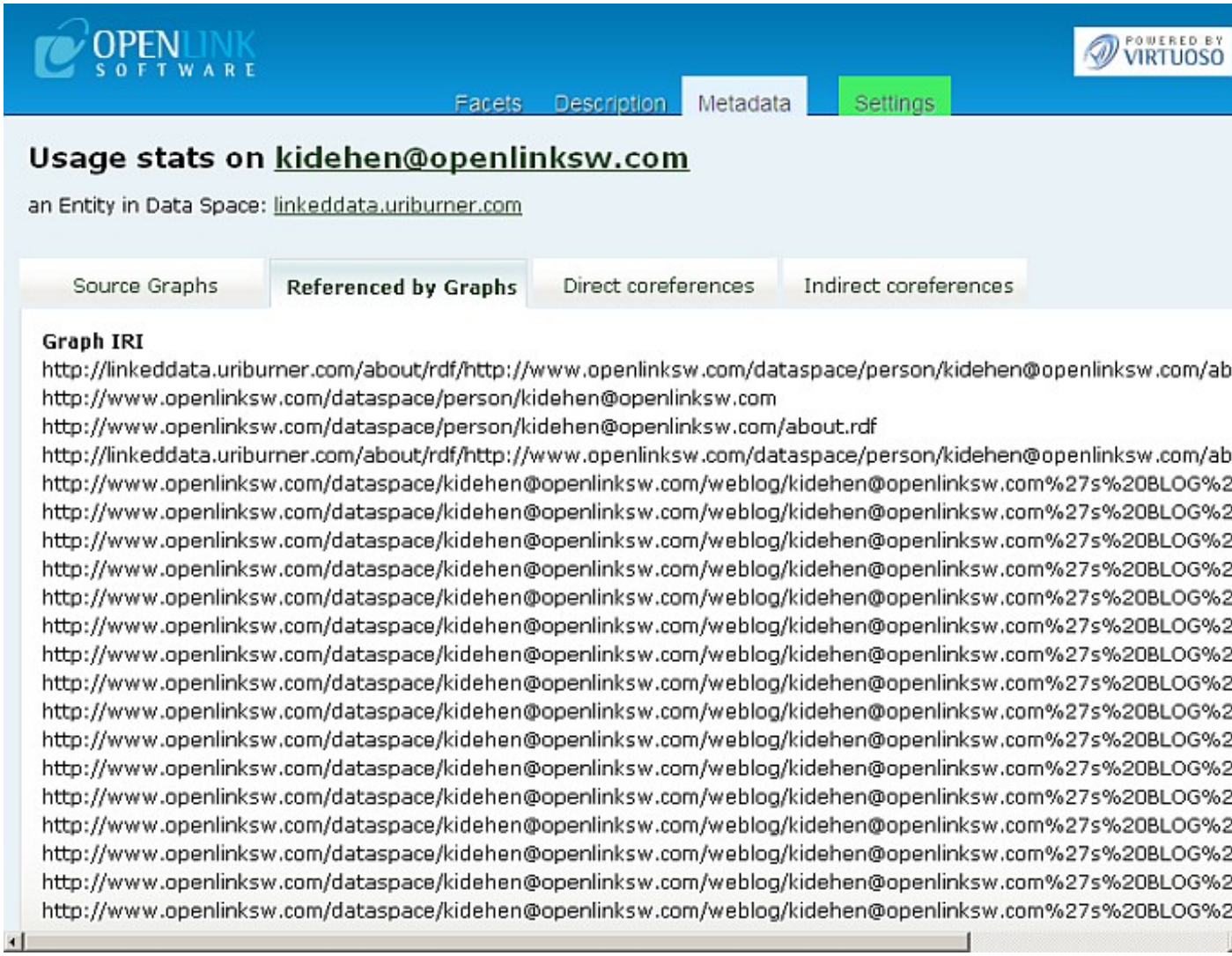
Usage stats on kidehen@openlinksw.com

an Entity in Data Space: linkeddata.uriburner.com

Graph IRI

http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com
http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com/about.rdf
http://linkeddata.uriburner.com/about/rdf/http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com/ab
http://linkeddata.uriburner.com/about/rdf/http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com/ab
http://www.openlinksw.com/dataspace/organization/dav
http://www.openlinksw.com/dataspace/organization/vdb
http://www.openlinksw.com/dataspace/organization/uda
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%2
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%2
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%2
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%2
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%2
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%2
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%2
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%2
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%2
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%2
http://linkeddata.uriburner.com/about/rdf/http://www.openlinksw.com/dataspace/kidehen@openlinksw.com
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%2

Figure 16.173. Faceted Navigation Example



Usage stats on kidehen@openlinksw.com

an Entity in Data Space: linkeddata.uriburner.com

Graph IRI

http://linkeddata.uriburner.com/about/rdf/http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com/ab
http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com
http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com/about.rdf
http://linkeddata.uriburner.com/about/rdf/http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com/ab
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%2
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%2
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%2
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%2
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%2
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%2
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%2
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%2
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%2
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%2
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%2
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%2
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%2
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%2
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%2
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%2

15. "Direct" and "InDirect" coreferences show other references (Identifiers) that relate associated with Kingsley Idehen (like saying: here are his other names or his know by this name in this other place):

    Figure 16.174. f21 Navigation Example

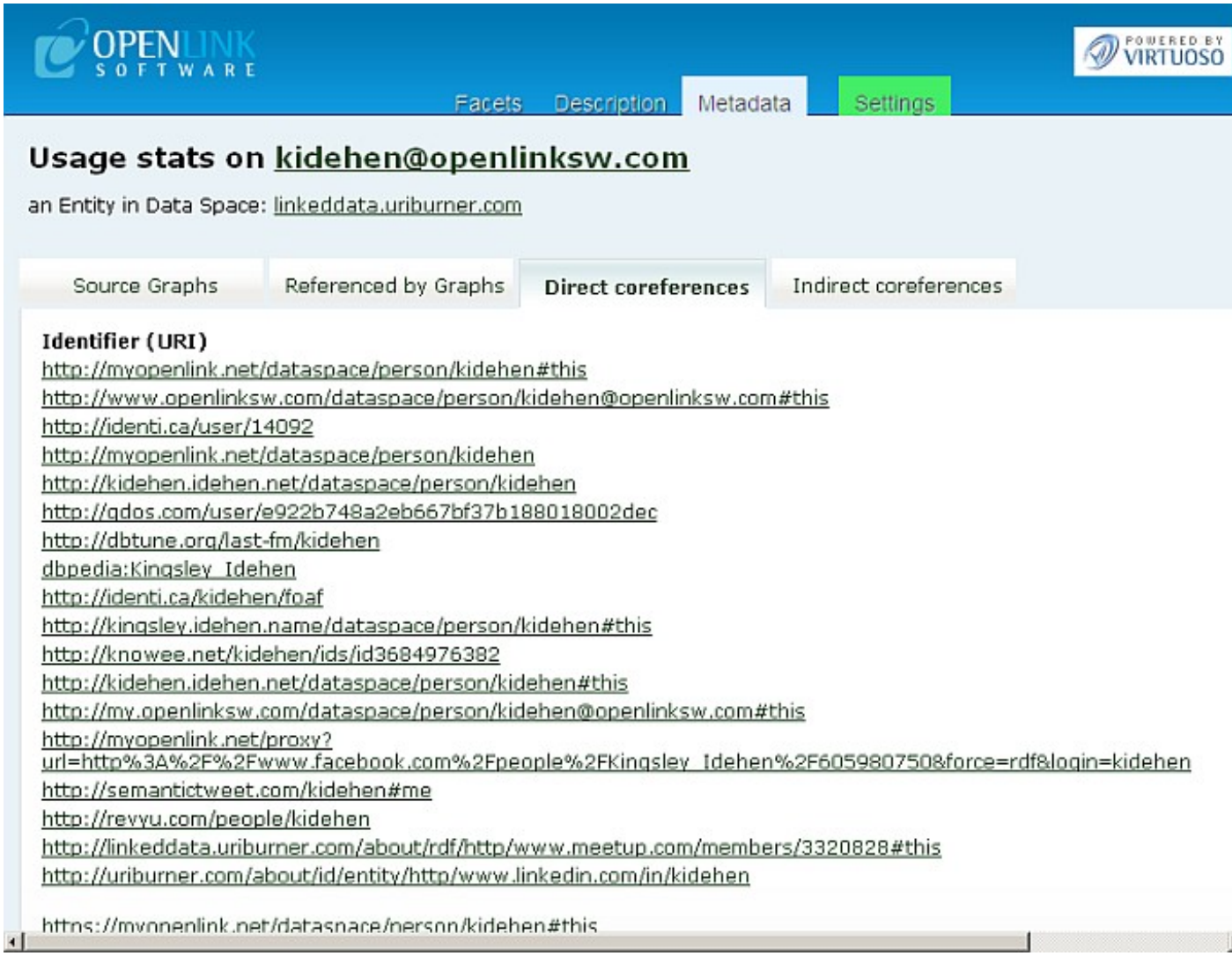
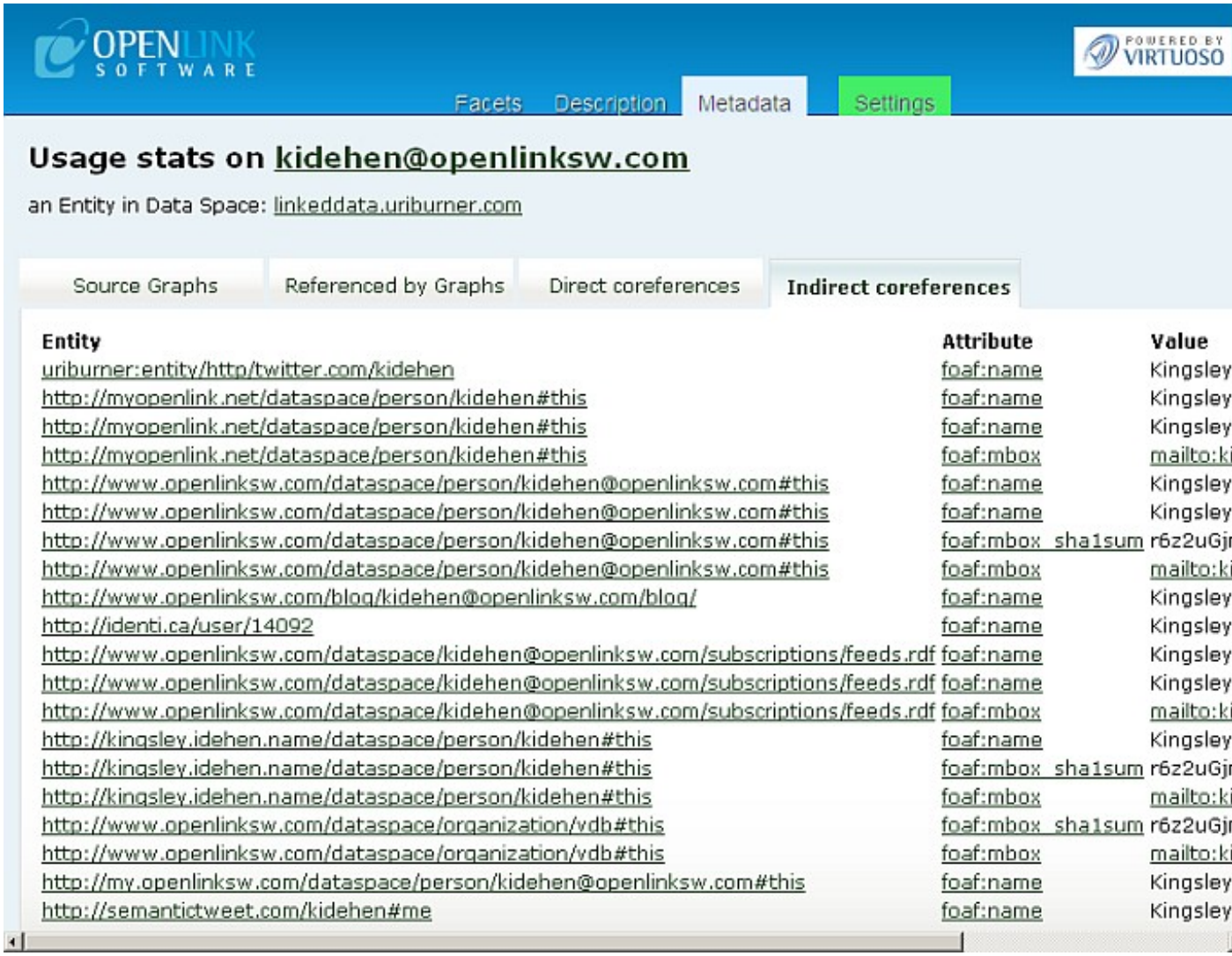
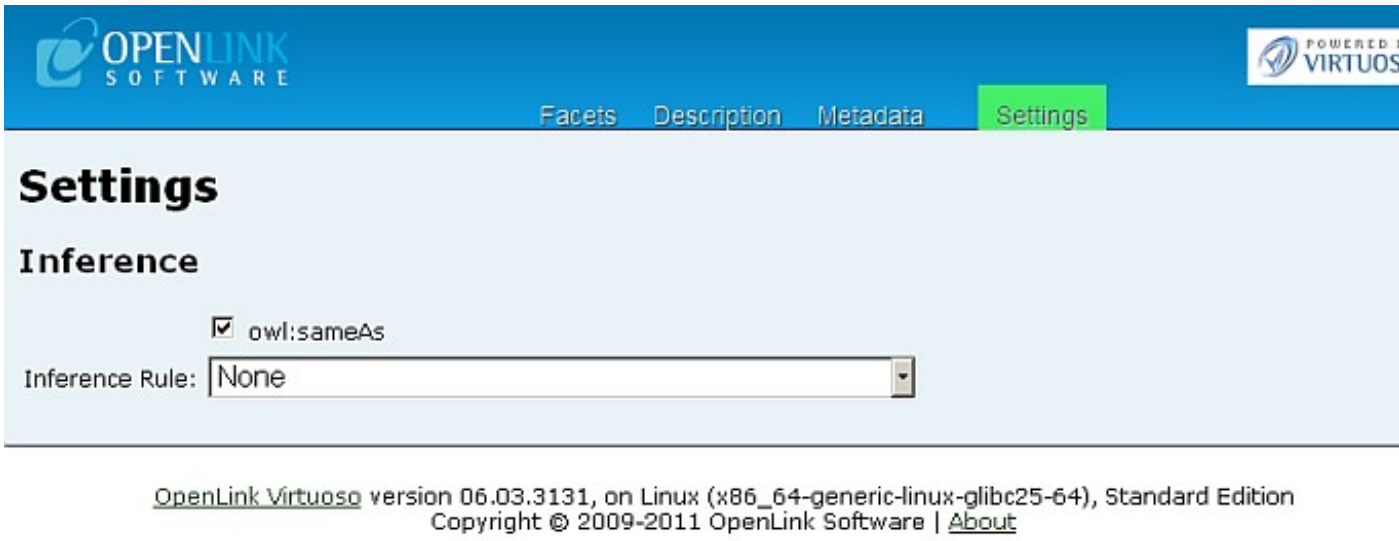


Figure 16.175. Faceted Navigation Example



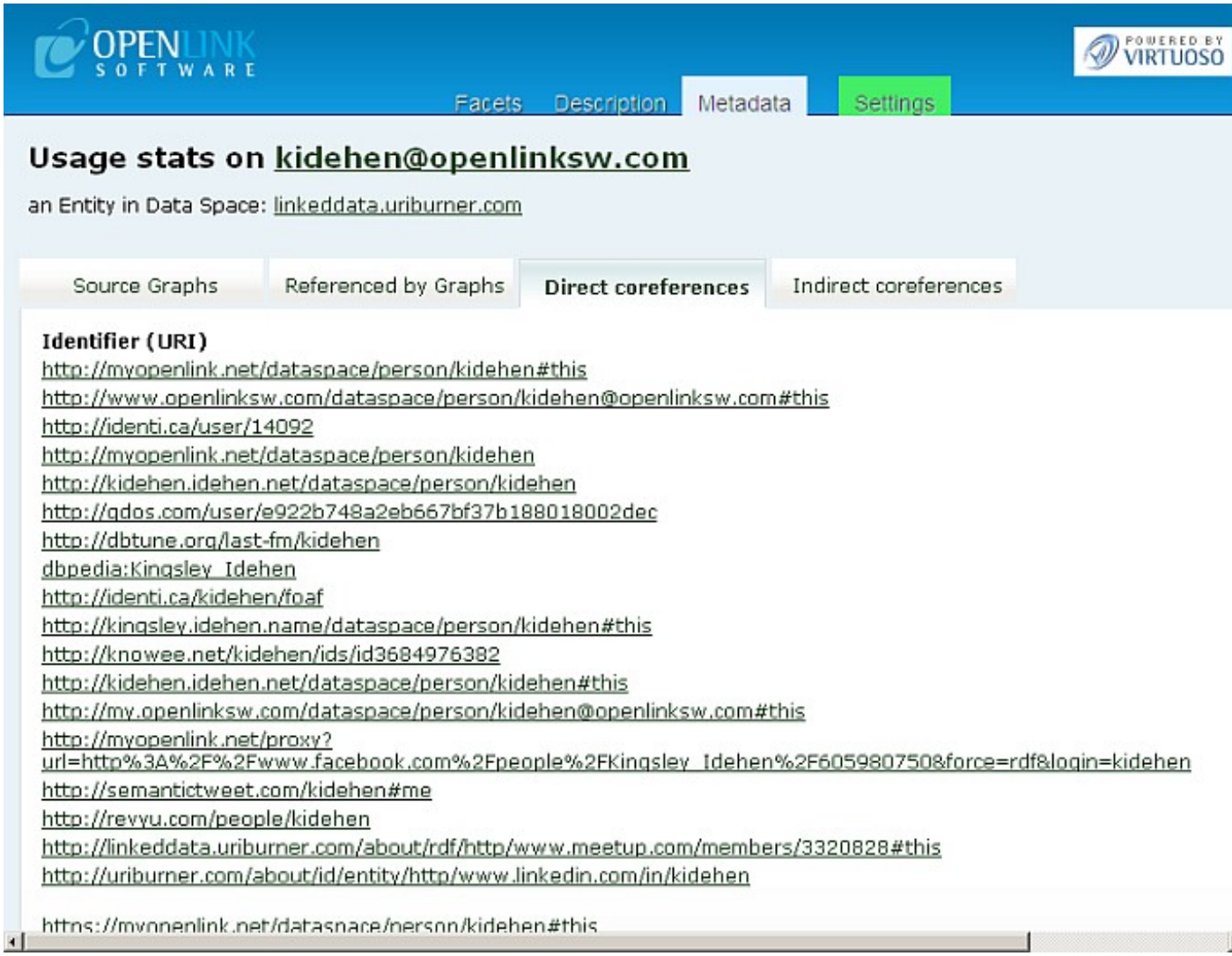
16. Click on "Settings" check "owl:sameAs" and it sets a context mode for the session (meaning: a set of rules to take place):

Figure 16.176. Faceted Navigation Example



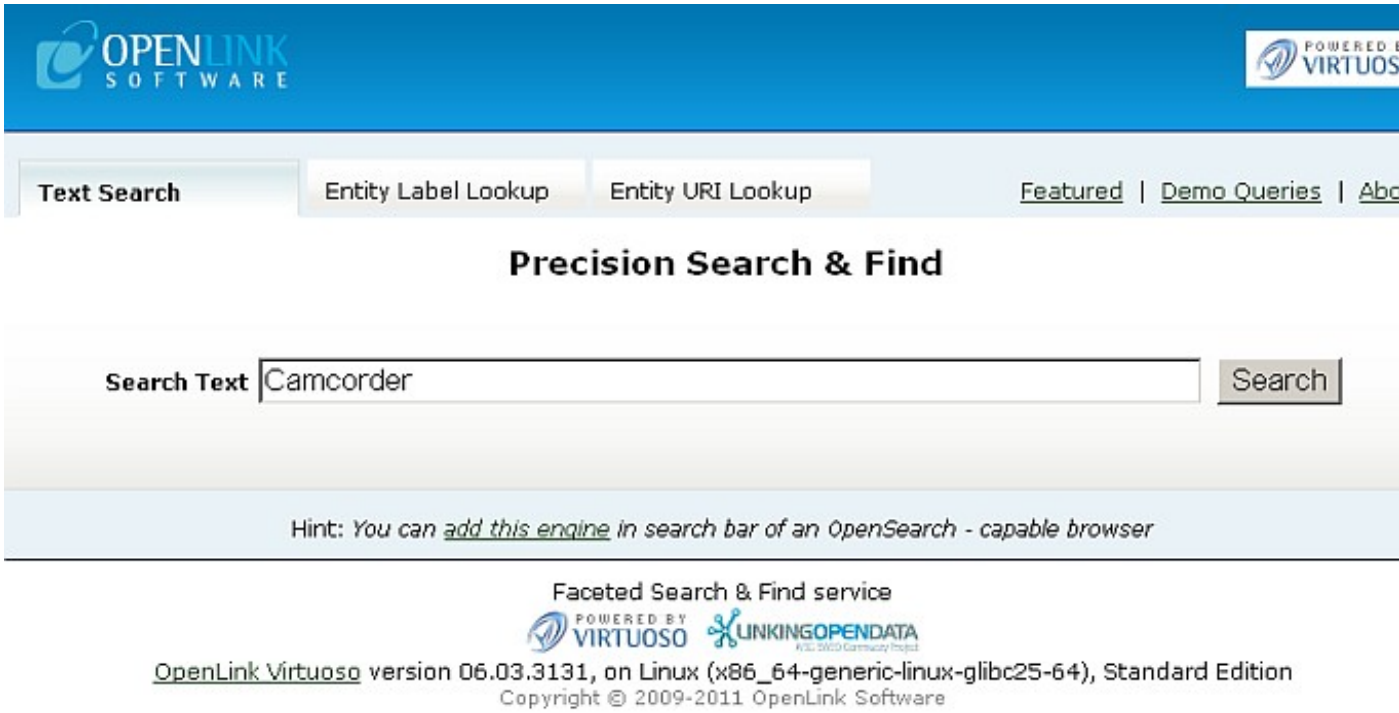
17. Go back to the "Direct Co-reference" tab:

Figure 16.177. Faceted Navigation Example



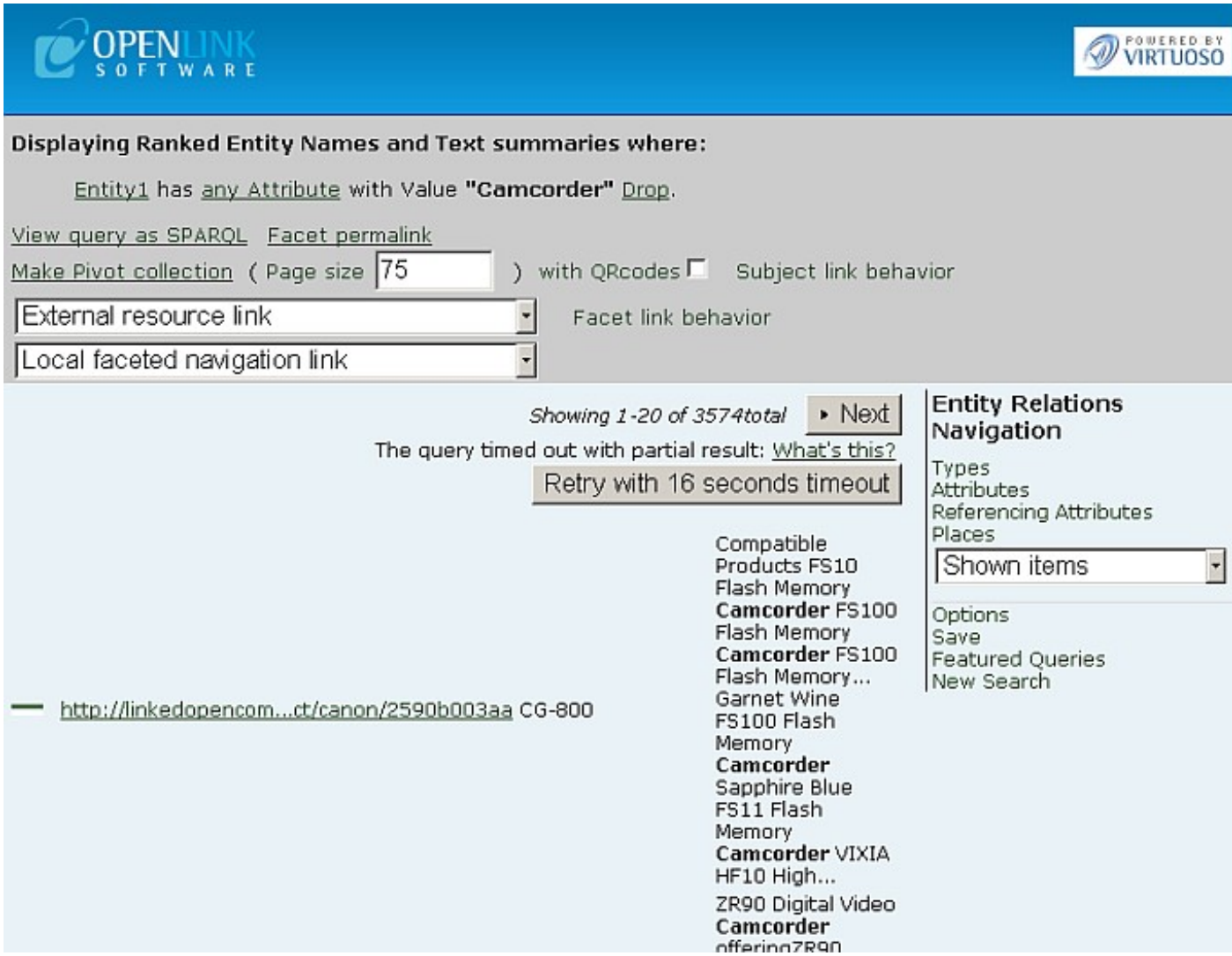
18. As result each link will unveil a union (combination) of all the the data associated with all Kingsley Idehen's other Identifiers (other Names in other places), i.e., they all show the same data.
19. Go to "Facets" and then from "Entity Relations Navigation" click "New Search".
20. Enter a free text search pattern (for example, " `Camcorder` " as consumer product), and click Search:

Figure 16.178. Faceted Navigation Example



21. Your initial query results page will display a list of literal value snippets where for each URI will be displayed a label:

Figure 16.179. Faceted Navigation Example



22. Click for ex. on the URL link ofthe first row result.
23. The product description page should be shown and a list of Attributes and Values will be presented. An URI label of the product also will be shown: "CG-800":

Figure 16.180. Faceted Navigation Example

Figure 16.181. Faceted Navigation Example

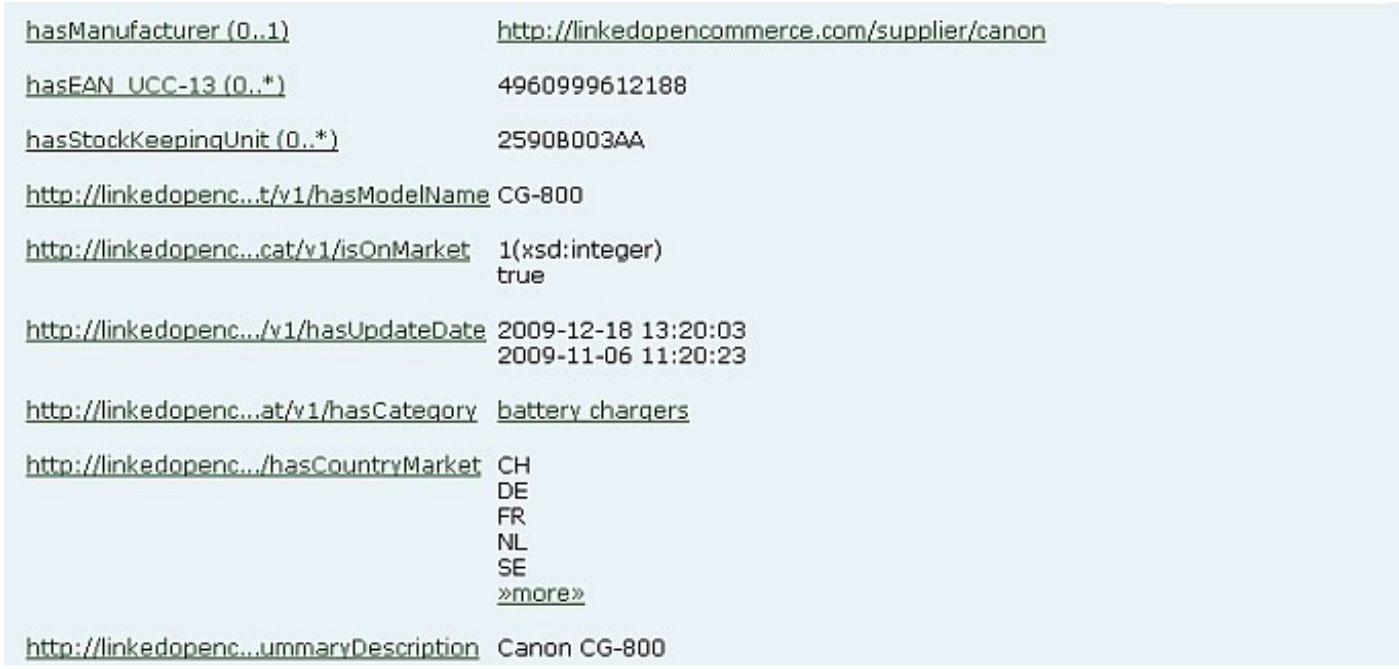


Figure 16.182. Faceted Navigation Example

16.12. Virtuoso Faceted Web Service

The Virtuoso Faceted web service is a general purpose RDF query facility for facet based browsing. It takes an XML description of the view desired and generates the reply as an XML tree containing the requested data. The user agent or a local web page can use XSLT for rendering this for the end user. The selection of facets and values is represented as an XML tree. The rationale for this is the fact that such a representation is easier to process in an application than the SPARQL source text or a parse tree of SPARQL and more compactly captures the specific subset of SPARQL needed for faceted browsing. The web service returns the SPARQL source text also, thus this can serve as a basis for and-crafted queries.

The top element of the tree is <query>, it must be in namespace "http://openlinksw.com/services/facets/1.0/".

This has the following attributes:

- graph="graph_iri" - default is search in all graphs but system defaults may override this
- timeout="no_of_msec" - default is no timeout, but system defaults may override this
- inference="name" where name is a name of an inference context declared with rdfs_rule_set.
- same-as="boolean" - If "boolean" is "yes", then owl:sameAs links will be considered in the query evaluation.

The result is a tree of the form:

```
<facets xmlns="http://openlinksw.com/services/facets/1.0/">
<result><row><column datatype="..." shortform="..." xml:lang="..">...</column></row></result>
<time>msecs</time>
<complete>yes or no</complete>
<db-activity>resource use string</db-activity>
<sparql>sparql statement text</sparql>
</facets>
```

By convention, the first column is the subject selected by the view element, typically a URI, the second a label of the URI and the third, if present, is either a count or a search summary.

The first column's text child is the text form of the value. The column element has the following attributes qualifying this further:

- datatype - The xsd type of the value. If this is a URI, the datatype is "uri"
- shortform - If the value is a URI, this is an abbreviated form where known namespaces are replaced with their prefixes and very long URI's are truncated preserving start and end.
- xml:lang - if the value is a language tagged string, this is the language

The query has the top level element <query>. The child elements of this represent conditions pertaining to a single subject. A join is expressed with the property or property-of element. This has in turn children which state conditions on a property of the first subject. property and property-of elements can be nested to an arbitrary depth and many can occur inside one containing element. In this way, tree-shaped structures of joins can be expressed.

Expressing more complex relationships, such as intermediate grouping, subqueries, arithmetic or such requires writing the query in SPARQL. The XML format is a shorthand for easy automatic composition of queries needed for showing facets, not a replacement for SPARQL.

A facet query contains a single view element. This specifies which subject of the joined subjects is shown. Its attributes specify the manner of viewing, e.g. list of distinct values, distinct values with occurrence counts, properties or classes of the selected subjects etc.

The top query element or any property or property-of element can have the following types of children:

```
<text property="iri">text pattern</text>
```

The subject has an O that matches the text pattern. If property is given, the text pattern must occur in a value of this property. If not specified, any property will do. The value "none" for property is the same as not specifying a property. This is restricted to occurring directly under the top level query element.

```
<class iri="iri" inference="ctx_name" />
```

The S must be an instance of this class. If inference is specified then option (input:inference "ctx_name" is added and applies to this pattern alone.

```
<property iri="iri" same_as="yes" inference="ctx_name">
```

The child elements of this are conditions that apply to the value of this property of the S that is in scope in the enclosing <query> or <property> element. If same_as is present, then option (input:same-as "yes") is added to the triple pattern which specifies this property. If inference is present, then option (input:inference "ctx_name") is added to the triple pattern for the property.

```
<property-of iri="iri" same_as="yes" inference="ctx_name" >
```

The child elements of this are conditions that apply to an S which has property "iri" whose object is the S in scope in the enclosing <query> or <property> element. The options are otherwise the same as with property.

```
<value datatype="type" xml:lang="lng" op="= | < | > | >= | <=">value </value>
```

When this occurs inside <property> or <property-of> this means that the property in scope has the specified relation to the value. type and language can be used for XML typed or language tagged literals. The "uri" type means that the value is a qualified name of a URI. If this occurs directly under the <query> element, this means that the query starts with a fixed subject. If this is so, then there must be property or propertyof elements or the view element must specify properties or classes, list is not allowed as a view type. This is so because the query must have at least one triple pattern.

```
<view type="view" limit="n" offset="n" >
```

This may occur once inside a <query> element but may occur either at top level or inside property or property-of elements. This specifies what which subject is presented in the result set.

The type can be:

- "properties"

```
SPARQL
SELECT ?p count (*) { ?this_s ?p ?any_o ...}
GROUP BY ?p
ORDER BY DESC 2
LIMIT l OFFSET 0
```

- "properties-in"

```
SPARQL
SELECT ?p count (*) { ?any_s ?p ?this_s ... }
GROUP BY ?p
ORDER BY DESC 2
LIMIT L OFFSET 0
```

- "classes"

```
SPARQL
SELECT ?c count (*)
WHERE { ?xx a ?c ... }
GROUP BY ?c
ORDER BY DESC 2
LIMIT l OFFSET 0
```

- "text"

```
SPARQL
SELECT DISTINCT ?s (bif:search_excerpt (sql:search_terms (""pattern"), ?o)) ...
LIMIT 1 OFFSET 0
```

- "list"

```
SPARQL
SELECT DISTINCT ?s long::sql:fct_label (?s) ...
LIMIT 1 OFFSET 0
```

- "list-count"

```
SPARQL
SELECT ?s COUNT (*) ....
GROUP BY ?s
ORDER BY DESC 2
```

- "alphabet"

```
SPARQL
SELECT (sql:subseq (?s, 0, 1)) count (*) ...
GROUP BY (sql:subseq (?s, 0, 1))
ORDER BY 1
```

- "geo"

```
SPARQL
SELECT DISTINCT ?lat ?long ?s
WHERE ?s geo:lat ?lat . ?s geo:long ?long . ... }
```

- "years"

```
SPARQL
SELECT sql::year (?s) count (*) ...
GROUP BY (bif:year (?s))
ORDER BY 1
OFFSET 0 LIMIT 1
```

- "months"

```
SPARQL
SELECT sql::round_month (?s) count (*) ...
GROUP BY (sql:round_month (?s))
ORDER BY 1 OFFSET 0 LIMIT 1
```

- "weeks"

```
SPARQL
SELECT sql::round_week (?s) COUNT (*) ...
GROUP BY (sql:round_week (?s))
ORDER BY 1 OFFSET 0 LIMIT 1
```

- "describe"

```
SPARQL describe ?s ... OFFSET 0 LIMIT 1
```

16.12.1. Customizing

The following types of customization will be generally useful:

- Resource accounting and limitations, managing access and login
- Localization, choice of labels shown with class/property/instance URI's
- Adding types of views, for example timelines, map or business graphics
- Controlling navigation, for example choosing what type of view is initially presented when opening a given property.
- Page layout, captions, help texts, etc.

The source code is divided in two SQL files and a number of XSLT sheets. The file facet.sql has the code for the web service. The facet_view.sql file contains the procedures for the sample HTML interface.

16.12.2. Examples

Note: in all examples the default namespace xmlns="http://openlinksw.com/services/facets/1.0/" is omitted for brevity.

For people called Mike:

```
<query>
  <text>Mike</text>
  <view type="text"/>
</query>
```

To open the list of people who Mike knows:

```
<query>
  <text>Mike</text>
  <view type="properties"/>
</query>
```

To show the list of subjects Mike knows:

```
<query>
  <text>Mike</text>
  <property iri="foaf:knows>
    <view type="list" />
  </property>
</query>
```

To show the properties of people Mike knows:

```
<query>
  <text>Mike</text>
  <property iri="foaf:knows>
    <view type="properties" />
  </property>
</query>
```

To show the names:

```
<query>
  <text>Mike</text>
  <property iri="foaf:knows>
    <property iri="foaf:name>
      <view type="list" />
    </property>
  </property>
</query>
```

To specify one named Joe:

```
<query>
  <text>Mike</text>
  <property iri="foaf:knows>
     <property iri="foaf:name>
        <value>Joe</value>
     </property>
    <view type="properties" />
  </property>
</query>
```

This lists the properties of the friends of Mike that are called Joe.

To show the Mikes that know a Joe, one would change the shown variable in the navigation and get:

```
<query>
  <text>Mike</text>
  <property iri="foaf:knows>
    <property iri="foaf:name>
      <value>Joe</value>
```

```
    </property>
   </property>
   <view type="text" />
</query>
```

This would be the search summaries of subjects with Mike in some field that know a subject with name Joe.

Now to specify that Mike must be a member of a discussion board:

```
<query>
  <text>Mike</text>
  <property iri="foaf:knows>
    <property iri="foaf:name>
     <value>Joe</value>
   </property>
  </property>
  <view type="property-in" />
</query>
```

This lists the properties of triples whom object is Mike. Pick sioc:member_of

```
<query>
  <text>Mike</text>
  <property iri="foaf:knows>
    <property iri="foaf:name>
      <value>Joe</value>
    </property>
  </property>
  <property-of iri="sioc:member_of>
    <view type="list" />
  </property-of>
</query>
```

This would show things where Mike is a member. To specify that the thing must be a forum:

```
<query>
  <text>Mike</text>
  <property iri="foaf:knows>
    <property iri="foaf:name>
      <value>Joe</value>
    </property>
  </property>
  <property-of iri="sioc:member_of>
    <view type="classes" />
  </property-of>
</query>
```

This shows classes of things where Mike is a member Clicking on sioc:Forum gives:

```
<query>
  <text>Mike</text>
  <property iri="foaf:knows>
    <property iri="foaf:name>
      <value>Joe</value>
    </property>
  </property>
  <property-of iri="sioc:member_of>
    <class iri="sioc:Forum" />
    <view type="classes"/>
  </property-of>
</query>
```

The view stays with classes, but now scoped to the classes of things where Mike is a member that are instances of sioc:Forum.

To go look at the list of Mikes with the added restriction, click the shown variable in the navigation and set it to s1.

```
<query>
  <text>Mike</text>
  <property iri="foaf:knows>
    <property iri="foaf:name>
```

```
        <value>Joe</value>
      </property>
    </property>
    <property-of iri="sioc:member_of>
     <class iri="sioc:Forum" />
    </property-of>
    <view type="list"/>
  </query>
```

To say that Joe must also have a geekCode, One clicks the shown variable and sets it to s2 and the view to properties.

```
  <query>
    <text>Mike</text>
    <property iri="foaf:knows>
      <property iri="foaf:name>
        <value>Joe</value>
      </property>
      <view type="properties"/>
    </property>
    <property-of iri="sioc:member_of>
      <class iri="sioc:Forum" />
     </property-of>
  </query>
```

Pick geekCode

```
  <query>
    <text>Mike</text>
    <property iri="foaf:knows>
      <property iri="foaf:name>
        <value>Joe</value>
      </property>
      <property iri="geekCode">
        <view type="list"/>
      </property>
    </property>
    <property-of iri="sioc:member_of>
      <class iri="sioc:Forum" />
    </property-of>
  </query>
```

We specify no restriction on the geekCode. Click the shown variable to take the focus back to Mike.

```
  <query>
    <text>Mike</text>
    <property iri="foaf:knows>
      <property iri="foaf:name>
        <value>Joe</value>
      </property>
      <property iri="geekCode"></property>
    </property>
    <property-of iri="sioc:member_of>
      <class iri="sioc:Forum" />
    </property-of>
    <view type="text"/>
  </query>
```

16.12.3. WebService Interface

REST interface

The Virtuoso Faceted web service provide following REST interface:

Service description:

- Endpoint: http://<cname>/fct/service for ex. http://lod.openlinksw.com/fct/service
- HTTP method: POST
- Content-Type: MUST be 'text/xml'
- The entity body must be XML document with top element 'query' as described above.

- The request response namespace MUST be "http://openlinksw.com/services/facets/1.0"

Error conditions:

The all error conditions are reported via 'Error explanation'

Files:

The facet_svc.sql contains web service code and virtual directory mapping, and it uses fct_req.xsl & fct_resp.xsl as request & response filters.

Example:

Using CURL program

```
curl -H "Content-Type: text/xml" -d @post.xml  http://lod.openlinksw.com/fct/service
```

Where 'post.xml' document contains query document:

```
<?xml version="1.0"?>
<query xmlns="http://openlinksw.com/services/facets/1.0" inference="" same-as="">
  <text> Seattle Mariners traveled all the way to Japan to watch</text>
  <view type="text" limit="20" offset=""/>
</query>
```

Produces following response:

```
<fct:facets xmlns:fct="http://openlinksw.com/services/facets/1.0/">
<fct:sparql>   SELECT distinct ?s1 as ?c1, (bif:search_excerpt (bif:vector ('THE', 'MARINERS', 'WAY', 'SE
<fct:time>116</fct:time>
<fct:complete>yes</fct:complete>
<fct:db-activity>   134R rnd  9.488KR seq      0P disk  8.966MB /    602 messages</fct:db-activity>
 <fct:result>
  <fct:row>
    <fct:column datatype="url" shortform="http://bobdupuy.mlbl...ld_baseball__6.html">http://bobdupuy.mlb
    <fct:column />
    <fct:column><span class="srch_xerpt">... While Chuck Armstrong president of <b>the</b> <b>Seattle</b>
  </fct:row>
  <fct:row>
    <fct:column datatype="url" shortform="http://bobdupuy.mlbl...ld_baseball__6.html">http://bobdupuy.mlb
    <fct:column />
    <fct:column><span class="srch_xerpt">Orlando While Chuck Armstrong president of <b>the</b> <b>Seattle
  </fct:row>
 </fct:result>
</fct:facets>
```

Virtuoso APIs for Faceted REST services

The Virtuoso APIs for FCT REST services are Virtuoso Stored Procedures that enable faceted browsing over Linked Data hosted in the RDF Quad Store. This also includes Linked Data that is progressively added to the Quad Store via URI de-referencing.

They enable the use Virtuoso's VSP/VSPX technology to produce (X)HTML-based Linked Data explorer pages that are endowed with high-performance (in-process) faceted browsing capability.

You can use this API with Virtuoso SQL calls that provide data to your VSP/VSPX, ASP.NET, PHP, etc., -based interfaces using ODBC, JDBC, ADO.NET, or XMLA connectivity (SPASQL) to Virtuoso.

API Definition

```
CREATE PROCEDURE
fct_exec
  (
    IN  tree     ANY ,
    IN  timeout  INT
  )
{
  DECLARE  start_time,
```

```
          view3,
          inx,
          n_rows      INT     ;
  DECLARE  sqls,
          msg,
          qr,
          qr2,
          act,
          query       VARCHAR ;
  DECLARE  md,
          res,
          results,
          more        ANY     ;
  DECLARE  tmp         ANY     ;
  DECLARE  offs,
          lim         INT     ;

  SET result_timeout = _min
                       (
                         timeout,
                         ATOI
                           (
                             registry_get ('fct_timeout_max')
                           )
                       )
  ;

  offs := xpath_eval ('//query/view/@offset', tree);
  lim := xpath_eval ('//query/view/@limit', tree);

  -- db_activity ();

  results := vector (null, null, null);
  more := vector ();

  IF
    (
      xpath_eval
        (
          '//query[@view3="yes"]//view[@type="text"]',
          tree
        )
      IS NOT NULL
    )
    {
      more := VECTOR ('classes', 'properties');
    }

  sqls := '00000';
  qr := fct_query
          (
            xpath_eval ('//query', tree, 1)
          )
  ;
  query := qr;
--  dbg_obj_print (qr);
  qr2 := fct_xml_wrap (tree, qr);
  start_time := msec_time ();

  dbg_printf('query: %s', qr2);

  EXEC
    (
      qr2,
      sqls,
      msg,
      vector (),
      0,
      md,
      res
    )
  ;
  n_rows := row_count ();
```

```
act := db_activity ();
SET result_timeout = 0;
IF (
     sqls <> '00000'
     AND
     sqls <> 'S1TAT'
   )
  SIGNAL (sqls, msg);
IF (
     NOT ISARRAY (res)
     OR
     0 = length (res)
     OR
     NOT ISARRAY (res[0])
     OR
     0 = length (res[0])
   )
  results[0] := xtree_doc ('<result/>');
ELSE
  results[0] := res[0][0];

inx := 1;

FOREACH (VARCHAR tp IN more) DO
  {
    tree := XMLUpdate (
                        tree,
                        '/query/view/@type',
                        tp,
                        '/query/view/@limit',
                        '40',
                        '/query/view/@offset',
                        '0'
                      )
    ;
    qr := fct_query (xpath_eval ('//query', tree, 1));
    qr2 := fct_xml_wrap (tree, qr);
    sqls := '00000';
    SET result_timeout = _min (
                                timeout,
                                ATOI
                                  (
                                    registry_get ('fct_timeout_max')
                                  )
                               )
    ;
    EXEC (
           qr2,
           sqls,
           msg,
           vector (),
           0,
           md,
           res
         );
    n_rows := row_count ();
    act := db_activity ();
    SET result_timeout = 0;
    IF ( sqls <> '00000'
         AND
         sqls <> 'S1TAT'
       )
  SIGNAL (sqls, msg);
    IF (
         ISARRAY (res)
         AND
         LENGTH (res)
         AND
         ISARRAY (res[0])
         AND
         LENGTH (res[0])
       )
  {
```

```
      tmp := res[0][0];
      tmp := XMLUpdate (tmp, '/result/@type', tp);
      results[inx] := tmp;
    }
      inx := inx + 1;
    }

  res := XMLELEMENT
         (
           "facets",
           XMLELEMENT
             ( "sparql", query ),
           XMLELEMENT
             ( "time", msec_time () - start_time ),
           XMLELEMENT
             (
               "complete",
               CASE WHEN sqls = 'S1TAT'
                    THEN 'no'
                    ELSE 'yes'
                END
              ),
           XMLELEMENT
             (
               "timeout",
               _min
                 (
                   timeout * 2,
                   ATOI
                     (
                       registry_get
                         ( 'fct_timeout_max' )
                     )
                 )
             ),
           XMLELEMENT
             ("db-activity", act),
           XMLELEMENT
             ("processed", n_rows),
           XMLELEMENT
             (
               "view",
               XMLATTRIBUTES
                 (
                   offs AS "offset",
                   lim AS "limit"
                 )
             ),
           results[0],
           results[1],
           results[2]
         );

---- for debugging:
--string_to_file ('ret.xml', serialize_to_UTF8_xml (res), -2);
--  dbg_obj_print (res);

  RETURN res;
}
;
```

Example

The following example shows how to use the fct_exec APi in vsp page to perform a "text" search for the word "Mike" assuming this exists in your Virtuoso RDF store (if not amend the query in the fct_example.vsp code sample below to search for text known to exist).

1. The service can be used in the following sample fct_example.vsp:

```
<?vsp

declare txt, reply, tree any;
```

```
declare timeout int;

tree := xtree_doc ('
  <query>
    <text>Mike</text>
    <view type="text"/>
  </query>
');

timeout := 3000;
reply := fct_exec (tree, timeout);

txt := string_output ();

http_value (xslt ('virt://WS.WS.SYS_DAV_RES.RES_FULL_PATH.RES_CONTENT:/DAV/fct_example.xsl',
                  reply,
                  vector ()),
                  null, txt);

http (txt);
?>
```

2. The xsl:

```
<?xml version="1.0" encoding="utf-8"?>

<xsl:stylesheet
    version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" encoding="ISO-8859-1"/>
  <xsl:variable name="rowcnt" select="count(/facets/result/row)"/>

  <xsl:template match="facets">
    <div
xml:id="res">
      <xsl:if test="/facets/complete = 'yes' and /facets/processed = 0 and $rowcnt = 0">
        <div class="empty_result">
          Nothing found.
        </div>
      </xsl:if>
      <xsl:for-each select="/facets/result">
        <xsl:call-template name="render-result"/>
      </xsl:for-each>
    </div>
    <!-- #res -->

  </xsl:template>

  <xsl:template name="render-result">
    <table class="result" border="1">
      <thead>
        <tr>
          <th>Entity</th>
          <th>Title</th>
          <th>Text excerpt</th>
        </tr>
      </thead>
      <tbody>
        <xsl:for-each select="row">
          <tr>
            <td class="rnk">
              <xsl:for-each select="column[@datatype='trank' or @datatype='erank']">
                <xsl:choose>
                  <xsl:when test="./@datatype='trank'">Text Rank:</xsl:when>
                  <xsl:when test="./@datatype='erank'">Entity Rank:</xsl:when>
                </xsl:choose>
                <xsl:value-of select="."/>
                <br/>
              </xsl:for-each>
            </td>
            <xsl:for-each select="column">
              <xsl:choose>
                <xsl:when test="'url' = ./@datatype">
                  <td>
```

```
                        <a>
                          <xsl:attribute name="href">http://lod.openlinksw.com/describe/?url=<xsl:valu
                          <xsl:attribute name="title"><xsl:value-of select="."/></xsl:attribute>
                          <xsl:choose>
                            <xsl:when test="'' != ./@shortform">
                              <xsl:value-of select="./@shortform"/>
                            </xsl:when>
                            <xsl:when test="'erank' = ./@datatype or 'trank' = ./@datatype">rank</xsl:
                            <xsl:otherwise>
                              <xsl:value-of select="."/>
                            </xsl:otherwise>
                          </xsl:choose>
                        </a>
                      </td>
                    </xsl:when>
                    <xsl:when test="'erank' = ./@datatype or 'trank' = ./@datatype"/>
                    <xsl:when test="'srch_xerpt' = ./span/@class">
                      <td>
                        <xsl:value-of select="."/>
                      </td>
                    </xsl:when>
                    <xsl:otherwise/>
                  </xsl:choose>
                </xsl:for-each>
              </tr>
            </xsl:for-each>
          </tbody>
        </table>
      </xsl:template>

      <xsl:template match="@* | node()">
        <xsl:copy>
          <xsl:apply-templates select="@* | node()"/>
        </xsl:copy>
      </xsl:template>
    </xsl:stylesheet>
```

3. The result of executing the fct_example.vsp should be:

Figure 16.183. Faceted API Example

Entity	Title	Text excerpt
Text Rank:4.5 Entity Rank:0.000000000000059	dbpedia:Mike_Keenan	Mike Keenan.
Text Rank:4.5 Entity Rank:0.000000000000059	dbpedia:Mike_McHugh	Mike McHugh.
Text Rank:4.5 Entity Rank:0.000000000000059	dbpedia:Mike_McKone	Mike McKone.
Text Rank:4.5 Entity Rank:0.000000000000059	dbpedia:Mike_Montgomery	Mike Montgomery.
Text Rank:4.5 Entity Rank:0.000000000000059	dbpedia:Mike_Salmon	Mike Salmon.
Text Rank:4.5 Entity Rank:0.000000000000059	dbpedia:Mike_Smith_%...e_hockey_b._1982%29	Mike Smith Eishockeyspieler.
Text Rank:4.5 Entity Rank:0.000000000000059	dbpedia:Mike_Vrabel	Mike Vrabel.

SOAP interface

The facet web service is also available via SOAP protocol.

The request message contains single element 'query' with syntax explained earlier. Also the SOAPAction HTTP header should be '#query' . After successful evaluation of the query, the service will return a SOAP envelope containing in the Body element single 'facets' element described above.

Example:

This example shows execution of same command as in example for REST interface here it using SOAP:

Request message:

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP:Body>
    <query xmlns="http://openlinksw.com/services/facets/1.0/" inference="" same-as="">
      <text>Seattle Mariners traveled all the way to Japan to watch</text>
      <view type="text" limit="20" offset="0"/>
    </query>
  </SOAP:Body>
</SOAP:Envelope>
```

Response message:

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP:Body>
    <fct:facets xmlns:fct="http://openlinksw.com/services/facets/1.0/">
      <fct:sparql>SELECT distinct ?s1 as ?c1, (bif:search_excerpt (bif:vector ('THE', 'MARINERS', 'WAY',
      <fct:time>114</fct:time>
      <fct:complete>yes</fct:complete>
      <fct:db-activity>   134R rnd  9.488KR seq      0P disk  8.966MB /    602 messages</fct:db-activity>
      <fct:result>
        <fct:row>
          <fct:column datatype="url" shortform="http://bobdupuy.mlbl...ld_baseball__6.html">http://bobdup
```

```
            <fct:column/>
            <fct:column><span class="srch_xerpt">... While Chuck Armstrong president of <b>the</b> <b>Seatt
          </fct:row>
          <fct:row>
            <fct:column datatype="url" shortform="http://bobdupuy.mlbl...ld_baseball__6.html">http://bobdup
            <fct:column/>
            <fct:column><span class="srch_xerpt">Orlando While Chuck Armstrong president of <b>the</b> <b>S
          </fct:row>
        </fct:result>
      </fct:facets>
    </SOAP:Body>
  </SOAP:Envelope>
```

16.13. Linked Data

There are many cases when RDF data should be retrieved from remote sources only when really needed. E.g., a scheduling application may read personal calendars from personal sites of its users. Calendar data expire quickly, so there's no reason to frequently re-load them in hope that they are queried before expired.

Virtuoso extends SPARQL so it is possible to download RDF resource from a given IRI, parse them and store the resulting triples in a graph, all three operations will be performed during the SPARQL query execution. The IRI of graph to store triples is usually equal to the IRI where the resource is download from, so the feature is named "IRI dereferencing" There are two different use cases for this feature. In simple case, a SPARQL query contains *from* clauses that enumerate graphs to process, but there are no triples in *DB.DBA.RDF_QUAD* that correspond to some of these graphs. The query execution starts with dereferencing of these graphs and the rest runs as usual. In more sophisticated case, the query is executed many times in a loop. Every execution produces a partial result. SPARQL processor checks for IRIs in the result such that resources with that IRIs may contain relevant data but not yet loaded into the *DB.DBA.RDF_QUAD* . After some iteration, the partial result is identical to the result of the previous iteration, because there's no more data to retrieve. As the last step, SPARQL processor builds the final result set.

16.13.1. IRI Dereferencing For FROM Clauses, "define get:..." Pragmas

Virtuoso extends SPARQL syntax of *from* and *from named* clauses. It allows additional list of options at end of clause: *option (param1 value1, param2 value2, ...)* where parameter names are QNames that start with *get:* prefix and values are "precode" expressions, i.e. expressions that does not contain variables other than external parameters. Names of allowed parameters are listed below.

- *get:soft*

  is the retrieval mode, supported values are "soft" and "replacing". If the value is "soft" then the SPARQL processor will not even try to retrieve triples if the destination graph is non-empty. Other

  get:...

  parameters are useless without this one.
- *get:uri*

  is the IRI to retrieve if it is not equal to the IRI of the

  from

  clause. These can be used if data should be retrieved from a mirror, not from original resource location or in any other case when the destination graph IRI differs from the location of the resource.
- ```
  SQL>SPARQL
  define get:uri "http://example.com/dataspace/person/kidehen"
  SELECT ?id
  FROM NAMED <http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.c
  WHERE { graph ?g { ?id a ?o } }
  LIMIT 10;

  id
  VARCHAR
  _______________________________________________________________________________

  http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLO
  http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLO
  ```
  ```

```
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLC
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLC
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLC
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLC
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLC
http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#this
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLC
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLC

10 Rows. -- 10 msec.
```

- *get:method*

  is the HTTP method that should be used to retrieve the resource, supported methods are "GET" for plain HTTP and "MGET" for URIQA web service endpoint. By default, "MGET" is used for IRIs that end with "/" and "GET" for everything else.
- *get:refresh*

  is the maximum allowed age of the cached resource, no matter what is specified by the server where the resource resides. The value is an positive integer (number of seconds). Virtuoso reads HTTP headers and uses "Date", "ETag", "Expires", "Last-Modified", "Cache-Control" and "Pragma: no-cache" fields to calculate when the resource should be reloaded, this value can become smaller due to

  get:refresh

  but can not be incremented.
-
  ```
  SQL>SPARQL
  define get:refresh "3600"
  SELECT ?id
  FROM NAMED <http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.c
  WHERE { graph ?g { ?id a ?o } }
  LIMIT 10;

  id
  VARCHAR
  _______________________________________________________________________________

  http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLC
  http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLC
  http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLC
  http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLC
  http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLC
  http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLC
  http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLC
  http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#this
  http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLC
  http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLC

  10 Rows. -- 10 msec.
  ```
- *get:proxy*

  address of the proxy server, as "host:port" string, if direct download is impossible; the default is to not use proxy.
-
  ```
  SQL>SPARQL
  define get:proxy "www.openlinksw.com:80"
  define get:method "GET"
  define get:soft "soft"
  SELECT ?id
  FROM NAMED <http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.c
  WHERE { graph ?g { ?id a ?o } }
  LIMIT 10;

  id
  VARCHAR
  _______________________________________________________________________________

  http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLC
  http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLC
  http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLC
  http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLC
  ```

```
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLO
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLO
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLO
http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#this
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLO
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLO

10 Rows. -- 10 msec.
SQL> limit 10;
```

- If a value of some *get:*... parameter repeats for every *from* clause then it can be written as a global pragma like *define get:soft "soft"* . The following two queries will work identically:

```
SQL>SPARQL
SELECT ?id
FROM NAMED <http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.c
  OPTION (get:soft "soft", get:method "GET")
FROM NAMED <http://www.openlinksw.com/dataspace/oerling/weblog/Orri%20Erling%27s%20Blog/sioc.ttl>
  OPTION (get:soft "soft", get:method "GET")
WHERE { graph ?g { ?id a ?o } }
LIMIT 10;

id
VARCHAR
_______________________________________________________________________________

http://www.openlinksw.com/dataspace/person/oerling#this
http://www.openlinksw.com/dataspace/oerling/weblog/Orri%20Erling%27s%20Blog
http://www.openlinksw.com/dataspace/oerling/weblog/Orri%20Erling%27s%20Blog
http://www.openlinksw.com/dataspace/oerling/weblog/Orri%20Erling%27s%20Blog/958
http://www.openlinksw.com/dataspace/oerling/weblog/Orri%20Erling%27s%20Blog/958
http://www.openlinksw.com/dataspace/oerling/weblog/Orri%20Erling%27s%20Blog/949
http://www.openlinksw.com/dataspace/oerling/weblog/Orri%20Erling%27s%20Blog/949

10 Rows. -- 862 msec.
```

```
SQL>SPARQL
define get:method "GET"
define get:soft "soft"
SELECT ?id
FROM NAMED <http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.c
FROM NAMED <http://www.openlinksw.com/dataspace/oerling/weblog/Orri%20Erling%27s%20Blog/sioc.ttl>
WHERE { graph ?g { ?id a ?o } }
LIMIT 10;

id
VARCHAR
_______________________________________________________________________________

http://www.openlinksw.com/dataspace/person/oerling#this
http://www.openlinksw.com/dataspace/oerling/weblog/Orri%20Erling%27s%20Blog
http://www.openlinksw.com/dataspace/oerling/weblog/Orri%20Erling%27s%20Blog
http://www.openlinksw.com/dataspace/oerling/weblog/Orri%20Erling%27s%20Blog/958
http://www.openlinksw.com/dataspace/oerling/weblog/Orri%20Erling%27s%20Blog/958
http://www.openlinksw.com/dataspace/oerling/weblog/Orri%20Erling%27s%20Blog/949
http://www.openlinksw.com/dataspace/oerling/weblog/Orri%20Erling%27s%20Blog/949

10 Rows. -- 10 msec.
```

- It can make text shorter and it is especially useful when the query text comes from client but the parameter should have a fixed value due to security reasons: the values set by *define get:*... can not be redefined inside the query and the application may prevent the text with desired pragmas before the execution.

  Note that the user should have *SPARQL_UPDATE* role in order to execute such a query. By default SPARQL web service endpoint is owned by *SPARQL* user that have *SPARQL_SELECT* but not *SPARQL_UPDATE* . It is possible in principle to grant *SPARQL_UPDATE* to *SPARQL* but this breaches the whole security of the RDF storage.
- *FROM CLAUSE with options*

  : options in OPTION() list should be delimited with commas. grab options are not allowed as they are global for the query. Only specific 'get:xxx' options are useful here.

```
SQL>SPARQL
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?friend
FROM NAMED  <http://example.com/dataspace/person/kidehen>
```

```
OPTION (get:soft "soft", get:method "GET")
WHERE
  {
      <http://example.com/dataspace/person/kidehen#this> foaf:knows
?friend .
  };
friend
VARCHAR
_______________________________________________________________________________

http://www.dajobe.org/foaf.rdf#i
http://www.w3.org/People/Berners-Lee/card#i
http://www.w3.org/People/Connolly/#me
http://my.opera.com/chaals/xml/foaf#me
http://www.w3.org/People/Berners-Lee/card#amy
http://www.w3.org/People/EM/contact#me
http://example.com/dataspace/person/ghard#this
http://example.com/dataspace/person/omfaluyi#this
http://example.com/dataspace/person/alanr#this
http://example.com/dataspace/person/bblfish#this
http://example.com/dataspace/person/danja#this
http://example.com/dataspace/person/tthibodeau#this
...
36 Rows. -- 1693 msec.
```

16.13.2. IRI Dereferencing For Variables, "define input:grab-..." Pragmas

Consider a set of personal data such that one resource can list many persons and point to resources where that persons are described in more details. E.g. resource about *user1* describes the user and also contain statements that *user2* and *user3* are persons and more data can be found in *user2.ttl* and *user3.ttl* , *user3.ttl* can contain statements that *user4* is also person and more data can be found in *user4.ttl* and so on. The query should find as many users as it is possible and return their names and e-mails.

If all data about all users were loaded into the database, the query could be quite simple:

```
SQL>SPARQL
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?id ?firstname ?nick
where
  {
    graph ?g
      {
        ?id rdf:type foaf:Person.
        ?id foaf:firstName ?firstname.
        ?id foaf:knows ?fn .
        ?fn foaf:nick ?nick.
      }
   }
limit 10;

id                                                    firstname  nick
VARCHAR                                               VARCHAR    VARCHAR
_______________________________________________________________________________

http://example.com/dataspace/person/pmitchell#this   LaRenda    sdmonroe
http://example.com/dataspace/person/pmitchell#this   LaRenda    kidehen{at}openlinksw.com
http://example.com/dataspace/person/pmitchell#this   LaRenda    alexmidd
http://example.com/dataspace/person/abm#this         Alan       kidehen{at}openlinksw.com
http://example.com/dataspace/person/igods#this       Cameron    kidehen{at}openlinksw.com
http://example.com/dataspace/person/goern#this       Christoph  captsolo
http://example.com/dataspace/person/dangrig#this     Dan        rickbruner
http://example.com/dataspace/person/dangrig#this     Dan        sdmonroe
http://example.com/dataspace/person/dangrig#this     Dan        lszczepa
http://example.com/dataspace/person/dangrig#this     Dan        kidehen

10 Rows. -- 80 msec.
```

It is possible to enable IRI dereferencing in such a way that all appropriate resources are loaded during the query execution even if names of some of them are not known a priori.

```
SQL>SPARQL
  define input:grab-var "?more"
  define input:grab-depth 10
  define input:grab-limit 100
  define input:grab-base "http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openl
  prefix foaf: <http://xmlns.com/foaf/0.1/>
  prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
  prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?id ?firstname ?nick
WHERE {
    graph ?g {
                ?id rdf:type foaf:Person.
                ?id foaf:firstName ?firstname.
                ?id foaf:knows ?fn .
                ?fn foaf:nick ?nick.
                OPTIONAL { ?id rdfs:SeeAlso ?more }
            }
}
LIMIT 10;

id                                                          firstname  nick
VARCHAR                                                     VARCHAR    VARCHAR
_______________________________________________________________________________

http://example.com/dataspace/person/ghard#this          Yrj+?n+?   kidehen
http://inamidst.com/sbp/foaf#Sean                          Sean       d8uv
http://example.com/dataspace/person/dangrig#this        Dan        rickbruner
http://example.com/dataspace/person/dangrig#this        Dan        sdmonroe
http://example.com/dataspace/person/dangrig#this        Dan        lszczepa
http://example.com/dataspace/person/dangrig#this        Dan        kidehen
http://captsolo.net/semweb/foaf-captsolo.rdf#Uldis_Bojars  Uldis      mortenf
http://captsolo.net/semweb/foaf-captsolo.rdf#Uldis_Bojars  Uldis      danja
http://captsolo.net/semweb/foaf-captsolo.rdf#Uldis_Bojars  Uldis      zool
http://example.com/dataspace/person/rickbruner#this     Rick       dangrig

10 Rows. -- 530 msec.
```

The IRI dereferencing is controlled by the following pragmas:

- *input:grab-var*

  specifies a name of variable whose values should be used as IRIs of resources that should be downloaded. It is not an error if the variable is sometimes unbound or gets values that can not be converted to IRIs (e.g., integers) -- bad values are silently ignored. It is also not an error if the IRI can not be retrieved, this makes IRI retrieval somewhat similar to "best effort union" in SQL. This pragma can be used more than once to specify many variable names. It is not an error if values of different variables result in same IRI or a variable gets same value many times -- no one IRI is retrieved more than once.
- *input:grab-iri*

  specifies an IRI that should be retrieved before executing the rest of the query, if it is not in the

  DB.DBA.RDF_QUAD

  already. This pragma can be used more than once to specify many IRIs. The typical use of this pragma is querying a set of related resources when only one "root" resource IRI is known but even that resource is not loaded.
- ```
  SQL>SPARQL
    define input:storage ""
    define input:grab-iri <http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen
    define input:grab-var "id"
    define input:grab-depth 10
    define input:grab-limit 100
    define input:grab-base "http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehe
  SELECT ?id
  WHERE { graph ?g { ?id a ?o } }
  LIMIT 10;

  id
  VARCHAR
  _______________________________________________________________________________
  ```
  ```

```
http://example.com/virtrdf-data-formats#default-iid
http://example.com/virtrdf-data-formats#default-iid-nullable
http://example.com/virtrdf-data-formats#default-iid-nonblank
http://example.com/virtrdf-data-formats#default-iid-nonblank-nullable
http://example.com/virtrdf-data-formats#default
http://example.com/virtrdf-data-formats#default-nullable
http://example.com/virtrdf-data-formats#sql-varchar
http://example.com/virtrdf-data-formats#sql-varchar-nullable
http://example.com/virtrdf-data-formats#sql-longvarchar
http://example.com/virtrdf-data-formats#sql-longvarchar-nullable

10 Rows. -- 530 msec.
```

- *input:grab-all*

  is the simplest possible way to enable the feature but the resulting performance can be very bad. It turns all variables and IRI constants in all graph, subject and object fields of all triple patterns of the query into values for

  input:grab-var

  and

  input:grab-iri

  , so the SPARQL processor will dereference everything what might be related to the text of the query.

```
SQL>SPARQL
  define input:grab-all "yes"
  define input:grab-depth 10
  define input:grab-limit 100
  define input:grab-base "http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehe
  prefix foaf: <http://xmlns.com/foaf/0.1/>
  prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?id ?firstname ?nick
where
  {
    graph ?g
     {
       ?id rdf:type foaf:Person.
       ?id foaf:firstName ?firstname.
       ?id foaf:knows ?fn .
       ?fn foaf:nick ?nick.
     }
  }
limit 10;

id                                                          firstname   nick
VARCHAR                                                     VARCHAR     VARCHAR
_______________________________________________________________________________

http://example.com/dataspace/person/pmitchell#this   LaRenda     sdmonroe
http://example.com/dataspace/person/pmitchell#this   LaRenda     kidehen{at}openlinksw.com
http://example.com/dataspace/person/pmitchell#this   LaRenda     alexmidd
http://example.com/dataspace/person/abm#this         Alan        kidehen{at}openlinksw.com
http://example.com/dataspace/person/igods#this       Cameron     kidehen{at}openlinksw.com
http://example.com/dataspace/person/goern#this       Christoph   captsolo
http://example.com/dataspace/person/dangrig#this     Dan         rickbruner
http://example.com/dataspace/person/dangrig#this     Dan         sdmonroe
http://example.com/dataspace/person/dangrig#this     Dan         lszczepa
http://example.com/dataspace/person/dangrig#this     Dan         kidehen

10 Rows. -- 660 msec.
```

- *input:grab-seealso*

  (and synonym

  input:grab-follow-predicate

) specifies an IRI of an predicate similar to foaf:seeAlso. Predicates of that sort suggest location of resources that contain more data about predicate subject. The IRI dereferencing routine may use these predicates to find additional IRIs for loading resources. This is especially useful when the text of the query comes from remote client and may lack triple patterns like

optional { ?id <SeeAlso> ?more }

from the previous example. The use of

input:grab-seealso

makes the SPARQL query nondeterministic, because the order and the number of retrieved documents will depend on execution plan and they may change from run to run. This pragma can be used more than once to specify many IRIs, but this feature is costly. Every additional predicate may result in significant number of lookups in the RDF storage, affecting total execution time.

- ```
  SQL>SPARQL
    define input:grab-iri <http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen
    define input:grab-var "id"
    define input:grab-depth 10
    define input:grab-limit 100
    define input:grab-base "http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehe
    define input:grab-seealso <foaf:maker>
      prefix foaf: <http://xmlns.com/foaf/0.1/>
  SELECT ?id
  where
    {
      graph ?g
        {
          ?id a foaf:Person .
        }
    }
  limit 10;

  id
  VARCHAR
  _______________________________________________________________________________

  mailto:somebody@example.domain
  http://example.com/dataspace/person/dav#this
  http://example.com/dataspace/person/dba#this
  mailto:2@F.D
  http://example.com/dataspace/person/test1#this
  http://www.openlinksw.com/blog/~kidehen/gems/rss.xml#Kingsley%20Uyi%20Idehen
  http://digitalmusic.weblogsinc.com/rss.xml#
  http://partners.userland.com/nytrss/books.xml#
  http://partners.userland.com/nytrss/arts.xml#

  9 Rows. -- 105 msec.
  ```

- *input:grab-limit*

  should be an integer that is a maximum allowed number of resource retrievals. The default value is pretty big (few millions of documents) so it is strongly recommended to set smaller value. Set it even if you're absolutely sure that the set of resources is small, because program errors are always possible. All resource downloads are counted, both successful and failed, both forced by

  input:grab-iri

  and forced by

  input:grab-var

  . Nevertheless, all constant IRIs specified by

  input:grab-iri

  (or
  ```

input:grab-all

) are downloaded before the first check of the

input:grab-limit

counter, so this limit will never prevent from downloading "root" resources.

- *input:grab-depth*

  should be an integer that is a maximum allowed number of query iterations. Every iteration may find new IRIs to retrieve, because resources loaded on previous iteration may add these IRIs to

  DB.DBA.RDF_QUAD

  and make result set longer. The default value is 1, so the SPARQL processor will retrieve only resources explicitly named in "root" resources or in quad that are in the database before the query execution.
- *input:grab-base*

  specifies a base IRI used to convert relative IRIs into absolute. The default is an empty string.
- ```
  SQL>SPARQL
    define input:grab-depth 10
    define input:grab-limit 100
    define input:grab-var "more"
    define input:grab-base "http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehe
    prefix foaf: <http://xmlns.com/foaf/0.1/>
  SELECT ?id
  where
    {
      graph ?g
       {
         ?id a foaf:Person .
         optional { ?id foaf:maker ?more }
       }
    }
  limit 10;

  id
  VARCHAR
  _____________________________________________________________________________

  mailto:somebody@example.domain
  http://example.com/dataspace/person/dav#this
  http://example.com/dataspace/person/dba#this
  mailto:2@F.D
  http://example.com/dataspace/person/test1#this
  http://www.openlinksw.com/blog/~kidehen/gems/rss.xml#Kingsley%20Uyi%20Idehen
  http://digitalmusic.weblogsinc.com/rss.xml#
  http://partners.userland.com/nytrss/books.xml#
  http://partners.userland.com/nytrss/arts.xml#

  9 Rows. -- 115 msec.
  ```
- *input:grab-resolver*

  is a name of procedure that resolve IRIs and determines the HTTP method of retrieval. The default is name of

  DB.DBA.RDF_GRAB_RESOLVER_DEFAULT()

  procedure that is described below. If other procedure is specified, the signature should match to the default one.
- *input:grab-destination*

  is to override the default behaviour of the IRI dereferencing and store all retrieved triples in a single graph. This is convenient when there's no logical difference where any given triple comes from, and changes in remote resources will only add triples but not make cached triples obsolete. A SPARQL query is usually faster when all graph IRIs are fixed and there are no graph group patterns with an unbound graph variable, so storing everything in one single graph is worth considering.
  ```

- *input:grab-loader*

  is a name of procedure that retrieve the resource via HTTP, parse it and store it. The default is name of

  DB.DBA.RDF_SPONGE_UP()

  procedure; this procedure also used by IRI dereferencing for FROM clauses. You will probably never need to write your own procedure of this sort but some Virtuoso plugins will provide ready-to-use functions that will retrieve non-RDF resources and extract their metadata as triples or will implement protocols other than HTTP.

Default resolver procedure is *DB.DBA.RDF_GRAB_RESOLVER_DEFAULT()* . Note that the function produce two absolute URIs, *abs_uri* and *dest_uri* . Default procedure returns two equal strings, but other may return different values, e.g., return primary and permanent location of the resource as *dest_uri* and the fastest known mirror location as *abs_uri* thus saving HTTP retrieval time. It can even signal an error to block the downloading of some unwanted resource.

```
DB.DBA.RDF_GRAB_RESOLVER_DEFAULT (
  in base varchar,         -- base IRI as specified by input:grab-base pragma
  in rel_uri varchar,      -- IRI of the resource as it is specified by input:grab-iri or a value of a va
  out abs_uri varchar,     -- the absolute IRI that should be downloaded
  out dest_uri varchar,    -- the graph IRI where triples should be stored after download
  out get_method varchar ) -- the HTTP method to use, should be "GET" or "MGET".
```

16.13.3. URL rewriting

URL rewriting is the act of modifying a source URL prior to the final processing of that URL by a Web Server.

The ability to rewrite URLs may be desirable for many reasons that include:

- Changing Web information resource URLs on the a Web Server without breaking existing bookmarks held in User Agents (e.g., Web browsers)
- URL compaction where shorter URLs may be constructed on a conditional basis for specific User Agents (e.g. Email clients)
- Construction of search engine friendly URLs that enable richer indexing since most search engines cannot process parameterized URLs effectively.

Using URL Rewriting to Solve Linked Data Deployment Challenges

URI naming schemes don't resolve the challenges associated with referencing data. To reiterate, this is demonstrated by the fact that the URIs http://demo.openlinksw.com/Northwind/Customer/ALFKI and http://demo.openlinksw.com/Northwind/Customer/ALFKI#this both appear as http://demo.openlinksw.com/Northwind/Customer/ALFKI to the Web Server, since data following the fragment identifier "#" never makes it that far.

The only way to address data referencing is by pre-processing source URIs (e.g. via regular expression or sprintf substitutions) as part of a URL rewriting processing pipeline. The pipeline process has to take the form of a set of rules that cater for elements such as HTTP Accept headers, HTTP response code, HTTP response headers, and rule processing order.

An example of such a pipeline is depicted in the table below.

Table 16.17. Pre-processing source URIs

URI Source(Regular Expression Pattern)	HTTP Accept Headers(Regular Expression)	HTTPResponse Code	HTTP Response Headers	Rule Processing Order
/Northwind/Customer/([^#]*)	None (meaning default)	200 or 303 redirect to a resource with default representation.	None	Normal (order irrelevant)
/Northwind/Customer/([^#]*)	(text/rdf.n3)	(application/rdf.xml)	303 redirect to location of a descriptive and associated resource (e.g. RESTful Web	None

URI Source(Regular Expression Pattern)	HTTP Accept Headers(Regular Expression)	HTTPResponse Code	HTTP Response Headers	Rule Processing Order
			Service that returns desired representation)	
/Northwind/Customer/([^#]*)	(text/html)	(application/xhtml.xml)	406 (Not Acceptable)or303 redirect to location of resource in requested representation	Vary: negotiate, acceptAlternates: {"ALFKI" 0.9 {type application/rdf+xml}}

The source URI patterns refer to virtual or physical directories for ex. at http://demo.openlinksw.com/. Rules can be placed at the head or tail of the pipeline, or applied in the order they are declared, by specifying a Rule Processing Order of First, Last, or Normal, respectively. The decision as to which representation to return for URI http://demo.openlinksw.com/Northwind/Customer/ALFKI is based on the MIME type(s) specified in any Accept header accompanying the request.

In the case of the last rule, the Alternates response header applies only to response code 406. 406 would be returned if there were no (X)HTML representation available for the requested resource. In the example shown, an alternative representation is available in RDF/XML.

When applied to matching HTTP requests, the last two rules might generate responses similar to those below:

```
$ curl -I -H "Accept: application/rdf+xml" http://demo.openlinksw.com/Northwind/Customer/ALFKI

HTTP/1.1 303 See Other
Server: Virtuoso/05.00.3016 (Solaris) x86_64-sun-solaris2.10-64 PHP5
Connection: close
Content-Type: text/html; charset=ISO-8859-1
Date: Mon, 16 Jul 2007 22:40:03 GMT
Accept-Ranges: bytes
Location: /sparql?query=CONSTRUCT+{+%3Chttp%3A//demo.openlinksw.com/Northwind/Customer/ALFKI%23this%3E+%3
Content-Length: 0
```

In the cURL exchange depicted above, the target Virtuoso server redirects to a SPARQL endpoint that retrieves an RDF/XML representation of the requested entity.

```
$ curl -I -H "Accept: text/html" http://demo.openlinksw.com/Northwind/Customer/ALFKI

HTTP/1.1 406 Not Acceptable
Server: Virtuoso/05.00.3016 (Solaris) x86_64-sun-solaris2.10-64 PHP5
Connection: close
Content-Type: text/html; charset=ISO-8859-1
Date: Mon, 16 Jul 2007 22:40:23 GMT
Accept-Ranges: bytes
Vary: negotiate,accept
Alternates: {"ALFKI" 0.9 {type application/rdf+xml}}
Content-Length: 0
```

In this second cURL exchange, the target Virtuoso server indicates that there is no resource to deliver in the requested representation. It provides hints in the form of an alternate resource representation and URI that may be appropriate, i.e., an RDF/XML representation of the requested entity.

The Virtuoso Rules-Based URL Rewriter

Virtuoso provides a URL rewriter that can be enabled for URLs matching specified patterns. Coupled with customizable HTTP response headers and response codes, Data-Web server administrators can configure highly flexible rules for driving content negotiation and URL rewriting. The key elements of the URL rewriter are:

- Rewriting rule
- Each rule describes how to parse a single source URL, and how to compose the URL of the page ultimately returned in the "Location:" response headers
- Every rewriting rule is uniquely identified internally (using IRIs).

- Two types of rule are supported, based on the syntax used to describe the source URL pattern matching: sprintf-based and regex-based.
- Rewrite rules list
- A named ordered list of rewrite rules or rule lists where rules of the list are processed from top to bottom or in line with processing pipeline precedence instructions
- Configuration API
- The rewriter configuration API defines functions for creating, dropping, and enumerating rules and rule lists.
- Virtual hosts and virtual paths
- URL rewriting is enabled by associating a rewrite rules list with a virtual directory

Virtual Domains (Hosts) & Directories

A Virtuoso virtual directory maps a logical path to a physical directory that is file system or WebDAV based. This mechanism allows physical locations to be hidden or simply reorganised. Virtual directory definitions are held in the system table DB.DBA.HTTP_PATH. Virtual directories can be administered in three basic ways:

- Using the Visual Administration Interface via a Web browser;
- Using the functions vhost_define() and vhost_remove(); and
- Using SQL statements to directly update the HTTP_PATH system table.

"Nice" URLs vs. "Long" URLs

Although we are approaching the URL Rewriter from the perspective of deploying linked data, the Rewriter was developed with additional objectives in mind. These in turn have influenced the naming of some of the formal argument names in the Configuration API function prototypes. In the following sections, long URLs are those containing a query string with named parameters; nice (aka. source) URLs have data encoded in some other format. The primary goal of the Rewriter is to accept a nice URL from an application and convert this into a long URL, which then identifies the page that should actually be retrieved.

Rule Processing Mechanics

When an HTTP request is accepted by the Virtuoso HTTP server, the received nice URL is passed to an internal path translation function. This function takes the nice URL and, if the current virtual directory has a url_rewrite option set to an existing ruleset name, tries to match the corresponding rulesets and rules; that is, it performs a recursive traversal of any rulelist associated with it. For every rule in the rulelist, the same logic is applied (only the logic for regex-based rules is described; that for sprintf-based rules is very similar):

- The input for the rule is the resource URL as received from the HTTP header, i.e., the portion of the URL from the first '/' after the host:port fields to the end of the URL.
- The input is normalized.
- The input is matched against the rule's regex. If the match fails, the rule is not applied and the next rule is tried. If the match succeeds, the result is a vector of values.
- If the URL contains a query string, the names and values of the parameters are decoded by split_and_decode().
- The names and values of any parameters in the request body are also decoded.
- The destination URL is composed
- The value of each parameter in the destination URL is taken from (in order of priority)
- The value of a parameter in the match result;
- The value of a named parameter in the query string of the input nice URL;
- If the original request was submitted by the POST method, the value of a named parameter in the body of the POST request; or
- if a parameter value cannot be derived from one of these sources, the rule is not applied and the next rule is tried.

The path translation function described above is internal to the Web server, so its signature is not appropriate for Virtuoso/PL calls and thus is not published. Virtuoso/PL developers can harness the same functionality using the DB.DBA.URLREWRITE_APPLY API call.

Enabling URL Rewriting via the Virtuoso Conductor UI

Virtuoso is a full-blown HTTP server in its own right. The HTTP server functionality co-exists with the product core (i.e., DBMS Engine, Web Services Platform, WebDAV filesystem, and other components of the Universal Server). As a result, it has the ability to multi-home Web domains within a single instance across a variety of domain name and port combinations. In addition, it also enables the creation of multiple virtual directories per domain.

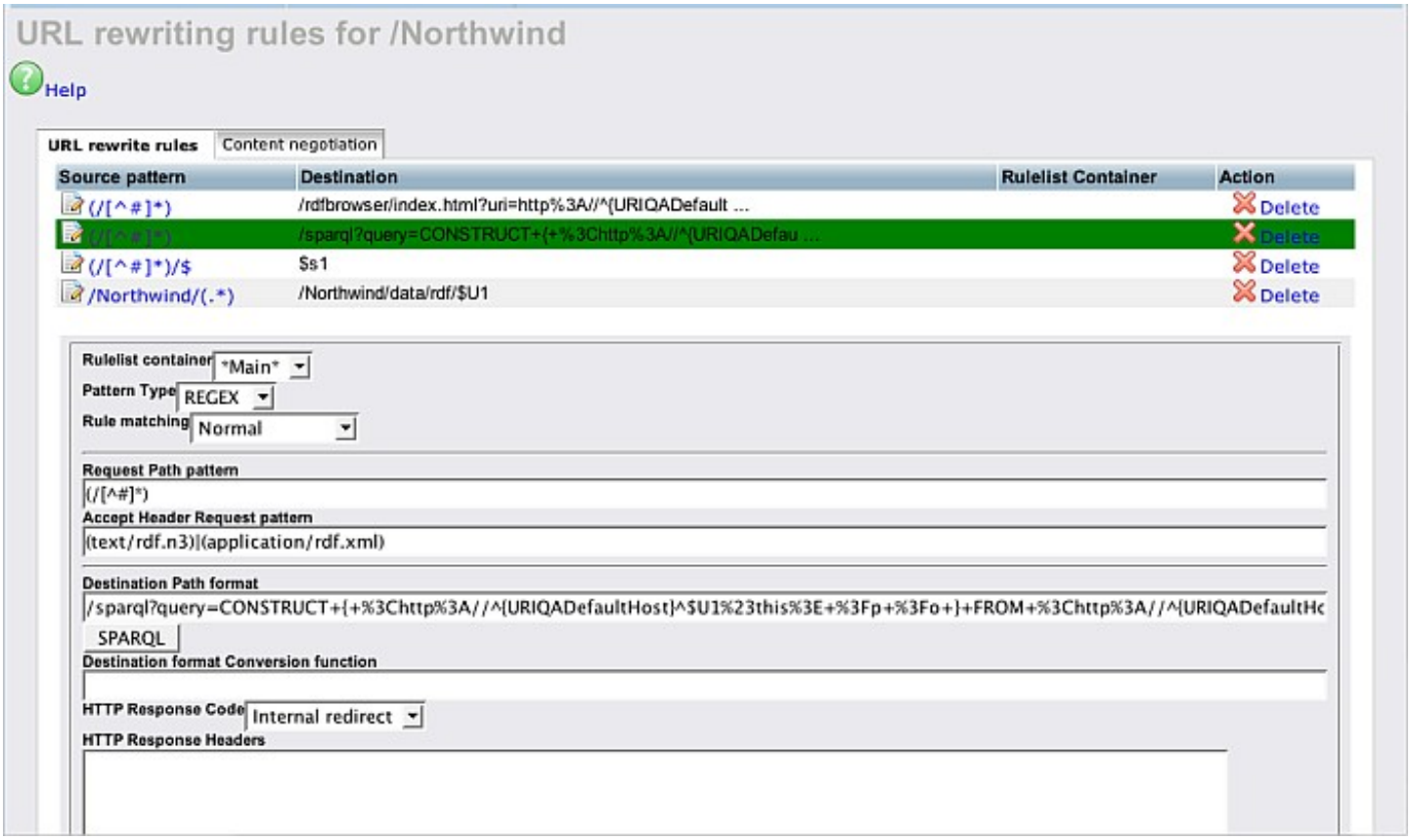
In addition to the basic functionality, Virtuoso facilitates the association of URL Rewriting rules with the virtual directories associated with a hosted Web domain.

In all cases, Virtuoso enables you to configure virtual domains, virtual directories and URL rewrite rules for one or more virtual directories, via the (X)HTML-based Conductor Admin User Interface or a collection of Virtuoso Stored Procedure Language (PL)-based APIs.

The steps for configuring URL Rewrite rules via the Virtuoso Conductor are as follows:

- Assuming you are using the local demonstration database, load http://example.com/conductor into your browser, and then proceed through the Conductor as follows:
- Click the "Web Application Server", and "Virtual Domains & Directories" tabs
- Pick the domain that contains the virtual directories to which the rules are to be applied (in this case the default was taken)
- Click on the "URL-rewrite" link to create, delete, or edit a rule as shown below:
- Create a Rule for HTML Representation Requests (via SPARQL SELECT Query)
- Create a Rule for RDF Representation Requests (via SPARQL CONSTRUCT Query)
- Then save and exit the Conductor, and test your rules with curl or any other User Agent.

Figure 16.184. URL-rewrite UI using Conductor



Enabling URL Rewriting via Virtuoso PL

The vhost_define()API is used to define virtual hosts and virtual paths hosted by the Virtuoso HTTP server. URL rewriting is enabled through this function's opts parameter. opts is of type ANY, e.g., a vector of field-value pairs. Numerous fields are recognized for controlling different options. The field value url_rewrite controls URL rewriting. The corresponding field value is the IRI of a rule list to apply.

Configuration API

Virtuoso includes the following functions for managing URL rewriting rules and rule lists. The names are self-explanatory.

```
-- Deletes a rewriting rule
DB.DBA.URLREWRITE_DROP_RULE
```

```
-- Creates a rewriting rule which uses sprintf-based pattern matching
DB.DBA.URLREWRITE_CREATE_SPRINTF_RULE

-- Creates a rewriting rule which uses regular expression (regex) based pattern matching
DB.DBA.URLREWRITE_CREATE_REGEX_RULE

-- Deletes a rewriting rule list
DB.DBA.URLREWRITE_DROP_RULELIST

-- Creates a rewriting rule list
DB.DBA.URLREWRITE_CREATE_RULELIST

-- Lists all the rules whose IRI match the specified 'SQL like' pattern
DB.DBA.URLREWRITE_ENUMERATE_RULES

-- Lists all the rule lists whose IRIs match the specified 'SQL like' pattern
DB.DBA.URLREWRITE_ENUMERATE_RULELISTS
```

Creating Rewriting Rules

Rewriting rules take two forms: sprintf-based or regex-based. When used for nice URL to long URL conversion, the only difference between them is the syntax of format strings. The reverse long to nice conversion works only for sprintf-based rules, whereas regex-based rules are unidirectional.

For the purposes of describing how to make dereferenceable URIs for linked data, we will stick with the nice to long conversion using regex-based rules.

Regex rules are created using the *URLREWRITE_CREATE_REGEX_RULE()* function.

Example - URL Rewriting For the Northwind Linked Data View

The Northwind schema is comprised of commonly understood SQL Tables that include: Customers, Orders, Employees, Products, Product Categories, Shippers, Countries, Provinces etc.

An Linked Data View of SQL data is an RDF named graph (RDF data set) comprised of RDF Linked Data (triples) stored in a Virtuoso Quad Store (the native RDF Data Management realm of Virtuoso).

In this example we are going interact with Linked Data deployed into the Data-Web from a live instance of Virtuoso, which uses the URL Rewrite rules from the prior section.

The components used in the example are as follows:

- Virtuoso SPARQL Endpoint: http://demo.openlinksw.com/sparql
- Named RDF Graph: http://demo.openlinksw.com/Northwind
- Entity ID - http://demo.openlinksw.com/Northwind/Customer/ALFKI#this
- Information Resource: http://demo.openlinksw.com/Northwind/Customer/ALFKI
- Interactive SPARQL Query Builder (iSPARQL) - http://demo.openlinksw.com/DAV/JS/isparql/index.html

Northwind URL Rewriting Verification Using curl

The curl utility provides a useful tool for verifying HTTP server responses and rewriting rules. The curl exchanges below show the URL rewriting rules defined for the Northwind RDF view being applied.

Example 1:

```
$ curl -I -H "Accept: text/html" http://demo.openlinksw.com/Northwind/Customer/ALFKI

HTTP/1.1 303 See Other
Server: Virtuoso/05.00.3016 (Solaris) x86_64-sun-solaris2.10-64 PHP5
Connection: close
Content-Type: text/html; charset=ISO-8859-1
Date: Tue, 14 Aug 2007 13:30:02 GMT
Accept-Ranges: bytes
Location: http://demo.openlinksw.com/about/html/http/demo.openlinksw.com/Northwind/Customer/ALFKI
Content-Length: 0
```

Example 2:

```
$ curl -I -H "Accept: application/rdf+xml" http://demo.openlinksw.com/Northwind/Customer/ALFKI

HTTP/1.1 303 See Other
Server: Virtuoso/05.00.3016 (Solaris) x86_64-sun-solaris2.10-64 PHP5
Connection: close
Content-Type: text/html; charset=ISO-8859-1
Date: Tue, 14 Aug 2007 13:30:22 GMT
Accept-Ranges: bytes
Location: /sparql?query=CONSTRUCT+{+%3Chttp%3A//demo.openlinksw.com/Northwind/Customer/ALFKI%23this%3E+%3
Content-Length: 0
```

Example 3:

```
$ curl -I -H "Accept: text/html" http://demo.openlinksw.com/Northwind/Customer/ALFKI#this

HTTP/1.1 404 Not Found
Server: Virtuoso/05.00.3016 (Solaris) x86_64-sun-solaris2.10-64 PHP5
Connection: Keep-Alive
Content-Type: text/html; charset=ISO-8859-1
Date: Tue, 14 Aug 2007 13:31:01 GMT
Accept-Ranges: bytes
Content-Length: 0
```

The output above shows how RDF entities from the Data-Web, in this case customer ALFKI, are exposed in the Document Web. The power of SPARQL coupled with URL rewriting enables us to produce results in line with the desired representation. A SPARQL SELECT or CONSTRUCT query is used depending on whether the requested representation is text/html or application/rdf+xml, respectively.

The 404 response in Example 3 indicates that no HTML representation is available for entity ALFKI#this. In most cases, a URI of this form (containing a '#' fragment identifier) will not reach the server. This example supposes that it does: i.e., the RDF client and network routing allows the suffixed request. The presence of the #this suffix implicitly states that this is a request for a data resource in the Data-Web realm, not a document resource from the Document Web.2

Rather than return 404, we could instead choose to construct our rewriting rules to perform a 303 redirect, so that the response for ALFKI#this in Example 3 becomes the same as that for ALFKI in Example 1.

Transparent Content Negotiation

So as not to overload our preceding description of Linked Data deployment with excessive detail, the description of content negotiation presented thus far was kept deliberately brief. This section discusses content negotiation in more detail.

HTTP/1.1 Content Negotiation

Recall that a resource (conceptual entity) identified by a URI may be associated with more than one representation (e.g. multiple languages, data formats, sizes, resolutions). If multiple representations are available, the resource is referred to as negotiable and each of its representations is termed a variant. For instance, a Web document resource, named 'ALFKI' may have three variants: alfki.xml, alfki.html and alfki.txt all representing the same data. Content negotiation provides a mechanism for selecting the best variant.

As outlined in the earlier brief discussion of content negotiation, when a user agent requests a resource, it can include with the request Accept headers (Accept, Accept-Language, Accept-Charset, Accept-Encoding etc.) which express the user preferences and user agent capabilities. The server then chooses and returns the best variant based on the Accept headers. Because the selection of the best resource representation is made by the server, this scheme is classed as server-driven negotiation.

Transparent Content Negotiation

An alternative content negotiation mechanism is Transparent Content Negotiation (TCN), a protocol defined by RFC2295 . TCN offers a number of benefits over standard HTTP/1.1 negotiation, for suitably enabled user agents.

RFC2295 introduces a number of new HTTP headers including the Negotiate request header, and the TCN and Alternates response headers. (Krishnamurthy et al. note that although the HTTP/1.1 specification reserved the Alternates header for use in agent driven negotiation, it was not fully specified. Consequently under a pure HTTP/1.1 implementation as defined by RFC2616, server-driven content negotiation is the only option. RFC2295 addresses this issue.)

Deficiencies of HTTP/1.1 Server-Driven Negotiation

Weaknesses of server-driven negotiation highlighted by RFCs 2295 and 2616 include:

- Inefficiency - Sending details of a user agent's capabilities and preferences with every request is very inefficient, not least because very few Web resources have multiple variants, and expensive in terms of the number of Accept headers required to fully describe all but the most simple browser's capabilities.
- Server doesn't always know 'best' - Having the server decide on the 'best' variant may not always result in the most suitable resource representation being returned to the client. The user agent might often be better placed to decide what is best for its needs.

Variant Selection By User Agent

Rather than rely on server-driven negotiation and variant selection by the server, a user agent can take full control over deciding the best variant by explicitly requesting transparent content negotiation through the Negotiate request header. The negotiation is 'transparent' because it makes all the variants on the server visible to the agent.

Under this scheme, the server sends the user agent a list, represented in an Alternates header, containing the available variants and their properties. The user agent can then choose the best variant itself. Consequently, the agent no longer needs to send large Accept headers describing in detail its capabilities and preferences. (However, unless caching is used, user-agent driven negotiation does suffer from the disadvantage of needing a second request to obtain the best representation. By sending its best guess as the first response, server driven negotiation avoids this second request if the initial best guess is acceptable.)

Variant Selection By Server

As well as variant selection by the user agent, TCN allows the server to choose on behalf of the user agent if the user agent explicitly allows it through the Negotiate request header. This option allows the user agent to send smaller Accept headers containing enough information to allow the server to choose the best variant and return it directly. The server's choice is controlled by a 'remote variant selection algorithm' as defined in RFC2296.

Variant Selection By End-User

A further option is to allow the end-user to select a variant, in case the choice made by negotiation process is not optimal. For instance, the user agent could display an HTML-based 'pick list' of variants constructed from the variant list returned by the server. Alternatively the server could generate this pick list itself and include it in the response to a user agent's request for a variant list. (Virtuoso currently responds this way.)

Transparent Content Negotiation in Virtuoso HTTP Server

The following section describes the Virtuoso HTTP server's TCN implementation which is based on RFC2295, but without "Feature" negotiation. OpenLink's RDF rich clients, iSparql and the OpenLink RDF Browser, both support TCN. User agents which do not support transparent content negotiation continue to be handled using HTTP/1.1 style content negotiation (whereby server-side selection is the only option - the server selects the best variant and returns a list of variants in an Alternates response header).

Describing Resource Variants

In order to negotiate a resource, the server needs to be given information about each of the variants. Variant descriptions are held in SQL table HTTP_VARIANT_MAP. The descriptions themselves can be created, updated or deleted using Virtuoso/PL or through the Conductor UI. The table definition is as follows:

```
create table DB.DBA.HTTP_VARIANT_MAP (
  VM_ID integer identity, -- unique ID
  VM_RULELIST varchar, -- HTTP rule list name
  VM_URI varchar, -- name of requested resource e.g. 'page'
  VM_VARIANT_URI varchar, -- name of variant e.g. 'page.xml', 'page.de.html' etc.
  VM_QS float, -- Source quality, a number in the range 0.001-1.000, with 3 digit precision
  VM_TYPE varchar, -- Content type of the variant e.g. text/xml
  VM_LANG varchar, -- Content language e.g. 'en', 'de' etc.
  VM_ENC varchar, -- Content encoding e.g. 'utf-8', 'ISO-8892' etc.
  VM_DESCRIPTION long varchar, -- a human readable description about the variant e.g. 'Profile in RDF for
  VM_ALGO int default 0, -- reserved for future use
  primary key (VM_RULELIST, VM_URI, VM_VARIANT_URI)
 )
```

```
create unique index HTTP_VARIANT_MAP_ID on DB.DBA.HTTP_VARIANT_MAP (VM_ID)
```

Configuration using Virtuoso/PL

Two functions are provided for adding or updating, or removing variant descriptions using Virtuoso/PL:

```
-- Adding or Updating a Resource Variant:
DB.DBA.HTTP_VARIANT_ADD (
  in rulelist_uri varchar, -- HTTP rule list name
  in uri varchar, -- Requested resource name e.g. 'page'
  in variant_uri varchar, -- Variant name e.g. 'page.xml', 'page.de.html' etc.
  in mime varchar, -- Content type of the variant e.g. text/xml
  in qs float := 1.0, -- Source quality, a floating point number with 3 digit precision in 0.001-1.000 ra
  in description varchar := null, -- a human readable description of the variant e.g. 'Profile in RDF for
  in lang varchar := null, -- Content language e.g. 'en', 'bg'. 'de' etc.
  in enc varchar := null -- Content encoding e.g. 'utf-8', 'ISO-8892' etc.
)

--Removing a Resource Variant
DB.DBA.HTTP_VARIANT_REMOVE (
  in rulelist_uri varchar, -- HTTP rule list name
  in uri varchar, -- Name of requested resource e.g. 'page'
  in variant_uri varchar := '%' -- Variant name filter
)
```

Configuration using Conductor UI

The Conductor 'Content negotiation' panel for describing resource variants and configuring content negotiation is depicted below. It can be reached by selecting the 'Virtual Domains & Directories' tab under the 'Web Application Server' menu item, then selecting the 'URL rewrite' option for a logical path listed amongst those for the relevant HTTP host, e.g. '{Default Web Site}'

The input fields reflect the supported 'dimensions' of negotiation which include content type, language and encoding. Quality values corresponding to the options for 'Source Quality' are as follows:

Table 16.18. Source Quality

Source Quality	Quality Value
perfect representation	1.000
threshold of noticeable loss of quality	0.900
noticeable, but acceptable quality reduction	0.800
barely acceptable quality	0.500
severely degraded quality	0.300
completely degraded quality	0.000

Variant Selection Algorithm

When a user agent instructs the server to select the best variant, Virtuoso does so using the selection algorithm below:

If a virtual directory has URL rewriting enabled (has the 'url_rewrite' option set), the web server:

- Looks in DB.DBA.HTTP_VARIANT_MAP for a VM_RULELIST matching the one specified in the 'url_rewrite' option
- If present, it loops over all variants for which VM_URI is equal to the resource requested
- For every variant it calculates the source quality based on the value of VM_QS and the source quality given by the user agent
- If the best variant is found, it adds TCN HTTP headers to the response and passes the VM_VARIANT_URI to the URL rewriter
- If the user agent has asked for a variant list, it composes such a list and returns an 'Alternates' HTTP header with response code 300
- If no URL rewriter rules exist for the target URL, the web server returns the content of the dereferenced VM_VARIANT_URI.

The server may return the best-choice resource representation or a list of available resource variants. When a user agent requests transparent negotiation, the web server returns the TCN header "choice". When a user agent asks for a variant list, the server

returns the TCN header "list".

Examples

In this example we assume the following files have been uploaded to the Virtuoso WebDAV server, with each containing the same information but in different formats:

- /DAV/TCN/page.xml - a XML variant
- /DAV/TCN/page.html - a HTML variant
- /DAV/TCN/page.txt - a text variant

We add TCN rules and define a virtual directory:

```
DB.DBA.HTTP_VARIANT_ADD ('http_rule_list_1', 'page', 'page.html','text/html', 0.900000, 'HTML variant');
DB.DBA.HTTP_VARIANT_ADD ('http_rule_list_1', 'page', 'page.txt', 'text/plain', 0.500000, 'Text document')
DB.DBA.HTTP_VARIANT_ADD ('http_rule_list_1', 'page', 'page.xml', 'text/xml', 1.000000, 'XML variant');
DB.DBA.VHOST_DEFINE (lpath=>'/DAV/TCN/',
                     ppath=>'/DAV/TCN/',
                     is_dav=>1,
                     vsp_user=>'dba',
                     opts=>vector ('url_rewrite', 'http_rule_list_1'));
```

Having done this we can now test the setup with a suitable HTTP client, in this case the curl command line utility. In the following examples, the curl client supplies Negotiate request headers containing content negotiation directives which include:

- "trans" - The user agent supports transparent content negotiation for the current request.
- "vlist" - The user agent requests that any transparently negotiated response for the current request includes an Alternates header with the variant list bound to the negotiable resource. Implies "trans".
- "*" - The user agent allows servers and proxies to run any remote variant selection algorithm.

The server returns a TCN response header signalling that the resource is transparently negotiated and either a choice or a list response as appropriate.

In the first curl exchange, the user agent indicates to the server that, of the formats it recognizes, HTML is preferred and it instructs the server to perform transparent content negotiation. In the response, the Vary header field expresses the parameters the server used to select a representation, i.e. only the Negotiate and Accept header fields are considered.

```
$ curl -i -H "Accept: text/xml;q=0.3,text/html;q=1.0,text/plain;q=0.5,*/*;
q=0.3" -H "Negotiate: *" http://example.com/DAV/TCN/page
HTTP/1.1 200 OK Server: Virtuoso/05.00.3021 (Linux) i686-pc-linux-gnu
VDB Connection: Keep-Alive Date: Wed, 31 Oct 2007 15:43:18
GMT Accept-Ranges: bytes TCN: choice Vary: negotiate,accept
Content-Location: page.html Content-Type: text/html
ETag: "14056a25c066a6e0a6e65889754a0602"
Content-Length: 49
<html> <body> some html </body> </html>
```

Next, the source quality values are adjusted so that the user agent indicates that XML is its preferred format.

```
$ curl -i -H "Accept: text/xml,text/html;q=0.7,text/plain;q=0.5,*/*;q=0.3" -H "Negotiate:
*" http://example.com/DAV/TCN/page HTTP/1.1 200 OK Server: Virtuoso/05.00.3021
(Linux) i686-pc-linux-gnu VDB Connection: Keep-Alive Date: Wed, 31 Oct 2007
15:44:07 GMT Accept-Ranges: bytes TCN: choice Vary: negotiate,accept
Content-Location: page.xml Content-Type: text/xml ETag:
"8b09f4b8e358fcb7fd1f0f8fa918973a" Content-Length: 39

<?xml version="1.0" ?> <a>some xml</a>
```

In the final example, the user agent wants to decide itself which is the most suitable representation, so it asks for a list of variants. The server provides the list, in the form of an Alternates response header, and, in addition, sends an HTML representation of the list so that the end user can decide on the preferred variant himself if the user agent is unable to.

```
$ curl -i -H "Accept: text/xml,text/html;q=0.7,text/plain;q=0.5,*/*;q=0.3" -H "Negotiate:
vlist" http://example.com/DAV/TCN/page HTTP/1.1 300 Multiple Choices Server:
Virtuoso/05.00.3021 (Linux) i686-pc-linux-gnu VDB Connection: close Content-Type:
text/html; charset=ISO-8859-1 Date: Wed, 31 Oct 2007 15:44:35 GMT Accept-Ranges:
```

```
bytes TCN: list Vary: negotiate,accept Alternates: {"page.html" 0.900000 {type text/html}},
{"page.txt" 0.500000 {type text/plain}}, {"page.xml" 1.000000 {type text/xml}} Content-Length: 368

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html>
<head>
<title>300 Multiple Choices</title>
</head>
<body>
<h1>Multiple Choices</h1>
Available variants:
<ul>
<li>
<a href="page.html">HTML variant</a>, type text/html</li>
<li><a href="page.txt">Text document</a>, type text/plain</li>
<li><a href="page.xml">XML variant</a>, type text/xml</li>
</ul>
</body>
</html>
```

16.13.4. Examples of other Protocol Resolvers

Example of *LSIDs* : A scientific name from UBio

```
SQL>SPARQL
define get:soft "soft"
SELECT *
FROM <urn:lsid:ubio.org:namebank:11815>
WHERE { ?s ?p ?o }
LIMIT 5;

s                                 p                                          o
VARCHAR                           VARCHAR                                    VARCHAR
_______________________________________________________________________________

urn:lsid:ubio.org:namebank:11815  http://purl.org/dc/elements/1.1/title       Pternistis leucoscepus
urn:lsid:ubio.org:namebank:11815  http://purl.org/dc/elements/1.1/subject     Pternistis leucoscepus (Gra
urn:lsid:ubio.org:namebank:11815  http://purl.org/dc/elements/1.1/identifier  urn:lsid:ubio.org:namebank:
urn:lsid:ubio.org:namebank:11815  http://purl.org/dc/elements/1.1/creator     http://www.ubio.org
urn:lsid:ubio.org:namebank:11815  http://purl.org/dc/elements/1.1/type        Scientific Name

5 Rows. -- 741 msec.
```

Example of *LSIDs* : A segment of the human genome from GDB

```
SQL>SPARQL
define get:soft "soft"
SELECT *
FROM <urn:lsid:gdb.org:GenomicSegment:GDB132938>
WHERE { ?s ?p ?o }
LIMIT 5;

s                                          p                                                  o
VARCHAR                                    VARCHAR                                            VARCHAR
_______________________________________________________________________________

urn:lsid:gdb.org:GenomicSegment:GDB132938  urn:lsid:gdb.org:DBObject-predicates:accessionID      GDB:1329
urn:lsid:gdb.org:GenomicSegment:GDB132938  http://www.ibm.com/LSID/2004/RDF/#lsidLink            urn:lsid
urn:lsid:gdb.org:GenomicSegment:GDB132938  urn:lsid:gdb.org:DBObject-predicates:objectClass      DBObject
urn:lsid:gdb.org:GenomicSegment:GDB132938  urn:lsid:gdb.org:DBObject-predicates:displayName      D20S95
urn:lsid:gdb.org:GenomicSegment:GDB132938  urn:lsid:gdb.org:GenomicSegment-predicates:variantsQ  nodeID:/

5 Rows. -- 822 msec.
```

Example of *OAI* : an institutional / departmental repository.

```
SQL>SPARQL
define get:soft "soft"
SELECT *
FROM <oai:etheses.bham.ac.uk:23>
WHERE { ?s ?p ?o }
```

```
LIMIT 5;

s                          p                                          o
VARCHAR                    VARCHAR                                    VARCHAR
_______________________________________________________________________________

oai:etheses.bham.ac.uk:23  http://purl.org/dc/elements/1.1/title      A study of the role of ATM mutati
oai:etheses.bham.ac.uk:23  http://purl.org/dc/elements/1.1/date       2007-07
oai:etheses.bham.ac.uk:23  http://purl.org/dc/elements/1.1/subject    RC0254 Neoplasms. Tumors. Oncolog
oai:etheses.bham.ac.uk:23  http://purl.org/dc/elements/1.1/identifier Austen, Belinda (2007) A study of
oai:etheses.bham.ac.uk:23  http://purl.org/dc/elements/1.1/identifier http://etheses.bham.ac.uk/23/1/Au

5 Rows. -- 461 msec.
```

Example of *DOI*

In order to execute correctly queries with doi resolver you need to have:

- the handle.dll file accessible from your system. For ex. you can put it in the Virtuoso bin folder where the rest of the server components are.
- in your Virtuoso database ini file in section Plugins added the hslookup.dll file, which location should be in the plugins folder under your Virtuoso server installation. For ex:

```
[Plugins]
LoadPath = ./plugin
...
Load6    = plain,hslookup
```

```
SQL>SPARQL
define get:soft "soft"
SELECT *
FROM <doi:10.1045/march99-bunker>
WHERE { ?s ?p ?o } ;

s                                                  p
VARCHAR                                            VARCHAR
_______________________________________________________________________________

http://www.dlib.org/dlib/march99/bunker/03bunker.html  http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://www.dlib.org/dlib/march99/bunker/03bunker.html  http://example.com/schemas/XHTML#title     Collabo

2 Rows. -- 12388 msec.
```

Other examples

```
SQL>SPARQL
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX doap: <http://usefulinc.com/ns/doap#>
SELECT DISTINCT ?name ?mbox ?projectName
WHERE {
 <http://dig.csail.mit.edu/2005/ajar/ajaw/data#Tabulator>
doap:developer ?dev .
 ?dev foaf:name ?name .
 OPTIONAL { ?dev foaf:mbox ?mbox }
 OPTIONAL { ?dev doap:project ?proj .
            ?proj foaf:name ?projectName }
};

name                 mbox    projectName
VARCHAR              VARCHAR VARCHAR
_____________________ _____________________________________________

Adam Lerer           NULL    NULL
Dan Connolly         NULL    NULL
David Li             NULL    NULL
David Sheets         NULL    NULL
James Hollenbach     NULL    NULL
Joe Presbrey         NULL    NULL
Kenny Lu             NULL    NULL
Lydia Chilton        NULL    NULL
Ruth Dhanaraj        NULL    NULL
```

```
Sonia Nijhawan       NULL    NULL
Tim Berners-Lee      NULL    NULL
Timothy Berners-Lee  NULL    NULL
Yuhsin Joyce Chen    NULL    NULL

13 Rows. -- 491 msec.

SQL>SPARQL
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?friendsname ?friendshomepage ?foafsname ?foafshomepage
WHERE
 {
  <http://example.com/dataspace/person/kidehen#this> foaf:knows ?friend .
  ?friend foaf:mbox_sha1sum ?mbox .
  ?friendsURI foaf:mbox_sha1sum ?mbox .
  ?friendsURI foaf:name ?friendsname .
  ?friendsURI foaf:homepage ?friendshomepage .
  OPTIONAL { ?friendsURI foaf:knows ?foaf .
             ?foaf foaf:name ?foafsname .
             ?foaf foaf:homepage ?foafshomepage .
           }
 }
LIMIT 10;

friendsname        friendshomepage                          foafsname        foafshomepage
ANY                ANY                                      ANY              ANY
 Tim Berners Lee   http://www.w3.org/People/Berners-Lee/    Dan Connolly     http://www.w3.org/People/Conn
 Tim Berners Lee   http://www.w3.org/People/Berners-Lee/    Henry J. Story   http://bblfish.net/
 Tim Berners Lee   http://www.w3.org/People/Berners-Lee/    Henry Story      http://bblfish.net/
 Tim Berners Lee   http://www.w3.org/People/Berners-Lee/    Henry J. Story   http://bblfish.net/people/hen
 Tim Berners Lee   http://www.w3.org/People/Berners-Lee/    Henry Story      http://bblfish.net/people/hen
 Tim Berners Lee   http://www.w3.org/People/Berners-Lee/    Ruth Dhanaraj    http://web.mit.edu/ruthdhan/w
 Tim Berners Lee   http://www.w3.org/People/Berners-Lee/    Dan Brickley     http://danbri.org/
 Tim Berners Lee   http://www.w3.org/People/Berners-Lee/    Dan Brickley     http://danbri.org/
 Tim Berners Lee   http://www.w3.org/People/Berners-Lee/    Daniel Krech     http://eikeon.com/
 Tim Berners Lee   http://www.w3.org/People/Berners-Lee/    Daniel Krech     http://eikeon.com/
```

16.13.5. Faceted Views over Large-Scale Linked Data

Faceted views over structured and semi structured data have been popular in user interfaces for some years. Deploying such views of arbitrary linked data at arbitrary scale has been hampered by lack of suitable back end technology. Many ontologies are also quite large, with hundreds of thousands of classes.

Also, the linked data community has been concerned with the processing cost and potential for denial of service presented by public SPARQL end points.

This section discusses how we use Virtuoso Cluster Edition for providing interactive browsing over billions of triples, combining full text search, structured querying and result ranking. We discuss query planning, run-time inferencing and partial query evaluation. This functionality is exposed through SPARQL, a specialized web service and a web user interface.

The transition of the web from a distributed document repository into a universal, ubiquitous database requires a new dimension of scalability for supporting rich user interaction. If the web is the database, then it also needs a query and report writing tool to match. A faceted user interaction paradigm has been found useful for aiding discovery and query of variously structured data. Numerous implementations exist but they are chiefly client side and are limited in the data volumes they can handle.

At the present time, linked data is well beyond prototypes and proofs of concept. This means that what was done in limited specialty domains before must now be done at real world scale, in terms of both data volume and ontology size. On the schema, or T box side, there exist many comprehensive general purpose ontologies such as Yago[1], OpenCyc[2], Umbel[3] and the DBpedia[4] ontology and many domain specific ones, such as [5]. For these to enter into the user experience, the platform must be able to support the user's choice of terminology or terminologies as needed, preferably without blow up of data and concomitant slowdown.

Likewise, in the LOD world, many link sets have been created for bridging between data sets. Whether such linkage is relevant will depend on the use case. Therefore we provide fine grained control over which owl:sameAs assertions will be followed, if any.

Against this background, we discuss how we tackle incremental interactive query composition on arbitrary data with Virtuoso Cluster .

Using SPARQL or a web/web service interface, the user can form combinations of text search and structured criteria, including joins to an arbitrary depth. If queries are precise and select a limited number of results, the results are complete. If queries would select tens of millions of results, partial results are shown.

The system being described is being actively developed as of this writing, early March of 2009 and is online at http://lod.openlinksw.com/. The data set is a combination of DBpedia, MusicBrainz, Freebase, UniProt, NeuroCommons, Bio2RDF, and web crawls from PingTheSemanticWeb.com.

The hardware consists of two 8-core servers with 16G RAM and 4 disks each. The system runs on Virtuoso 6 Cluster Edition. All application code is written in SQL procedures with limited client side Ajax, the Virtuoso platform itself is in C.

The facets service allows the user to start with a text search or a fixed URI and to refine the search by specifying classes, property values etc., on the selected subjects or any subjects referenced therefrom.

This process generates queries involving combinations of text and structured criteria, often dealing with property and class hierarchies and often involving aggregation over millions of subjects, specially at the initial stages of query composition. To make this work with in interactive time, two things are needed:

1. a query optimizer that can almost infallibly produce the right join order based on cardinalities of the specific constants in the query
2. a query execution engine that can return partial results after a timeout.

It is often the case, specially at the beginning of query formulation, that the user only needs to know if there are relatively many or few results that are of a given type or involve a given property. Thus partially evaluating a query is often useful for producing this information. This must however be possible with an arbitrary query, simply citing precomputed statistics is not enough.

It has for a long time been a given that any search-like application ranks results by relevance. Whenever the facets service shows a list of results, not an aggregation of result types or properties, it is sorted on a composite of text match score and link density.

The section is divided into the following parts:

- SPARQL query optimization and execution adapted for run time inference over large subclass structures.
- Resolving identity with inverse functional properties
- Ranking entities based on graph link density
- SPARQL partial query evaluation for displaying partial results in fixed time
- a facets web service providing an XML interface for submitting queries, so that the user interface is not required to parse SPARQL
- a sample web interface for interacting with this
- sample queries and their evaluation times against combinations of large LOD data sets

Processing Large Hierarchies in SPARQL

Virtuoso has for a long time had built-in superclass and superproperty inference. This is enabled by specifying the *DEFINE input:inference "context"* option, where context is previously declared to be all subclass, subproperty, equivalence, inverse functional property and same as relations defined in a given graph. The ontology file is loaded into its own graph and this is then used to construct the context. Multiple ontologies and their equivalences can be loaded into a single graph which then makes another context which holds the union of the ontology information from the merged source ontologies.

Let us consider a sample query combining a full text search and a restriction on the class of the desired matches:

```
DEFINE  input:inference  "yago"
PREFIX  cy:  <http://dbpedia.org/class/yago/>
SELECT DISTINCT ?s1 AS ?c1
                ( bif:search_excerpt
                  ( bif:vector ( 'Shakespeare' ), ?o1 )
                ) AS ?c2
WHERE
  {
    ?s1  ?s1textp ?o1                          .
    FILTER
```

```
      ( bif:contains (?o1, '"Shakespeare"') ) .
    ?s1  a        cy:Performer110415638
  }
LIMIT 20
```

This selects all Yago performers that have a property that contains "Shakespeare" as a whole word.

The *DEFINE input:inference "yago"* clause means that subclass, subproperty and inverse functions property statements contained in the inference context called yago are considered when evaluating the query. The built-in function *bif:search_excerpt* makes a search engine style summary of the found text, highlighting occurrences of Shakespeare.

The *bif:contains* function in the filter specifies the full text search condition on ?o1.

This query is a typical example of queries that are executed all the time when a user refines a search. We will now look at how we can make an efficient execution plan for the query. First, we must know the cardinalities of the search conditions.

To see the count of subclasses of Yago performer, we can do:

```
SPARQL
PREFIX  cy:  <http://dbpedia.org/class/yago/>
SELECT COUNT (*)
FROM <http://dbpedia.org/yago.owl>
WHERE
  {
    ?s  rdfs:subClassOf  cy:Performer110415638
    OPTION (TRANSITIVE, T_DISTINCT)
  }
```

There are 4601 distinct subclasses, including indirect ones. Next we look at how many Shakespeare mentions there are:

```
SPARQL
SELECT COUNT (*)
WHERE
  {
    ?s  ?p  ?o .
    FILTER
      ( bif:contains (?o, 'Shakespeare') )
  }
```

There are 10267 subjects with Shakespeare mentioned in some literal.

```
SPARQL
DEFINE input:inference "yago"
PREFIX cy: <http://dbpedia.org/class/yago/>
SELECT COUNT (*)
WHERE
  {
    ?s1  a  cy:Performer110415638
  }
```

There are 184885 individuals that belong to some subclass of performer.

This is the data that the SPARQL compiler must know in order to have a valid query plan. Since these values will wildly vary depending on the specific constants in the query, the actual database must be consulted as needed while preparing the execution plan. This is regular query processing technology but is now specially adapted for deep subclass and subproperty structures.

Conditions in the queries are not evaluated twice, once for the cardinality estimate and once for the actual run. Instead, the cardinality estimate is a rapid sampling of the index trees that reads at most one leaf page.

Consider a B tree index, which we descend from top to the leftmost leaf containing a match of the condition. At each level, we count how many children would match and always select the leftmost one. When we reach a leaf, we see how many entries are on the page. From these observations, we extrapolate the total count of matches.

With this method, the guess for the count of performers is 114213, which is acceptably close to the real number. Given these numbers, we see that it makes sense to first find the full text matches and then retrieve the actual classes of each and see if this class is a subclass of performer. This last check is done against a memory resident copy of the Yago hierarchy, the same copy that

was used for enumerating the subclasses of performer.

However, the query

```
SPARQL
DEFINE input:inference "yago"
PREFIX cy: <http://dbpedia.org/class/yago/>
SELECT DISTINCT ?s1 AS ?c1,
                ( bif:search_excerpt
                  ( bif:vector ('Shakespeare'), ?o1 )
                ) AS ?c2
WHERE
  {
    ?s1  ?s1textp  ?o1                          .
    FILTER
      ( bif:contains (?o1, '"Shakespeare"') )   .
    ?s1  a          cy:ShakespeareanActors
  }
```

will start with Shakespearean actors since this is a leaf class with only 74 instances and then check if the properties contain Shakespeare and return their search summaries.

In principle, this is common cost based optimization but is here adapted to deep hierarchies combined with text patterns. An unmodified SQL optimizer would have no possibility of arriving at these results.

The implementation reads the graphs designated as holding ontologies when first needed and subsequently keeps a memory based copy of the hierarchy on all servers. This is used for quick iteration over sub/superclasses or properties as well as for checking if a given class or property is a subclass/property of another. Triples with OWL predicates *equivalentClass* , *equivalentProperty* and *sameAs* are also cached in the same data structure if they occur in the ontology graphs.

Also cardinality estimates for members of classes near the root of the class hierarchy take some time since a sample of each subclass is needed. These are cached for some minutes in the inference context, so that repeated queries will not redo the sampling.

Inverse Functional Properties and Same As

Specially when navigating social data, as in FOAF and SIOC spaces, there are many blank nodes that are identified by properties only. For this, we offer an option for automatically joining to subjects which share an IFP value with the subject being processed. For example, the query for the friends of friends of Kjetil Kjernsmo returns empty:

```
SPARQL
SELECT COUNT (?f2)
WHERE
  {
    ?s   a             foaf:Person          ;
         ?p            ?o                   ;
         foaf:knows    ?f1                  .
    ?o   bif:contains  "'Kjetil Kjernsmo'"  .
    ?f1  foaf:knows    ?f2
  }
```

But with the option

```
SPARQL
DEFINE input:inference "b3sifp"
SELECT COUNT (?f2)
WHERE
  {
    ?s   a             foaf:Person          ;
         ?p            ?o                   ;
         foaf:knows    ?f1                  .
    ?o   bif:contains  "'Kjetil Kjernsmo'"  .
    ?f1  foaf:knows    ?f2
  }
```

we get 4022. We note that there are many duplicates since the data is blank nodes only, with people easily represented 10 times. The context *b3sifp* simple declares that *foaf:name* and *foaf:mbox* sha1sum should be treated as inverse functional properties (IFP).

The name is not an IFP in the actual sense but treating it as such for the purposes of this one query makes sense, otherwise nothing would be found.

This option is controlled by the choice of the inference context, which is selectable in the interface discussed below.

The IFP inference can be thought of as a transparent addition of a subquery into the join sequence. The subquery joins each subject to its synonyms given by sharing IFPs. This subquery has the special property that it has the initial binding automatically in its result set. It could be expressed as:

```
SPARQL
SELECT ?f
WHERE
  {
    ?k foaf:name "Kjetil Kjernsmo"  .
    {
      SELECT ?org ?syn
      WHERE
        {
          ?org ?p ?key  .
          ?syn ?p ?key  .
          FILTER
            ( bif:rdf_is_sub
                ( "b3sifp", ?p, <b3s:any_ifp>, 3 )
              &&
              ?syn  !=  ?org
            )
        }
    }
    OPTION
      (
        TRANSITIVE ,
        T_IN (?org),
        T_OUT (?syn),
        T_MIN (0),
        T_MAX (1)
      )
    FILTER ( ?org = ?k ) .
    ?syn foaf:knows ?f .
  }
```

It is true that each subject shares IFP values with itself but the transitive construct with 0 minimum and 1 maximum depth allows passing the initial binding of *?org* directly to *?syn* , thus getting first results more rapidly. The *rdf_is_sub* function is an internal that simply tests whether *?p* is a subproperty of *b3s:any_ifp* .

Internally, the implementation has a special query operator for this and the internal form is more compact than would result from the above but the above could be used to the same effect.

Our general position is that identity criteria are highly application specific and thus we offer the full spectrum of choice between run time and precomputing. Further, weaker identity statements than sameness are difficult to use in queries, thus we prefer identity with semantics of *owl:sameAs* but make this an option that can be turned on and off query by query.

Entity Ranking

It is a common end user expectation to see text search results sorted by their relevance. The term entity rank refers to a quantity describing the relevance of a URI in an RDF graph.

This is a sample query using entity rank:

```
SPARQL
PREFIX  yago:  <http://dbpedia.org/class/yago/>
PREFIX  prop:  <http://dbpedia.org/property/>
SELECT DISTINCT ?s2 AS ?c1
WHERE
  {
    ?s1  ?s1textp      ?o1                    .
    ?o1  bif:contains  'Shakespeare'          .
    ?s1  a             yago:Writer110794014   .
    ?s2  prop:writer   ?s1
```

```
  }
ORDER BY DESC ( <LONG::IRI_RANK> (?s2) )
LIMIT 20
OFFSET 0
```

This selects works where a writer with Shakespeare in some property is the writer.

Here the query returns subjects, thus no text search summaries, so only the entity rank of the returned subject is used. We order text results by a composite of text hit score and entity rank of the RDF subject where the text occurs. The entity rank of the subject is defined by the count of references to it, weighed by the rank of the referrers and the outbound link count of referrers. Such techniques are used in text based information retrieval.

Example with Entity Ranking and Score

```
## Searching over labels, with text match
## scores and additional ranks for each
## iri / resource:

SELECT ?s ?page ?label
  ?textScore AS ?Text_Score_Rank
  ( <LONG::IRI_RANK> (?s) ) AS ?Entity_Rank
WHERE
  {
    ?s foaf:page ?page ;
     rdfs:label ?label .
    FILTER( lang( ?label ) = "en" ) .
    ?label bif:contains 'adobe and flash'
    OPTION (score ?textScore ) .
  }
```

One interesting application of entity rank and inference on IFPs and *owl:sameAs* is in locating URIs for reuse. We can easily list synonym URIs in order of popularity as well as locate URIs based on associated text. This can serve in application such as the Entity Name Server

Entity ranking is one of the few operations where we take a precomputing approach. Since a rank is calculated based on a possibly long chain of references, there is little choice but to precompute. The precomputation itself is straightforward enough: First all outbound references are counted for all subjects. Next all ranks of subjects are incremented by 1 over the referrer's outbound link count. On successive iterations, the increment is based on the rank increment the referrer received in the previous round.

The operation is easily partitioned, since each partition increments the ranks of subjects it holds. The referrers are spread throughout the cluster, though. When rank is calculated, each partition accesses every other partition. This is done with relatively long messages, referee ranks are accessed in batches of several thousand at a time, thus absorbing network latency.

On the test system, this operation performs a single pass over the corpus of 2.2 billion triples and 356 million distinct subjects in about 30 minutes. The operation has 100% utilization of all 16 cores. Adding hardware would speed it up, as would implementing it in C instead of the SQL procedures it is written in at present.

The main query in rank calculation is:

```
SPARQL
SELECT O            ,
       P            ,
       iri_rank (S)
FROM rdf_quad TABLE
OPTION (NO CLUSTER)
WHERE isiri_id(O)
ORDER BY O
```

This is the SQL cursor iterated over by each partition. The no cluster option means that only rows in this process's partition are retrieved. The RDF_QUAD table holds the RDF quads in the store, i.e., triple plus graph. The S, P, O columns are the subject, predicate, and object respectively. The graph column is not used here. The textttiri rank is a partitioned SQL function. This works by using the S argument to determine which cluster node should run the function. The specifics of the partitioning are declared elsewhere. The calls are then batched for each intended recipient and sent when the batches are full. The SQL compiler automatically generates the relevant control structures. This is like an implicit map operation in the map-reduce terminology.

An SQL procedure loops over this cursor, adds up the rank and when seeing a new O, the added rank is persisted into a table. Since links in RDF are typed, we can use the semantics of the link to determine how much rank is transferred by a reference. With extraction of named entities from text content, we can further place a given entity into a referential context and use this as a weighting factor. This is to be explored in future work. The experience thus far shows that we greatly benefit from Virtuoso being a general purpose DBMS, as we can create application specific data structures and control flows where these are efficient. For example, it would make little sense to store entity ranks as triples due to space consumption and locality considerations. With these tools, the whole ranking functionality took under a week to develop.

Note: In order to use the IRI_RANK feature you need to have the Facet (fct) vad package installed as the procedure is part of this vad.

Query Evaluation Time Limits

When scaling the Linked Data model, we have to take it as a given that the workload will be unexpected and that the query writers will often be unskilled in databases. Insofar possible, we wish to promote the forming of a culture of creative reuse of data. To this effect, even poorly formulated questions deserve an answer that is better than just timeout.

If a query produces a steady stream of results, interrupting it after a certain quota is simple. However, most interesting queries do not work in this way. They contain aggregation, sorting, maybe transitivity.

When evaluating a query with a time limit in a cluster setup, all nodes monitor the time left for the query. When dealing with a potentially partial query to begin with, there is little point in transactionality. Therefore the facet service uses read committed isolation. A read committed query will never block since it will see the before-image of any transactionally updated row. There will be no waiting for locks and timeouts can be managed locally by all servers in the cluster.

Thus, when having a partitioned count, for example, we expect all the partitions to time out around the same time and send a ready message with the timeout information to the cluster node coordinating the query. The condition raised by hitting a partial evaluation time limit differs from a run time error in that it leaves the query state intact on all participating nodes. This allows the timeout handling to come fetch any accumulated aggregates.

Let us consider the query for the top 10 classes of things with "Shakespeare" in some literal. This is typical of the workload generated by the faceted browsing web service:

```
SPARQL
DEFINE  input:inference  "yago"
SELECT ?c
       COUNT (*)
WHERE
  {
    ?s  a             ?c             ;
        ?p            ?o             .
    ?o  bif:contains  "Shakespeare"
  }
GROUP BY ?c
ORDER BY DESC 2
LIMIT 10
```

On the first execution with an entirely cold cache, this times out after 2 seconds and returns:

```
?c                                      COUNT (*)
yago:class/yago/Entity100001740         566
yago:class/yago/PhysicalEntity100001930 452
yago:class/yago/Object100002684         452
yago:class/yago/Whole100003553          449
yago:class/yago/Organism100004475       375
yago:class/yago/LivingThing100004258    375
yago:class/yago/CausalAgent100007347    373
yago:class/yago/Person100007846         373
yago:class/yago/Abstraction100002137    150
yago:class/yago/Communicator109610660   125
```

The next repeat gets about double the counts, starting with 1291 entities.

With a warm cache, the query finishes in about 300 ms (4 core Xeon, Virtuoso 6 Cluster) and returns:

```
?c                                        COUNT (*)
yago:class/yago/Entity100001740           13329
yago:class/yago/PhysicalEntity100001930   10423
yago:class/yago/Object100002684           10408
yago:class/yago/Whole100003553            10210
yago:class/yago/LivingThing100004258       8868
yago:class/yago/Organism100004475          8868
yago:class/yago/CausalAgent100007347       8853
yago:class/yago/Person100007846            8853
yago:class/yago/Abstraction100002137       3284
yago:class/yago/Entertainer109616922       2356
```

It is a well known fact that running from memory is thousands of times faster than from disk.

The query plan begins with the text search. The subjects with "Shakespeare" in some property get dispatched to the partition that holds their class. Since all partitions know the class hierarchy, the superclass inference runs in parallel, as does the aggregation of the group by. When all partitions have finished, the process coordinating the query fetches the partial aggregates, adds them up and sorts them by count.

If a timeout occurs, it will most likely occur where the classes of the text matches are being retrieved. When this happens, this part of the query is reset, but the aggregate states are left in place. The process coordinating the query then goes on as if the aggregates had completed. If there are many levels of nested aggregates, each timeout terminates the innermost aggregation that is still accumulating results, thus a query is guaranteed to return in no more than n timeouts, where n is the number of nested aggregations or subqueries.

Faceted Web Service and Linked Data

The Virtuoso Faceted Web Service is a general purpose RDF query facility for Faceted based browsing. It takes an XML description of the view desired and generates the reply as an XML tree containing the requested data. The user agent or a local web page can use XSLT for rendering this for the end user. The selection of facets and values is represented as an XML tree. The rationale for this is the fact that such a representation is easier to process in an application than the SPARQL source text or a parse tree of SPARQL and more compactly captures the specific subset of SPARQL needed for faceted browsing. All such queries internally generate SPARQL and the SPARQL generated is returned with the results. One can therefore use this is a starting point for hand crafted queries.

The query has the top level element. The child elements of this represents conditions pertaining to a single subject. A join is expressed with the property or propertyof element. This has in turn children which state conditions on a property of the first subject. Property and propertyof elements can be nested to an arbitrary depth and many can occur inside one containing element. In this way, tree-shaped structures of joins can be expressed.

Expressing more complex relationships, such as intermediate grouping, subqueries, arithmetic or such requires writing the query in SPARQL. The XML format is for easy automatic composition of queries needed for showing facets, not a replacement for SPARQL.

Consider composing a map of locations involved with Napoleon. Below we list user actions and the resulting XML query descriptions.

- Enter in the search form "Napoleon":

```
<query inference="" same-as="" view3="" s-term="e" c-term="type">
  <text>napoleon</text>
  <view type="text" limit="20" offset="" />
</query>
```

- Select the "types" view:

```
<query inference="" same-as="" view3="" s-term="e" c-term="type">
  <text>napoleon</text>
  <view type="classes" limit="20" offset="0" location-prop="0" />
</query>
```

- Choose "MilitaryConflict" type:

```
<query inference="" same-as="" view3="" s-term="e" c-term="type">
  <text>napoleon</text>
  <view type="classes" limit="20" offset="0" location-prop="0" />
  <class iri="yago:ontology/MilitaryConflict" />
```

```
</query>
```

- Choose "NapoleonicWars":

```
<query inference="" same-as="" view3="" s-term="e" c-term="type">
  <text>napoleon</text>
  <view type="classes" limit="20" offset="0" location-prop="0" />
  <class iri="yago:ontology/MilitaryConflict" />
  <class iri="yago:class/yago/NapoleonicWars" />
</query>
```

- Select "any location" in the select list beside the "map" link; then hit "map" link:

```
<query inference="" same-as="" view3="" s-term="e" c-term="type">
  <text>napoleon</text>
  <class iri="yago:ontology/MilitaryConflict" />
  <class iri="yago:class/yago/NapoleonicWars" />
  <view type="geo" limit="20" offset="0" location-prop="any" />
</query>
```

This last XML fragment corresponds to the below text of SPARQL query:

```
SPARQL
SELECT ?location AS ?c1
       ?lat1     AS ?c2
       ?lng1     AS ?c3
WHERE
  {
    ?s1        ?s1textp  ?o1                              .
    FILTER
      ( bif:contains (?o1, '"Napoleon"') )  .
    ?s1        a         <yago:ontology/MilitaryConflict>  .
    ?s1        a         <yago:class/yago/NapoleonicWars>  .
    ?s1        ?anyloc   ?location                         .
    ?location  geo:lat   ?lat1                             ;
               geo:long  ?lng1
  }
LIMIT 200
OFFSET 0
```

The query takes all subjects with some literal property with "Napoleon" in it, then filters for military conflicts and Napoleonic wars, then takes all objects related to these where the related object has a location. The map has the objects and their locations.

See Also:

- Virtuoso Faceted Web Service
- Virtuoso APIs for Faceted REST services

voiD Discoverability

A long awaited addition to the LOD cloud is the Vocabulary of Interlinked Data (voiD). Virtuoso automatically generates voiD descriptions of data sets it hosts. Virtuoso incorporates an SQL function *rdf_void_gen* which returns a Turtle representation of a given graph's voiD statistics.

Test System and Data

The test system consists of two 2x4 core Xeon 5345, 2.33 GHz servers with 16G RAM and 4 disks each. The machines are connected by two 1Gbit Ethernet connections. The software is Virtuoso 6 Cluster. The Virtuoso server is split into 16 partitions, 8 for each machine. Each partition is managed by a separate server process.

The test database has the following data sets:

- DBpedia 3.2
- MusicBrainz
- Bio2RDF
- NeuroCommons
- UniProt

- Freebase (95M triples)
- PingTheSemanticWeb (1.6M miscellaneous files from http://www.pingthesemanticweb.com/).

Ontologies:

- Yago
- OpenCyc
- Umbel
- DBpedia

The database is 2.2 billion triples with 356 million distinct URIs.

See Also:

- Virtuoso Faceted Browser Installation and configuration

16.14. Inference Rules & Reasoning

16.14.1. Introduction

Virtuoso SPARQL can use an inference context for inferring triples that are not physically stored. This functionality applies to physically stored quads and not to virtual triples generated from relational data with Linked Data Views. Such an inference context can be built from one or more graphs containing RDF Schema triples. The supported RDF Schema or OWL constraints are imported from these graphs and are grouped together into rule bases. A rule base is a persistent entity that can be referenced by a SPARQL query or end point. Queries running with a given rule base work as if the triples asserted by this rule base were included in the graph or graphs accessed by the query.

As of version 5.0, Virtuoso recognizes *rdfs:subClassOf* and *rdfs:subPropertyOf* . owl:sameAs is considered for arbitrary subjects and objects if specially enabled by a pragma in the query. As of 5.00.3031, owl:sameAs, owl:equivalentClass and owl:equivalentProperty are also considered when determining subclass or subproperty relations. If two classes are equivalent, they share all instances, subclasses and superclasses directly or indirectly stated in the data for either class. Other RDF Schema or OWL information is not taken into account.

16.14.2. Making Rule Sets

Since RDF Schema and OWL schemas are RDF graphs, these can be loaded into the triple store. Thus, in order to use such a schema as query context, one first loads the corresponding document into the triple store using ttlp() or rdf_load_rdfxml() or related functions. After the schema document is loaded, one can add the assertions there into an inference context with the rdfs_rule_set() function. This function specifies a logical name for the rule set plus a graph URI. It is possible to combine multiple schema graphs into a single rule set. A single schema graph may also independently participate in multiple rule sets.

The *DB.DBA.SYS_RDF_SCHEMA* table contains information for all RDF rule sets in a Virtuoso instance. This table may be queried to, for instance, verify rdfs_rule_set() activity:

```
CREATE TABLE DB.DBA.SYS_RDF_SCHEMA (
  RS_NAME VARCHAR,  -- The name of the rdf rule set
  RS_URI  VARCHAR,  -- The name of the graph
  RS_G    VARCHAR,  -- Column for system usage only
  PRIMARY KEY (RS_NAME, RS_URI))
)
```

16.14.3. Changing Rule Sets

Changing a rule set affects queries made after the change. Some queries may have been previously compiled and will not be changed as a result of modifying the rule set. When a rule set is changed, i.e. when *rdfs_rule_set* is called with the first argument set to a pre-existing rule set's name, all the graphs associated with this name are read and the relevant facts are added to a new empty rule set. Thus, if triples are deleted from or added to the graphs comprising the rule set, calling *rdfs_rule_set* will refresh the rule set to correspond to the state of the stored graphs.

16.14.4. Subclasses and Subproperties

Virtuoso SPARQL supports RDF Schema subclasses and subproperties.

The predicates *rdfs:subClassOf* and *rdfs:subPropertyOf* are recognized when they appear in graphs included in a rule set. When such a rule set is specified as a context for a SPARQL query, the following extra triples are generated as needed.

For every *?s rdf:type ?class* , a triple *?s rdf:type ?superclass* is considered to exist, such that *?superclass* is a direct or indirect superclass of *?class* . Direct superclasses are declared with the *rdfs:subClassOf* predicate in the rule set graph. Transitivity of superclasses is automatically taken into account, meaning that if a is a superclass of b and b a superclass of c, then a is a superclass of c also. Cyclic superclass relations are not allowed. If such occur in the rule set data, the behavior is undefined but will not involve unterminating recursion.

For every *?s ?subpredicate ?o* , a triple *?s ?superpredicate ?o* is considered to exist if the rule context declares *?superpredicate* to be a superpredicate of *?predicate* . This is done by having the triple *?subpredicate rdfs:subPropertyOf ?superpredicate* as part of the graphs making up the rule context. Transitivity is observed, thus if a is a subpredicate of b and b a subpredicate of c, then a is also a subpredicate of c.

Two methods can be used for typical recursions, transitivity on inference and plain transitive patterns (or subqueries).

The advantage of inference is that queries are short and one inference rule set may be maintained for numerous queries.

If queries are about trees of classes or properties, or about equivalences of nodes, consider using inference rule sets.

Transitive patterns are inconvenient and may easily result in queries that runs too long or hard to debug, but they're unavoidable in traversing social networks or plain querying of RDF lists.

So consider a rule set, a handful of nodes with classes from the rule set and a couple of RDF Lisp-style lists defined on demo.openlinksw.com:

```
SQL> SPARQL CLEAR GRAPH <http://example.com/2/owl>;
callret-0
VARCHAR
_______________________________________________________________________________

Clear <http://example.com/2/owl>  -- done

1 Rows. -- 0 msec.

SQL> TTLP (' @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>  .
  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>  .
  @prefix owl: <http://www.w3.org/2002/07/owl#>  .
  @prefix e: <http://example.com/e/>  .
  e:c1 rdfs:subClassOf e:c1or2 .
  e:c2 rdfs:subClassOf e:c1or2 .
  e:c1-10 rdfs:subClassOf e:c1 .
  e:c1-20 rdfs:subClassOf e:c1 .
  e:c2-30 rdfs:subClassOf e:c2 .
  e:c2-40 rdfs:subClassOf e:c2 .
  ', 'http://example.com/2/owl', 'http://example.com/2/owl' );

Done. -- 0 msec.
```

You can also use the SPARUL equivalent variant:

```
SPARQL
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX e: <http://example.com/e/>

INSERT IN GRAPH <http://example.com/2/owl>
  {
    e:c1 rdfs:subClassOf e:c1or2 .
    e:c2 rdfs:subClassOf e:c1or2 .
    e:c1-10 rdfs:subClassOf e:c1 .
    e:c1-20 rdfs:subClassOf e:c1 .
```

```
    e:c2-30 rdfs:subClassOf e:c2 .
    e:c2-40 rdfs:subClassOf e:c2 .
  } ;
```

Define the inference rule:

```
SQL> rdfs_rule_set ('http://example.com/2/owl', 'http://example.com/2/owl');

Done. -- 0 msec.

SQL> SPARQL CLEAR GRAPH <http://example.com/2/data> ;
callret-0
VARCHAR
_______________________________________________________________________________

Clear <http://example.com/2/data>  -- done

1 Rows. -- 0 msec.

SQL>  TTLP ('
  @prefix e: <http://example.com/e/>  .
  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>  .
  @prefix owl: <http://www.w3.org/2002/07/owl#>  .
  e:s1 a e:c1 ; e:p1 "Value of p1 for s1" .
  e:s2 a e:c2 ; e:p1 "Value of p1 for s2" .
  e:s1-10 a e:c1-10 ; e:p1 "Value of p1 for s1-10" .
  e:s1-20 a e:c1-20 ; e:p1 "Value of p1 for s1-20" .
  e:s2-30 a e:c2-30 ; e:p1 "Value of p1 for s2-30" .
  e:s2-40 a e:c2-40 ; e:p1 "Value of p1 for s2-40" .
  e:lists
       rdf:_1 ( e:list1-item1 e:list1-item2 e:list1-item3 ) ;
       rdf:_2 (
           [ e:p2 "Value of p2 of item1 of list2" ; e:p3 "Value of p3 of item1 of list2" ]
           [ e:p2 "Value of p2 of item2 of list2" ; e:p3 "Value of p3 of item2 of list2" ]
           [ e:p2 "Value of p2 of item3 of list2" ; e:p3 "Value of p3 of item3 of list2" ] ) .
  ', 'http://example.com/2/data', 'http://example.com/2/data' );

Done. -- 0 msec.
```

You can also use the SPARUL equivalent variant:

```
SPARQL
PREFIX e: <http://example.com/e/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

INSERT IN GRAPH <http://example.com/2/data>
  {
    e:s1 a e:c1 ; e:p1 "Value of p1 for s1" .
    e:s2 a e:c2 ; e:p1 "Value of p1 for s2" .
    e:s1-10 a e:c1-10 ; e:p1 "Value of p1 for s1-10" .
    e:s1-20 a e:c1-20 ; e:p1 "Value of p1 for s1-20" .
    e:s2-30 a e:c2-30 ; e:p1 "Value of p1 for s2-30" .
    e:s2-40 a e:c2-40 ; e:p1 "Value of p1 for s2-40" .
    e:lists
      rdf:_1 ( e:list1-item1 e:list1-item2 e:list1-item3 ) ;
      rdf:_2 (
           [ e:p2 "Value of p2 of item1 of list2" ; e:p3 "Value of p3 of item1 of list2" ]
           [ e:p2 "Value of p2 of item2 of list2" ; e:p3 "Value of p3 of item2 of list2" ]
           [ e:p2 "Value of p2 of item3 of list2" ; e:p3 "Value of p3 of item3 of list2" ] )
  };
```

SPARQL DESCRIBE works fine with inference, deriving additional type information:

```
DEFINE input:inference <http://example.com/2/owl>
DESCRIBE <http://example.com/e/s1>
FROM <http://example.com/2/data>

fmtaggret-
LONG VARCHAR
_______________________________________________________________________________
```

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>  .
@prefix ns1: <http://example.com/e/>  .
ns1:s1 rdf:type ns1:c1or2 ,
ns1:c1 ;
ns1:p1 "Value of p1 for s1" .

1 Rows. -- 0 msec.
```

Example links against Virtuoso Demo Server SPARQL Endpoint with SPARQL Protocol URLs:

- View results page
- View editor page

```
DEFINE input:inference <http://example.com/2/owl>
DESCRIBE <http://example.com/e/s2>
FROM <http://example.com/2/data>
fmtaggret-
LONG VARCHAR
_____________________________________________________________________________

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>  .
@prefix ns1: <http://example.com/e/>  .
ns1:s2 rdf:type ns1:c1or2 ,
ns1:c2 ;
ns1:p1 "Value of p1 for s2" .

1 Rows. -- 0 msec.
```

Example links against Virtuoso Demo Server SPARQL Endpoint with SPARQL Protocol URLs:

- View results page
- View editor page

Querying is simple as well:

```
SQL>SPARQL DEFINE input:inference <http://example.com/2/owl>
PREFIX e:<http://example.com/e/>
SELECT *
FROM <http://example.com/2/data>
WHERE
  {
    ?s a e:c1or2 ;
          e:p1 ?o
  }

s                           o
VARCHAR                     VARCHAR
_____________________________

http://example.com/e/s1     Value of p1 for s1
http://example.com/e/s1-10  Value of p1 for s1-10
http://example.com/e/s1-20  Value of p1 for s1-20
http://example.com/e/s2-30  Value of p1 for s2-30
http://example.com/e/s2-40  Value of p1 for s2-40
http://example.com/e/s2     Value of p1 for s2

6 Rows. -- 0 msec.
```

Example links against Virtuoso Demo Server SPARQL Endpoint with SPARQL Protocol URLs:

- View results page
- View editor page

```
SQL>SPARQL DEFINE input:inference <http://example.com/2/owl>
PREFIX e:<http://example.com/e/>
SELECT * FROM <http://example.com/2/data>
WHERE
  {
    ?s a e:c1 ;
```

```
        e:p1 ?o
  }

s                           o
VARCHAR                     VARCHAR
_______________________________

http://example.com/e/s1     Value of p1 for s1
http://example.com/e/s1-10  Value of p1 for s1-10
http://example.com/e/s1-20  Value of p1 for s1-20

3 Rows. -- 0 msec.
```

Example links against Virtuoso Demo Server SPARQL Endpoint with SPARQL Protocol URLs:

- View results page
- View editor page

However you should care about duplicates if both types and properties are queried: the join will result in all combinations of types and property values.

```
SQL>SPARQL DEFINE input:inference <http://example.com/2/owl>
PREFIX e:<http://example.com/e/>
SELECT * FROM <http://example.com/2/data>
WHERE
  {
    ?s a ?t ;
     e:p1 ?o
  }

s                           t                           o
VARCHAR                     VARCHAR                     VARCHAR
_______________________________

http://example.com/e/s1     http://example.com/e/c1     Value of p1 for s1
http://example.com/e/s1     http://example.com/e/c1or2  Value of p1 for s1
http://example.com/e/s1-10  http://example.com/e/c1-10  Value of p1 for s1-10
http://example.com/e/s1-10  http://example.com/e/c1     Value of p1 for s1-10
http://example.com/e/s1-10  http://example.com/e/c1or2  Value of p1 for s1-10
http://example.com/e/s1-20  http://example.com/e/c1-20  Value of p1 for s1-20
http://example.com/e/s1-20  http://example.com/e/c1     Value of p1 for s1-20
http://example.com/e/s1-20  http://example.com/e/c1or2  Value of p1 for s1-20
http://example.com/e/s2-30  http://example.com/e/c2-30  Value of p1 for s2-30
http://example.com/e/s2-30  http://example.com/e/c2     Value of p1 for s2-30
http://example.com/e/s2-30  http://example.com/e/c1or2  Value of p1 for s2-30
http://example.com/e/s2-40  http://example.com/e/c2-40  Value of p1 for s2-40
http://example.com/e/s2-40  http://example.com/e/c2     Value of p1 for s2-40
http://example.com/e/s2-40  http://example.com/e/c1or2  Value of p1 for s2-40
http://example.com/e/s2     http://example.com/e/c2     Value of p1 for s2
http://example.com/e/s2     http://example.com/e/c1or2  Value of p1 for s2

16 Rows. -- 0 msec.
```

Example links against Virtuoso Demo Server SPARQL Endpoint with SPARQL Protocol URLs:

- View results page
- View editor page

Transitive queries are convenient as SPARQL 1.1 "predicate+" equivalent. The equivalent of "predicate*" requires the use of a union:

```
SQL>SPARQL PREFIX e:<http://example.com/e/>
SELECT ?item
FROM <http://example.com/2/data>
WHERE
  {
    {
      ?lists rdf:_1 ?node
    }
    UNION
```

```
    {
      ?lists rdf:_1 ?l .
      ?l rdf:rest ?node option (transitive) .
    }
    ?node rdf:first ?item
  }

item
VARCHAR
_______________________________________________________________________________

http://example.com/e/list1-item1
http://example.com/e/list1-item2
http://example.com/e/list1-item3

3 Rows. -- 0 msec.
```

Example links against Virtuoso Demo Server SPARQL Endpoint with SPARQL Protocol URLs:

- View results page
- View editor page

```
SQL> SPARQL PREFIX e:<http://example.com/e/>
SELECT ?p ?o
FROM <http://example.com/2/data>
WHERE
  {
    {
      ?lists rdf:_2 ?node
    }
    UNION
    {
      ?lists rdf:_2 ?l .
      ?l rdf:rest ?node option (transitive) .
    }
    ?node rdf:first ?item .
    ?item ?p ?o
  }

p                         o
VARCHAR                   VARCHAR
_________________________

http://example.com/e/p2   Value of p2 of item1 of list2
http://example.com/e/p3   Value of p3 of item1 of list2
http://example.com/e/p2   Value of p2 of item2 of list2
http://example.com/e/p3   Value of p3 of item2 of list2
http://example.com/e/p2   Value of p2 of item3 of list2
http://example.com/e/p3   Value of p3 of item3 of list2

6 Rows. -- 0 msec.
```

Example links against Virtuoso Demo Server SPARQL Endpoint with SPARQL Protocol URLs:

- View results page
- View editor page

Note that the result set can be in order of items in the list, but it don't have to. If the order should be preserved, then fix the direction of transitive scan, get step number as a variable, order by that variable.

```
-- Line 82:
SQL> SPARQL PREFIX e:<http://example.com/e/>
SELECT ?p ?o bif:coalesce(?step_no, 0)
FROM <http://example.com/2/data>
WHERE
  {
    {
      ?lists rdf:_2 ?node
    }
    UNION
```

```
    {
      ?lists rdf:_2 ?l .
      ?l rdf:rest ?node OPTION (transitive, t_direction 1, t_step("step_no") as ?step_no) .
    }
    ?node rdf:first ?item .
    ?item ?p ?o
  }
ORDER BY ASC (?step_no)

p                        o                              callret-2
VARCHAR                  VARCHAR                        VARCHAR
_________________________

http://example.com/e/p2   Value of p2 of item1 of list2   0
http://example.com/e/p3   Value of p3 of item1 of list2   0
http://example.com/e/p2   Value of p2 of item2 of list2   1
http://example.com/e/p3   Value of p3 of item2 of list2   1
http://example.com/e/p2   Value of p2 of item3 of list2   2
http://example.com/e/p3   Value of p3 of item3 of list2   2

6 Rows. -- 7 msec.
```

Example links against Virtuoso Demo Server SPARQL Endpoint with SPARQL Protocol URLs:

- View results page
- View editor page

16.14.5. OWL sameAs Support

Virtuoso has limited support for the OWL sameAs predicate.

If sameAs traversal is enabled and a triple pattern with a given subject or object is being matched, all the synonyms of the S and O will be tried and results generated for all the tried bindings of S and O. The set of synonyms is generated at run time by following all owl:sameAs triples where the IRI in question is either the subject or the object. These are followed recursively from object to subject and subject to object until the complete transitive closure is generated. All sameAs triples from all the graphs applicable to instantiating the triple pattern at hand are considered.

Thus for example:

The inital SPARQL query:

```
SQL>SPARQL
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix sioc: <http://rdfs.org/sioc/ns#>
SELECT *
from <http://example.com/dataspace>
where
 {
    ?person a foaf:Person FILTER REGEX(?person ,"http://example.com/dataspace/person/kidehen#this").
    ?person foaf:name ?name .
    ?person owl:sameAs ?sameas .
  }
limit 8;
person                                              name             sameas
VARCHAR                                             VARCHAR          VARCHAR
_____________________________________________________________________________

http://example.com/dataspace/person/kidehen#this  Kingsley Idehen  http://www.openlinksw.com/dataspace/pe
http://example.com/dataspace/person/kidehen#this  Kingsley Idehen  http://my.openlinksw.com/dataspace/per
http://example.com/dataspace/person/kidehen#this  Kingsley Idehen  http://kidehen.idehen.net/dataspace/pe
http://example.com/dataspace/person/kidehen#this  Kingsley Idehen  http://qdos.com/user/e922b748a2eb667bf
http://example.com/dataspace/person/kidehen#this  Kingsley Idehen  http://knowee.net/kidehen/ids/id368497
http://example.com/dataspace/person/kidehen#this  Kingsley Idehen  http://dbpedia.org/resource/Kingsley_I
http://example.com/dataspace/person/kidehen#this  Kingsley Idehen  http://identi.ca/user/14092
http://example.com/dataspace/person/kidehen#this  Kingsley Idehen  http://myopenlink.net/proxy?url=http%3

8 Rows. -- 181 msec.
```

So if we have:

```
<http://example.com/dataspace/person/kidehen#this>      <http://www.w3.org/2002/07/owl#sameAs> <http://ww
<http://example.com/dataspace/person/kidehen#this>      <http://xmlns.com/foaf/0.1/name>           Kingsl
```

and we instantiate *?s <http://xmlns.com/foaf/0.1/name> "Kingsley Idehen"* we get *?s* bound to *<http://example.com/dataspace/person/kidehen#this>* .

If we instantiate *<http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#this> <http://xmlns.com/foaf/0.1/name> ?l* we get *?l* bound to *"Kingsley Idehen"* because the subject was given and it was expanded to its synonyms.

If binding a variable in a pattern where the variable was free, we do not expand the value to the complete set of its synonyms.

Same-as expansion is enabled in a query by *define input:same-as "yes"* in the beginning of the SPARQL query. This has a significant run time cost but is in some cases useful when joining data between sets which are mapped to each other with same-as.

We note that the number of same-as expansions will depend on the join order used for the SPARQL query. The compiler does not know the number of synonyms and cannot set the join order accordingly. Regardless of the join order we will however get at least one IRI of the each synonym set as answer. Also when interactively navigating a graph with a browser, the same-as expansion will take all synonyms into account.

For getting the complete entailment of same-as, a forward chaining approach should be used, effectively asserting all the implied triples.

OWL sameAs Example

```
SQL>SPARQL
DEFINE input:same-as "yes"
SELECT *
WHERE
 {
   ?s <http://xmlns.com/foaf/0.1/name> "Kingsley Idehen" .
 }
LIMIT 10;

s
VARCHAR
_______________________________________________________
http://example.com/dataspace/person/kidehen#this
http://example.com/dataspace/person/kidehen#this
http://example.com/dataspace/person/kidehen#this
http://example.com/dataspace/person/kidehen#this
http://example.com/dataspace/person/kidehen#this
http://example.com/dataspace/person/kidehen#this
http://demo.openlinksw.com/dataspace/kingsley#person
http://example.com/dataspace/person/kidehen#this
http://example.com/dataspace/person/kidehen#this
http://example.com/dataspace/person/kidehen#this
No. of rows in result: 10
```

16.14.6. Implementation

Triples entailed by subclass or subproperty statements in an inference context are not physically stored. Such triples are added to the result set by the query run time as needed. Also queries involving subclass or subproperty rules are not rewritten into unions of all the possible triple patterns that might imply the pattern that is requested. Instead, the SQL compiler adds special nodes that iterate over subclasses or subproperties at run time. The cost model also takes subclasses and subproperties into account when determining the approximate cardinality of triple patterns.

In essence, Virtuoso's support for subclasses and subproperties is backward chaining, i.e. it does not materialize all implied triples but rather looks for the basic facts implying these triples at query evaluation time.

16.14.7. Enabling Inferencing

In a SPARQL query, the define input:inference clause is used to instruct the compiler to use the rules in the named rule set. For example:

```
SQL> rdfs_rule_set ('sample', 'rule_graph');

SQL> SPARQL
define input:inference "sample"
SELECT *
FROM <g>
WHERE {?s ?p ?o};
```

will include all the implied triples in the result set, using the rules in the sample rule set.

Inference can be enabled triple pattern by triple pattern. This is done with the option (inference 'rule_set') clause after the triple pattern concerned. Specifying option (inference none) will disable inference for the pattern concerned while the default inference context applies to the rest of the patterns. Note that the keyword is input:inference in the query header and simply inference in the option clause. See the examples section below for examples.

In SQL, if RDF_QUAD occurs in a select from clause, inference can be added with the table option *WITH* , as follows:

```
SPARQL
SELECT *
FROM rdf_quad table OPTION (with 'sample')
WHERE g = iri_to_id ('xx', 0);
```

This is about the same as:

```
SPARQL
define input:inference "sample"
SELECT *
FROM <xx>
WHERE {?s ?p ?o}
```

16.14.8. Examples

Example for loading data space instance data Triples into a Named Graph for schema/ontology data

The following example shows how to load data space instance data Triples into a Named Graph: <http://example.com/test>, for schema/ontology data called: <http://example.com/schema/test> that expresses assertions about subclasses and subproperties.

```
ttlp ('
   <http://example.com/dataspace>                                <http://www.w3.org/1999/02/22-rdf-syntax-
   <http://example.com/dataspace/test2/weblog/test2tWeblog>  <http://www.w3.org/1999/02/22-rdf-syntax-ns#
   <http://example.com/dataspace/discussion/oWiki-test1Wiki> <http://www.w3.org/1999/02/22-rdf-syntax-ns#
   <http://example.com/dataspace>                               <http://rdfs.org/sioc/ns#link>
   <http://example.com/dataspace/test2/weblog/test2tWeblog>  <http://rdfs.org/sioc/ns#link>
   <http://example.com/dataspace/discussion/oWiki-test1Wiki> <http://rdfs.org/sioc/ns#link>
   ', '', 'http://example.com/test');

ttlp (' @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
  <http://rdfs.org/sioc/ns#Space> rdfs:subClassOf <http://www.w3.org/2000/01/rdf-schema#Resource> .
  <http://rdfs.org/sioc/ns#Container> rdfs:subClassOf <http://rdfs.org/sioc/ns#Space> .
  <http://rdfs.org/sioc/ns#Forum> rdfs:subClassOf <http://rdfs.org/sioc/ns#Container> .
  <http://rdfs.org/sioc/types#Weblog> rdfs:subClassOf <http://rdfs.org/sioc/ns#Forum> .
  <http://rdfs.org/sioc/types#MessageBoard> rdfs:subClassOf <http://rdfs.org/sioc/ns#Forum> .
  <http://rdfs.org/sioc/ns#link> rdfs:subPropertyOf <http://rdfs.org/sioc/ns> .
  ', '', 'http://example.com/schema/test');

rdfs_rule_set ('http://example.com/schema/property_rules1', 'http://example.com/schema/test');
```

This defines the rule context http://example.com/schema/property_rules1 that is initialized from the contents of graph http://example.com/schema/test.

```
SQL>SPARQL
define input:inference "http://example.com/schema/property_rules1"
```

```
SELECT ?s
FROM <http://example.com/test>
WHERE {?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>  <http://rdfs.org/sioc/ns#Space> };
s
VARCHAR
_______________________________________________________________________________

http://example.com/dataspace/test2/weblog/test2tWeblog
http://example.com/dataspace/discussion/oWiki-test1Wiki
http://example.com/dataspace

3 Rows. -- 0 msec.
```

This returns the instances of http://rdfs.org/sioc/ns#Space. Since http://rdfs.org/sioc/types#Weblog and http://rdfs.org/sioc/types#MessageBoard are subclasses of http://rdfs.org/sioc/ns#Space, instances of http://rdfs.org/sioc/ns#Space, http://rdfs.org/sioc/types#Weblog and http://rdfs.org/sioc/types#MessageBoard are all returned. This results in the subjects http://example.com/dataspace, http://example.com/dataspace/test2/weblog/test2tWeblog and http://example.com/dataspace/discussion/oWiki-test1Wiki.

```
SQL>SELECT id_to_iri (s)
FROM rdf_quad table option (with 'http://example.com/schema/property_rules1')
WHERE g = iri_to_id ('http://example.com/test',0)
  AND p = iri_to_id ('http://www.w3.org/1999/02/22-rdf-syntax-ns#type', 0)
  AND o = iri_to_id ('http://rdfs.org/sioc/ns#Space', 0);
callret
VARCHAR
_______________________________________________________________________________

http://example.com/dataspace/test2/weblog/test2tWeblog
http://example.com/dataspace/discussion/oWiki-test1Wiki
http://example.com/dataspace

3 Rows. -- 10 msec.
```

This is the corresponding SQL query, internally generated by the SPARQL query.

Below we first look for all instances of http://rdfs.org/sioc/ns#Space with some property set to http://example.com/dataspace/test2/weblog/test2tWeblog. We get the subject http://example.com/dataspace/test2/weblog/test2tWeblog and the properties http://rdfs.org/sioc/ns#link and http://rdfs.org/sioc/ns. The join involves both subclass and subproperty inference. Then we turn off the inference for the second pattern and only get the property http://rdfs.org/sioc/ns#link. Then we do the same but now specify that inference should apply only to the first triple pattern.

```
SQL>SPARQL
define input:inference  "http://example.com/schema/property_rules1"
SELECT *
FROM <http://example.com/test>
WHERE
  {
    ?s ?p <http://rdfs.org/sioc/ns#Space> .
    ?s ?p1 <http://example.com/dataspace/test2/weblog/test2tWeblog> .
  };

s             p              p1
VARCHAR       VARCHAR       VARCHAR
_______________________________________________________________________________

http://example.com/dataspace/test2/weblog/test2tWeblog  http://www.w3.org/1999/02/22-rdf-syntax-ns#type h
http://example.com/dataspace/test2/weblog/test2tWeblog  http://www.w3.org/1999/02/22-rdf-syntax-ns#type h

2 Rows. -- 0 msec.

SQL>SPARQL
SELECT *
FROM <http://example.com/test>
WHERE
  {
    ?s ?p <http://rdfs.org/sioc/ns#Space> OPTION (inference 'http://example.com/schema/property_rules1')
    ?s ?p1 <http://example.com/dataspace/test2/weblog/test2tWeblog> .
  };
```

```
s               p               p1
VARCHAR         VARCHAR         VARCHAR
_______________________________________________________________________________

http://example.com/dataspace/test2/weblog/test2tWeblog  http://www.w3.org/1999/02/22-rdf-syntax-ns#type h

1 Rows. -- 10 msec.
```

DBpedia example

```
ttlp ('
 prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
 <http://dbpedia.org/property/birthcity> rdfs:subPropertyOf <http://dbpedia.org/property/birthPlace> .
 <http://dbpedia.org/property/birthcountry> rdfs:subPropertyOf  <http://dbpedia.org/property/birthPlace>
 <http://dbpedia.org/property/cityofbirth> rdfs:subPropertyOf <http://dbpedia.org/property/birthPlace> .
 <http://dbpedia.org/property/countryofbirth> rdfs:subPropertyOf <http://dbpedia.org/property/birthPlace>
 <http://dbpedia.org/property/countyofbirth> rdfs:subPropertyOf <http://dbpedia.org/property/birthPlace>
 <http://dbpedia.org/property/cityofdeath> rdfs:subPropertyOf <http://dbpedia.org/property/deathPlace> .
 <http://dbpedia.org/property/countryofdeath> rdfs:subPropertyOf <http://dbpedia.org/property/deathPlace>
 'http://dbpedia.org/inference/rules#') ;

rdfs_rule_set ('http://dbpedia.org/schema/property_rules1', 'http://dbpedia.org/inference/rules#');

SQL>SPARQL
define input:inference "http://dbpedia.org/schema/property_rules1"
prefix p: <http://dbpedia.org/property/>
SELECT ?s
FROM <http://dbpedia.org>
WHERE {?s p:birthcity ?o }
LIMIT 50
s
VARCHAR
_______________________________________________________________________________

http://dbpedia.org/resource/Britt_Janyk
http://dbpedia.org/resource/Chiara_Costazza
http://dbpedia.org/resource/Christoph_Gruber
http://dbpedia.org/resource/Daron_Rahlves
http://dbpedia.org/resource/Finlay_Mickel
http://dbpedia.org/resource/Genevi%C3%A8ve_Simard
http://dbpedia.org/resource/Johann_Grugger
http://dbpedia.org/resource/Kalle_Palander
http://dbpedia.org/resource/Marc_Gini
http://dbpedia.org/resource/Mario_Scheiber
http://dbpedia.org/resource/Prince_Hubertus_of_Hohenlohe-Langenburg
http://dbpedia.org/resource/Resi_Stiegler
http://dbpedia.org/resource/Steven_Nyman
http://dbpedia.org/resource/Hannes_Reichelt
http://dbpedia.org/resource/Jeremy_Transue

15 Rows. -- 167 msec.

SQL>SPARQL
define input:inference "http://dbpedia.org/schema/property_rules1"
prefix p: <http://dbpedia.org/property/>
SELECT ?s
FROM <http://dbpedia.org>
WHERE {?s p:countryofbirth ?o }
LIMIT 50
s
VARCHAR
_______________________________________________________________________________

http://dbpedia.org/resource/A._J._Wood
http://dbpedia.org/resource/A._J._Godbolt
http://dbpedia.org/resource/Ac%C3%A1cio_Casimiro
http://dbpedia.org/resource/Adam_Fry
http://dbpedia.org/resource/Adam_Gilchrist
http://dbpedia.org/resource/Adam_Griffin
http://dbpedia.org/resource/Adam_Gross
...
```

```
50 Rows. -- 324 msec.

SQL>SPARQL
define input:inference "http://dbpedia.org/schema/property_rules1"
prefix p: <http://dbpedia.org/property/>
SELECT ?s
FROM <http://dbpedia.org>
WHERE {?s p:countyofbirth ?o }
LIMIT 50

s
VARCHAR
_______________________________________________________________________________

http://dbpedia.org/resource/Eddie_Colman

1 Rows. -- 163 msec.

SQL>SPARQL
define input:inference "http://dbpedia.org/schema/property_rules1"
prefix p: <http://dbpedia.org/property/>
SELECT ?s
FROM <http://dbpedia.org>
WHERE {?s p:birthPlace ?o }

s
VARCHAR
_______________________________________________________________________________

http://dbpedia.org/resource/Eddie_Colman
http://dbpedia.org/resource/Jeremy_Transue
http://dbpedia.org/resource/Finlay_Mickel
http://dbpedia.org/resource/Prince_Hubertus_of_Hohenlohe-Langenburg
http://dbpedia.org/resource/Hannes_Reichelt
http://dbpedia.org/resource/Johann_Grugger
http://dbpedia.org/resource/Chiara_Costazza
...
155287 Rows. -- 342179 msec.
```

Example for loading script of the Yago Class hierarchy as inference rules

```
--- Load Class Hierarchy into a Named Graph
SELECT ttlp_mt (file_to_string_output ('yago-class-hierarchy_en.nt'),
'', 'http://dbpedia.org/resource/classes/yago#');

-- Create an  Inference Rule that references the Yago Class Hierarchy
Named Graph

SQL>rdfs_rule_set ('http://dbpedia.org/resource/inference/rules/yago#',
'http://dbpedia.org/resource/classes/yago#');

-- Query for the "The Lord of the Rings" which is a "Fantasy Novel" as explicitly
-- claimed in the DBpedia data set (instance data)

SQL>SPARQL
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbpedia: <http://dbpedia.org/property/>
PREFIX yago: <http://dbpedia.org/class/yago/>
SELECT ?s
FROM <http://dbpedia.org>
WHERE
{
  ?s a <http://dbpedia.org/class/yago/FantasyNovels> .
  ?s dbpedia:name "The Lord of the Rings"@en .
};

s
VARCHAR
_______________________________________________________________________________

http://dbpedia.org/resource/The_Lord_of_the_Rings
```

```
1 Rows. -- 241 msec.

-- Query aimed at Novel via query scoped to the "Fiction" class of
-- which it is a subclass in the Yago Hierarchy
SQL>SPARQL
define input:inference "http://dbpedia.org/resource/inference/rules/yago#"
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbpedia: <http://dbpedia.org/property/>
PREFIX yago: <http://dbpedia.org/class/yago/>

SELECT ?s
FROM <http://dbpedia.org>
WHERE {
?s a <http://dbpedia.org/class/yago/Fiction106367107> .
?s dbpedia:name "The Lord of the Rings"@en .
};

s
VARCHAR
_______________________________________________________________________________

http://dbpedia.org/resource/The_Lord_of_the_Rings
http://dbpedia.org/resource/The_Lord_of_the_Rings
http://dbpedia.org/resource/The_Lord_of_the_Rings
http://dbpedia.org/resource/The_Lord_of_the_Rings

4 Rows. -- 4767 msec.

-- # Variant of query with Virtuoso's Full Text Index extension: bif:contains
SQL>SPARQL
define input:inference "http://dbpedia.org/resource/inference/rules/yago#"
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbpedia: <http://dbpedia.org/property/>
PREFIX yago: <http://dbpedia.org/class/yago/>

SELECT ?s ?n
FROM <http://dbpedia.org>
WHERE {
?s a <http://dbpedia.org/class/yago/Fiction106367107> .
?s dbpedia:name ?n .
?n bif:contains 'Lord and Rings'
};

s                                                       n
VARCHAR                                                 VARCHAR
_______________________________________________________________________________

http://dbpedia.org/resource/The_Lord_of_the_Rings    The Lord of the Rings
http://dbpedia.org/resource/The_Lord_of_the_Rings    The Lord of the Rings
http://dbpedia.org/resource/The_Lord_of_the_Rings    The Lord of the Rings
http://dbpedia.org/resource/The_Lord_of_the_Rings    The Lord of the Rings

4 Rows. -- 5538 msec.

-- Retrieve all individuals instances of the FantasyNovels Class
SQL>SPARQL
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbpedia: <http://dbpedia.org/property/>
PREFIX yago: <http://dbpedia.org/class/yago/>

SELECT ?s ?n
FROM <http://dbpedia.org>
WHERE
{
  ?s a <http://dbpedia.org/class/yago/FantasyNovels> .
  ?s dbpedia:name ?n .
}
limit 10;
s                                                                       n
VARCHAR                                                                 VARCHAR
_______________________________________________________________________________
```

```
http://dbpedia.org/resource/ATLA_-_A_Story_of_the_Lost_Island                Atla
http://dbpedia.org/resource/A_Crown_of_Swords                                A Crown of Swords
http://dbpedia.org/resource/A_Game_of_Thrones                                A Game of Thrones
http://dbpedia.org/resource/A_Secret_Atlas                                   A Secret Atlas
http://dbpedia.org/resource/A_Storm_of_Swords                                A Storm of Swords
http://dbpedia.org/resource/A_Voyage_to_Arcturus                             A Voyage to Arcturus
http://dbpedia.org/resource/A_Wizard_Alone                                   A Wizard Alone
http://dbpedia.org/resource/Above_the_Veil                                   Above the Veil
http://dbpedia.org/resource/Black_Easter                                     Black Easter
http://dbpedia.org/resource/Lord_of_Chaos                                    Lord of Chaos

10 Rows. -- 781 msec.

-- Retrieve all individuals instances of Fiction Class which should
-- include all Novels.
SQL>SPARQL
define input:inference "http://dbpedia.org/resource/inference/rules/yago#"
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbpedia: <http://dbpedia.org/property/>
PREFIX yago: <http://dbpedia.org/class/yago/>

SELECT ?s ?n
FROM <http://dbpedia.org>
WHERE {
?s a <http://dbpedia.org/class/yago/Fiction106367107> .
?s dbpedia:name ?n .
};
s                                                                            n
VARCHAR                                                                      VARCHAR
_______________________________________________________________________________

http://dbpedia.org/resource/Last_Son_of_Krypton                              Last Son of Krypton
http://dbpedia.org/resource/Tuvaluan_language                                Tuvaluan
http://dbpedia.org/resource/Card_Walker                                      E. Cardon Walker
http://dbpedia.org/resource/Les_Clark                                        Les Clark
http://dbpedia.org/resource/Marc_Davis                                       Marc Davis
http://dbpedia.org/resource/Eric_Larson                                      Eric Larson
http://dbpedia.org/resource/Marty_Sklar                                      Marty Sklar
http://dbpedia.org/resource/Peter_Ellenshaw                                  Peter Ellenshaw
http://dbpedia.org/resource/Adriana_Caselotti                                Adriana Caselotti
http://dbpedia.org/resource/Jimmie_Dodd                                      Jimmie Dodd
...
15296 Rows.
```

Pure SPARQL Example

```
-- Query aimed at Fantasy Novel via query scoped to the "Fiction" class of
-- which it is a subclass in the Yago Hierarchy

SQL>SPARQL
define input:inference "http://dbpedia.org/resource/inference/rules/yago#"
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbpedia: <http://dbpedia.org/property/>
PREFIX yago: <http://dbpedia.org/class/yago/>
SELECT ?s
FROM <http://dbpedia.org>
WHERE {
?s a <http://dbpedia.org/class/yago/Fiction106367107> .
?s dbpedia:name "The Lord of the Rings"@en .
};
s
VARCHAR
_______________________________________________________________________________

http://dbpedia.org/resource/The_Lord_of_the_Rings
http://dbpedia.org/resource/The_Lord_of_the_Rings
http://dbpedia.org/resource/The_Lord_of_the_Rings
http://dbpedia.org/resource/The_Lord_of_the_Rings

4 Rows. -- 150 msec.
```

Example for equivalence between classes

This example is based on UMBEL and DBpedia integration:

```
-- Load UMBEL & DBpedia Instance Level Cross-Links (owl:sameAs) Triples
SELECT ttlp_mt (file_to_string_output ('umbel_dbpedia_linkage_v071.n3'), '', 'http://dbpedia.org');

-- Load UMBEL and DBpedia Type (rdf:type) association Triples
SELECT ttlp_mt (file_to_string_output ('umbel_dbpedia_types_v071.n3'), '', 'http://dbpedia.org');

--- Load UMBEL Subject Concept Class Hierarchy into a Named Graph
SELECT ttlp_mt (file_to_string_output ('umbel_class_hierarchy_v071.n3'), '', 'http://dbpedia.org/resource

--- load UMBEL Subject Concepts Instance Data
SELECT ttlp_mt (file_to_string_output ('umbel_subject_concepts.n3'), '', 'http://dbpedia.org/resource/cla

--- Load UMBEL Abstract Concepts Instance Data
SELECT ttlp_mt (file_to_string_output ('umbel_abstract_concepts.n3'), '', 'http://dbpedia.org/resource/cl

-- Load UMBEL External Ontology Mapping into a Named Graph
SELECT ttlp_mt (file_to_string_output ('umbel_external_ontologies_linkage.n3'), '', 'http://dbpedia.org/r

-- Create UMBEL Inference Rules

rdfs_rule_set ('http://dbpedia.org/resource/inference/rules/umbel#', 'http://dbpedia.org/resource/classes
```

Now let's execute the following queries:

```
SQL>SPARQL define input:inference "http://dbpedia.org/resource/inference/rules/umbel#"
prefix umbel: <http://umbel.org/umbel/sc/>
PREFIX dbpedia: <http://dbpedia.org/property/>
prefix opencyc: <http://sw.opencyc.org/2008/06/10/concept/en/>
SELECT ?s
where
{
 ?s a opencyc:Motorcycle.
 ?s dbpedia:name ?n.
 ?n bif:contains "BMW".
};

s
_______________________________________________
http://dbpedia.org/resource/BMW_K1200GT
http://dbpedia.org/resource/BMW_F650CS
http://dbpedia.org/resource/BMW_C1
http://dbpedia.org/resource/BMW_R75
4 Rows. -- 26 msec.

SQL>SPARQL define input:inference "http://dbpedia.org/resource/inference/rules/umbel#"
prefix umbel: <http://umbel.org/umbel/sc/>
PREFIX dbpedia: <http://dbpedia.org/property/>
prefix opencyc: <http://sw.opencyc.org/2008/06/10/concept/en/>
SELECT ?s
where
{
 ?s a umbel:Motorcycle.
 ?s dbpedia:name ?n.
 ?n bif:contains "BMW".
};

s
_______________________________________________
http://dbpedia.org/resource/BMW_K1200GT
http://dbpedia.org/resource/BMW_F650CS
http://dbpedia.org/resource/BMW_C1
http://dbpedia.org/resource/BMW_R75
4 Rows. -- 26 msec.
```

Example for finding celebrities which are not fans of their own fans

```
SQL>SPARQL
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX sioc: <http://rdfs.org/sioc/ns#>
INSERT INTO GRAPH <urn:rules.skos> { foaf:knows rdfs:subPropertyOf sioc:follows . };

callret-0
VARCHAR
Insert into <urn:rules.skos>, 1 triples -- done
No. of rows in result: 1

SQL>rdfs_rule_set ('foaf-trans', 'urn:rules.skos');

Done.

SPARQL>SPARQL
DEFINE input:inference "foaf-trans"
PREFIX sioc: <http://rdfs.org/sioc/ns#>
SELECT ?celeb COUNT (*)
WHERE
  {
    ?claimant sioc:follows ?celeb .
    FILTER
      (
        !bif:exists
         (
           ( SELECT (1)
             WHERE
               {
                 ?celeb sioc:follows ?claimant
               }
           )
         )
      )
  }
GROUP BY ?celeb
ORDER BY DESC 2
LIMIT 10

celeb                                                                       callret-1
ANY                                                                         ANY
_________________________________________________________________________________________
http://localhost.localdomain/about/id/entity/http/twitter.com/kidehen        100
http://localhost.localdomain/about/id/entity/http/twitter.com/shawnafennell  77
http://localhost.localdomain/about/id/entity/http/twitter.com/thines01       71
http://localhost.localdomain/about/id/entity/http/twitter.com/mhausenblas    50
http://localhost.localdomain/about/id/entity/http/twitter.com/DirDigEng      2
http://localhost.localdomain/about/id/entity/http/twitter.com/SarahPalinUSA  1
http://localhost.localdomain/about/id/entity/http/twitter.com/zbrox          1
http://localhost.localdomain/about/id/entity/http/twitter.com/LamarLee       1
http://localhost.localdomain/about/id/entity/http/twitter.com/HackerChick    1
http://localhost.localdomain/about/id/entity/http/twitter.com/programmingfeed 1
No. of rows in result: 10
```

16.14.9. Identity With Inverse Functional Properties

A graph used used with rdfs_rule_set may declare certain properties to be inversely functional. If one or more inverse functional properties (IFP's) are declared in the inference context used with the query, enabled with define input:inference = "context_name", then the following semantics apply:

1. If a literal is compared with an IRI, then the literal is substituted by all the subject IRI's where this literal occurs as a value of an IFP.
2. If two IRI's are compared for equality, they will be considered the same if there is an IFP P such that the same P has the same value on both subjects.
3. If an IRI is processed for distinctness in either distinct or group by, the IRI is first translated to be the IRI with the lowest ID among all IRI's that share an IFP value with this IRI.

Thus, if two IRI's are compared for distinctness, they will count as one if there is an IFP P with the same value with both IRI's. Literal data types are not translated into IRI's even if these literals occurred as IFP values of some subject.

It is possible to declare that specific values, even if they occur as values of an IFP in more than onme subject do not constitute identity between the subjects. For example, if two subjects were inferred to be the same because they had the same foaf:mbox_sha1sum, the SHA1 hash of mailto:// would be excluded. Two individuals have an email address that has a common default value are not the same.

In an ontology graph, a property IRI is declared to be inversely functional by making it an instance of the owl:InverseFunctionalProperty class. A value of an IFP can be declared null, i.e. sharing the value does not imply identity by by giving the IFP IRI a <http://www.openlinksw.com/schemas/virtrdf#nullIFPValue> property with the value to be ignored as the object.

Example

```
SQL>ttlp ('
<john1> a <person> .
<john2> a <person> .
<mary> a <person> .
<mike> a <person> .
<john1> <name> "John" .
<john2> <name> "John" .
<john1> <address> "101 A street" .
<john2> <address> "102 B street" .
<john2> <knows> <mike> .
<john1> <http://www.w3.org/2002/07/owl#sameAs> <john2> .
<mary> <knows> "John" .
<mike> <knows> <john1> .
<mike> <knows> <john2> .
<john1> <name> "Tarzan" .
<mike> <nam> "Tarzan" .
', '', 'ifps');

SQL>ttlp ('
<name> a <http://www.w3.org/2002/07/owl#InverseFunctionalProperty> .
<name> <http://www.openlinksw.com/schemas/virtrdf#nullIFPValue> "Tarzan" .
', '', 'ifp_list');

SQL>rdfs_rule_set ('ifps', 'ifp_list');

SQL>SPARQL define input:inference "ifps"  SELECT * FROM <ifps> WHERE {<john1> ?p ?o};

p                                                                                o
VARCHAR                                                                          VARCHAR
_______________________________________________________________________________

address                                                                          101 A street
name                                                                             John
http://www.w3.org/2002/07/owl#sameAs                                             john2
http://www.w3.org/1999/02/22-rdf-syntax-ns#type                                  person
name                                                                             Tarzan
name                                                                             John
knows                                                                            mike
http://www.w3.org/1999/02/22-rdf-syntax-ns#type                                  person
address                                                                          102 B street
```

We see that we get the properties of <john2> also.

```
SQL>SPARQL define input:inference "ifps" SELECT distinct ?p FROM <ifps> WHERE { ?p a <person>};

john2
mike
mary
```

We see that we get only one John. But John is not the same as Mike because they share the name Tarzan which is not considered as implying identity. Which John we get is a matter of which gets the lowest internal ID. This is variable and arbitrary at load time but once loaded this is permanent as long as the set of subjects with the name John does not change.

16.14.10. Inference Rules and SPARQL with Transitivity Option

- See example with an inference rule to cater data being skos:broader based, which is no longer transitive.
- See example with an inference rule to find entities that are subcategories of Protestant Churches, no deeper than 3 levels within the concept scheme hierarchy, filtered by a specific subcategory.

16.14.11. Inference Rules, OWL Support and Relationship Ontology

This section provides queries usage for inference rules, owl support and Relationship Vocabulary.

Example 1

Example based on Relationship Vocab:

```
## Verify Ontology Data is in Quad Store
## Ontology: <http://vocab.org/relationship/> (Relationship Ontology)
## Use pragma to put latest in Quad store.

DEFINE get:soft "replace"
SELECT *
FROM <http://vocab.org/relationship/>
WHERE {?s ?p ?o}

## Clean up

CLEAR GRAPH <urn:owl.tests>

## Create Instance Data for Relationship Ontology
PREFIX rel: <http://purl.org/vocab/relationship/>

INSERT into GRAPH <urn:owl.tests>
  {
    <http://dbpedia.org/resource/Prince_William_of_Wales> rel:siblingOf <http://dbpedia.org/resource/Prin
    <http://dbpedia.org/resource/Elizabeth_Bowes-Lyon> rel:ancestorOf <http://dbpedia.org/resource/Elizab
    <http://dbpedia.org/resource/Elizabeth_II_of_the_United_Kingdom> rel:ancestorOf
    <http://dbpedia.org/resource/Charles%2C_Prince_of_Wales>.
    <http://dbpedia.org/resource/Charles%2C_Prince_of_Wales> rel:ancestorOf <http://dbpedia.org/resource/
  };

## Verify

SELECT *
FROM  <urn:owl.tests>
WHERE
  {
    ?s ?p ?o
  }

## Create an  Inference Rule that references the Relationship Ontology Named Graph

rdfs_rule_set ('urn:owl.tests', 'http://vocab.org/relationship') ;

## Verify Rule's existence

SELECT * FROM SYS_RDF_SCHEMA ;
```

Example 2

```
## Test owl:TransitiveProperty Reasoning
## Start with a specific URI
## Goal: See inferred Triples
## In this case, relationship between: <http://dbpedia.org/resource/Elizabeth_Bowes-Lyon>
## and her descendants: Queen Elizabeth, Prince Charles, Prince William, and Prince Harry)

DEFINE input:inference 'urn:owl.tests'
PREFIX rel: <http://purl.org/vocab/relationship/>
SELECT *
FROM <urn:owl.tests>
WHERE
  {
```

```
    <http://dbpedia.org/resource/Elizabeth_Bowes-Lyon> rel:ancestorOf ?o
  }
```

Example 3

```
## Test owl:SymmetricalProperty Reasoning
## Should show same result irrespective of rel:siblingOf URI in Subject or Object slots of Triple

DEFINE input:inference 'urn:owl.tests'
PREFIX rel: <http://purl.org/vocab/relationship/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT *
FROM <urn:owl.tests>
WHERE
  {
    <http://dbpedia.org/resource/Prince_William_of_Wales> rel:siblingOf ?o
  }

## OR

DEFINE input:inference 'urn:owl.tests'
PREFIX rel: <http://purl.org/vocab/relationship/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT *
FROM <urn:owl.tests>
WHERE
  {
    ?s rel:siblingOf <http://dbpedia.org/resource/Prince_William_of_Wales>
  }
```

Example 4

```
## Test owl:inverseOf Reasoning
## Should show triples exposing the inverseOf relation.
## In this case rel:ancestorOf instance data triples exist,so the system must infer rel:descendant Of tri

DEFINE input:inference 'urn:owl.tests'
PREFIX rel: <http://purl.org/vocab/relationship/>

SELECT *
FROM <urn:owl.tests>
WHERE
  {
    <http://dbpedia.org/resource/Elizabeth_II_of_the_United_Kingdom> rel:descendantOf ?o
  }

## OR with Transitivity Option applied

DEFINE input:inference 'urn:owl.tests'
PREFIX rel: <http://purl.org/vocab/relationship/>

SELECT *
FROM <urn:owl.tests>
WHERE
  {
    <http://dbpedia.org/resource/Prince_William_of_Wales> rel:descendantOf ?o
    OPTION (T_DISTINCT)
  }
```

Example 5

```
## Test owl:inverseOf Reasoning
## Should show triples exposing the inverseOf relation.
## In this case rel:employedBy instance data triples exist,
## the system must infer rel:employerOf triples.

DEFINE input:inference 'urn:owl.tests'
PREFIX rel: <http://purl.org/vocab/relationship/>

SELECT *
```

```
FROM <urn:owl.tests>
WHERE
  {
    ?s rel:employerOf ?o
  }
```

Example 6

Example based on Relationship Vocab and SKOS

Assume the following test10.rdf file:

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
        xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:skos="http://www.w3.org/2004/02/skos/core#"
        xmlns:owl="http://www.w3.org/2002/07/owl#"
        xmlns:turnguard="http://www.turnguard.com#"
        xmlns:dc="http://purl.org/dc/elements/1.1/"
        xmlns:sea="http://www.2sea.org/"
        xmlns:foaf="http://xmlns.com/foaf/0.1/"
        xmlns:dct="http://purl.org/dc/terms/">

<rdf:Description rdf:about="http://www.turnguard.com/Music">
        <rdf:type rdf:resource="http://www.w3.org/2004/02/skos/core#Concept"/>
        <skos:prefLabel xml:lang="en">Music</skos:prefLabel>
        <skos:narrower rdf:resource="http://www.turnguard.com/Pop" />
</rdf:Description>
<rdf:Description rdf:about="http://www.turnguard.com/Pop">
      <rdf:type rdf:resource="http://www.w3.org/2004/02/skos/core#Concept"/>
                <skos:prefLabel xml:lang="en">POP</skos:prefLabel>
      <skos:narrower rdf:resource="http://www.turnguard.com/TechnoPop" />
</rdf:Description>
<rdf:Description rdf:about="http://www.turnguard.com/TechnoPop">
        <rdf:type rdf:resource="http://www.w3.org/2004/02/skos/core#Concept"/>
        <skos:prefLabel xml:lang="en">TECHNOPOP</skos:prefLabel>
        <skos:narrower rdf:resource="http://www.turnguard.com/ElectroPop" />
</rdf:Description>

<rdf:Description rdf:about="http://www.turnguard.com/ElectroPop">
        <rdf:type rdf:resource="http://www.w3.org/2004/02/skos/core#Concept"/>
        <skos:prefLabel xml:lang="en">ELECTROPOP</skos:prefLabel>
</rdf:Description>
<rdf:Description rdf:about="http://www.turnguard.com/KrautRock">
        <rdf:type rdf:resource="http://www.w3.org/2004/02/skos/core#Concept"/>
        <skos:prefLabel xml:lang="en">KrautRock</skos:prefLabel>
        <skos:related rdf:resource="http://www.turnguard.com/ElectroPop" />
</rdf:Description>
</rdf:RDF>
```

Execute the following steps:

```
## Graph Cleanup
CLEAR GRAPH <urn:owl.test2.tbox>
CLEAR GRAPH <http://turnguard.com/virtuoso/test10.rdf>

## Load Instance Data into Quad Store
## PL Procedure

## SQL realm
DB.DBA.RDF_LOAD_RDFXML
  (
    http_get('http://www.w3.org/2009/08/skos-reference/skos-owl1-dl.rdf'),
    'no',
    'urn:owl.test2.tbox'
  );
DB.DBA.RDF_LOAD_RDFXML
  (
```

```
    http_get ('http://www.w3.org/2002/07/owl.rdf'),
    'no',
    'urn:owl.test2.tbox'
  );
DB.DBA.RDF_LOAD_RDFXML
  (
    file_to_string ('test10.rdf'),
    'no',
    'http://turnguard.com/virtuoso/test10.rdf'
  );

SELECT *
FROM <http://www.w3.org/2004/02/skos/core>
WHERE
  {
    {
      <http://www.w3.org/2004/02/skos/core#related> ?p ?o
    }
    UNION
    {
      ?s ?p <http://www.w3.org/2004/02/skos/core#related>
    }
  }

## Create Rules
## SQL Realm

rdfs_rule_set ('urn:owl.test2.rules', 'urn:owl.test2.tbox');

## Transitivity Query re. SKOS concept hierarchy

DEFINE input:inference "urn:owl.test2.rules"
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>

SELECT *
FROM <http://turnguard.com/virtuoso/test10.rdf>
WHERE
  {
    <http://www.turnguard.com/ElectroPop> skos:broaderTransitive ?o
    OPTION  (T_DISTINCT).
  }
```

16.15. RDF and Geometry

A geometry may occur as an object of an RDF quad. The SQL MM functions can then be used for geospatial queries.

For geometry functions, see the SQL Geometry support section .

A geometry may occur as an object value in an RDF quad. In such a case, the bare geometry object is not used but instead a special RDF typed literal is made with the type virtrdf:Geometry. Such a literal is automatically indexed in an R tree index containing all distinct geometries occurring in any quad of any graph under any predicate. Normally, WGS84, SRID 4326 is the SRID of any such geometry.

In this section, the geo namespace prefix is used to mean <http://www.w3.org/2003/01/geo/wgs84_pos#>.

The preferred way of adding geometries to RDF graphs is with the ttlp and related functions which parse a text string in the Turtle syntax and insert the result in a graph.

For example:

```
ttlp ('@prefix virtrdf: <http://www.openlinksw.com/schemas/virtrdf#>
<point> geo:geometry "point(1.2 22.4"^^virtrdf:Geometry .',
'xxx', 'graph');
```

A typed literal whose text is a WKT representation of a geometry and whose type is virtrdf:geometry creates a geometry object and adds it to the R tree index of all RDF geometries.

Geometries can be queried with geometry predicates, st_intersects, st_contains and st_within, as follows. As usual, the bif: namespace is used since these are SQL built-in functions.

```
SQL>
SPARQL
SELECT ?c COUNT (*)
WHERE
  {
    ?m geo:geometry ?geo .
    ?m a ?c .
    FILTER (bif:st_intersects (?geo, bif:st_point (0, 52), 100))
  }
GROUP BY ?c
ORDER BY DESC 2;

c                                                    callret-1
VARCHAR                                              VARCHAR
_____________________________________________________________
http://linkedgeodata.org/vocabulary#node             2317684
http://linkedgeodata.org/vocabulary#way              85315
http://linkedgeodata.org/vocabulary#building         14257
http://dbpedia.org/class/yago/Landmark108624891      9093
http://linkedgeodata.org/vocabulary#wood             7155
http://linkedgeodata.org/vocabulary#gate             7079
http://www.w3.org/2002/07/owl#Thing                  6788
http://linkedgeodata.org/vocabulary#post_box         6144
http://linkedgeodata.org/vocabulary#pub              5697
http://dbpedia.org/ontology/Place                    5670
http://linkedgeodata.org/vocabulary#hedge            5391
...
```

This would return the classes of things within 100 km of 0, 52, which is near London.

```
SQL>
SPARQL
SELECT ?m (bif:st_distance (?geo, bif:st_point (0, 52)))
WHERE
  {
    ?m geo:geometry ?geo .
    ?m a <http://dbpedia.org/ontology/City> .
    FILTER (bif:st_intersects (?geo, bif:st_point (0, 52), 30))
  }
ORDER BY DESC 2
LIMIT 20;

m                                                                                    callret-1
VARCHAR                                                                              VARCHAR
_____________________________________________________________________________________

http://dbpedia.org/resource/Kingston%2C_Cambridgeshire                               39.13180985471543
http://dbpedia.org/resource/Kingston%2C_Cambridgeshire                               39.13180985471543
http://dbpedia.org/resource/Kingston%2C_Cambridgeshire                               39.13180985471543
http://dbpedia.org/resource/Kingston%2C_Cambridgeshire                               39.13180985471543
http://dbpedia.org/resource/Kingston%2C_Cambridgeshire                               37.36907252285992
http://dbpedia.org/resource/Kingston%2C_Cambridgeshire                               34.49432513061792
http://dbpedia.org/resource/Kingston%2C_Cambridgeshire                               33.7676326404143
http://dbpedia.org/resource/Kingston%2C_Cambridgeshire                               33.24238654570499
http://dbpedia.org/resource/Kingston%2C_Cambridgeshire                               32.60139660515003
http://dbpedia.org/resource/Kingston%2C_Cambridgeshire                               32.60139660515003
http://dbpedia.org/resource/Kingston%2C_Cambridgeshire                               32.17414911350438
http://dbpedia.org/resource/Kingston%2C_Cambridgeshire                               31.45681319171456
http://dbpedia.org/resource/Kingston%2C_Cambridgeshire                               31.17750625349044
http://dbpedia.org/resource/Kingston%2C_Cambridgeshire                               31.115377038
http://dbpedia.org/resource/Kingston%2C_Cambridgeshire                               31.115377038
http://dbpedia.org/resource/Kingston%2C_Cambridgeshire                               30.56388658524301
http://dbpedia.org/resource/Kingston%2C_Cambridgeshire                               29.89662974046085
http://dbpedia.org/resource/Kingston%2C_Cambridgeshire                               29.85090625132639
http://dbpedia.org/resource/Kingston%2C_Cambridgeshire                               29.82605254366244
http://dbpedia.org/resource/Kingston%2C_Cambridgeshire                               29.60102064794003

20 Rows. -- 13600 msec.
```

This would be the cities within 20 km of 0, 52, ordered by increasing distance from this point.

When SPARQL is called from SQL, the geometries can be bound to SQL variables as anything else returned from SPARQL. The *st_* functions can then be used for retrieving properties of these objects.

See Also

`st_point`

`st_x`

`st_y`

`st_distance`

`ST_SRID`

`ST_SetSRID`

`st_astext`

`st_geomfromtext`

`st_contains`

`st_intersects`

`st_within`

`isgeometry`

`geo_insert`

`geo_delete`

16.15.1. Programmatic Manipulation of Geometries in RDF

The `ttlp` function is the preferred way of inserting geometries. The more are inserted at one time, the more efficient the operation is. This loader function will also deal with cluster message optimization.

For deleting quads with geometries, normal SPARUL operations apply.

A geometry occurring in an RDF quad object is a member of the RDF box data type. This data type stands for a typed RDF literal. The type of all geometries is 256. This is mapped to a URI in the RDF_DATATYPE system table.

A geometry does not occur directly in the object position of a quad. It is referenced by an id that is stored in the RDF typed literal box and references RO_ID of the RDF_OBJ system table. To translate a geometry into a RDF box that can be stored, do as in the example below:

```
INSERT INTO RDF_QUAD (g, s, p, o)
VALUES (
         "g",
         "s",
         iri_to_id ('http://www.w3.org/2003/01/geo/wgs84_pos#geometry'),
         DB.DBA.rdf_geo_add (rdf_box (st_point (lng, lat), 256, 257, 0, 1)));
```

The DB.DBA.RDF_GEO_ADD function looks if an identical geometry already exists and if so assigns the existing id to it. If the geometry is new, it gets a new ID and is stored in the RDF literals table RDF_OBJ. At this time it is also automatically inserted into the RDF geometry index.

In a cluster situation one should use the dpipe mechanism for inserting into RDF quad so as to get large numbers of inserts into a single message. This is essential for performance.

16.15.2. Creating Geometries From RDF Data

Many data sets use the geo:lat and geo:long properties for describing a position. Virtuoso comes with a function for converting these properties into geometries. This operation reads through all graphs and for each subject with at least one geo:lat and geo:long, a point geometry is made for each distinct lat/long pair where lat and long are in the same graph. It should not happen in practice that a single subject has multiple lats or long within one graph. If this still happens, a geometry is made for each combination. The geometry is added to the subject with the lat and long as the value of the geo:geometry property. This is added to the same graph where the lat and long were.

The SQL procedure DB.DBA.RDF_GEO_FILL () performs this operation. This is performed in parallel on multiple threads and is optimized for cluster execution. This is done without transaction logging and is not transactional. To make the result persistent, the operator should do an explicit checkpoint. This is done by executing:

```
SQL>cl_exec ('checkpoint');
```

on any process of a cluster or single server. Otherwise the result may be lost if the server terminates abnormally before an automatic checkpoint is made.

The DB.DBA.RDF_GEO_FILL procedure may in principle be called several times but it will read every lat and long in the database. This is inefficient if there are large numbers of geometries.

Application logic must generally be used for constructing geometries and adding these to RDF subjects. It is easiest for the application to construct a text representation of the geometries in TTL and to use the `ttlp` function for loading this.

16.15.3. Using Geometries With Existing Databases

The geometry feature is compatible with any Virtuoso 6 databases. Once geometries are used, the database should not be opened with a server older than the one used for first inserting geometries, older servers will consider the storage format a physical corruption.

16.15.4. GEO Spatial Examples

Example 1

```
## Get All Stuff For Given Coordinates
SQL>SPARQL
SELECT ?c COUNT (*)
WHERE
   {
     ?m geo:geometry ?geo .
     ?m a ?c .
     FILTER (bif:st_intersects (?geo, bif:st_point (0, 52), 100))
   }
GROUP BY ?c
ORDER BY desc 2;

c                                                                                callret-1
VARCHAR                                                                          VARCHAR
_______________________________________________________________________________

http://linkedgeodata.org/vocabulary#node                                         2317684
http://linkedgeodata.org/vocabulary#way                                          85315
http://linkedgeodata.org/vocabulary#building                                     14257
http://dbpedia.org/class/yago/Landmark108624891                                  9093
http://linkedgeodata.org/vocabulary#wood                                         7155
....
```

Example 2

```
## Get City Stuff Around Catholic Churches In Paris
SQL>
SPARQL
SELECT ?m (bif:st_distance (?geo, bif:st_point (0, 52)))
WHERE
  {
    ?m geo:geometry ?geo .
```

```
    ?m a <http://dbpedia.org/ontology/City> .
    FILTER (bif:st_intersects (?geo, bif:st_point (0, 52), 30))
  }
ORDER BY DESC 2
LIMIT 20;
```

m	callret-1
VARCHAR	VARCHAR
http://dbpedia.org/resource/Stansted_Mountfitchet	39.13180985471543
http://dbpedia.org/resource/Stansted_Mountfitchet	39.13180985471543
http://dbpedia.org/resource/Stansted_Mountfitchet	39.13180985471543
http://dbpedia.org/resource/Stansted_Mountfitchet	39.13180985471543
http://dbpedia.org/resource/Stansted_Mountfitchet	37.36907252285992
http://dbpedia.org/resource/Stansted_Mountfitchet	34.49432513061792
http://dbpedia.org/resource/Stansted_Mountfitchet	33.7676326404143
http://dbpedia.org/resource/Stansted_Mountfitchet	33.24238654570499
http://dbpedia.org/resource/Stansted_Mountfitchet	32.60139660515003
http://dbpedia.org/resource/Stansted_Mountfitchet	32.60139660515003
http://dbpedia.org/resource/Stansted_Mountfitchet	31.45681319171456
http://dbpedia.org/resource/Stansted_Mountfitchet	31.115377038
http://dbpedia.org/resource/Stansted_Mountfitchet	31.115377038
http://dbpedia.org/resource/Stansted_Mountfitchet	30.56388658524301
http://dbpedia.org/resource/Stansted_Mountfitchet	29.89662974046085
http://dbpedia.org/resource/Stansted_Mountfitchet	29.85090625132639
http://dbpedia.org/resource/Stansted_Mountfitchet	29.82605254366244
http://dbpedia.org/resource/Stansted_Mountfitchet	29.60102064794003
http://dbpedia.org/resource/Stansted_Mountfitchet	29.44147385851453
http://dbpedia.org/resource/Stansted_Mountfitchet	29.421242437379

Example 3

```
## Get City Stuff Around Catholic Churches In Paris Extended
SQL>
SPARQL
SELECT ?m (bif:st_distance (?geo, bif:st_point (0, 52)))
WHERE
  {
    ?m geo:geometry ?geo .
    ?m a <http://dbpedia.org/ontology/City> .
    FILTER (bif:st_intersects (?geo, bif:st_point (0, 52), 100))
  }
ORDER BY DESC 2
LIMIT 20;
```

m	callret-1
VARCHAR	VARCHAR
http://dbpedia.org/resource/Weston-on-Trent	138.7082197019335
http://dbpedia.org/resource/Weston-on-Trent	137.7213767969613
http://dbpedia.org/resource/Weston-on-Trent	136.4597167847218
http://dbpedia.org/resource/Weston-on-Trent	134.1807668663677
http://dbpedia.org/resource/Weston-on-Trent	133.104337839536
http://dbpedia.org/resource/Weston-on-Trent	133.104337839536
http://dbpedia.org/resource/Nonington	132.7368236183588
http://dbpedia.org/resource/Nonington	132.1339163200362
http://dbpedia.org/resource/Nonington	132.1339163200362
http://dbpedia.org/resource/Nonington	130.5478483560461
http://dbpedia.org/resource/Nonington	130.1620410981843
http://dbpedia.org/resource/Nonington	129.8549842943355
http://dbpedia.org/resource/Nonington	129.6459280567849
http://dbpedia.org/resource/Nonington	129.4504858595742
http://dbpedia.org/resource/Nonington	129.2790713235814
http://dbpedia.org/resource/Nonington	128.9081040147881
http://dbpedia.org/resource/Nonington	128.8845164618929
http://dbpedia.org/resource/Nonington	128.6676189617872
http://dbpedia.org/resource/Nonington	128.2565253458452
http://dbpedia.org/resource/Nonington	128.2551696344652

```
20 Rows. -- 120 msec.
```

Example 4

```
## Text Or Geo
SQL>
SPARQL
SELECT ?c COUNT (*)
WHERE
  {
    ?m geo:geometry ?geo .
    ?m a ?c .
    FILTER (bif:st_intersects (?geo, bif:st_point (0, 52), 100)  &&  REGEX (str (?c), "London") )
  }
GROUP BY ?c
ORDER BY DESC 2
LIMIT 10;

c                                                                    callret-1
_____________________________________________________________________________

http://dbpedia.org/class/yago/DistrictsOfLondon                      861
http://dbpedia.org/class/yago/GradeIListedBuildingsInLondon          199
http://dbpedia.org/class/yago/MuseumsInLondon                        107
http://dbpedia.org/class/yago/ArtMuseumsAndGalleriesInLondon         92
http://dbpedia.org/class/yago/GradeIIListedBuildingsInLondon         89
http://dbpedia.org/class/yago/SportsVenuesInLondon                   80
http://dbpedia.org/class/yago/RoyalBuildingsInLondon                 72
http://dbpedia.org/class/yago/LondonOvergroundStations               69
http://dbpedia.org/class/yago/NationalGovernmentBuildingsInLondon    69
http://dbpedia.org/class/yago/SkyscrapersInLondon                    60
```

Example 5

```
## Example "Places Of Worship, Within 5 km Of Paris":

## Describes places of worship, within 5 km of Paris,
## that have cafes in close proximity(0.2 km).
## The query requires V6 or higher.
SQL>
PREFIX lgv: <http://linkedgeodata.org/vocabulary#>
DESCRIBE ?cafe ?church
WHERE
  {
    ?church a lgv:place_of_worship .
    ?church geo:geometry ?churchgeo .
    ?church lgv:name ?churchname .
    ?cafe a lgv:cafe .
    ?cafe lgv:name ?cafename .
    ?cafe geo:geometry ?cafegeo .
    ?cafe geo:lat ?lat .
    ?cafe geo:long ?long .
    FILTER ( bif:st_intersects ( ?churchgeo, bif:st_point ( 2.3498, 48.853 ), 5 ) &&
             bif:st_intersects ( ?cafegeo, ?churchgeo, 0.2 ) )
  }
LIMIT 10;

@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ns1:    <http://linkedgeodata.org/triplify/node/243360870#> .
@prefix ns2:    <http://linkedgeodata.org/vocabulary#> .
ns1:id  rdf:type        ns2:place_of_worship ,
                ns2:node .
@prefix geo:    <http://www.w3.org/2003/01/geo/wgs84_pos#> .
ns1:id  geo:lat 48.8794 ;
        geo:long        2.3748 ;
        ns2:created_by  "Potlatch 0.6c" ;
        ns2:name        "Saint-Georges de la Villette" ;
        ns2:religion    "christian" ,
                ns2:christian .
@prefix virtrdf:        <http://www.openlinksw.com/schemas/virtrdf#> .
ns1:id  geo:geometry    "POINT(2.3748 48.8794)"^^virtrdf:Geometry .
@prefix ns5:    <http://linkedgeodata.org/triplify/node/266632049#> .
ns5:id  rdf:type        ns2:node ,
                ns2:cafe ;
```

```
        geo:lat 48.8518 ;
        geo:long        2.325 ;
        ns2:created_by  "Potlatch 0.9a" ;
        ns2:name        "Le Babylone" ;
        geo:geometry    "POINT(2.325 48.8518)"^^virtrdf:Geometry .
....
```

Example 6

```
## Count Geo
SQL>
SPARQL
SELECT ?c COUNT (*)
WHERE
   {
     ?s geo:geometry ?geo .
     FILTER (bif:st_intersects (?geo, bif:st_point (2.3498, 48.853), 5)) .
     ?s a ?c
   }
GROUP BY ?c
ORDER BY desc 2
LIMIT 10;

c                                                                                callret-1
VARCHAR                                                                          VARCHAR
_______________________________________________________________________________________

http://linkedgeodata.org/vocabulary#node                                         37792
http://dbpedia.org/class/yago/Landmark108624891                                  4003
http://linkedgeodata.org/vocabulary#way                                          1688
http://linkedgeodata.org/vocabulary#building                                     719
http://linkedgeodata.org/vocabulary#station                                      257
http://linkedgeodata.org/vocabulary#post_box                                     247
http://www.w3.org/2002/07/owl#Thing                                              227
http://linkedgeodata.org/vocabulary#park                                         208
http://linkedgeodata.org/vocabulary#restaurant                                   198
http://dbpedia.org/ontology/Place                                                192

10 Rows. -- 932 msec.
```

Example 7

```
## Get Stuff Around Notre Dame De Paris
SQL>
SPARQL
SELECT ?c COUNT (*)
WHERE
  {
    ?s a ?c .
    ?s geo:geometry ?geo .
    FILTER (bif:st_intersects (?geo, bif:st_point (2.3498, 48.853), 0.3))
  }
GROUP BY ?c
ORDER BY desc 2
LIMIT 10;

c                                                                                callret-1
VARCHAR                                                                          VARCHAR
_______________________________________________________________________________________

http://linkedgeodata.org/vocabulary#node                                         408
http://dbpedia.org/class/yago/Landmark108624891                                  134
http://linkedgeodata.org/vocabulary#way                                          17
http://dbpedia.org/class/yago/RomanCatholicChurchesInParis                       17
http://dbpedia.org/class/yago/TallBuildingsAndStructuresInParis                  13
http://dbpedia.org/class/yago/CathedralsInFrance                                 13
http://sw.opencyc.org/2008/06/10/concept/Mx4rvVigPpwpEbGdrcN5Y29ycA              13
http://sw.opencyc.org/2008/06/10/concept/Mx4rjm5QanS6EdaAAACgyZzFrg              13
http://sw.opencyc.org/2008/06/10/concept/Mx4rwQwtGpwpEbGdrcN5Y29ycA              13
http://www.w3.org/2002/07/owl#Thing                                              10

10 Rows. -- 241 msec.
```

Example 8

```
## Things within 10 km proximity of place of worship
SQL>
SPARQL
PREFIX lgv: <http://linkedgeodata.org/vocabulary#>
SELECT ?c COUNT (*)
WHERE
  {
    ?s a ?c .
    ?s a lgv:place_of_worship .
    ?s geo:geometry ?geo .
    FILTER (bif:st_intersects (?geo, bif:st_point (2.3498, 48.853), 10))
  }
GROUP BY ?c
ORDER BY desc 2
LIMIT 10;

c                                                                                callret-1
VARCHAR                                                                          VARCHAR
_______________________________________________________________________________

http://linkedgeodata.org/vocabulary#place_of_worship                             147
http://linkedgeodata.org/vocabulary#node                                         146
http://linkedgeodata.org/vocabulary#way                                          46
http://linkedgeodata.org/vocabulary#building                                     36
http://linkedgeodata.org/vocabulary#attraction                                   3
http://linkedgeodata.org/vocabulary#church                                       1

6 Rows. -- 120 msec.
```

Example 9

```
## Get Stuff Around Notre Dame De Paris with Names
SQL>
SPARQL
PREFIX lgv: <http://linkedgeodata.org/vocabulary#>
SELECT ?cn
WHERE
  {
    ?s lgv:name ?cn  .
    ?s geo:geometry ?geo .
    FILTER (bif:st_intersects (?geo, bif:st_point (2.3498, 48.853), 0.3))
  }
LIMIT 20;
cn
VARCHAR
_______________________________________________________________________________

Parking Lagrange
Maitre Albert B&B
Le Grenier de Notre Dame
Eglise Saint-Julien-le-Pauvre
Eglise Saint Julien le Pauvre
Polly Magoo
Point 0 des Routes de France
Square Jean XXIII
....
20 Rows. -- 140 msec.
```

Example 10

```
## Get Churches With The Most Bars
SQL>
SPARQL
PREFIX lgv: <http://linkedgeodata.org/vocabulary#>
SELECT ?churchname ?cafename (bif:st_distance (?churchgeo, ?cafegeo))
WHERE
  {
    ?church a lgv:place_of_worship .
    ?church geo:geometry ?churchgeo .
    ?church lgv:name ?churchname .
```

```
    ?cafe a lgv:cafe .
    ?cafe lgv:name ?cafename .
    ?cafe geo:geometry ?cafegeo .
    FILTER (bif:st_intersects (?churchgeo, bif:st_point (2.3498, 48.853), 5)
      && bif:st_intersects (?cafegeo, ?churchgeo, 0.2))
  }
LIMIT 10;

churchname                          cafename                        callret-2
VARCHAR                             VARCHAR                         VARCHAR
_______________________________________________________________________________

Eglise Saint-Julien-le-Pauvre       Le Saint R+?-?gis               0.09759308692691648
Eglise Saint-Germain des Pr+?-?s    Caf+?-? de Flore                0.08774468391412803
Eglise Saint-Germain des Pr+?-?s    Les Deux Magots                 0.05235923473923059
Eglise Saint-Germain des Pr+?-?s    Caf+?-? Mabillon                0.1712042770289815
Eglise Saint-Germain-des-Pr+?-?s    Caf+?-? de Flore                0.1466502865197912
Eglise Saint-Germain-des-Pr+?-?s    Les Deux Magots                 0.1096767137079839
Eglise Saint-Germain-des-Pr+?-?s    Bar du march+?-?                0.1831441251868126
Eglise Saint-Germain-des-Pr+?-?s    Caf+?-? Mabillon                0.1174051745495528
Synagogue                           La Chaise au Plafond            0.1038387283609551
Synagogue                           Le Loir dans la Th+?-?i+?-?re   0.1632848322062273

10 Rows. -- 511225 msec.
```

Example 11

```
## Things around highly populated places
SQL>
SPARQL
SELECT ?s ( sql:num_or_null (?o) ) COUNT (*)
WHERE
  {
    ?s <http://dbpedia.org/ontology/populationTotal> ?o .
        FILTER ( sql:num_or_null (?o) > 6000000 ) .
    ?s geo:geometry ?geo .
        FILTER ( bif:st_intersects (?pt, ?geo,2) ) .
    ?xx geo:geometry ?pt
  }
GROUP BY ?s ( sql:num_or_null (?o) )
ORDER BY desc 3
LIMIT 20;

s                                                   callret-1       callret-2
VARCHAR                                             VARCHAR         VARCHAR
_______________________________________________________________________________

http://dbpedia.org/resource/London                  7556900         312307
http://dbpedia.org/resource/Toronto                 8102163         115859
http://dbpedia.org/resource/New_York_City           8363710         95629
http://dbpedia.org/resource/The_Hague               6659300         84410
http://dbpedia.org/resource/Tokyo                   12790000        78618
http://dbpedia.org/resource/Philadelphia            6385461         67115
http://dbpedia.org/resource/Los_Angeles             17755322        64394
http://dbpedia.org/resource/Bangkok                 8160522         62519
http://dbpedia.org/resource/Barcelona               2147483648      57635
http://dbpedia.org/resource/Cairo                   6758581         52738
http://dbpedia.org/resource/Istanbul                12697164        50745
http://dbpedia.org/resource/Seoul                   10421782        43962
http://dbpedia.org/resource/Beijing                 17430000        35979
http://dbpedia.org/resource/Purmerend               6659300         33508
http://dbpedia.org/resource/Baghdad                 6554126         33426
http://dbpedia.org/resource/Bogot%C3%A1             6776009         30429
http://dbpedia.org/resource/Mexico_City             8836045         30127
http://dbpedia.org/resource/Jakarta                 8500000         28944
http://dbpedia.org/resource/Boston                  7514759         27705
http://dbpedia.org/resource/Baden-W%C3%BCrttemberg  10755000        25112

20 Rows. -- 4296 msec.
```

Example 12

```
## Example "Places Of Worship, Within 5 km Of Paris":

## Constructs a custom Linked Data Mesh (graph) about
## places of worship, within 5 km of Paris, that have
## cafes in close proximity(0.2 km).

## Note: we have distinct pin colors that identify
## for places of worship distinct from cafes.

## The query requires V6 or higher.
SQL>
SPARQL
PREFIX lgv: <http://linkedgeodata.org/vocabulary#>
PREFIX rtb: <http://www.openlinksw.com/schemas/oat/rdftabs#>
CONSTRUCT
  {
    ?cafe geo:geometry ?cafegeo        ;
                  rtb:useMarker '01'   ;
                 lgv:name ?cafename    .
    ?church geo:geometry ?churchgeo    ;
                  rtb:useMarker '02'   ;
                lgv:name ?churchname   .
  }
WHERE
  {
    ?church a lgv:place_of_worship .
    ?church geo:geometry ?churchgeo .
    ?church lgv:name ?churchname .
    ?cafe a lgv:cafe .
    ?cafe lgv:name ?cafename .
    ?cafe geo:geometry ?cafegeo .
    ?cafe geo:lat ?lat .
    ?cafe geo:long ?long .
    FILTER ( bif:st_intersects ( ?churchgeo, bif:st_point ( 2.3498, 48.853 ), 5 ) &&
             bif:st_intersects ( ?cafegeo, ?churchgeo, 0.2 ) )
  }
LIMIT 10;

@prefix ns0:    <http://linkedgeodata.org/vocabulary#> .
@prefix ns1:    <http://linkedgeodata.org/triplify/node/237435716#> .
ns1:id  ns0:name        "Chapelle du Val de Gr\u00C3\u00A2ce" .
@prefix ns2:    <http://www.openlinksw.com/schemas/oat/rdftabs#> .
ns1:id  ns2:useMarker   "02" .
@prefix virtrdf:        <http://www.openlinksw.com/schemas/virtrdf#> .
@prefix geo:    <http://www.w3.org/2003/01/geo/wgs84_pos#> .
ns1:id  geo:geometry    "POINT(2.3418 48.8406)"^^virtrdf:Geometry .
@prefix ns5:    <http://linkedgeodata.org/triplify/node/218147750#> .
ns5:id  ns0:name        "Synagogue" ;
        ns2:useMarker   "02" ;
        geo:geometry    "POINT(2.3593 48.857)"^^virtrdf:Geometry .
@prefix ns6:    <http://linkedgeodata.org/triplify/node/218145208#> .
ns6:id  ns0:name        "Synagogue" ;
        ns2:useMarker   "02" ;
        geo:geometry    "POINT(2.3589 48.8567)"^^virtrdf:Geometry .
...
```

Example 13

```
## Example "Places Of Worship, Within 5 km Of Paris":

## Asks for places of worship, within 5 km of Paris,
## that have cafes in close proximity(0.2 km).
## The query requires V6 or higher.

SQL>
SPARQL
PREFIX lgv: <http://linkedgeodata.org/vocabulary#>
ASK
WHERE
  {
```

```
    ?church a lgv:place_of_worship .
    ?church geo:geometry ?churchgeo .
    ?church lgv:name ?churchname .
    ?cafe a lgv:cafe .
    ?cafe lgv:name ?cafename .
    ?cafe geo:geometry ?cafegeo .
    ?cafe geo:lat ?lat .
    ?cafe geo:long ?long .
    FILTER ( bif:st_intersects ( ?churchgeo, bif:st_point ( 2.3498, 48.853 ), 5 ) &&
             bif:st_intersects ( ?cafegeo, ?churchgeo, 0.2 ) )
  };

Done.

true
```

Example 14

```
## Places of worship, within 5 km of Paris,
## that have cafes in close proximity(0.2 km)
SQL>
SPARQL
PREFIX lgv: <http://linkedgeodata.org/vocabulary#>
SELECT DISTINCT ?cafe ?lat ?long ?cafename ?churchname
(bif:round(bif:st_distance (?churchgeo, ?cafegeo)))
WHERE
  {
    ?church a lgv:place_of_worship .
    ?church geo:geometry ?churchgeo .
    ?church lgv:name ?churchname .
    ?cafe a lgv:cafe .
    ?cafe lgv:name ?cafename .
    ?cafe geo:geometry ?cafegeo .
    ?cafe geo:lat ?lat.
    ?cafe geo:long ?long.
    FILTER ( bif:st_intersects (?churchgeo, bif:st_point (2.3498, 48.853), 5) &&
                  bif:st_intersects (?cafegeo, ?churchgeo, 0.2) )
  }
LIMIT 10;

cafe                                                    lat       long     cafename
VARCHAR                                                 VARCHAR   VARCHAR  VARCHAR
_______________________________________________________________________________________

http://linkedgeodata.org/triplify/node/321932192#id  48.8522   2.3484   Le Saint R+?-?gis
http://linkedgeodata.org/triplify/node/251699776#id  48.8541   2.3326   Caf+?-? de Flore
http://linkedgeodata.org/triplify/node/251699775#id  48.854    2.3331   Les Deux Magots
http://linkedgeodata.org/triplify/node/315769036#id  48.8533   2.3358   Caf+?-? Mabillon
http://linkedgeodata.org/triplify/node/251699776#id  48.8541   2.3326   Caf+?-? de Flore
http://linkedgeodata.org/triplify/node/251699775#id  48.854    2.3331   Les Deux Magots
http://linkedgeodata.org/triplify/node/315769035#id  48.8539   2.3371   Bar du march+?-?
http://linkedgeodata.org/triplify/node/315769036#id  48.8533   2.3358   Caf+?-? Mabillon
http://linkedgeodata.org/triplify/node/251126326#id  48.8572   2.3577   La Chaise au Plafond
http://linkedgeodata.org/triplify/node/251043135#id  48.8562   2.361    Le Loir dans la Th+?-?i+?-?re

10 Rows. -- 120 msec.
```

Example 15

```
## Stuff around Notre Dame de Paris
SQL>
SPARQL
PREFIX lgv: <http://linkedgeodata.org/vocabulary#>
SELECT ?s ?cn ?lat ?long
WHERE
  {
    ?s lgv:name ?cn  .
    ?s geo:geometry ?geo .
    ?s geo:lat ?lat.
    ?s geo:long ?long.
    FILTER ( bif:st_intersects (?geo, bif:st_point (2.3498, 48.853), 0.3) )
  }
```

```
LIMIT 20;

s                                                    cn
VARCHAR                                              VARCHAR
_____________________________________________________________________________________________________

http://linkedgeodata.org/triplify/node/237004656#id    Parking Lagrange
http://linkedgeodata.org/triplify/node/237003117#id    Mus+?-?e de l'Assistance Publique H+?-opitaux de P
http://linkedgeodata.org/triplify/way/23071565#id      Jardin de la Rue de Bi+?-?vre
http://linkedgeodata.org/triplify/node/251652818#id    Maitre Albert B&B
http://linkedgeodata.org/triplify/node/251373384#id    Le Grenier de Notre Dame
http://linkedgeodata.org/triplify/node/205266764#id    Eglise Saint-Julien-le-Pauvre
http://linkedgeodata.org/triplify/way/19741083#id      Eglise Saint Julien le Pauvre
http://linkedgeodata.org/triplify/node/251474112#id    Polly Magoo
http://linkedgeodata.org/triplify/node/251531803#id    H+?-otel Esmerelda
http://linkedgeodata.org/triplify/node/191031796#id    Point 0 des Routes de France
http://linkedgeodata.org/triplify/way/20444455#id      Square Jean XXIII
http://linkedgeodata.org/triplify/way/19740745#id      Square Ren+?-? Viviani
http://linkedgeodata.org/triplify/node/321932192#id    Le Saint R+?-?gis
http://linkedgeodata.org/triplify/node/27440965#id     Notre-Dame de Paris
http://linkedgeodata.org/triplify/node/243461762#id    Parking Notre-Dame
http://linkedgeodata.org/triplify/way/21816758#id      Notre-Dame de Paris
http://linkedgeodata.org/triplify/way/22972062#id      La Seine
http://linkedgeodata.org/triplify/way/25463927#id      La Seine
http://linkedgeodata.org/triplify/node/251128395#id    H+?-otel Hospitel
http://linkedgeodata.org/triplify/way/14155323#id      H+?-otel Dieu

20 Rows. -- 167 msec.
```

Example 16

```
## Stuff around Notre Dame de Paris
SQL>
SPARQL
PREFIX lgv: <http://linkedgeodata.org/vocabulary#>
DESCRIBE ?s
WHERE
  {
    ?s lgv:name ?cn  .
    ?s geo:geometry ?geo .
    ?s geo:lat ?lat.
    ?s geo:long ?long.
    FILTER (bif:st_intersects (?geo, bif:st_point (2.3498, 48.853), 0.3))
  }
LIMIT 20;

@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ns1:    <http://linkedgeodata.org/triplify/node/27440966#> .
@prefix ns2:    <http://linkedgeodata.org/vocabulary#> .
ns1:id  rdf:type        ns2:node ,
                ns2:police .
@prefix geo:    <http://www.w3.org/2003/01/geo/wgs84_pos#> .
ns1:id  geo:lat 48.8542 ;
        geo:long        2.3473 ;
        ns2:created_by  "Potlatch 0.6a" ;
        ns2:name        "Pr\u00C3\u00A9fecture de Police de Paris" ,
                "Pr\u00E9fecture de Police de Paris" .
@prefix virtrdf:        <http://www.openlinksw.com/schemas/virtrdf#> .
ns1:id  geo:geometry    "POINT(2.3473 48.8542)"^^virtrdf:Geometry .
@prefix ns5:    <http://linkedgeodata.org/triplify/node/27440965#> .
ns5:id  rdf:type        ns2:node ,
                ns2:place_of_worship ;
        geo:lat 48.853 ;
        geo:long        2.3499 ;
        ns2:denomination        "catholic" ;
        ns2:name        "Notre-Dame de Paris" ;
        ns2:religion    "christian" ,
                ns2:christian ;
        geo:geometry    "POINT(2.3499 48.853)"^^virtrdf:Geometry .

......
```

Example 17

```
## Cities within 30 km proximity of London
SQL>
SPARQL
SELECT ?m (bif:round(bif:st_distance (?geo, ?gm)))
WHERE
  {
    <http://dbpedia.org/resource/London> geo:geometry ?gm .
    ?m geo:geometry ?geo .
    ?m a <http://dbpedia.org/ontology/City> .
    FILTER (bif:st_intersects (?geo, ?gm, 30))
  }
ORDER BY DESC 2
LIMIT 20;

m                                                    callret-1
VARCHAR                                              VARCHAR
_____________________________________________________________

http://dbpedia.org/resource/Bletchingley             30
http://dbpedia.org/resource/Bletchingley             30
http://dbpedia.org/resource/Bletchingley             30
http://dbpedia.org/resource/Bletchingley             30
http://dbpedia.org/resource/Bletchingley             30
http://dbpedia.org/resource/Bletchingley             30
http://dbpedia.org/resource/Bletchingley             30
http://dbpedia.org/resource/Bletchingley             30
http://dbpedia.org/resource/Bletchingley             30
http://dbpedia.org/resource/Bletchingley             30
http://dbpedia.org/resource/Bletchingley             30
http://dbpedia.org/resource/Bletchingley             30
http://dbpedia.org/resource/Bletchingley             30
http://dbpedia.org/resource/Bletchingley             30
http://dbpedia.org/resource/Bletchingley             30
http://dbpedia.org/resource/Ebbsfleet_Valley         30
http://dbpedia.org/resource/Bletchingley             30
http://dbpedia.org/resource/Bletchingley             30
http://dbpedia.org/resource/Bletchingley             30
http://dbpedia.org/resource/Bletchingley             30

20 Rows. -- 727666 msec.
```

Example 18

```
## Motorways across England & Scotland from DBpedia
SQL>
SPARQL
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbpprop: <http://dbpedia.org/property/>
PREFIX yago: <http://dbpedia.org/class/yago/>
SELECT ?road ?services ?lat ?long
WHERE
  {
    {
      ?services dbpprop:road ?road .
      ?road a yago:MotorwaysInEngland .
      ?services dbpprop:lat ?lat .
      ?services dbpprop:long ?long .
     }
    UNION
    {
      ?services dbpprop:road ?road .
      ?road a yago:MotorwaysInScotland .
      ?services dbpprop:lat ?lat .
      ?services dbpprop:long ?long .
     }
  }
LIMIT 20;

road                                          services
```

```
VARCHAR                                          VARCHAR
_________________________________________________________________________________

http://dbpedia.org/resource/M90_motorway         http://dbpedia.org/resource/Kinross_services
http://dbpedia.org/resource/M1_motorway          http://dbpedia.org/resource/Leicester_Forest_East_services
http://dbpedia.org/resource/M1_motorway          http://dbpedia.org/resource/Woodall_services
http://dbpedia.org/resource/M1_motorway          http://dbpedia.org/resource/Tibshelf_services
http://dbpedia.org/resource/M1_motorway          http://dbpedia.org/resource/London_Gateway_services
http://dbpedia.org/resource/M1_motorway          http://dbpedia.org/resource/Donington_Park_services
http://dbpedia.org/resource/M1_motorway          http://dbpedia.org/resource/Watford_Gap_services
http://dbpedia.org/resource/M1_motorway          http://dbpedia.org/resource/Newport_Pagnell_services
http://dbpedia.org/resource/M1_motorway          http://dbpedia.org/resource/Trowell_services
http://dbpedia.org/resource/M1_motorway          http://dbpedia.org/resource/Woolley_Edge_services
http://dbpedia.org/resource/M1_motorway          http://dbpedia.org/resource/Toddington_services
http://dbpedia.org/resource/M1_motorway          http://dbpedia.org/resource/Northampton_services
http://dbpedia.org/resource/M4_motorway          http://dbpedia.org/resource/Chieveley_services
http://dbpedia.org/resource/M4_motorway          http://dbpedia.org/resource/Magor_services
http://dbpedia.org/resource/M4_motorway          http://dbpedia.org/resource/Pont_Abraham_services
http://dbpedia.org/resource/M4_motorway          http://dbpedia.org/resource/Swansea_services
http://dbpedia.org/resource/M4_motorway          http://dbpedia.org/resource/Leigh_Delamere_services
http://dbpedia.org/resource/M4_motorway          http://dbpedia.org/resource/Reading_services
http://dbpedia.org/resource/M4_motorway          http://dbpedia.org/resource/Cardiff_West_services
http://dbpedia.org/resource/M4_motorway          http://dbpedia.org/resource/Heston_services

20 Rows. -- 531 msec.
```

Example 19

```
SELECT DISTINCT ?s (bif:round(?lat)) as ?lat (bif:round(?long)) as ?long
WHERE
  {
    {
      SELECT ?g ?s WHERE
        {
          graph ?g {
            ?s geo:geometry ?geo }
        }
      LIMIT 100
    }
    graph ?g {
      ?s geo:lat ?lat .
      ?s geo:long ?long . }
    FILTER (datatype (?lat) in (xsd:integer, xsd:float, xsd:double)) .
    FILTER (datatype (?long) in (xsd:integer, xsd:float, xsd:double))
  }
```

```
s                                                                                lat        long
ANY                                                                              ANY        ANY
_____________________________________________________________________________________________________

http://dbpedia.org/resource/QUaD                                                 -90        -139
http://dbpedia.org/resource/Amundsen-Scott_South_Pole_Station                    -90        -139
http://dbpedia.org/resource/Amundsen-Scott_South_Pole_Station                    -90        0
http://dbpedia.org/resource/Degree_Angular_Scale_Interferometer                  -90        -139
http://dbpedia.org/resource/South_Pole_Telescope                                 -90        -139
http://dbpedia.org/resource/Arcminute_Cosmology_Bolometer_Array_Receiver         -90        -139
http://dbpedia.org/resource/Viper_telescope                                      -90        -139
http://dbpedia.org/resource/Mount_Weaver                                         -87        -154
http://dbpedia.org/resource/Axel_Heiberg_Glacier                                 -85        -163
http://dbpedia.org/resource/Mount_Ray                                            -85        -171
http://linkedgeodata.org/triplify/node/275487234#id                              -85        -142
http://linkedgeodata.org/triplify/node/303732928#id                              -85        -142
http://linkedgeodata.org/triplify/node/332036611#id                              -85        -85
http://linkedgeodata.org/triplify/node/303732935#id                              -85        -143
http://linkedgeodata.org/triplify/node/303732951#id                              -85        -144
http://linkedgeodata.org/triplify/node/303732953#id                              -85        -144
http://linkedgeodata.org/triplify/node/276208684#id                              -85        -166
```

Example 19

```
## "Find things within 20km of New York City":
```

```
SELECT DISTINCT ?resource ?label ?location
WHERE
  {
    <http://dbpedia.org/resource/New_York_City>
        geo:geometry ?sourcegeo                    .
    ?resource geo:geometry ?location               ;
                rdfs:label ?label .
    FILTER( bif:st_intersects( ?location, ?sourcegeo, 20 ) ) .
    FILTER( lang(?label) = "en" )
  }
```

Example 20

```
## "Find Distance between New York City
## and London, England":

SELECT ( bif:st_distance( ?nyl,?ln ) )
      AS ?distanceBetweenNewYorkCityAndLondon
WHERE
  {
    <http://dbpedia.org/resource/New_York_City>
        geo:geometry ?nyl .
    <http://dbpedia.org/resource/London>
        geo:geometry ?ln  .
  }
```

Example 21

```
## "Find "All Educational Institutions
## within 10km of Oxford, UK; ordered by
## date of establishment":

SELECT DISTINCT ?thing AS ?uri
      ?thingLabel AS ?name
      ?date AS ?established
      ?matchgeo AS ?location
WHERE
  {
    <http://dbpedia.org/resource/Oxford>
             geo:geometry ?sourcegeo .
    ?resource geo:geometry ?matchgeo .
    FILTER( bif:st_intersects( ?matchgeo, ?sourcegeo, 5 ) ) .
       ?thing ?somelink ?resource                          ;
       <http://dbpedia.org/ontology/established> ?date     ;
       rdfs:label ?thingLabel                              .
    FILTER( lang(?thingLabel) = "en" )
  }
ORDER BY ASC( ?date )
```

Example 22

```
## "Find Historical cross section of events related
## to Edinburgh and the surrounding area (within 30km)
## during the 19th century":

SELECT DISTINCT ?thing ?thingLabel
      ?dateMeaningLabel ?date ?matchgeo
WHERE
  {
    {
      SELECT DISTINCT ?thing ?matchgeo
      WHERE
        {
          <http://dbpedia.org/resource/Edinburgh>
              geo:geometry ?sourcegeo .
            ?resource geo:geometry ?matchgeo .
            FILTER( bif:st_intersects (
                ?matchgeo, ?sourcegeo, 30 ) ) .
            ?thing ?somelink ?resource
        }
    }
    {
```

```
      ?property rdf:type owl:DatatypeProperty ;
                         rdfs:range xsd:date
    } .
    ?thing ?dateMeaning ?date .
    FILTER( ?dateMeaning IN ( ?property ) ) .
    FILTER( ?date >= xsd:gYear("1800")
         && ?date <= xsd:gYear("1900") )
    ?dateMeaning rdfs:label ?dateMeaningLabel .
  FILTER( lang(?dateMeaningLabel) = "en" ) .
  ?thing rdfs:label ?thingLabel .
  FILTER( lang(?thingLabel) = "en" )
  }
ORDER BY ASC ( ?date )
```

16.16. RDF Replication

Tables of RDF storage, such as DB.DBA.RDF_QUAD and DB.DBA.RDF_OBJ, can not be replicated in a usual way, because it's content is cached in memory in special ways and synchronized with values outside these tables, such as current values of special sequence objects.

Moreover, same IRI may have different internal IRI_IDs on different boxes, because the assigned IDs vary if new IRIs appear in data in different order. Similarly, there will be different IDs of RDF literal, datatypes and languages, blocking any attempt of one-to-one replication between RDF storages.

However, a special asynchronous RDF replication makes it possible to configure a "publisher" Virtuoso instance to keep the log of changes in some RDF graphs and subscribe some Virtuoso instances to replay all these changes.

Configuration functions are quite straightforward.

RDF graphs to replicate are all members of <http://www.openlinksw.com/schemas/virtrdf#rdf_repl_graph_group> graph group. That group can be filled in with graphs like any other graph group, but it is better to get the advantage of proper security check made by `DB.DBA.RDF_REPL_GRAPH_INS()` that inserts a graph to the group and `DB.DBA.RDF_REPL_GRAPH_DEL()` that removes a graph from the group.

Only publicly readable graphs can be replicated, an error is signalled otherwise, and it is better to know about a security issue as early as possible.

The `DB.DBA.RDF_REPL_START()` function starts the RDF replication at the publishing side. It creates replication "publication" named '__rdf_repl' and makes a log file '__rdf_repl.log' to record changes in replicated graphs. If the replication has been started before then an error is signalled; passing value 1 for parameter "quiet" elimintaes the error so the incorrect call has no effect at all. If the replication is enabled then the value of registry variable 'DB.DBA.RDF_REPL' indicates the moment of replication start.

The `DB.DBA.RDF_REPL_START()` function performs a security check before starting the replication to check.

The `DB.DBA.RDF_REPL_STOP()` stops the RDF replication at the publishing side. It calls `repl_unpublish()` but does not make empty reates replication "publication" named '__rdf_repl' and makes a log file '__rdf_repl.log' to record changes in replicated graphs.

Replication is asynchronous and the order of insertion and removal operations at the subscriber's side may not match the order at the publisher. As a result, it is not recommended to make few subscriptions that writes changes of few publishers into one common graph. A client-side application can force the synchronuzation by calling `DB.DBA.RDF_REPL_SYNC()` that acts like `repl_sync()` but for an RDF subscription. `DB.DBA.RDF_REPL_SYNC()` will not only initial synchronisation but also wait for the end of subscription to guarantee that the total effect of INSERT and DELETE operations is correct even if these operations were made in an order that differs from the original one.

16.17. RDF Performance Tuning

For RDF query performance, we have the following possible questions:

- Is the Virtuoso process properly configured to handle big data sets?
- Is the graph always specified?

- Are public web service endpoints protected against bad queries?
- Are there patterns where only a predicate is given?
- Is there a bad query plan because of cost model error?

16.17.1. General

When running with large data sets, one should configure the Virtuoso process to use between 2/3 to 3/5 of system RAM and to stripe storage on all available disks. See NumberOfBuffers , MaxDirtyBuffers , and Striping INI file parameters.

```
; default installation
NumberOfBuffers          = 2000
MaxDirtyBuffers          = 1200
```

Typical sizes for the NumberOfBuffers and MaxDirtyBuffers (3/4 of NumberOfBuffers) parameters in the Virtuoso configuration file (virtuoso.ini) for various memory sizes are as follows, with each buffer consisting of 8K bytes:

Table 16.19. recommended NumberOfBUffers and MaxDirtyBuffers

System RAM	NumberOfBuffers	MaxDirtyBuffers
2 GB	170000	130000
4 GB	340000	250000
8 GB	680000	500000
16 GB	1360000	1000000
32 GB	2720000	2000000
48 GB	4000000	3000000
64 GB	5450000	4000000

Also, if running with a large database, setting MaxCheckpointRemap to 1/4th of the database size is recommended. This is in pages, 8K per page.

16.17.2. RDF Index Scheme

Starting with version 6.00.3126, the default RDF index scheme consists of 2 full indices over RDF quads plus 3 partial indices. This index scheme is generally adapted to all kinds of workloads, regardless of whether queries generally specify a graph. As indicated the default index scheme in Virtuoso is almost always applicable as is, whether one has a RDF database with very large numbers of small graphs or just one or a few large graphs. With Virtuoso 7 the indices are column-wise by default, which results in them to consuming usually about 1/3 of the space the equivalent row-wise structures would consume.

Alternate indexing schemes are possible but will not be generally needed. For upgrading old databases with a different index scheme see the corresponding documentation.

The index scheme consists of the following indices:

- *PSOG*

  - primary key
- *POGS*

  - bitmap index for lookups on object value.
- *SP*

  - partial index for cases where only S is specified.
- *OP*

  - partial index for cases where only O is specified.
- *GS*

  - partial index for cases where only G is specified.

This index scheme is created by the following statements:

```
CREATE TABLE DB.DBA.RDF_QUAD (
  G IRI_ID_8,
  S IRI_ID_8,
  P IRI_ID_8,
  O ANY,
  PRIMARY KEY (P, S, O, G)
  )
ALTER INDEX RDF_QUAD ON DB.DBA.RDF_QUAD
  PARTITION (S INT (0hexffff00));

CREATE DISTINCT NO PRIMARY KEY REF BITMAP INDEX RDF_QUAD_SP
  ON RDF_QUAD (S, P)
  PARTITION (S INT (0hexffff00));

CREATE BITMAP INDEX RDF_QUAD_POGS
  ON RDF_QUAD (P, O, G, S)
  PARTITION (O VARCHAR (-1, 0hexffff));

CREATE DISTINCT NO PRIMARY KEY REF BITMAP INDEX RDF_QUAD_GS
  ON RDF_QUAD (G, S)
  PARTITION (S INT (0hexffff00));

CREATE DISTINCT NO PRIMARY KEY REF INDEX RDF_QUAD_OP
  ON RDF_QUAD (O, P)
  PARTITION (O VARCHAR (-1, 0hexffff));
```

The idea is to favor queries where the predicate is specified in triple patterns. The entire quad can be efficiently accessed when `P` and at least one of `S` and `O` are known. This has the advantage of clustering data by the predicate which improves working set. A page read from disk will only have entries pertaining to the same predicate; chances of accessing other entries of the page are thus higher than if the page held values for arbitrary predicates. For less frequent cases where only `S` is known, as in `DESCRIBE` , the distinct `P` s of the `S` are found in the `SP` index. These `SP` pairs are then used for accessing the `PSOG` index to get the `O` and `G` . For cases where only the `G` is known, as when dropping a graph, the distinct `S` s of the `G` are found in the `GS` index. The `P` s of the `S` are then found in the `SP` index. After this, the whole quad is found in the `PSOG` index.

The `SP` , `OP` , and `GS` indices do not store duplicates. If an `S` has many values of the `P` , there is only one entry. Entries are not deleted from `SP` , `OP` , or `GS` . This does not lead to erroneous results since a full index (that is, either `POSG` or `PSOG`) is always consulted in order to know if a quad actually exists. When updating data, most often a graph is entirely dropped and a substantially similar graph inserted in its place. The `SP` , `OP` , and `GS` indices get to stay relatively unaffected.

Still, over time, especially if there are frequent updates and values do not repeat between consecutive states, the `SP` , `OP` , and `GS` indices will get polluted, which may affect performance. Dropping and recreating the index will remedy this situation.

In cases where this is not practical, the index scheme should only have full indices; i.e., each key holds all columns of the primary key of the quad. This will be the case if the `DISTINCT NO PRIMARY KEY REF` options are not specified in the `CREATE INDEX` statement. In such cases, all indices remain in strict sync across deletes.

Many RDF workloads have bulk-load and read-intensive access patterns with few deletes. The default index scheme is optimized for these. With these situations, this scheme offers significant space savings, resulting in better working set. Typically, this layout takes 60-70% of the space of a layout with 4 full indices.

16.17.3. Index Scheme Selection

The indexes in place on the `RDF_QUAD table` can greatly affect the performance of SPARQL queries, as can be determined by running the `STATISTICS` command on the table as follows:

```
SQL> STATISTICS DB.DBA.RDF_QUAD;
Showing SQLStatistics of table(s) 'DB.DBA.RDF_QUAD'
TABLE_QUALIFIER  TABLE_OWNER      TABLE_NAME       NON_UNIQUE  INDEX_QUALIFIER  INDEX_NAME       TYPE
VARCHAR          VARCHAR          VARCHAR          SMALLINT    VARCHAR          VARCHAR          SMALLINT
_______________________________________________________________________________

DB               DBA              RDF_QUAD         NULL        NULL             NULL             0
DB               DBA              RDF_QUAD         0           DB               RDF_QUAD         3
DB               DBA              RDF_QUAD         0           DB               RDF_QUAD         3
DB               DBA              RDF_QUAD         0           DB               RDF_QUAD         3
```

```
DB              DBA             RDF_QUAD        0       DB              RDF_QUAD        3
DB              DBA             RDF_QUAD        1       DB              RDF_QUAD_GS     3
DB              DBA             RDF_QUAD        1       DB              RDF_QUAD_GS     3
DB              DBA             RDF_QUAD        1       DB              RDF_QUAD_OP     3
DB              DBA             RDF_QUAD        1       DB              RDF_QUAD_OP     3
DB              DBA             RDF_QUAD        1       DB              RDF_QUAD_POGS   3
DB              DBA             RDF_QUAD        1       DB              RDF_QUAD_POGS   3
DB              DBA             RDF_QUAD        1       DB              RDF_QUAD_POGS   3
DB              DBA             RDF_QUAD        1       DB              RDF_QUAD_POGS   3
DB              DBA             RDF_QUAD        1       DB              RDF_QUAD_SP     3
DB              DBA             RDF_QUAD        1       DB              RDF_QUAD_SP     3

15 Rows. -- 24 msec.
SQL>
```

With only one index (`OGPS`) created by default, if the graph is always given, as with one or more `FROM` or `FROM NAMED` clauses, and there are no patterns where only graph and predicate are given, then the default indices should be sufficient. If predicate and graph are given but subject is not, then it is sometimes useful to add:

```
CREATE BITMAP INDEX RDF_QUAD_PGOS
  ON DB.DBA.RDF_QUAD (G, P, O, S)
  PARTITION (O VARCHAR (-1, 0hexffff));
```

Note: If the server version is pre-5.0.7, leave out the partitioning clause.

Making the `PGOS` index can help in some cases even if it is not readily apparent from the queries that one is needed. This is so, for example, if the predicate by itself is selective; i.e., there is a predicate that occurs in only a few triples.

There is one known application scenario that requires a small alteration to the default index scheme. If the application has a large number of small graphs, e.g. millions of graphs of tens or hundreds of triples each, and it commonly happens that large numbers of graphs contain exactly the same triple, for example the same triple is found in 100000 or one million graphs, then some operations will become inefficient with the default index scheme. In specific, dropping a graph may have to scan through large amounts of data in order to find the right quad to delete from the set of quads that differ only in the graph.

This will affect a graph replace and a graph drop or generally any deletion that falls on a quad of the described sort. If this is the situation in the application, then dropping the `RDF_QUAD_GS` distinct projection and replacing it with a covering index that starts with `G` is appropriate:

```
Drop index RDF_QUAD_GS;
Create column index RDF_QUAD_GPSO on RDF_QUAD (G, P, S, O) partition (S int (0hexffff00);
```

The partition clause only affects cluster settings and is ignored in the single server case. Partitioning on `S` is usually better than on `O` since the distribution of `S` is generally less skewed than that of `O` . That is, there usually are some very common `O` values (e.g. class "thing"). This will increase space consumption by maybe 25% compared to the default scheme.

16.17.4. Manage Public Web Service Endpoints

Public web service endpoints have proven to be sources of especially bad queries. While local application developers can obtain instructions from database administrators and use ISQL access to the database to tune execution plans, "external" clients do not know details of configuration and/or lack appropriate skills. The most common problem is that public endpoints usually get requests that do not mention the required graph, because the queries were initially written for use with triple stores. If the web service provides access to a single graph (or to a short list of graphs), then it is strongly recommended to configure it by adding a row into `DB.DBA.SYS_SPARQL_HOST` .

```
create table DB.DBA.SYS_SPARQL_HOST (
  SH_HOST       varchar not null primary key, -- host mask
  SH_GRAPH_URI varchar,                 -- default graph uri
  SH_USER_URI   varchar,                  -- reserved for any use in applications
  SH_BASE_URI varchar,                  -- for future use (not used currently) to set BASE in sparql quer
  SH_DEFINES long varchar,              -- additional defines for requests
  PRIMARY KEY (SH_HOST)
)
```

You can find detailed descriptions of the table columns here .

The idea is that if the client specifies the default graph in the request, or uses named graphs and group graph patterns, then he is probably smarter than average and will provide meaningful queries. If no graph names are specified, then the query will benefit from preset graph because this will give the compiler some more indexes to choose from -- indexes that begin with `G` .

Sometimes web service endpoints are used to access data of only one application, not all data in the system. In that case, one may wish to declare a separate storage that consists of only RDF Views made by that application and `define input:storage` in the appropriate row of `DB.DBA.SYS_SPARQL_HOST` .

16.17.5. Erroneous Cost Estimates and Explicit Join Order

The selectivity of triple patterns is determined at query compile time from sampling the data. It is possible that misleading data is produced. To see if the cardinality guesses are generally valid, look at the query plan with the `explain` () function.

Below is a sample from the LUBM qualification data set in the Virtuoso distribution. After running *make test* in *binsrc/test/lubm* , there is a loaded database with the data. Start a server in the same directory to see the data.

```
SQL> EXPLAIN
  ('SPARQL
  PREFIX  ub:  <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
  SELECT *
  FROM <lubm>
  WHERE { ?x  rdf:type  ub:GraduateStudent }
  ');

REPORT
VARCHAR
_______________________________________________________________________________

{

Precode:
      0: $25 "callret" := Call __BOX_FLAGS_TWEAK (<constant (lubm)>, <constant (1)>)
      5: $26 "lubm" := Call DB.DBA.RDF_MAKE_IID_OF_QNAME_SAFE ($25 "callret")
      12: $27 "callret" := Call __BOX_FLAGS_TWEAK (<constant (http://www.w3.org/1999/02/22-rdf-syntax-ns#
      17: $28 "-ns#type" := Call DB.DBA.RDF_MAKE_IID_OF_QNAME_SAFE ($27 "callret")
      24: $29 "callret" := Call __BOX_FLAGS_TWEAK (<constant (http://www.lehigh.edu/~zhp2/2004/0401/univ-
      29: $30 "owl#GraduateStudent" := Call DB.DBA.RDF_MAKE_IID_OF_QNAME_SAFE ($29 "callret")
      36: BReturn 0
from DB.DBA.RDF_QUAD by RDF_QUAD_OGPS    1.9e+03 rows
Key RDF_QUAD_OGPS  ASC ($32 "s-3-1-t0.S")
<col=415 O = $30 "owl#GraduateStudent"> , <col=412 G = $26 "lubm"> , <col=414 P = $28 "-ns#type">
row specs: <col=415 O LIKE <constant (T)>>

Current of: <$34 "<DB.DBA.RDF_QUAD s-3-1-t0>" spec 5>

After code:
      0: $35 "x" := Call ID_TO_IRI ($32 "s-3-1-t0.S")
      5: BReturn 0
Select ($35 "x", <$34 "<DB.DBA.RDF_QUAD s-3-1-t0>" spec 5>)
}

22 Rows. -- 1 msec.
```

This finds the graduate student instances in the LUBM graph. First the query converts the IRI literals to IDs. Then, using a match of `OG` on `OGPS` , it finds the IRIs of the graduate students. Then, it converts the IRI ID to return to the string form.

The cardinality estimate of 1.9e+03 rows is on the `FROM` line.

Doing an `EXPLAIN()` on the queries will show the cardinality estimates. To drill down further, one can split the query into smaller chunks and see the estimates for these, up to doing it at the triple pattern level. To indicate a variable that is bound but whose value is not a literal known at compile time, one can use the parameter marker *??* .

```
SQL> EXPLAIN
  ('
      SPARQL
      DEFINE  sql:table-option "order"
      PREFIX  ub:  <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
      SELECT *
```

```
        FROM <lubm>
        WHERE { ?x  rdf:type  ?? }
    ');
```

This will not know the type but will know that a type will be provided. So instead of guessing 1900 matches, this will guess a smaller number, which is obviously less precise. Thus literals are generally better.

In some cases, generally to work around an optimization error, one can specify an explicit *JOIN* order. This is done with the *sql:select-option "order"* clause in the SPARQL query prefix:

```
SQL> SELECT SPARQL_to_sql_text
  ('
      DEFINE sql:select-option "order"
      PREFIX  ub:  <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
      SELECT *
      FROM <lubm>
      WHERE
        {
          ?x  rdf:type        ub:GraduateStudent                                    .
          ?x  ub:takesCourse  <http://www.Department0.University0.edu/GraduateCourse0>
        }
  ');
```

shows the SQL text with the order option at the end.

If an estimate is radically wrong then this should be reported as a bug.

If there is a `FROM` with a `KEY` on the next line and no column specs then this is a full table scan. The more columns are specified the less rows will be passed to the next operation in the chain. In the example above, there are three columns whose values are known before reading the table and these columns are leading columns of the index in use so column specs are:

```
<col=415 O = $30 "owl#GraduateStudent"> ,
<col=412 G = $26 "lubm"> ,
<col=414 P = $28 "-ns#type">
```

Note

Note: A `KEY` with only a row spec is a full table scan with the row spec applied as a filter. This is usually not good unless this is specifically intended.

If queries are compiled to make full table scans when this is not specifically intended, this should be reported as a bug. The `explain ()` output and the query text should be included in the report.

Consider:

```
SQL> EXPLAIN
  ('
      SPARQL
      DEFINE sql:select-option "order, loop"
      PREFIX  ub:  <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
      SELECT *
      FROM <lubm>
      WHERE
        {
          ?x  ub:takesCourse  ?c                  .
          ?x  rdf:type        ub:GraduateStudent
        }
  ');
```

One will see in the output that the first table access is to retrieve all in the LUBM graph which take some course and then later to check if this is a graduate student. This is obviously not the preferred order but the *sql:select-option "order"* forces the optimizer to join from left to right.

It is very easy to end up with completely unworkable query plans in this manner but if the optimizer really is in error, then this is the only way of overriding its preferences. The effect of *sql:select-option* is pervasive, extending inside unions, optionals, subqueries etc. within the statement.

We note that if, in the above query, both the course taken by the student and the type of the student are given, the query compilation will be, at least for all non-cluster cases, an index intersection. This is not overridden by the `sql:select-option` clause since an index intersection is always a safe guess, regardless of the correctness of the cardinality guesses of the patterns involved.

Translate and Analyze modes for analyzing sparql queries

Virtuoso Release 6.4 ISQL offers 2 new modes for analyzing sparql queries:

1. Translate a sparql query into the correspondent sql:

```
SQL> SET SPARQL_TRANSLATE ON;
SQL> SELECT * FROM <graph> WHERE {?S a ?O};
SQL> SET SPARQL_TRANSLATE OFF;
```

2. Analyze a given SQL query:

```
SQL> SET EXPLAIN ON;
SQL> SELECT * FROM TABLE WHERE field = 'text';
SQL> SET EXPLAIN OFF;
```

   - `explain` () is much more difficult to use since you cannot just cut and past a query as all quotes need to be doubled inside the `explain (' ... ')` :

```
SQL> explain('select * from table where field = ''text''');
```

Here is simple example of how to combine the two options to get a full explain plan for a simple SPARQL query:

1. Assume the following query:

```
SELECT *
FROM <http://dbpedia.org>
WHERE
  {
    ?s a ?o
  }
LIMIT 10
```

2. Connect using the ISQL command line tool to your database and execute:

```
SQL> SET BLOBS ON;                        -- in case output is very large
SQL> SET SPARQL_TRANSLATE ON;
SQL> SELECT * FROM <http://dbpedia.org> WHERE {?s a ?o} LIMIT 10;

SPARQL_TO_SQL_TEXT
VARCHAR
_______________________________________________________________________________

SELECT TOP 10 __id2i ( "s_1_0-t0"."S" ) AS "s",
  __ro2sq ( "s_1_0-t0"."O" ) AS "o"
FROM DB.DBA.RDF_QUAD AS "s_1_0-t0"
WHERE "s_1_0-t0"."G" = __i2idn ( __bft( 'http://dbpedia.org' , 1))
  AND  "s_1_0-t0"."P" = __i2idn ( __bft( 'http://www.w3.org/1999/02/22-rdf-syntax-ns#type' , 1))
OPTION (QUIETCAST)

1 Rows. -- 1 msec.

SQL> SET SPARQL_TRANSLATE OFF;
```

3. Use mouse to select the above query output and paste it after the `SET EXPLAIN ON;` command. After pasting in the command, followed by the ENTER key:

```
SQL> SET EXPLAIN ON;
SQL> SELECT TOP 10 __id2i ( "s_1_0-t0"."S" ) AS "s", __ro2sq ( "s_1_0-t0"."O" ) AS "o"
 FROM DB.DBA.RDF_QUAD AS "s_1_0-t0"
 WHERE "s_1_0-t0"."G" = __i2idn ( __bft( 'http://dbpedia.org' , 1))
   AND  "s_1_0-t0"."P" = __i2idn ( __bft( 'http://www.w3.org/1999/02/22-rdf-syn tax-ns#type' , 1))
 OPTION (QUIETCAST)
;

REPORT
```

```
VARCHAR
_______________________________________________________________________________

{
from DB.DBA.RDF_QUAD by RDF_QUAD_POGS    4.5e+05 rows
Key RDF_QUAD_POGS  ASC ($22 "s_1_0-t0.S", $21 "s_1_0-t0.O")
 inlined <col=556 P =  #type >
row specs: <col=554 G =  #http://dbpedia.org >

After code:
      0: $25 "s" := Call __id2i ($22 "s_1_0-t0.S")
      5: $26 "o" := Call __ro2sq ($21 "s_1_0-t0.O")
      10: BReturn 0
Select (TOP  10 ) ($25 "s", $26 "o", <$24 "<DB.DBA.RDF_QUAD s_1_0-t0>" spec 5>)
}

13 Rows. -- 1 msec.

SQL> SET EXPLAIN OFF;
```

16.17.6. Using "swappiness" parameter (Linux only)

For Linux users only , there is a kernel tuning parameter called "`swappiness` " that controls how much the kernel favors swap over RAM.

When hosting large data sets, it is recommended that this parameter be changed from its default value of 60 to something closer to 10, to reduce the amount of swapping that takes place on the server. Useful tidbits regarding swappiness include:

- The swappiness setting is found in the file `/proc/sys/vm/swappiness` .
- The command `/sbin/sysctl vm.swappiness` can be used to view its setting.
- The command `/sbin/sysctl -w vm.swappiness=10` can be used to change its value.
- Adding `vm.swappiness = 10` to the file `/etc/sysctl.conf` will force the value to be set at machine boot time.

16.17.7. Get All Graphs

In order to get all graphs URIs, one might use the Virtuoso `DB.DBA.SPARQL_SELECT_KNOWN_GRAPHS()` built-in function.

16.17.8. Rename RDF Graph and RDF Graph Groups

A RDF Graph in the Virtuoso Quad Store can be renamed without copying each assertion from the old graph to the new graph using a SQL statement, this being what the Conductor "rename" option does, which is:

```
UPDATE DB.DBA.RDF_QUAD TABLE OPTION (index RDF_QUAD_GS)
   SET g = iri_to_id ('new')
 WHERE g = iri_to_id ('old', 0);
```

Note: this operation must be run in row-autocommit mode i.e. log_enable (3), and then restore back to the default logging mode of 1.

For Virtuoso Graph Groups two tables need to be updated:

```
UPDATE DB.DBA.RDF_GRAPH_GROUP_MEMBER
   SET RGGM_GROUP_IID = iri_to_id ('new')
 WHERE RGGM_GROUP_IID = iri_to_id (old)
```

and

```
UPDATE DB.DBA.RDF_GRAPH_GROUP
   SET RGG_IID = iri_to_id ('new') , RGG_IRI = 'new'
 WHERE RGG_IRI = 'old'
```

16.17.9. Dump and Reload Graphs

What?

How to export RDF model data from Virtuoso's Quad Store.

Why?

Every DBMS needs to offer a mechanism for bulk export and import of data.

Virtuoso supports dumping and reloading graph model data (e.g., RDF), as well as relational data (e.g., SQL) (discussed elsewhere).

How?

We have created stored procedures for the task. The dump procedures leverage SPARQL to facilitate selective data dump(s) from one or more Named Graphs, each denoted by an IRI.

Dump One Graph

The procedure dump_one_graph can be used to dump any single Named Graph.

Parameters

- IN

  srcgraph

  VARCHAR -- source graph
- IN

  out_file

  VARCHAR -- output file
- IN

  file_length_limit

  INTEGER -- maximum length of dump files

Procedure source

```
CREATE PROCEDURE dump_one_graph
  ( IN  srcgraph           VARCHAR
  , IN  out_file           VARCHAR
  , IN  file_length_limit  INTEGER  := 1000000000
  )
  {
    DECLARE  file_name     VARCHAR;
    DECLARE  env,  ses           ANY;
    DECLARE  ses_len
          ,  max_ses_len
          ,  file_len
          ,  file_idx      INTEGER;
   SET ISOLATION = 'uncommitted';
   max_ses_len  := 10000000;
   file_len     := 0;
   file_idx     := 1;
   file_name    := sprintf ('%s%06d.ttl', out_file, file_idx);
   string_to_file ( file_name || '.graph',
                    srcgraph,
                    -2
                  );
    string_to_file ( file_name,
                     sprintf ( '# Dump of graph <%s>, as of %s\n@base <> .\n',
                               srcgraph,
                               CAST (NOW() AS VARCHAR)
                             ),
```

```
                    -2
                  );
    env := vector (dict_new (16000), 0, '', '', '', 0, 0, 0, 0, 0);
    ses := string_output ();
    FOR (SELECT * FROM ( SPARQL DEFINE input:storage ""
                         SELECT ?s ?p ?o { GRAPH `iri(?:srcgraph)` { ?s ?p ?o } }
                       ) AS sub OPTION (LOOP)) DO
       {
         http_ttl_triple (env, "s", "p", "o", ses);
         ses_len := length (ses);
         IF (ses_len > max_ses_len)
           {
             file_len := file_len + ses_len;
             IF (file_len > file_length_limit)
               {
                 http (' .\n', ses);
                 string_to_file (file_name, ses, -1);
                 gz_compress_file (file_name, file_name||'.gz');
                 file_delete (file_name);
                 file_len := 0;
                 file_idx := file_idx + 1;
                 file_name := sprintf ('%s%06d.ttl', out_file, file_idx);
                 string_to_file ( file_name,
                                  sprintf ( '# Dump of graph <%s>, as of %s (part %d)\n@base <> .\n',
                                            srcgraph,
                                            CAST (NOW() AS VARCHAR),
                                            file_idx),
                                  -2
                                );
                  env := VECTOR (dict_new (16000), 0, '', '', '', 0, 0, 0, 0, 0);
               }
             ELSE
               string_to_file (file_name, ses, -1);
             ses := string_output ();
           }
       }
     IF (LENGTH (ses))
       {
         http (' .\n', ses);
         string_to_file (file_name, ses, -1);
         gz_compress_file (file_name, file_name||'.gz');
         file_delete (file_name);
       }
   }
;
```

To load the procedure into Virtuoso, it can simply be copied and pasted into the Virtuoso `isql` command line tool or Interactive SQL UI of the Conductor and executed.

Example

Call the `dump_one_graph` procedure with appropriate arguments:

```
$ pwd
/Applications/OpenLink Virtuoso/Virtuoso 6.1/database

$ grep DirsAllowed virtuoso.ini
DirsAllowed              = ., ../vad,

$ /opt/virtuoso/bin/isql 1111
Connected to OpenLink Virtuoso
Driver: 06.01.3127 OpenLink Virtuoso ODBC Driver
OpenLink Interactive SQL (Virtuoso), version 0.9849b.
Type HELP; for help and EXIT; to exit.
SQL> dump_one_graph ('http://daas.openlinksw.com/data#', './data_', 1000000000);
Done. -- 1438 msec.

$
As a result, a dump of the graph <http://daas.openlinksw.com/data#> will be found in the files data_XX
```

```
$ ls
data_000001.ttl
data_000002.ttl
....
data_000001.ttl.graph
```

Dump Multiple Graphs

The *dump_graphs* procedure can be used to dump all the graphs in a Virtuoso server to a set of turtle (.ttl) data files in the specified dump directory.

Parameters

The procedure dump_graphs has the following parameters:

- IN

  dir

  VARCHAR -- dump directory
- IN

  file_length_limit

  INTEGER -- maximum length of dump files

Note: The dump directory must be included in the DirsAllowed parameter of the Virtuoso configuration file (virtuoso.ini), or the Virtuoso server will not be able to create or access the data files.

Source

The procedure *dump_graphs* has the following source:

```
CREATE PROCEDURE dump_graphs
  ( IN  dir               VARCHAR  :=  'dumps'   ,
    IN  file_length_limit INTEGER  :=  1000000000
  )
  {
    DECLARE inx INT;
    inx := 1;
    SET ISOLATION = 'uncommitted';
    FOR ( SELECT *
            FROM ( SPARQL DEFINE input:storage ""
                   SELECT DISTINCT ?g { GRAPH ?g { ?s ?p ?o } .
                                        FILTER ( ?g != virtrdf: )
                                      }
                 ) AS sub OPTION ( LOOP )) DO
      {
        dump_one_graph ( "g",
                         sprintf ('%s/graph%06d_', dir, inx),
                         file_length_limit
                       );
        inx := inx + 1;
      }
  }
;
```

Example

1. Call the

   dump_graphs

   procedure:

   ```
   $ pwd
   ```

```
/Applications/OpenLink Virtuoso/Virtuoso 6.1/database

$ grep DirsAllowed virtuoso.ini
DirsAllowed              = ., ../vad, ./dumps

$ /opt/virtuoso/bin/isql 1111
Connected to OpenLink Virtuoso
Driver: 06.01.3127 OpenLink Virtuoso ODBC Driver
OpenLink Interactive SQL (Virtuoso), version 0.9849b.
Type HELP; for help and EXIT; to exit.
SQL> dump_graphs();

Done. -- 998 msec.
SQL> quit;
```

2. As a result, a dump of the graph will be found in the files dumps/data_XX (located in your Virtuoso db folder):

```
<verbatim>
$ ls dumps
graph000001_000001.ttl          graph000005_000001.ttl
graph000001_000001.ttl.graph    graph000005_000001.ttl.graph
graph000002_000001.ttl          graph000006_000001.ttl
graph000002_000001.ttl.graph    graph000006_000001.ttl.graph
graph000003_000001.ttl          graph000007_000001.ttl
graph000003_000001.ttl.graph    graph000007_000001.ttl.graph
graph000004_000001.ttl          graph000008_000001.ttl
graph000004_000001.ttl.graph    graph000008_000001.ttl.graph
```

Load Graphs

The stored procedure *load_graphs* procedure performs a bulk load from a file.

Parameters

The procedure load_graphs has the following parameters:

- IN

  dir

  VARCHAR -- dump directory

Note: The dump directory must be included in the DirsAllowed parameter of the Virtuoso configuration file (virtuoso.ini), or the Virtuoso server will not be able to create or access the data files.

Source

The procedure *load_graphs* has the following source:

```
CREATE PROCEDURE load_graphs
  ( IN  dir  VARCHAR := 'dumps/' )
{
  DECLARE arr ANY;
  DECLARE g VARCHAR;

  arr := sys_dirlist (dir, 1);
  log_enable (2, 1);
  FOREACH (VARCHAR f IN arr) DO
    {
      IF (f LIKE '*.ttl')
        {
          DECLARE CONTINUE HANDLER FOR SQLSTATE '*'
            {
              log_message (sprintf ('Error in %s', f));
            };
          g := file_to_string (dir || '/' || f || '.graph');
          DB.DBA.TTLP_MT (file_open (dir || '/' || f), g, g, 255);
        }
    }
  EXEC ('CHECKPOINT');
}
```

```
;
```

Example

```
$ /opt/virtuoso/bin/isql 1112
Connected to OpenLink Virtuoso
Driver: 06.01.3127 OpenLink Virtuoso ODBC Driver
OpenLink Interactive SQL (Virtuoso), version 0.9849b.
Type HELP; for help and EXIT; to exit.
SQL> load_graphs();

Done. -- 2392 msec.
SQL>
```

16.17.10. RDF dumps from Virtuoso Quad store hosted data into NQuad dumps

What?

How to export RDF model data from Virtuoso's Quad Store in NQuad format.

Why?

When exporting RDF model data from Virtuoso's Quad Store, having the ability to retain and reflect Named Graph IRI based data partitioning is provide significant value to a variety of application profiles.

How?

We have created stored procedures for the task. The dump procedure *dump_nquads* leverage SPARQL to facilitate data dump(s) for all graphs excluding the predefined "virtrdf:" one.

Dump NQuads

Parameters

The procedure *dump_nquads* has the following parameters:

- IN

  dir

  VARCHAR -- folder where the dumps will be stored
- IN

  outstart_fromfile

  INTEGER -- output start from number n
- IN

  file_length_limit

  INTEGER -- maximum length of dump files
- IN

  comp

  INTEGER -- when set to 0, then no gzip will be done. By default is set to 1.

Source

The procedure *dump_nquads* has the following source:

```
create procedure dump_nquads (in dir varchar := 'dumps', in start_from int := 1, in file_length_limit int
{
  declare inx, ses_len int;
```

```
  declare file_name varchar;
  declare env, ses any;

  inx := start_from;
  set isolation = 'uncommitted';
  env := vector (0,0,0);
  ses := string_output (10000000);
  for (select * from (sparql define input:storage "" select ?s ?p ?o ?g { graph ?g { ?s ?p ?o } . filter
    {
      declare exit handler for sqlstate '22023'
        {
          goto next;
        };
      http_nquad (env, "s", "p", "o", "g", ses);
      ses_len := length (ses);
      if (ses_len >= file_length_limit)
        {
          file_name := sprintf ('%s/output%06d.nq', dir, inx);
          string_to_file (file_name, ses, -2);
          if (comp)
            {
              gz_compress_file (file_name, file_name||'.gz');
              file_delete (file_name);
            }
          inx := inx + 1;
          env := vector (0,0,0);
          ses := string_output (10000000);
        }
      next:;
    }
  if (length (ses))
    {
      file_name := sprintf ('%s/output%06d.nq', dir, inx);
      string_to_file (file_name, ses, -2);
      if (comp)
        {
          gz_compress_file (file_name, file_name||'.gz');
          file_delete (file_name);
        }
      inx := inx + 1;
      env := vector (0,0,0);
    }
}
;
```

Example

This example demonstrates calling the *dump_nquads* procedure in order to dump all graphs in a compressed nquad dumps, each uncompressed with length 10Mb (./dumps/output000001.nq.gz) :

```
SQL> dump_nquads ('dumps', 1, 10000000, 1);
```

16.17.11. Dump Linked Data View Graph to n3

The RDF_QM_TREE_DUMP procedure and its associated procedures below are used for dumping one or more RDFView Graphs in a Virtuoso server to a set of turtle ttl dataset files in the specified dump directory. The dump generation is made as fast as possible by grouping mappings by underlying tables so many properties from neighbor database columns can be extracted in one table scan. The size of the generated files is limited to 5MB. The dump process creates internal stored procedures; their texts are saved in file .dump_procedures.sql in the directory of dump files for debugging purposes.

Note that the dump directory must be included in the `DirsAllowed` parameter of the Virtuoso configuration file (e.g., `virtuoso.ini`), or the server will not be allowed to create nor access the dataset file(s).

The Virtuoso RDF bulk loader scripts can then be used to load the dumped datasets for the RDFView graphs directly into a Virtuoso RDF QUAD store.

Parameters

- in *dest_dir*

  VARCHAR - dump directory
- in *graph_iri*

  VARCHAR - IRI of the graph to be dumped; triples from other graphs will be excluded. If NULL, then there's no restriction by graph.
- in *storage*

  VARCHAR - IRI of the quad map storage to use. NULL means use default storage.
- in *root*

  VARCHAR - IRI of the quad map to use, e.g., an IRI of an Linked Data View (or its part). NULL means use all Linked Data Views of the storage (and the default mapping as well).

Procedure Code

```
CREATE PROCEDURE DB.DBA.RDF_QM_TREE_DUMP
  ( in  dest_dir  VARCHAR,
    in  graph_iri VARCHAR := NULL,
    in  storage   VARCHAR := NULL,
    in  root      VARCHAR := NULL
  )
{
 DECLARE all_qms,
         grouped_qmvs,
         launcher_text  ANY;
 DECLARE grp_ctr,
         qm_ctr,
         qm_count       INTEGER;
 DECLARE sql_file,
         launcher_name  VARCHAR;
 IF (NOT (dest_dir LIKE '%/'))
   dest_dir := dest_dir || '/';
 sql_file := dest_dir || '.dump_procedures.sql';
 IF (storage IS NULL)
   storage := 'http://www.openlinksw.com/schemas/virtrdf#DefaultQuadStorage';
 string_to_file (
   sql_file,
   '-- This file contains procedure created by DB.DBA.RDF_QM_TREE_DUMP() for storage '
      || COALESCE (storage, 'NULL')
      || ' and root quad map '
      || COALESCE (root, 'NULL')
      || '\n\n',
   -2);
 all_qms := dict_list_keys (DB.DBA.RDF_QM_CONTENT_OF_QM_TREE (graph_iri, storage, root), 2);
 grouped_qmvs := DB.DBA.RDF_QM_GROUP_BY_SOURCE_TABLES (all_qms);
 launcher_name := 'RDF_QM_TREE_DUMP_BATCH_' || md5 (serialize (graph_iri) || storage || serialize (root))
 launcher_text := string_output ();
 http ('CREATE PROCEDURE DB.DBA."' || launcher_name || '" (in dest_dir VARCHAR)\n{\n', launcher_text);
 FOR (grp_ctr := length (grouped_qmvs); grp_ctr > 0; grp_ctr := grp_ctr-2)
   {
     DECLARE tables, qms, proc_text ANY;
     DECLARE group_key, proc_name, dump_prefix, cmt VARCHAR;
     tables := grouped_qmvs [grp_ctr-2];
     qms := grouped_qmvs [grp_ctr-1];
     qm_count := length (qms);
     group_key := md5 (serialize (graph_iri) || storage || serialize (root) || serialize (tables));
     proc_name := 'RDF_QM_TREE_DUMP_GRP_' || group_key;
     proc_text := string_output ();
     cmt := sprintf ('%d quad maps on join of', qm_count);
     FOREACH (VARCHAR t IN tables) DO cmt := cmt || ' ' || t;
     http ('  --  ' || cmt || '\n', launcher_text);
     http ('  DB.DBA."' || proc_name || '" (dest_dir);\n', launcher_text);
     http ('CREATE PROCEDURE DB.DBA."' || proc_name || '" (in dest_dir VARCHAR)\n', proc_text);
     http ('{\n', proc_text);
     http ('  -- ' || cmt || '\n', proc_text);
     http ('  DECLARE ses, env ANY;\n', proc_text);
```

```
      http ('  DECLARE file_ctr, cmt_len INTEGER;\n', proc_text);
      http ('  file_ctr := 0;\n', proc_text);
      http ('  dbg_obj_princ (' || WS.WS.STR_SQL_APOS (cmt) || ', '', file '', file_ctr);\n', proc_text);
      http ('  ses := string_output ();\n', proc_text);
      http ('  http (' || WS.WS.STR_SQL_APOS ('#' || cmt || '\n') || ', ses);\n', proc_text);
      http ('  env := VECTOR (dict_new (16000), 0, '''', '''', '''', 0, 0, 0, 0, 0);\n', proc_text);
      http ('  cmt_len := LENGTH (ses);\n', proc_text);
      http ('  FOR (SPARQL DEFINE input:storage <' || storage || '>\n', proc_text);
      http ('    SELECT ?s1, ?p1, ?o1\n', proc_text);
      IF (graph_iri IS NOT NULL)
        {
          http ('    WHERE { GRAPH <', proc_text); http_escape (graph_iri, 12, proc_text, 1, 1); http ('>
        }
      ELSE
        http ('    WHERE { GRAPH ?g1 {\n', proc_text);
      FOR (qm_ctr := 0; qm_ctr < qm_count; qm_ctr := qm_ctr + 1)
        {
          IF (qm_ctr > 0) http ('            UNION\n', proc_text);
          http ('            { quad map <' || qms[qm_ctr] || '> { ?s1 ?p1 ?o1 } }\n', proc_text);
        }
      http ('          } } ) DO {\n', proc_text);
      http ('      http_ttl_triple (env, "s1", "p1", "o1", ses);\n', proc_text);
      http ('      IF (LENGTH (ses) > 5000000)\n', proc_text);
      http ('        {\n', proc_text);
      http ('          http ('' .\\n'', ses);\n', proc_text);
      http ('          string_to_file (sprintf (''%s' || group_key || '_%05d.ttl'', dest_dir, file_ctr), s
      http ('          file_ctr := file_ctr + 1;\n', proc_text);
      http ('          dbg_obj_princ (' || WS.WS.STR_SQL_APOS (cmt) || ', '', file '', file_ctr);\n', proc
      http ('          ses := string_output ();\n', proc_text);
      http ('          http (' || WS.WS.STR_SQL_APOS ('#' || cmt || '\n') || ', ses);\n', proc_text);
      http ('          env := VECTOR (dict_new (16000), 0, '''', '''', '''', 0, 0, 0, 0, 0);\n', proc_text
      http ('        }\n', proc_text);
      http ('    }\n', proc_text);
      http ('  IF (LENGTH (ses) > cmt_len)\n', proc_text);
      http ('    {\n', proc_text);
      http ('      http ('' .\\n'', ses);\n', proc_text);
      http ('      string_to_file (sprintf (''%s' || group_key || '_%05d.ttl'', dest_dir, file_ctr), ses,
      http ('    }\n', proc_text);
      http ('}\n', proc_text);
      proc_text := string_output_string (proc_text);
      string_to_file (sql_file, proc_text || ';\n\n' , -1);
      EXEC (proc_text);
    }
  http ('}\n', launcher_text);
  launcher_text := string_output_string (launcher_text);
  string_to_file (sql_file, launcher_text || ';\n\n' , -1);
  EXEC (launcher_text);
  CALL ('DB.DBA.' || launcher_name)(dest_dir);
}
;

CREATE FUNCTION DB.DBA.RDF_QM_CONTENT_OF_QM_TREE
  ( in  graph_iri  VARCHAR := NULL,
    in  storage    VARCHAR := NULL,
    in  root       VARCHAR := NULL,
    in  dict       ANY := NULL
  ) returns ANY
{
  DECLARE res, subqms any;
  DECLARE graphiri varchar;
  graphiri := DB.DBA.JSO_SYS_GRAPH();
  IF (storage IS NULL)
    storage := 'http://www.openlinksw.com/schemas/virtrdf#DefaultQuadStorage';
  DB.DBA.RDF_QM_ASSERT_STORAGE_FLAG (storage, 0);
  IF (dict IS NULL)
    dict := dict_new ();
  IF (root IS NULL)
    {
      subqms := ((SELECT DB.DBA.VECTOR_AGG (sub."qmiri")
          FROM (
            SPARQL DEFINE input:storage ""
            SELECT DISTINCT (str(?qm)) AS ?qmiri
            WHERE { GRAPH `iri(?:graphiri)` {
```

```
                    { `iri(?:storage)` virtrdf:qsUserMaps ?lst .
                      ?lst ?p ?qm .
                      FILTER (0 = bif:strstr (str(?p), str(rdf:_)))
                    } UNION {
                      `iri(?:storage)` virtrdf:qsDefaultMap ?qm .
                    } } } ) AS sub ) );
    FOREACH (varchar qmid IN subqms) DO
      DB.DBA.RDF_QM_CONTENT_OF_QM_TREE (graph_iri, storage, qmid, dict);
    RETURN dict;
  }
 DB.DBA.RDF_QM_ASSERT_JSO_TYPE (root, 'http://www.openlinksw.com/schemas/virtrdf#QuadMap');
 IF (graph_iri IS NOT NULL AND
   EXISTS ((SPARQL DEFINE input:storage ""
       SELECT (1) WHERE {
           GRAPH `iri(?:graphiri)` {
               `iri(?:root)` virtrdf:qmGraphRange-rvrFixedValue ?g .
               FILTER (str (?g) != str(?:graph_iri))
             } } ) ) )
   RETURN dict;
 IF (NOT EXISTS ((SPARQL DEFINE input:storage ""
       SELECT (1) WHERE {
           GRAPH `iri(?:graphiri)` {
               `iri(?:root)` virtrdf:qmMatchingFlags virtrdf:SPART_QM_EMPTY .
             } } ) ) )
   dict_put (dict, root, 1);
 subqms := ((SELECT DB.DBA.VECTOR_AGG (sub."qmiri")
     FROM (
       SPARQL DEFINE input:storage ""
       SELECT DISTINCT (str(?qm)) as ?qmiri
       WHERE { GRAPH `iri(?:graphiri)` {
                `iri(?:root)` virtrdf:qmUserSubMaps ?lst .
               ?lst ?p ?qm .
               FILTER (0 = bif:strstr (str(?p), str(rdf:_)))
             } } ) AS sub ) );
 FOREACH (VARCHAR qmid IN subqms) DO
   DB.DBA.RDF_QM_CONTENT_OF_QM_TREE (graph_iri, storage, qmid, dict);
 RETURN dict;
}
;

CREATE FUNCTION DB.DBA.RDF_QM_GROUP_BY_SOURCE_TABLES (in qms ANY) returns ANY
{
 DECLARE res ANY;
 DECLARE ctr INTEGER;
 DECLARE graphiri VARCHAR;
 graphiri := DB.DBA.JSO_SYS_GRAPH();
 res := dict_new (LENGTH (qms) / 20);
 FOREACH (VARCHAR qmiri IN qms) DO
   {
     DECLARE tbls, acc ANY;
     tbls := ((SELECT DB.DBA.VECTOR_AGG (sub."tbl")
         FROM (SELECT subsub."tbl"
           FROM (
             SPARQL DEFINE input:storage ""
             SELECT DISTINCT ?tbl
             WHERE { GRAPH `iri(?:graphiri)` {
                       { `iri(?:qmiri)` virtrdf:qmTableName ?tbl .
                       } UNION {
                         `iri(?:qmiri)` virtrdf:qmATables ?atbls .
                         ?atbls ?p ?atbl .
                         ?atbl virtrdf:qmvaTableName ?tbl
                       } UNION {
                         `iri(?:qmiri)` ?fldmap ?qmv .
                         ?qmv virtrdf:qmvATables ?atbls .
                         ?atbls ?p ?atbl .
                         ?atbl virtrdf:qmvaTableName ?tbl .
                       } } } ) subsub
           ORDER BY 1 ) AS sub ) );
     acc := dict_get (res, tbls);
     IF (acc IS NULL)
       vectorbld_init (acc);
     vectorbld_acc (acc, qmiri);
     dict_put (res, tbls, acc);
```

```
    }
  res := dict_to_vector (res, 2);
  FOR (ctr := LENGTH (res); ctr > 0; ctr := ctr-2)
    {
      DECLARE acc ANY;
      acc := aref_set_0 (res, ctr-1);
      vectorbld_final (acc);
      aset_zap_arg (res, ctr-1, acc);
    }
  RETURN res;
}
;

--test dbg_obj_princ (DB.DBA.RDF_QM_GROUP_BY_SOURCE_TABLES (dict_list_keys (DB.DBA.RDF_QM_CONTENT_OF_QM_T
--test dbg_obj_princ (dict_list_keys (DB.DBA.RDF_QM_CONTENT_OF_QM_TREE (null), 2));
--test DB.DBA.RDF_QM_TREE_DUMP ('dump/demo', null, null, null);
--test DB.DBA.RDF_QM_TREE_DUMP ('dump/tpch', 'http://localhost:8600/tpch', null, null);
```

16.17.12. Loading RDF

There are many functions for loading RDF text, in RDF/XML and Turtle.

For loading RDF/XML, the best way is to split the data to be loaded into multiple streams and load these in parallel using `RDF_LOAD_RDFXML ()` . To avoid running out of rollback space for large files and in order to have multiple concurrent loads not interfere with each other, the row autocommit mode should be enabled.

For example,

```
log_enable (2);
-- switch row-by-row autocommit on and logging off for this session
DB.DBA.RDF_LOAD_RDFXML (file_to_string_output ('file.xml'), 'base_uri', 'target_graph');
-- more files here ...
checkpoint;
```

Loading a file with text like the above with isql will load the data. Since the transaction logging is off, make a manual checkpoint at the end to ensure that data is persisted upon server restart since there is no roll forward log.

If large amounts of data are to be loaded, run multiple such streams in parallel. One may have for example 6 streams for 4 cores. This means that if up to two threads wait for disk, there is still work for all cores.

Having substantially more threads than processors or disks is not particularly useful.

There exist multithreaded load functions which will load one file on multiple threads: the DB.DBA.TTLP_MT() function and the DB.DBA.RDF_LOAD_RDFXML_MT() function . Experience shows that loading multiple files on one thread per file is better.

For loading Turtle, some platforms may have a non-reentrant Turtle parser. This means that only one load may run at a time. One can try this by calling `ttlp ()` from two sessions at the same time. If these do not execute concurrently, then the best way may be to try `ttlp_mt` and see if this runs faster than a single threaded ttlp call.

RDF Bulk Load Utility

The RDF loader utility facilitates parallel bulk loading of multiple RDF files. The utility maintains a database table containing a list of files to load and the status of each file, whether not loaded, loaded or loaded with error. The table also records load start and end times.

One must have a dba group login for using this and the virtuoso.ini file access control list must be set up so that the Virtuoso server can open the files to load.

Files are added to the load list with the function `ld_dir` :

```
ld_dir (in dir_path varchar, in file_mask varchar, in target_graph varchar);
```

The file mask is a SQL like pattern to match against the files in the directory. For example:

```
ld_dir ('/data8/2848260', '%.gz', 'http://bsbm.org');
```

would load the RDF in all files ending in .gz from the directory given as first parameter. The RDF would be loaded in the http://bsbm.org graph.

If NULL is given for the graph, each file may go to a different graph specified in a separate file with the name of the RDF source file plus the extension .graph.

A .graph file contains the target graph URI without any other content or whitespace.

The layout of the load_list table is as follows:

```
create table DB.DBA.LOAD_LIST (
  ll_file varchar,
  ll_graph varchar,
  ll_state int default 0, -- 0 not started, 1 going, 2 done
  ll_started datetime,
  ll_done datetime,
  ll_host int,
  ll_work_time integer,
  ll_error varchar,
  primary key (ll_file))
alter index LOAD_LIST on DB.DBA.LOAD_LIST partition (ll_file varchar)
create index LL_STATE on DB.DBA.LOAD_LIST (ll_state, ll_file, ll_graph) partition (ll_state int)
;
```

This table may be checked at any time during bulk load for the progress of the load. ll_state is 1 for files being loaded and 2 for files whose loading has finished. ll_error is NULL if the load finished without error, else it is the error message.

In order to load data from the files in load_list, run as dba:

```
DB.DBA.rdf_loader_run ();
```

One may run several of these commands on parallel sessions for better throughput.

On a cluster one can do:

```
cl_exec ('rdf_ld_srv ()');
```

This will start one rdf_loader_run() on each node of the cluster. Note that in such a setting all the server processes must see the same files at the same path.

On an isql session one may execute rdf_loader_run () & several times, forking a new isql for each such command, similarly to what a Unix shell does.

Because this load is non-transactional and non-logged, one must do an explicit checkpoint after the load to guarantee a persistent state.

On a single server do:

```
checkpoint;
```

On a cluster do:

```
cl_exec ('checkpoint');
```

The server(s) are online and can process queries and transactions while a bulk load is in progress. Periodic checkpoints may occur during the load but the state is guaranteed to be consistent only after running a checkpoint after all the bulk load threads have finished.

A bulk load should not be forcibly stopped. To make a controlled stop, run:

```
rdf_load_stop ();
```

This will cause the files being loaded at the time to finish load but no new loads will start until explicitly started with rdf_loader_run() .

Specially note that on a cluster the database will be inconsistent if one server process does a checkpoint and another does not. Thus guaranteeing a checkpoint on all is necessary. This is easily done with an isql script with the following content:

```
ld_dir ('/data8/2848260', '%.gz', 'http://bsbm.org');

-- Record CPU time
select getrusage ()[0] + getrusage ()[1];

rdf_loader_run () &
rdf_loader_run () &
rdf_loader_run () &
rdf_loader_run () &
rdf_loader_run () &
rdf_loader_run () &
rdf_loader_run () &
rdf_loader_run () &

wait_for_children;
checkpoint;

-- Record CPU time
select getrusage ()[0] + getrusage ()[1];
```

For a cluster, the equivalent is:

```
ld_dir ('/data8/2848260', '%.gz', 'http://bsbm.org');

cl_exec ('DB.DBA.RDF_LD_SRV (2)');

cl_exec ('checkpoint');
```

rdf_loader_run() recognizes several file types, including .ttl, .nt, .xml, .rdf, .owl, .nq, .n4, and others. Internally the function uses `DB.DBA.ttlp()` or `DB.DBA.rdf_load_rdfxml` , as appropriate.

See the next section for detailed description of the rdf_loader_run() function.

Loading LOD RDF data

To load the rdf data to LOD instance, perform the following steps:

- Configure & start cluster
- Execute the file:

```
--
--  $Id$
--
--  Alternate RDF index scheme for cases where G unspecified
--
--  This file is part of the OpenLink Software Virtuoso Open-Source (VOS)
--  project.
--
--  Copyright (C) 1998-2024 OpenLink Software
--
--  This project is free software; you can redistribute it and/or modify it
--  under the terms of the GNU General Public License as published by the
--  Free Software Foundation; only version 2 of the License, dated June 1991.
--
--  This program is distributed in the hope that it will be useful, but
--  WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
--  General Public License for more details.
--
--  You should have received a copy of the GNU General Public License along
--  with this program; if not, write to the Free Software Foundation, Inc.,
--  51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
--
--

drop index RDF_QUAD_OGPS;
checkpoint;
```

```
create table R2 (G iri_id_8, S iri_id_8, P iri_id_8, O any, primary key (S, P, O, G))
alter index R2 on R2 partition (S int (0hexffff00));

log_enable (2);
insert into R2 (G, S, P, O) SELECT G, S, P, O from rdf_quad;

drop table RDF_QUAD;
alter table r2 rename RDF_QUAD;
checkpoint;
create bitmap index RDF_QUAD_OPGS on RDF_QUAD (O, P, G, S) partition (O varchar (-1, 0hexffff));
create bitmap index RDF_QUAD_POGS on RDF_QUAD (P, O, G, S) partition (O varchar (-1, 0hexffff));
create bitmap index RDF_QUAD_GPOS on RDF_QUAD (G, P, O, S) partition (O varchar (-1, 0hexffff));

checkpoint;
```

- Execute:

```
SQL>cl_exec ('checkpoint);
```

- Execute ld_dir ('directory' , 'mask' , 'graph'), for ex:

```
SQL>ld_dir ('/dbs/data', '*.gz', 'http://dbpedia.org');
```

- Execute on every node with separate client:

```
SQL>rdf_loader_run();
```

Loading UniProt RDF data

To load the uniprot data, create a function for example such as:

```
create function DB.DBA.UNIPROT_LOAD (in log_mode integer := 1)
{
  DB.DBA.RDF_LOAD_RDFXML_MT (file_to_string_output('filename1'),'http://base_uri_1', 'destination_graph_1
  DB.DBA.RDF_LOAD_RDFXML_MT (file_to_string_output('filename2'),'http://base_uri_2', 'destination_graph_2
  ...
  DB.DBA.RDF_LOAD_RDFXML_MT (file_to_string_output('filename9'),'http://base_uri_9', 'destination_graph_9
}
```

If you are starting from blank database and you can drop it and re-create in case of error signaled, use it this way:

```
checkpoint;
checkpoint_interval(6000);
DB.DBA.UNIPROT_LOAD (0),
checkpoint;
checkpoint_interval(60);
```

If the database contains important data already and there's no way to stop it and backup before the load then use:

```
checkpoint;
checkpoint_interval(6000);
DB.DBA.UNIPROT_LOAD (),
checkpoint;
checkpoint_interval(60);
```

Note that the 'number of threads' parameter of DB.DBA.RDF_LOAD_RDFXML() mentions threads used to process data from file, an extra thread will read the text and parse it, so for 4 CPU cores there's no need in parameter value greater than 3. Three processing threads per one parsing tread is usually good ratio because parsing is usually three times faster than the rest of loading so CPU loading is well balanced. If for example you are using 2 x Quad Xeon, then you can choose between 8 single-threaded parsers or 2 parsers with 3 processing threads each. With 4 cores you may simply load file after file with 3 processing threads. The most important performance tuning is to set the [Parameters] section of virtuoso configuration file:

```
NumberOfBuffers = 1000000
MaxDirtyBuffers = 800000
MaxCheckpointRemap = 1000000
DefaultIsolation = 2
```

Note: these numbers are reasonable for 16 GB RAM Linux box. Usually when there are no such massive operations as loading huge database, you can set up the values as:

```
NumberOfBuffers = 1500000
```

```
MaxDirtyBuffers = 1200000
MaxCheckpointRemap = 1500000
DefaultIsolation = 2
```

See Also:

Virtuoso Configuration Options

Tip:

Thus after loading all data you may wish to shutdown, tweak and start server again. If you have ext2fs or ext3fs filesystem, then it's better to have enough free space on disk not to make it more than 80% full. When it's almost full it may allocate database file badly, resulting in measurable loss of disk access speed. That is not Virtuoso-specific fact, but a common hint for all database-like applications with random access to big files.

Here is an example of using awk file for splitting big file smaller ones:

```
BEGIN {
        file_part=1000
        e_line = "</rdf:RDF>"
        cur=0
        cur_o=0
        file=0
        part=file_part
      }
        {
            res_file_i="res/"FILENAME
            line=$0
            s=$1
            res_file=res_file_i"_"file".rdf"

            if (index (s, "</rdf:Description>") == 1)
            {
                cur=cur+1
                part=part-1
            }

            if (part > 0)
            {
                print line >> res_file
            }

            if (part == 0)
            {
#               print "==================== " cur
                print line >> res_file
                print e_line >> res_file
                close (res_file)
                file=file+1
                part=file_part
                res_file=res_file_i"_"file".rdf"
                system ("cp beg.txt " res_file)
            }
        }
END { }
```

Loading DBPedia RDF data

You can use the following script as an example for loading DBPedia RDF data in Virtuoso:

```
#!/bin/sh

PORT=$1
USER=$2
PASS=$3
file=$4
g=$5
LOGF=`basename $0`.log

if [ -z "$PORT" -o -z "$USER" -o -z "$PASS" -o -z "$file" -o -z "$g" ]
then
  echo "Usage: `basename $0` [DSN] [user] [password] [ttl-file] [graph-iri]"
```

```
  exit
fi

if [ ! -f "$file" -a ! -d "$file" ]
then
    echo "$file does not exists"
    exit 1
fi

mkdir READY 2>/dev/null
rm -f $LOGF $LOGF.*

echo "Starting..."
echo "Logging into: $LOGF"

DOSQL ()
{
    isql $PORT $USER $PASS verbose=on banner=off prompt=off echo=ON errors=stdout exec="$1" > $LOGF
}

LOAD_FILE ()
{
    f=$1
    g=$2
    echo "Loading $f (`cat $f | wc -l` lines) `date \"+%H:%M:%S\"`" | tee -a $LOG

    DOSQL "ttlp_mt (file_to_string_output ('$f'), '', '$g', 17); checkpoint;" > $LOGF

    if [ $? != 0 ]
    then
        echo "An error occurred, please check $LOGF"
        exit 1
    fi

    line_no=`grep Error $LOGF | awk '{ match ($0, /line [0-9]+/, x) ; match (x[0], /[0-9]+/, y); print y[
    newf=$f.part
    inx=1
    while [ ! -z "$line_no" ]
    do
        cat $f |  awk "BEGIN { i = 1 } { if (i==$line_no) { print \$0; exit; } i = i + 1 }"  >> bad.nt
        line_no=`expr $line_no + 1`
        echo "Retrying from line $line_no"
        echo "@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> ." > tmp.nt
        cat $f |  awk "BEGIN { i = 1 } { if (i>=$line_no) print \$0; i = i + 1 }"  >> tmp.nt
        mv tmp.nt $newf
        f=$newf
        mv $LOGF $LOGF.$inx
        DOSQL "ttlp_mt (file_to_string_output ('$f'), '', '$g', 17); checkpoint;" > $LOGF

        if [ $? != 0 ]
        then
            echo "An error occurred, please check $LOGF"
            exit 1
        fi
        line_no=`grep Error $LOGF | awk '{ match ($0, /line [0-9]+/, x) ; match (x[0], /[0-9]+/, y); prin
        inx=`expr $inx + 1`
    done
    rm -f $newf 2>/dev/null
    echo "Loaded.  "
}

echo "========================================"
echo "Loading started."
echo "========================================"

if [ -f "$file" ]
then
    LOAD_FILE $file $g
    mv $file READY 2>> /dev/null
elif [ -d "$file" ]
then
    for ff in `find $file -name '*.nt'`
    do
```

```
        LOAD_FILE $ff $g
        mv $ff READY 2>> /dev/null
    done
else
   echo "The input is not file or directory"
fi
echo "======================================="
echo "Final checkpoint."
DOSQL "checkpoint;" > temp.res
echo "======================================="
echo "Check bad.nt file for skipped triples."
echo "======================================="

exit 0
```

Loading Bio2RDF data

The shell script below was used to import files in n3 notation into OpenLink Virtuoso RDF storage.

When an syntax error it will cut content from next line and will retry. This was used on ubuntu linux to import bio2rdf and freebase dumps.

Note it uses gawk, so it must be available on system where is tried. Also for recovery additional disk space is needed at max the size of original file.

```
#!/bin/bash

PASS=$1
f=$2
g=$3

# Usage
if [ -z "$PASS" -o -z "$f" -o -z "$g" ]
then
  echo "Usage: $0 [password] [ttl-file] [graph-iri]"
  exit
fi

if [ ! -f "$f" ]
then
    echo "$f does not exists"
    exit
fi

# Your port here
PORT=1111  #`inifile -f dbpedia.ini -s Parameters -k ServerPort`
if test -z "$PORT"
then
    echo "Cannot find INI and inifile command"
    exit
fi

# Initial run
isql $PORT dba $PASS verbose=on banner=off prompt=off echo=ON errors=stdout exec="ttlp_mt (file_to_string

# If disconnect etc.
if [ $? != 0 ]
then
    echo "An error occurred, please check $0.log"
    exit
fi

# Check for error
line_no=`grep Error $0.log | awk '{ match ($0, /line [0-9]+/, x) ; match (x[0], /[0-9]+/, y); print y[0]
newf=$f.part
inx=1

# Error recovery
while [ ! -z "$line_no" ]
do
```

```
    cat $f |  awk "BEGIN { i = 0 } { if (i==$line_no) { print \$0; exit; } i = i + 1 }"  >> bad.nt
    line_no=`expr $line_no + 1`
    echo "Retrying from line $line_no"
    cat $f |  awk "BEGIN { i = 0 } { if (i>=$line_no) print \$0; i = i + 1 }"  > tmp.nt
    mv tmp.nt $newf
    f=$newf
    mv $0.log $0.log.$inx
    # Run the recovered part
    isql $PORT dba $PASS verbose=on banner=off prompt=off echo=ON errors=stdout exec="ttlp_mt (file_to_st

    if [ $? != 0 ]
    then
        echo "An error occurred, please check $0.log"
        exit
    fi
   line_no=`grep Error $0.log | awk '{ match ($0, /line [0-9]+/, x) ; match (x[0], /[0-9]+/, y); print y[
   inx=`expr $inx + 1`
done
```

16.17.13. Using SPARUL

Since SPARUL updates are generally not meant to be transactional, it is best to run these in `log_enable (2)` mode, which commits every operation as it is done. This prevents one from running out of rollback space. Also for bulk updates, transaction logging can be turned off. If so, one should do a manual checkpoint after the operation to ensure persistence across server restart since there is no roll forward log.

To have a roll forward log and row by row autocommit, one may use `log_enable (3)` . This will write constantly into the log which takes extra time. Having no logging and doing a checkpoint when the whole work is finished is faster.

Many SPARUL operations can be run in parallel in this way. If they are independent with respect to their input and output, they can run in parallel and row by row autocommit will ensure they do not end up waiting for each others' locks.

16.17.14. DBpedia Benchmark

We ran the DBpedia benchmark queries again with different configurations of Virtuoso. Comparing numbers given by different parties is a constant problem. In the case reported here, we loaded the full DBpedia 3, all languages, with about 198M triples, onto Virtuoso v5 and Virtuoso Cluster v6, all on the same 4 core 2GHz Xeon with 8G RAM. All databases were striped on 6 disks. The Cluster configuration was with 4 processes in the same box. We ran the queries in two variants:

- With graph specified in the SPARQL FROM clause, using the default indices.
- With no graph specified anywhere, using an alternate indexing scheme.

The times below are for the sequence of 5 queries. As there is a query in the set that specifies no condition on S or O and only P, thus cannot be done with the default indices With Virtuoso v5. With Virtuoso Cluster v6 it can, because v6 is more space efficient. So we added the index:

```
create bitmap index rdf_quad_pogs on rdf_quad (p, o, g, s);
```

Table 16.20.

	Virtuoso v5 with gspo, ogps, pogs	Virtuoso Cluster v6 with gspo, ogps	Virtuoso Cluster v6 with gspo, ogps, pogs
cold	210 s	136 s	33.4 s
warm	0.600 s	4.01 s	0.628 s

Now let us do it without a graph being specified. Note that alter index is valid for v6 or higher. For all platforms, we drop any existing indices, and:

```
create table r2 (g iri_id_8, s, iri_id_8, p iri_id_8, o any, primary key (s, p, o, g))
alter index R2 on R2 partition (s int (0hexffff00));

log_enable (2);
insert into r2 (g, s, p, o) SELECT g, s, p, o from rdf_quad;
```

```
drop table rdf_quad;
alter table r2 rename RDF_QUAD;
create bitmap index rdf_quad_opgs on rdf_quad (o, p, g, s) partition (o varchar (-1, 0hexffff));
create bitmap index rdf_quad_pogs on rdf_quad (p, o, g, s) partition (o varchar (-1, 0hexffff));
create bitmap index rdf_quad_gpos on rdf_quad (g, p, o, s) partition (o varchar (-1, 0hexffff));
```

The code is identical for v5 and v6, except that with v5 we use iri_id (32 bit) for the type, not iri_id_8 (64 bit). We note that we run out of IDs with v5 around a few billion triples, so with v6 we have double the ID length and still manage to be vastly more space efficient.

With the above 4 indices, we can query the data pretty much in any combination without hitting a full scan of any index. We note that all indices that do not begin with s end with s as a bitmap. This takes about 60% of the space of a non-bitmap index for data such as DBpedia.

If you intend to do completely arbitrary RDF queries in Virtuoso, then chances are you are best off with the above index scheme.

Table 16.21.

	Virtuoso v5 with gspo, ogps, pogs	Virtuoso Cluster v6 with gspo, ogps, pogs
warm	0.595 s	0.617 s

The cold times were about the same as above, so not reproduced.

It is in the SPARQL spirit to specify a graph and for pretty much any application, there are entirely sensible ways of keeping the data in graphs and specifying which ones are concerned by queries. This is why Virtuoso is set up for this by default.

On the other hand, for the open web scenario, dealing with an unknown large number of graphs, enumerating graphs is not possible and questions like which graph of which source asserts x become relevant. We have two distinct use cases which warrant different setups of the database, simple as that.

The latter use case is not really within the SPARQL spec, so implementations may or may not support this.

Once the indices are right, there is no difference between specifying a graph and not specifying a graph with the queries considered. With more complex queries, specifying a graph or set of graphs does allow some optimizations that cannot be done with no graph specified. For example, bitmap intersections are possible only when all leading key parts are given.

The best warm cache time is with v5; the five queries run under 600 ms after the first go. This is noted to show that all-in-memory with a single thread of execution is hard to beat.

Cluster v6 performs the same queries in 623 ms. What is gained in parallelism is lost in latency if all operations complete in microseconds. On the other hand, Cluster v6 leaves v5 in the dust in any situation that has less than 100% hit rate. This is due to actual benefit from parallelism if operations take longer than a few microseconds, such as in the case of disk reads. Cluster v6 has substantially better data layout on disk, as well as fewer pages to load for the same content.

This makes it possible to run the queries without the pogs index on Cluster v6 even when v5 takes prohibitively long.

The purpose is to have a lot of RAM and space-efficient data representation.

For reference, the query texts specifying the graph are below. To run without specifying the graph, just drop the FROM <http://dbpedia.org> from each query. The returned row counts are indicated below each query's text.

```
SQL>SPARQL
SELECT ?p ?o
FROM <http://dbpedia.org>
WHERE
  {
    <http://dbpedia.org/resource/Metropolitan_Museum_of_Art> ?p ?o .
  };

p                                                                                o
VARCHAR                                                                          VARCHAR
_______________________________________________________________________________
```

```
http://www.w3.org/1999/02/22-rdf-syntax-ns#type                                  http://umbel.org/umbel/
http://www.w3.org/1999/02/22-rdf-syntax-ns#type                                  http://dbpedia.org/clas
http://www.w3.org/1999/02/22-rdf-syntax-ns#type                                  http://dbpedia.org/clas
http://www.w3.org/1999/02/22-rdf-syntax-ns#type                                  http://dbpedia.org/clas
..
-- 335 rows

SQL>SPARQL
PREFIX p: <http://dbpedia.org/property/>
SELECT ?film1 ?actor1 ?film2 ?actor2
FROM <http://dbpedia.org>
WHERE
  {
    ?film1 p:starring <http://dbpedia.org/resource/Kevin_Bacon> .
    ?film1 p:starring ?actor1 .
    ?film2 p:starring ?actor1 .
    ?film2 p:starring ?actor2 .
};

film1                                       actor1                                    film2
VARCHAR                                     VARCHAR                                   VARCHAR
http://dbpedia.org/resource/The_River_Wild  http://dbpedia.org/resource/Kevin_Bacon   http://dbpedia.org/
http://dbpedia.org/resource/The_River_Wild  http://dbpedia.org/resource/Kevin_Bacon   http://dbpedia.org/
http://dbpedia.org/resource/The_River_Wild  http://dbpedia.org/resource/Kevin_Bacon   http://dbpedia.org/
http://dbpedia.org/resource/The_River_Wild  http://dbpedia.org/resource/Kevin_Bacon   http://dbpedia.org/
http://dbpedia.org/resource/The_River_Wild  http://dbpedia.org/resource/Kevin_Bacon   http://dbpedia.org/
...
--  23910 rows

SQL>SPARQL
PREFIX p: <http://dbpedia.org/property/>
SELECT ?artist ?artwork ?museum ?director
FROM <http://dbpedia.org>
WHERE
  {
    ?artwork p:artist ?artist .
    ?artwork p:museum ?museum .
    ?museum p:director ?director
  };

artist                                          artwork                                       muse
VARCHAR                                         VARCHAR                                       VARC
_______________________________________________

http://dbpedia.org/resource/Paul_C%C3%A9zanne   http://dbpedia.org/resource/The_Basket_of_Apples     http
http://dbpedia.org/resource/Paul_Signac         http://dbpedia.org/resource/Neo-impressionism       http
http://dbpedia.org/resource/Georges_Seurat      http://dbpedia.org/resource/Neo-impressionism       http
http://dbpedia.org/resource/Edward_Hopper       http://dbpedia.org/resource/Nighthawks              http
http://dbpedia.org/resource/Mary_Cassatt        http://dbpedia.org/resource/The_Child%27s_Bath      http
..
-- 303 rows

SQL>SPARQL
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?s ?homepage
FROM <http://dbpedia.org>
WHERE
  {
    <http://dbpedia.org/resource/Berlin> geo:lat ?berlinLat .
    <http://dbpedia.org/resource/Berlin> geo:long ?berlinLong .
    ?s geo:lat ?lat .
    ?s geo:long ?long .
    ?s foaf:homepage ?homepage .
    FILTER (
      ?lat        <=     ?berlinLat + 0.03190235436 &&
      ?long       >=     ?berlinLong - 0.08679199218 &&
      ?lat        >=     ?berlinLat - 0.03190235436 &&
      ?long       <=     ?berlinLong + 0.08679199218) };

s                                                                                homepage
```

```
VARCHAR                                                                                    VARCHAR
_______________________________________________________________________________

http://dbpedia.org/resource/Berlin_University_of_the_Arts                                 http://www.udk-berlin.d
http://dbpedia.org/resource/Berlin_University_of_the_Arts                                 http://www.udk-berlin.d
http://dbpedia.org/resource/Berlin_Zoological_Garden                                      http://www.zoo-berlin.d
http://dbpedia.org/resource/Federal_Ministry_of_the_Interior_%28Germany%29                http://www.bmi.bund.de
http://dbpedia.org/resource/Neues_Schauspielhaus                                          http://www.goya-berlin.
http://dbpedia.org/resource/Bauhaus_Archive                                               http://www.bauhaus.de/e
http://dbpedia.org/resource/Canisius-Kolleg_Berlin                                        http://www.canisius-kol
http://dbpedia.org/resource/Franz%C3%B6sisches_Gymnasium_Berlin                           http://www.fg-berlin.ci
..
-- 48 rows

SQL>SPARQL
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX p: <http://dbpedia.org/property/>
SELECT ?s ?a ?homepage
FROM <http://dbpedia.org>
WHERE
  {
    <http://dbpedia.org/resource/New_York_City> geo:lat ?nyLat .
    <http://dbpedia.org/resource/New_York_City> geo:long ?nyLong .
    ?s geo:lat ?lat .
    ?s geo:long ?long .
    ?s p:architect ?a .
    ?a foaf:homepage ?homepage .
    FILTER (
      ?lat        <=     ?nyLat + 0.3190235436 &&
      ?long       >=     ?nyLong - 0.8679199218 &&
      ?lat        >=     ?nyLat - 0.3190235436 &&
      ?long       <=     ?nyLong + 0.8679199218) };
s                                                                                          a               homepag
VARCHAR                                                                                    VARCHAR              VA
_______________________________________________________________________________

http://dbpedia.org/resource/GE_Building                                                   http://dbpedia.org/reso
http://dbpedia.org/resource/Giants_Stadium                                                http://dbpedia.org/reso
http://dbpedia.org/resource/Fort_Tryon_Park_and_the_Cloisters                             http://dbpedia.org/reso
http://dbpedia.org/resource/Central_Park                                                  http://dbpedia.org/reso
http://dbpedia.org/resource/Prospect_Park_%28Brooklyn%29                                  http://dbpedia.org/reso
http://dbpedia.org/resource/Meadowlands_Stadium                                           http://dbpedia.org/reso
http://dbpedia.org/resource/Citi_Field                                                    http://dbpedia.org/reso
http://dbpedia.org/resource/Citigroup_Center                                              http://dbpedia.org/reso
http://dbpedia.org/resource/150_Greenwich_Street                                          http://dbpedia.org/reso
http://dbpedia.org/resource/Freedom_Tower                                                 http://dbpedia.org/reso
http://dbpedia.org/resource/7_World_Trade_Center                                          http://dbpedia.org/reso
http://dbpedia.org/resource/The_New_York_Times_Building                                   http://dbpedia.org/reso
http://dbpedia.org/resource/Trump_World_Tower                                             http://dbpedia.org/reso

13 Rows. -- 2183 msec.
```

16.17.15. RDF Store Benchmarks

Introduction

In a particular RDF Store Benchmarks there is difference if the queries are executed with specified graph or with specified multiple graphs. As Virtuoso is quad store, not triple store with many tables, it runs queries inefficiently if graphs are specified and there are no additional indexes except pre-set GSPO and OGPS. Proper use of the FROM clause or adding indexes with graph column will contribute for better results.

Using bitmap indexes

If is known in advance for the current RDF Store Benchmarks that some users will not indicate specific graphs then should be done:

- either create indexes with graph in last position
- or load everything into single graph and specify it somewhere in querying application.

Both methods do not require any changes in query texts

- For users using Virtuoso 5 is strongly recommended is the usage of additional bitmap indexes:

```
SQL> create bitmap index RDF_QUAD_POGS on DB.DBA.RDF_QUAD (P,O,G,S);
SQL> create bitmap index RDF_QUAD_PSOG on DB.DBA.RDF_QUAD (P,S,O,G);
```

- For users using Virtuoso 6 or higher, see the new layout here .

You can create other indexes as well. Bitmap indexes are preferable, but if O is the last column, then the index can not be bitmap, so it could be, for e.g.:

```
create index RDF_QUAD_PSGO on DB.DBA.RDF_QUAD (P, S, G, O);
```

but cannot be:

```
create bitmap index RDF_QUAD_PSGO on DB.DBA.RDF_QUAD (P, S, G, O);
```

16.17.16. Fast Approximate RDF Graph Diff and Patch

Two algorithms described below resemble "unified diff" and "patch by unified diff" but they work on RDF graphs, not on plain texts.

They work reasonably for graphs composed from CBDs (concise bounded descriptions) of some subjects, if these subjects are either "named" IRIs or can be identified by values of their inverse functional properties.

Many sorts of commonly used graphs match these restrictions, including all graphs without blank nodes, most of FOAF files, graphs that can be "pretty-printed" in JSON, most of dumps of relational databases etc.

The basic idea is as simple as zipper:

- Place one graph at the left and one to the right,
- Find a retainer box at the right and a matching pin at the left,
- Join them
- Pull the slider as long as possible.
- Repeat this while there are pins and boxes that can be matched and sliders that can be moved.

An IRI in left graph `(say, G1)` matches to same IRI in right graph `(G2)` as pin to box. The same is true for literals too.

Functional and inverse functional properties are teeth that form chains, algorithm "moves sliders" along these chains, incrementally connecting more and more nodes.

If there is a match of this sort `(O1 in G1 matches O2 in G2)` and the matched nodes are values of same inverse functional property `P` (there are `{ S1 P O1 }` in `G1` and `{ S2 P O2 }` in `G2`) then we guess that `S1` matches `S2` .

If `S1` in `G1` matches `S2` in `G2` and the matched nodes are subjects of same functional property `P` (there are `{ S1 P N1 }` in `G1` and `{ S2 P N2 }` in `G2`) then we guess that `N1` matches `N2` , now it's possible to try same interaction on triples where `N1` and `N2` are in subject position, that's how slides move. A typical example of a long zipper is closed list with matched heads.

Make a Diff And Use It

- Using DB.DBA.RDF_GRAPH_DIFF
- Using DB.DBA.RDF_SUO_DIFF_TTL
- Using DB.DBA.RDF_SUO_APPLY_PATCH

Collect Functional And Inverse Functional Properties

Lists of functional properties can be retrieved from an ontology graph by query like:

```
SPARQL define output:valmode "LONG"
SELECT (<LONG::sql:VECTOR_AGG(?s))
FROM <my-ontology-graph>
WHERE
```

```
  {
    ?s a owl:functionalProperty
  }
```

Inverse functional properties could be retrieved by a similar query, but unfortunately the ontology may mention so called NULL values that can be property values for many subjects. Current implementation of diff and patch does not recognize NULL values so they can cause patch with "false alarm" errors. The workaround is to retrieve only properties that have no NULL values declared:

```
SPARQL define output:valmode "LONG"
SELECT (<LONG::sql:VECTOR_AGG(?s))
FROM <my-ontology-graph>
WHERE
  {
    ?s a owl:inverseFunctionalProperty .
    OPTIONAL { ?s owl:nullIFPValue ?v }
    FILTER (!Bound(?v))
  }
```

If no ontology is available then appropriate predicates can be obtained from sample graphs using DB.DBA.RDF_GRAPH_COLLECT_FP_LIST .

Implementation-Specific Extensions of GUO Ontology

Note : This section contains implementation details that are needed only if you want to write your own patch or diff procedure, you don't have to worry about internals if you want to use existing procedures.

Basic GUO ontology is not expressive enough to work with blank nodes, so some custom extensions $ are needed.

In the rest of the description:

```
@prefix guo: <http://webr3.org/owl/guo#>
```

is assumed.

The diff contains one node of `rdf:type guo:diff` .

For debugging purpose it has properties `guo:graph1` and `guo:graph2` that corespond to `gfrom` and `gto` arguments of DB.DBA.RDF_SUO_DIFF_TTL .

The diff also contains zero or more nodes of `rdf:type guo:UpdateInstruction` . These nodes are as described in basic GUO ontology, but `guo:target_graph` is now optional, `guo:target_subject` can be a blank node and objects of predicates "inside" values of `guo:insert` and `guo:delete` can also be blank nodes. These blank nodes are "placeholders" for values, calculated according to the most important GUO extension - rule nodes.

There are eight sorts of rule nodes, four for `gfrom` side of diff and four similar for `gto` side. Out of four sorts related to one side, two are for functional properties and two similar are for inverse functional properties. Thus `rdf:type-s` of these nodes are:

```
guo:from-rule-FP0,
guo:from-rule-FP1,
guo:from-rule-IFP0,
guo:from-rule-IFP1
```

and

```
guo:to-rule-FP0,
guo:to-rule-FP1,
guo:to-rule-IFP ,
guo:to-rule-IFP1 .
```

Each rule node has property `guo:order` that is an non-negative integer.

These integers enumerate all `guo:from-rule-` ... nodes, starting from zero.

When patch procedure works, these rules are used in this order, the result of each rule is a blank node that either exists in the graph or just created.

All results are remembered for use in the rest of the patch procedure.

Similarly, other sequence of these integers enumerate all `guo:to-rule-` ... nodes, also starting from zero.

Consider a sequence of `guo:from-rule-` ... nodes, because `guo:to-rule-` nodes have identical properties.

A rule node can have zero or more values of `guo:dep` property, each value is a bnode that is rule node that should be calculated before the current one.

Every rule has exactly one predicate `guo:path` that is a blank node. Each property of this blank node describes one possible "move of slider": predicate to follow is in predicate position and a node to start from is in object position. An IRI or a literal in object position is used as is, a blank node in object position should be of type `guo:from-rule-` ... and have smaller `guo:order` so it refers to already calculated result bnode of some preceding rule.

Rule of form:

```
R a guo:from-rule-IFP1 ;
  guo:path [ P1 O1 ; P2 O2 ; ... ; Pn On ] .
```

searches for a unique blank node `_:Rres` that is a common subject of triples:

```
 _:Rres P1 O1
 _:Rres P2 O2
  . . .
 _:Rres Pn On
```

in the gfrom graph.

If subjects differ in these triples or some triples are not found or the subject is not a blank node then an appropriate error is logged and rule fails, otherwise `_:Rres` is remembered as the result of the rule.

Similarly, rule of form:

```
R a guo:from-rule-FP1 ;
  guo:path [ P1 O1 ; P2 O2 ; ... ; Pn On ] .
```

searches for a unique blank node `_:Rres` that is a common object of triples:

```
 O1 P1 _:Rres
 O2 P2 _:Rres
  . . .
 On Pn _:Rres
```

in the gfrom graph.

Rule of form:

```
R a guo:from-rule-IFP0 ;
  guo:path [ P1 O1 ; P2 O2 ; ... ; Pn On ] .
```

ensures that the `gfrom` graph does not contain any triple like:

```
 _:Rres P1 O1
 _:Rres P2 O2
```

or

```
_:Rres Pn On
```

It is an error if something exists. If nothing found then the result of the rule is newly created unique blank node. That's how patch procedure creates new blank nodes when it inserts "totally new" data.

Similarly, rule of form:

```
R a guo:from-rule-IFP0 ;
  guo:path [ P1 O1 ; P2 O2 ; ... ; Pn On ] .
```

ensures that the `gfrom` graph does not contain any triple like:

```
O1 P1 _:Rres
O2 P2 _:Rres
```

or

```
 On Pn _:Rres
```

Current version of patch procedure does not use rules `guo:to-rule-` ... , however they can be used by custom procedure of few sorts. First, these rules can be used to produce a "reversed diff". Next, these rules can be used to validate the result of the patch - if the patch can not be reverted then the result is "suspicious".

16.17.17. RDB2RDF Triggers

Linked Data Views have many advantages, if compared to static dumps of the database in RDF triples. However, they does not solve few problems. First, inference is supported only for physically stored triples, so one had to chose between convenience of inference and convenience of Linked Data Views. Next, algorithms that selects triples with non-constant graphs and predicates tend to produce enormous texts of SQL queries if Linked Data Views are complicated enough. Finally, there may be a need in export of big and fresh static RDF dump but preparing this dump would take too much time via both RDF Views and traditional methods.

The solution is set of triggers on source tables of an Linked Data View that edit parts of physical dump on each change of source data. Unlike Linked Data Views that cost nothing while not queried, these triggers add a significant overhead on any data manipulation on sources, continuously. To compensate this, the dump should be in an intensive use and not replaceable by Linked Data Views. In other cases, do not add these triggers.

It is next to impossible to write such triggers by hands so a small API is provided to generate SQL texts from metadata of Linked Data Views.

First of all, views in an RDF storage does not work in full isolation from each other. Some of them may partially disable others due to OPTION(EXCLUSIVE) and some may produce one triple in different ways. As a result, triggers are not made on per-view basis. Instead, a special RDF storage is introduced, namely virtrdf:SyncToQuads , all required triples are added to it and triggers are created for the whole storage. Typically an Linked Data View is created in some other storage, e.g., virtrdf:DefaultQuadStorage and then added to virtrdf:SyncToQuads via:

```
sparql alter quad storage virtrdf:SyncToQuads {
   create <my_rdf_view> using storage virtrdf:DefaultQuadStorage };
```

The following example procedure copies all user-defined Linked Data Views from default quad storage to virtrdf:SyncToQuads:

```
create procedure DB.DBA.RDB2RDF_COPY_ALL_RDF_VIEWS_TO_SYNC ()
{
  for (sparql define input:storage ""
    select (bif:aref(bif:sprintf_inverse (str(?idx), bif:concat (str(rdf:_), "%d"), 0), 0)) ?qm
    from virtrdf:
    where { virtrdf:DefaultQuadStorage-UserMaps ?idx ?qm . ?qm a virtrdf:QuadMap }
    order by asc (bif:sprintf_inverse (bif:concat (str(rdf:_), "%d"), str (?idx), 1)) ) do
    exec (sprintf ('sparql alter quad storage virtrdf:SyncToQuads { create <%s> using storage virtrdf:Def
}
;
```

When the virtrdf:SyncToQuads storage is fully prepared, two API functions can be used:

- DB.DBA.SPARQL_RDB2RDF_LIST_TABLES : The function returns a vector of names of tables that are used as sources for Linked Data Views. Application developer should decide what to do with each of them - create triggers or do some application-specific workarounds.

Note that if some SQL views are used as sources for Linked Data Views and these views does not have INSTEAD triggers then workarounds become mandatory for them, not just a choice, because BEFORE or AFTER triggers on views are not allowed if there is no appropriate INSTEAD trigger. The mode argument should be zero in current version.

- DB.DBA.SPARQL_RDB2RDF_CODEGEN : The function creates an SQL text for a given table and an operation specified by an opcode.

In some cases, Linked Data Views are complicated enough so that BEFORE UPDATE and AFTER DELETE triggers are required in additional to the minimal set. In this case, sparql_rdb2rdf_codegen calls will return a vector of two string sessions, not single string session, and both sessions are sql texts to inspect or execute. In this case, the BEFORE trigger will not delete obsolete quads from RDF_QUAD table, instead it will create records in a special table RDF_QUAD_DELETE_QUEUE as guesses what can be deleted. The AFTER trigger will re-check these guesses, delete related quads if needed and shorten the RDF_QUAD_DELETE_QUEUE.

The extra activity of triggers on RDF_QUAD, RDF_OBJ, RDF_QUAD_DELETE_QUEUE and other tables and indexes of the storage of "physical" triples may cause deadlocks so the application should be carefully checked for proper support of deadlocks if they were very seldom before turning RDB2RDF triggers on. In some cases, the whole processing of RDB2RDF can be moved to a separate server and connected to the main workhorse server via replication.

The following example functions create texts of all triggers, save them to files in for further studying and try to load them. That's probably quite bad scenario for a production database, because it's better to read procedures before loading them, especially if they're triggers, especially if some of them may contain errors.

```
-- This creates one or two files with one or two triggers or other texts and try to load the
generated sql texts.
create procedure DB.DBA.RDB2RDF_EXEC_CODEGEN1_FOR_TABLE
(  in dump_prefix varchar,
   in tbl varchar,
   in dump_id any,
   in txt any )
{
  declare fname varchar;
  declare stat, msg varchar;
  if (isinteger (dump_id))
    dump_id := cast (dump_id as varchar);
  if (__tag of vector = __tag (txt))
    {
      DB.DBA.RDB2RDF_EXEC_CODEGEN1_FOR_TABLE (dump_prefix, tbl, dump_id, txt[0]);
      DB.DBA.RDB2RDF_EXEC_CODEGEN1_FOR_TABLE (dump_prefix, tbl, dump_id || 'p' , txt[1]);
      return;
    }
  if (__tag of varchar <> __tag (txt))
    txt := string_output_string (txt);
  fname := sprintf ('%s_Rdb2Rdf.%s.%s.sql', dump_prefix, tbl, dump_id);
  string_to_file (fname, txt || '\n;\n', -2);
  if ('0' = dump_id)
    return;
  stat := '00000';
  msg := '';
  exec (txt, stat, msg);
  if ('00000' <> stat)
    {
      string_to_file (fname, '\n\n- - - - - 8< - - - - -\n\nError ' || stat || ' ' || msg, -1);
      if (not (subseq (msg, 0, 5) in ('SQ091')))
        signal (stat, msg);
    }
}
;

-- This creates and loads all triggers, init procedure and debug dump related to one table.
create procedure DB.DBA.RDB2RDF_PREPARE_TABLE (in dump_prefix varchar, in tbl varchar)
{
  declare ctr integer;
  for (ctr := 0; ctr <= 4; ctr := ctr+1 )
    DB.DBA.RDB2RDF_EXEC_CODEGEN1_FOR_TABLE (dump_prefix, tbl, ctr, sparql_rdb2rdf_codegen (tbl, ctr));
}
;

-- This creates and loads all triggers, init procedure and debug dump related to all tables used by and L
create procedure DB.DBA.RDB2RDF_PREPARE_ALL_TABLES (in dump_prefix varchar)
```

```
{
  declare tbl_list any;
  tbl_list := sparql_rdb2rdf_list_tables (0);
  foreach (varchar tbl in tbl_list) do
    {
      DB.DBA.RDB2RDF_PREPARE_TABLE (dump_prefix, tbl);
    }
}
;
```

The following combination of calls prepares all triggers for all Linked Data Views of the default storage:

```
DB.DBA.RDB2RDF_COPY_ALL_RDF_VIEWS_TO_SYNC ();
DB.DBA.RDB2RDF_PREPARE_ALL_TABLES (cast (now() as varchar));
```

This does not copy the initial state of RDB2RDF graphs to the physical storage, because this can be dangerous for existing RDF data and even if all procedures will work as expected then they may produce huge amounts of RDF data, run out of transaction log limits and thus require application-specific precautions. It is also possible to make initial loading by a SPARUL statements like:

```
SPARQL
INSERT IN <snapshot-graph> { ?s ?p ?o }
FROM <snapshot-htaph>
WHERE
 { quad map <id-of-rdf-view>
          { ?s ?p ?o }
 };
```

16.18. RDF Data Access Providers (Drivers)

16.18.1. Virtuoso Jena Provider

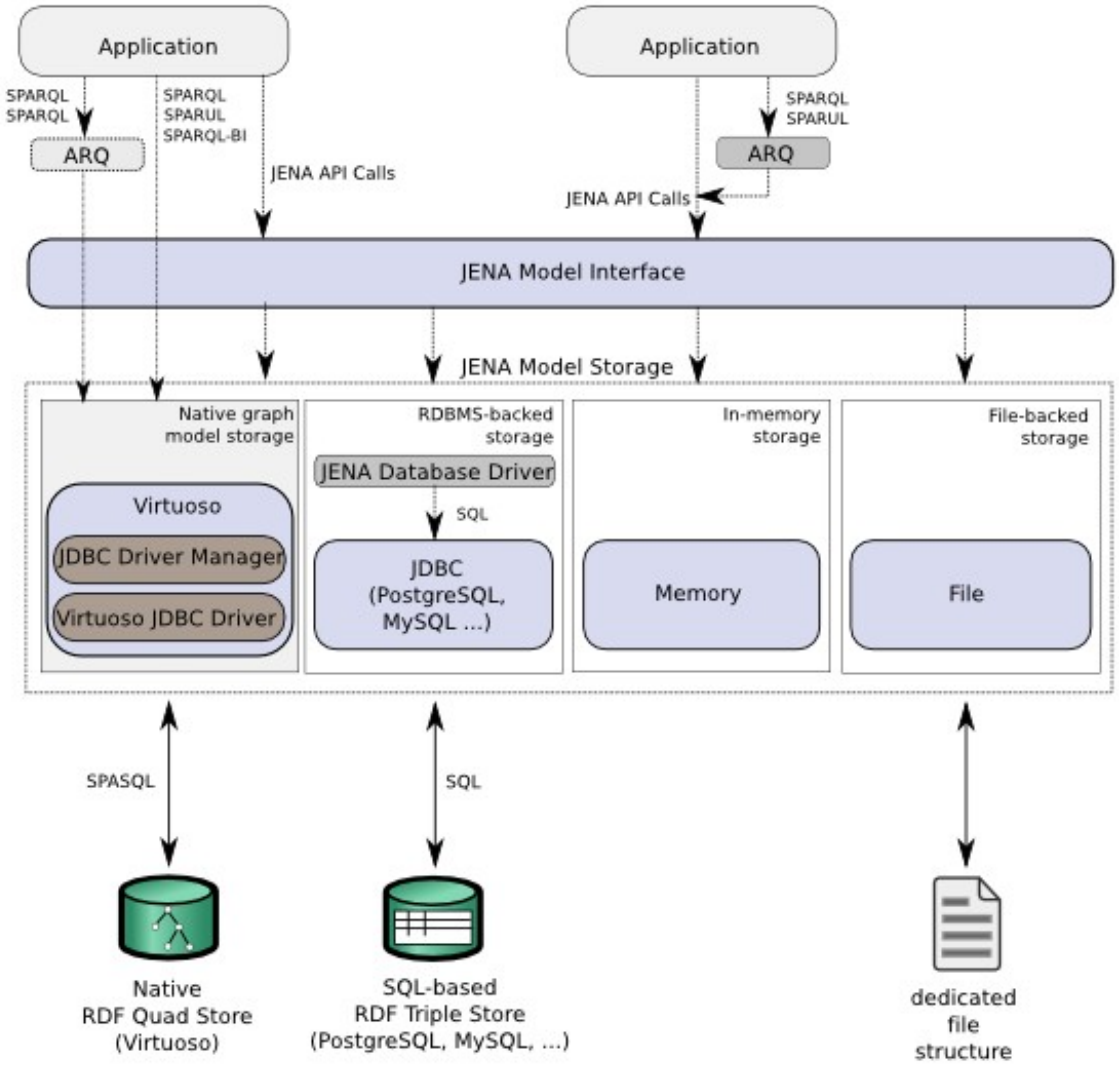
What is Jena

Jena is an open source Semantic Web framework for Java. It provides an API to extract data from and write to RDF graphs. The graphs are represented as an abstract "model". A model can be sourced with data from files, databases, URIs or a combination of these. A Model can also be queried through SPARQL and updated through SPARUL.

What is the Virtuoso Jena Provider

The Virtuoso Jena RDF Data Provider is a fully operational Native Graph Model Storage Provider for the Jena Framework, which enables Semantic Web applications written using the Jena RDF Frameworks to directly query the Virtuoso RDF Quad Store. Providers are available for the latest Jena 2.6.x and 2.10.x versions.

Figure 16.185. Virtuoso Jena RDF Data Provider



Setup

Prerequisites

Download the latest Virtuoso Jena Provider for your Jena framework version, Virtuoso JDBC 3 Driver, Jena Framework, and associated classes and sample programs.

- *Note:*

  The Jena Provider is explicitly bound to the Virtuoso JDBC 3 Driver. You cannot use the Virtuoso JDBC 4 Driver for this purpose at this time.
- The version of the Jena Provider (virt_jena.jar) can be verified with the command:

```
$ java -jar virt_jena.jar
OpenLink Virtuoso(TM) Provider for Jena(TM) Version 2.6.2 [Build 1.2]
```

- Files contained in the zip files are generally older than specifically linked downloads (e.g., the Virtuoso JDBC Driver, virtjdbc3.jar), so don't replace if prompted during extraction. Instead, rename the file extracted from the zip, and compare their versions to be sure you keep only the most recent.

```
$ java -cp virtjdbc3.jar virtuoso.jdbc3.Driver
OpenLink Virtuoso(TM) Driver for JDBC(TM) Version 3.x [Build 3.57]
$ java -cp virtjdbc3.fromzip.jar virtuoso.jdbc3.Driver
OpenLink Virtuoso(TM) Driver for JDBC(TM) Version 3.x [Build 3.11]
```

- Downloads:
  - Virtuoso Jena Provider JAR file, virt_jena.jar
  - Virtuoso JDBC Driver 3 JAR file, virtjdbc3.jar

- Virtuoso JDBC Driver 4 JAR file, virtjdbc4.jar
- Jena Framework and associated classes, jenajars.zip
- Sample programs, virtjenasamples.zip

Compiling Jena Sample Programs

1. Edit the sample programs VirtuosoSPARQLExampleX.java, where X = 1 to 9. Set the JDBC connection strings within to point to a valid Virtuoso Server instance of the form:

   ```
   "jdbc:virtuoso://localhost:1111/charset=UTF-8/log_enable=2"
   ```

   - charset=UTF-8 will be added by Jena provider, if it isn't in connection string. So now you don't need add "charset=UTF-8" to the connection string any more, it is done by Jena provider.
   - log_enable=2: to use row auto commit
   - use these settings to process large rdf data.
2. Ensure that full paths to

   jena.jar, arq.jar,

   and

   virtjdbc3.jar

   are included in the active CLASSPATH setting.
3. Compile the Jena Sample applications using the following command:

   ```
   javac -cp "jena.jar:arq.jar:virtjdbc3.jar:virt_jena.jar:." VirtuosoSPARQLExample1.java
   VirtuosoSPARQLExample2.java VirtuosoSPARQLExample3.java VirtuosoSPARQLExample4.java
   VirtuosoSPARQLExample5.java VirtuosoSPARQLExample6.java VirtuosoSPARQLExample7.java
   VirtuosoSPARQLExample8.java VirtuosoSPARQLExample9.java
   ```

Testing

Once the Provider classes and sample program have been successfully compiled, the Provider can be tested using the sample programs included. Ensure your active CLASSPATH includes full paths to all of the following files, before executing the example commands:

- icu4j_3_4.jar
- iri.jar
- xercesImpl.jar
- axis.jar
- commons-logging-1.1.1.jar
- jena.jar
- arq.jar
- virtjdbc3.jar
- virt_jena.jar

1. VirtuosoSPARQLExample1 returns the contents of the RDF Quad store of the targeted Virtuoso instance, with the following command:

   ```
   java VirtuosoSPARQLExample1
   ```
2. VirtuosoSPARQLExample2 reads in the contents of the following FOAF URIs --

   ```
   http://kidehen.idehen.net/dataspace/person/kidehen#this
   http://www.w3.org/People/Berners-Lee/card#i
   http://demo.openlinksw.com/dataspace/person/demo#this
   ```

   -- and returns the RDF data stored, with the following command:

   ```
   java VirtuosoSPARQLExample2
   ```
3. VirtuosoSPARQLExample3 performs simple addition and deletion operation on the content of the triple store, with the following command:

```
java VirtuosoSPARQLExample3
```

4. VirtuosoSPARQLExample4 demonstrates the use of the *graph.contains* method for searching triples, with the following command:

```
java VirtuosoSPARQLExample4
```

5. VirtuosoSPARQLExample5 demonstrates the use of the *graph.find* method for searching triples, with the following command:

```
java VirtuosoSPARQLExample5
```

6. VirtuosoSPARQLExample6 demonstrates the use of the *graph.getTransactionHandler* method, with the following command:

```
java VirtuosoSPARQLExample6
```

7. VirtuosoSPARQLExample7 demonstrates the use of the graph.getBulkUpdateHandler method, with the following command:

```
java VirtuosoSPARQLExample7
```

8. VirtuosoSPARQLExample8 demonstrates how to insert triples into a graph, with the following command:

```
java VirtuosoSPARQLExample8
```

9. VirtuosoSPARQLExample9 demonstrates the use of the *CONSTRUCT, DESCRIBE,* and *ASK* SPARQL query forms, with the following command:

```
java VirtuosoSPARQLExample9
```

Examples

VirtJenaSPARQLExample1

```
import com.hp.hpl.jena.query.*;
import com.hp.hpl.jena.rdf.model.RDFNode;

import virtuoso.jena.driver.*;

public class VirtuosoSPARQLExample1 {

        /**
         * Executes a SPARQL query against a virtuoso url and prints results.
         */
        public static void main(String[] args) {

                String url;
                if(args.length == 0)
                    url = "jdbc:virtuoso://localhost:1111";
                else
                    url = args[0];

/*                      STEP 1                  */
                VirtGraph set = new VirtGraph (url, "dba", "dba");

/*                      STEP 2                  */

/*                      STEP 3                  */
/*              Select all data in virtuoso     */
                Query sparql = QueryFactory.create("SELECT * WHERE { GRAPH ?graph { ?s ?p ?o } } limit 10

/*                      STEP 4                  */
                VirtuosoQueryExecution vqe = VirtuosoQueryExecutionFactory.create (sparql, set);

                ResultSet results = vqe.execSelect();
                while (results.hasNext()) {
                        QuerySolution result = results.nextSolution();
                    RDFNode graph = result.get("graph");
                    RDFNode s = result.get("s");
                    RDFNode p = result.get("p");
                    RDFNode o = result.get("o");
                    System.out.println(graph + " { " + s + " " + p + " " + o + " . }");
                }
        }
```

```
}
```

VirtJenaSPARQLExample2

```
import com.hp.hpl.jena.query.*;
import com.hp.hpl.jena.rdf.model.RDFNode;

import virtuoso.jena.driver.*;

public class VirtuosoSPARQLExample2 {

        /**
         * Executes a SPARQL query against a virtuoso url and prints results.
         */
        public static void main(String[] args) {

                String url;
                if(args.length == 0)
                    url = "jdbc:virtuoso://localhost:1111";
                else
                    url = args[0];

/*                      STEP 1                  */
                VirtGraph graph = new VirtGraph ("Example2", url, "dba", "dba");

/*                      STEP 2                  */
/*              Load data to Virtuoso           */
                graph.clear ();

                System.out.print ("Begin read from 'http://www.w3.org/People/Berners-Lee/card#i'  ");
                graph.read("http://www.w3.org/People/Berners-Lee/card#i", "RDF/XML");
                System.out.println ("\t\t\t Done.");

                System.out.print ("Begin read from 'http://demo.openlinksw.com/dataspace/person/demo#this
                graph.read("http://demo.openlinksw.com/dataspace/person/demo#this", "RDF/XML");
                System.out.println ("\t Done.");

                System.out.print ("Begin read from 'http://kidehen.idehen.net/dataspace/person/kidehen#th
                graph.read("http://kidehen.idehen.net/dataspace/person/kidehen#this", "RDF/XML");
                System.out.println ("\t Done.");

/*                      STEP 3                  */
/*              Select only from VirtGraph      */
                Query sparql = QueryFactory.create("SELECT ?s ?p ?o WHERE { ?s ?p ?o }");

/*                      STEP 4                  */
                VirtuosoQueryExecution vqe = VirtuosoQueryExecutionFactory.create (sparql, graph);

                ResultSet results = vqe.execSelect();
                while (results.hasNext()) {
                        QuerySolution result = results.nextSolution();
                    RDFNode graph_name = result.get("graph");
                    RDFNode s = result.get("s");
                    RDFNode p = result.get("p");
                    RDFNode o = result.get("o");
                    System.out.println(graph_name + " { " + s + " " + p + " " + o + " . }");
                }

                System.out.println("graph.getCount() = " + graph.getCount());
        }
}
```

VirtJenaSPARQLExample3

```
import java.util.*;

import com.hp.hpl.jena.query.*;
import com.hp.hpl.jena.rdf.model.RDFNode;
import com.hp.hpl.jena.graph.Node;
import com.hp.hpl.jena.graph.Triple;

import virtuoso.jena.driver.*;
```

```
public class VirtuosoSPARQLExample3
{
    public static void main(String[] args)
    {
        String url;

        if(args.length == 0)
            url = "jdbc:virtuoso://localhost:1111";
        else
            url = args[0];

        Node foo1 = Node.createURI("http://example.org/#foo1");
        Node bar1 = Node.createURI("http://example.org/#bar1");
        Node baz1 = Node.createURI("http://example.org/#baz1");

        Node foo2 = Node.createURI("http://example.org/#foo2");
        Node bar2 = Node.createURI("http://example.org/#bar2");
        Node baz2 = Node.createURI("http://example.org/#baz2");

        Node foo3 = Node.createURI("http://example.org/#foo3");
        Node bar3 = Node.createURI("http://example.org/#bar3");
        Node baz3 = Node.createURI("http://example.org/#baz3");

        List <Triple> triples = new ArrayList <Triple> ();

        VirtGraph graph = new VirtGraph ("Example3", url, "dba", "dba");

        graph.clear ();

        System.out.println("graph.isEmpty() = " + graph.isEmpty());
        System.out.println("Add 3 triples to graph <Example3>.");

        graph.add(new Triple(foo1, bar1, baz1));
        graph.add(new Triple(foo2, bar2, baz2));
        graph.add(new Triple(foo3, bar3, baz3));

        System.out.println("graph.isEmpty() = " + graph.isEmpty());
        System.out.println("graph.getCount() = " + graph.getCount());

        triples.add(new Triple(foo1, bar1, baz1));
        triples.add(new Triple(foo2, bar2, baz2));

        graph.isEmpty();

        System.out.println("Remove 2 triples from graph <Example3>");
        graph.remove(triples);
        System.out.println("graph.getCount() = " + graph.getCount());
        System.out.println("Please check result with isql tool.");

        /* EXPECTED RESULT:

SQL> SPARQL
SELECT ?s ?p ?o
FROM <Example3>
WHERE {?s ?p ?o};
s                                                    p
VARCHAR                                      VARCHAR                                                  VARCHAR
_____________________________________________________________________________________________________

http://example.org/#foo3             http://example.org/#bar3                                http://example.org

1 Rows. -- 26 msec.
SQL>

*/

        }
}
```

VirtJenaSPARQLExample4

```
import java.util.*;

import com.hp.hpl.jena.query.*;
import com.hp.hpl.jena.rdf.model.RDFNode;
import com.hp.hpl.jena.graph.Node;
import com.hp.hpl.jena.graph.Triple;

import virtuoso.jena.driver.*;

public class VirtuosoSPARQLExample4
{

    public static void main(String[] args)
    {
        String url;
        if(args.length == 0)
            url = "jdbc:virtuoso://localhost:1111";
        else
            url = args[0];

        Node foo1 = Node.createURI("http://example.org/#foo1");
        Node bar1 = Node.createURI("http://example.org/#bar1");
        Node baz1 = Node.createURI("http://example.org/#baz1");

        Node foo2 = Node.createURI("http://example.org/#foo2");
        Node bar2 = Node.createURI("http://example.org/#bar2");
        Node baz2 = Node.createURI("http://example.org/#baz2");

        Node foo3 = Node.createURI("http://example.org/#foo3");
        Node bar3 = Node.createURI("http://example.org/#bar3");
        Node baz3 = Node.createURI("http://example.org/#baz3");

        VirtGraph graph = new VirtGraph ("Example4", url, "dba", "dba");

        graph.clear ();

        System.out.println("graph.isEmpty() = " + graph.isEmpty());

        System.out.println("Add 3 triples to graph <Example4>.");

        graph.add(new Triple(foo1, bar1, baz1));
        graph.add(new Triple(foo2, bar2, baz2));
        graph.add(new Triple(foo3, bar3, baz3));

        System.out.println("graph.isEmpty() = " + graph.isEmpty());
        System.out.println("graph.getCount() = " + graph.getCount());

        System.out.println ("graph.contains(new Triple(foo2, bar2, baz2) - " + graph.contains(new Triple(
        System.out.println ("graph.contains(new Triple(foo2, bar2, baz3) - " + graph.contains(new Triple(

        graph.clear ();

    }
}
```

VirtJenaSPARQLExample5

```
import java.util.*;

import com.hp.hpl.jena.query.*;
import com.hp.hpl.jena.util.iterator.ExtendedIterator;
import com.hp.hpl.jena.graph.Node;
import com.hp.hpl.jena.graph.Triple;

import virtuoso.jena.driver.*;

public class VirtuosoSPARQLExample5
{

    public static void main(String[] args)
    {
```

```
        String url;
        if(args.length == 0)
            url = "jdbc:virtuoso://localhost:1111";
        else
            url = args[0];

        Node foo1 = Node.createURI("http://example.org/#foo1");
        Node bar1 = Node.createURI("http://example.org/#bar1");
        Node baz1 = Node.createURI("http://example.org/#baz1");

        Node foo2 = Node.createURI("http://example.org/#foo2");
        Node bar2 = Node.createURI("http://example.org/#bar2");
        Node baz2 = Node.createURI("http://example.org/#baz2");

        Node foo3 = Node.createURI("http://example.org/#foo3");
        Node bar3 = Node.createURI("http://example.org/#bar3");
        Node baz3 = Node.createURI("http://example.org/#baz3");

        VirtGraph graph = new VirtGraph ("Example5", url, "dba", "dba");

        graph.clear ();

        System.out.println("graph.isEmpty() = " + graph.isEmpty());

        System.out.println("Add 3 triples to graph <Example5>.");

        graph.add(new Triple(foo1, bar1, baz1));
        graph.add(new Triple(foo2, bar2, baz2));
        graph.add(new Triple(foo3, bar3, baz3));
        graph.add(new Triple(foo1, bar2, baz2));
        graph.add(new Triple(foo1, bar3, baz3));

        System.out.println("graph.isEmpty() = " + graph.isEmpty());
        System.out.println("graph.getCount() = " + graph.getCount());

        ExtendedIterator iter = graph.find(foo1, Node.ANY, Node.ANY);
        System.out.println ("\ngraph.find(foo1, Node.ANY, Node.ANY) \nResult:");
        for ( ; iter.hasNext() ; )
            System.out.println ((Triple) iter.next());

        iter = graph.find(Node.ANY, Node.ANY, baz3);
        System.out.println ("\ngraph.find(Node.ANY, Node.ANY, baz3) \nResult:");
        for ( ; iter.hasNext() ; )
            System.out.println ((Triple) iter.next());

        iter = graph.find(foo1, Node.ANY, baz3);
        System.out.println ("\ngraph.find(foo1, Node.ANY, baz3) \nResult:");
        for ( ; iter.hasNext() ; )
            System.out.println ((Triple) iter.next());

        graph.clear ();

    }
}
```

VirtJenaSPARQLExample6

```
import java.util.*;

import com.hp.hpl.jena.query.*;
import com.hp.hpl.jena.util.iterator.ExtendedIterator;
import com.hp.hpl.jena.graph.Node;
import com.hp.hpl.jena.graph.Triple;

import virtuoso.jena.driver.*;

public class VirtuosoSPARQLExample6
{

    public static void main(String[] args)
    {
        String url;
        if(args.length == 0)
```

```
            url = "jdbc:virtuoso://localhost:1111";
        else
            url = args[0];

        Node foo1 = Node.createURI("http://example.org/#foo1");
        Node bar1 = Node.createURI("http://example.org/#bar1");
        Node baz1 = Node.createURI("http://example.org/#baz1");

        Node foo2 = Node.createURI("http://example.org/#foo2");
        Node bar2 = Node.createURI("http://example.org/#bar2");
        Node baz2 = Node.createURI("http://example.org/#baz2");

        Node foo3 = Node.createURI("http://example.org/#foo3");
        Node bar3 = Node.createURI("http://example.org/#bar3");
        Node baz3 = Node.createURI("http://example.org/#baz3");

        VirtGraph graph = new VirtGraph ("Example6", url, "dba", "dba");

        graph.clear ();

        System.out.println("graph.isEmpty() = " + graph.isEmpty());

        System.out.println("test Transaction Commit.");
        graph.getTransactionHandler().begin();
        System.out.println("begin Transaction.");
        System.out.println("Add 3 triples to graph <Example6>.");

        graph.add(new Triple(foo1, bar1, baz1));
        graph.add(new Triple(foo2, bar2, baz2));
        graph.add(new Triple(foo3, bar3, baz3));

        graph.getTransactionHandler().commit();
        System.out.println("commit Transaction.");
        System.out.println("graph.isEmpty() = " + graph.isEmpty());
        System.out.println("graph.getCount() = " + graph.getCount());

        ExtendedIterator iter = graph.find(Node.ANY, Node.ANY, Node.ANY);
        System.out.println ("\ngraph.find(Node.ANY, Node.ANY, Node.ANY) \nResult:");
        for ( ; iter.hasNext() ; )
            System.out.println ((Triple) iter.next());

        graph.clear ();
        System.out.println("\nCLEAR graph <Example6>");
        System.out.println("graph.isEmpty() = " + graph.isEmpty());

        System.out.println("Add 1 triples to graph <Example6>.");
        graph.add(new Triple(foo1, bar1, baz1));

        System.out.println("test Transaction Abort.");
        graph.getTransactionHandler().begin();
        System.out.println("begin Transaction.");
        System.out.println("Add 2 triples to graph <Example6>.");

        graph.add(new Triple(foo2, bar2, baz2));
        graph.add(new Triple(foo3, bar3, baz3));

        graph.getTransactionHandler().abort();
        System.out.println("abort Transaction.");
        System.out.println("graph.isEmpty() = " + graph.isEmpty());
        System.out.println("graph.getCount() = " + graph.getCount());

        iter = graph.find(Node.ANY, Node.ANY, Node.ANY);
        System.out.println ("\ngraph.find(Node.ANY, Node.ANY, Node.ANY) \nResult:");
        for ( ; iter.hasNext() ; )
            System.out.println ((Triple) iter.next());

        graph.clear ();
        System.out.println("\nCLEAR graph <Example6>");

    }
}
```

VirtJenaSPARQLExample7

```
import java.util.*;

import com.hp.hpl.jena.query.*;
import com.hp.hpl.jena.util.iterator.ExtendedIterator;
import com.hp.hpl.jena.graph.Node;
import com.hp.hpl.jena.graph.Triple;

import virtuoso.jena.driver.*;

public class VirtuosoSPARQLExample7
{

    public static void main(String[] args)
    {
        String url;
        if(args.length == 0)
            url = "jdbc:virtuoso://localhost:1111";
        else
            url = args[0];

        Node foo1 = Node.createURI("http://example.org/#foo1");
        Node bar1 = Node.createURI("http://example.org/#bar1");
        Node baz1 = Node.createURI("http://example.org/#baz1");

        Node foo2 = Node.createURI("http://example.org/#foo2");
        Node bar2 = Node.createURI("http://example.org/#bar2");
        Node baz2 = Node.createURI("http://example.org/#baz2");

        Node foo3 = Node.createURI("http://example.org/#foo3");
        Node bar3 = Node.createURI("http://example.org/#bar3");
        Node baz3 = Node.createURI("http://example.org/#baz3");

        List triples1 = new ArrayList();
        triples1.add(new Triple(foo1, bar1, baz1));
        triples1.add(new Triple(foo2, bar2, baz2));
        triples1.add(new Triple(foo3, bar3, baz3));

        List triples2 = new ArrayList();
        triples2.add(new Triple(foo1, bar1, baz1));
        triples2.add(new Triple(foo2, bar2, baz2));

        VirtGraph graph = new VirtGraph ("Example7", url, "dba", "dba");

        graph.clear ();

        System.out.println("graph.isEmpty() = " + graph.isEmpty());
        System.out.println("Add List with 3 triples to graph <Example7> via BulkUpdateHandler.");

        graph.getBulkUpdateHandler().add(triples1);

        System.out.println("graph.isEmpty() = " + graph.isEmpty());
        System.out.println("graph.getCount() = " + graph.getCount());

        ExtendedIterator iter = graph.find(Node.ANY, Node.ANY, Node.ANY);
        System.out.println ("\ngraph.find(Node.ANY, Node.ANY, Node.ANY) \nResult:");
        for ( ; iter.hasNext() ; )
            System.out.println ((Triple) iter.next());

        System.out.println("\n\nDelete List of 2 triples from graph <Example7> via BulkUpdateHandler.");

        graph.getBulkUpdateHandler().delete(triples2);

        System.out.println("graph.isEmpty() = " + graph.isEmpty());
        System.out.println("graph.getCount() = " + graph.getCount());

        iter = graph.find(Node.ANY, Node.ANY, Node.ANY);
        System.out.println ("\ngraph.find(Node.ANY, Node.ANY, Node.ANY) \nResult:");
        for ( ; iter.hasNext() ; )
            System.out.println ((Triple) iter.next());

        graph.clear ();
```

```
            System.out.println("\nCLEAR graph <Example7>");

        }
    }
```

VirtJenaSPARQLExample8

```
    import com.hp.hpl.jena.query.*;
    import com.hp.hpl.jena.rdf.model.RDFNode;

    import virtuoso.jena.driver.*;

    public class VirtuosoSPARQLExample8 {

            /**
             * Executes a SPARQL query against a virtuoso url and prints results.
             */
            public static void main(String[] args) {

                    String url;
                    if(args.length == 0)
                        url = "jdbc:virtuoso://localhost:1111";
                    else
                        url = args[0];

    /*                      STEP 1                  */
                    VirtGraph set = new VirtGraph (url, "dba", "dba");

    /*                      STEP 2                  */
    System.out.println("\nexecute: CLEAR GRAPH <http://test1>");
                    String str = "CLEAR GRAPH <http://test1>";
                    VirtuosoUpdateRequest vur = VirtuosoUpdateFactory.create(str, set);
                    vur.exec();

    System.out.println("\nexecute: INSERT INTO GRAPH <http://test1> { <aa> <bb> 'cc' . <aa1> <bb1> 123. }");
                    str = "INSERT INTO GRAPH <http://test1> { <aa> <bb> 'cc' . <aa1> <bb1> 123. }";
                    vur = VirtuosoUpdateFactory.create(str, set);
                    vur.exec();

    /*                      STEP 3                  */
    /*              Select all data in virtuoso     */
    System.out.println("\nexecute: SELECT * FROM <http://test1> WHERE { ?s ?p ?o }");
                    Query sparql = QueryFactory.create("SELECT * FROM <http://test1> WHERE { ?s ?p ?o }");

    /*                      STEP 4                  */
                    VirtuosoQueryExecution vqe = VirtuosoQueryExecutionFactory.create (sparql, set);

                    ResultSet results = vqe.execSelect();
                    while (results.hasNext()) {
                            QuerySolution rs = results.nextSolution();
                        RDFNode s = rs.get("s");
                        RDFNode p = rs.get("p");
                        RDFNode o = rs.get("o");
                        System.out.println(" { " + s + " " + p + " " + o + " . }");
                    }

    System.out.println("\nexecute: DELETE FROM GRAPH <http://test1> { <aa> <bb> 'cc' }");
                    str = "DELETE FROM GRAPH <http://test1> { <aa> <bb> 'cc' }";
                    vur = VirtuosoUpdateFactory.create(str, set);
                    vur.exec();

    System.out.println("\nexecute: SELECT * FROM <http://test1> WHERE { ?s ?p ?o }");
                    vqe = VirtuosoQueryExecutionFactory.create (sparql, set);
                    results = vqe.execSelect();
                    while (results.hasNext()) {
                            QuerySolution rs = results.nextSolution();
                        RDFNode s = rs.get("s");
                        RDFNode p = rs.get("p");
                        RDFNode o = rs.get("o");
                        System.out.println(" { " + s + " " + p + " " + o + " . }");
                    }

            }
```

```
}
```

VirtJenaSPARQLExample9

```
import com.hp.hpl.jena.query.*;
import com.hp.hpl.jena.rdf.model.RDFNode;
import com.hp.hpl.jena.graph.Triple;
import com.hp.hpl.jena.graph.Node;
import com.hp.hpl.jena.graph.Graph;
import com.hp.hpl.jena.rdf.model.*;
import java.util.Iterator;

import virtuoso.jena.driver.*;

public class VirtuosoSPARQLExample9 {

        /**
         * Executes a SPARQL query against a virtuoso url and prints results.
         */
        public static void main(String[] args) {

                String url;
                if(args.length == 0)
                    url = "jdbc:virtuoso://localhost:1111";
                else
                    url = args[0];

/*                      STEP 1                  */
                VirtGraph set = new VirtGraph (url, "dba", "dba");

/*                      STEP 2                  */
                String str = "CLEAR GRAPH <http://test1>";
                VirtuosoUpdateRequest vur = VirtuosoUpdateFactory.create(str, set);
                vur.exec();

                str = "INSERT INTO GRAPH <http://test1> { <http://aa> <http://bb> 'cc' . <http://aa1> <ht
                vur = VirtuosoUpdateFactory.create(str, set);
                vur.exec();

/*              Select all data in virtuoso     */
                Query sparql = QueryFactory.create("SELECT * FROM <http://test1> WHERE { ?s ?p ?o }");
                VirtuosoQueryExecution vqe = VirtuosoQueryExecutionFactory.create (sparql, set);
                ResultSet results = vqe.execSelect();
                System.out.println("\nSELECT results:");
                while (results.hasNext()) {
                        QuerySolution rs = results.nextSolution();
                    RDFNode s = rs.get("s");
                    RDFNode p = rs.get("p");
                    RDFNode o = rs.get("o");
                    System.out.println(" { " + s + " " + p + " " + o + " . }");
                }

                sparql = QueryFactory.create("DESCRIBE <http://aa> FROM <http://test1>");
                vqe = VirtuosoQueryExecutionFactory.create (sparql, set);

                Model model = vqe.execDescribe();
                Graph g = model.getGraph();
                System.out.println("\nDESCRIBE results:");
                for (Iterator i = g.find(Node.ANY, Node.ANY, Node.ANY); i.hasNext();)
                   {
                      Triple t = (Triple)i.next();
                      System.out.println(" { " + t.getSubject() + " " +
                                                 t.getPredicate() + " " +
                                                 t.getObject() + " . }");
                }

                sparql = QueryFactory.create("CONSTRUCT { ?x <http://test> ?y } FROM <http://test1> WHERE
                vqe = VirtuosoQueryExecutionFactory.create (sparql, set);

                model = vqe.execConstruct();
                g = model.getGraph();
                System.out.println("\nCONSTRUCT results:");
                for (Iterator i = g.find(Node.ANY, Node.ANY, Node.ANY); i.hasNext();)
```

```
                    {
                        Triple t = (Triple)i.next();
                        System.out.println(" { " + t.getSubject() + " " +
                                                   t.getPredicate() + " " +
                                                   t.getObject() + " . }");
                    }

                    sparql = QueryFactory.create("ASK FROM <http://test1> WHERE { <http://aa> <http://bb> ?y
                    vqe = VirtuosoQueryExecutionFactory.create (sparql, set);

                    boolean res = vqe.execAsk();
                    System.out.println("\nASK results: "+res);

            }
    }
```

Javadoc API Documentation

Javadocs covers the complete set of classes, interfaces, and methods implemented by the provider:

- Javadoc API Documentation for the Jena 2.6 Provider
- Javadoc API Documentation for the Jena 2.10+ Provider

16.18.2. Virtuoso Sesame Provider

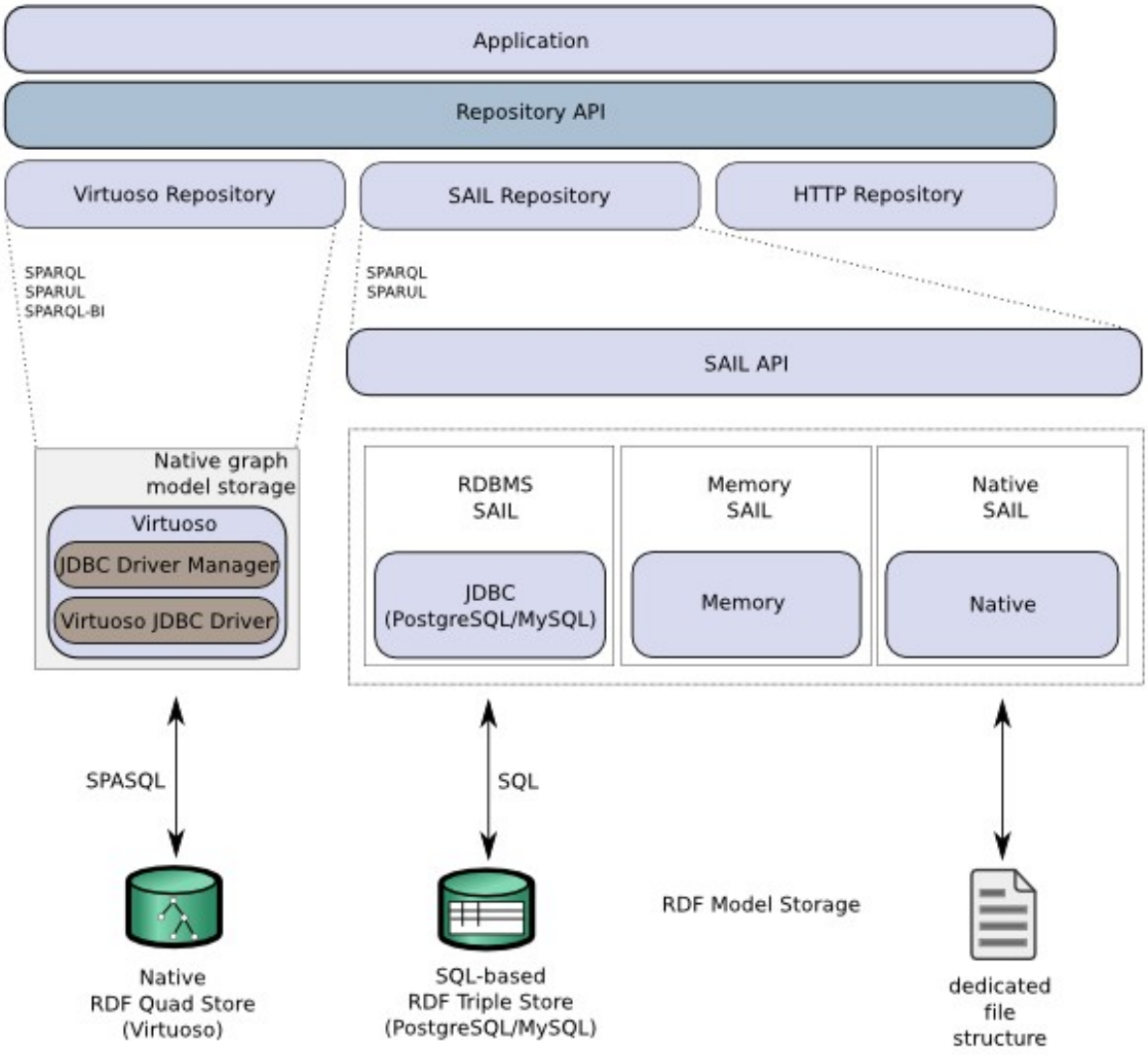
What is Sesame

Sesame is an open source Java framework for storing, querying, and reasoning with RDF and RDF Schema. It can be used as a database for RDF and RDF Schema, or as a Java library for applications that need to work with RDF internally. For example, suppose you need to read a big RDF file, find the relevant information for your application, and use that information. Sesame provides you with the necessary tools to parse, interpret, query, and store all this information, embedded in your own application if you want, or, if you prefer, in a separate database or even on a remote server. More generally: Sesame provides an application developer with a toolbox that contains useful hammers, screwdrivers, etc., for 'Do-It-Yourself' projects with RDF.

What is the Virtuoso Sesame Provider

The Virtuoso Sesame Provider is a fully operational Native Graph Model Storage Provider for the Sesame Framework, allowing users of Virtuoso to leverage the Sesame framework to modify, query, and reason with the Virtuoso quad store using the Java language. The Sesame Repository API offers a central access point for connecting to the Virtuoso quad store. Its purpose is to provides a Java-friendly access point to Virtuoso. It offers various methods for querying and updating the data, while abstracting the details of the underlying machinery. The Provider has been tested against the two latest currently available versions, Sesame 2.6.x, 2.7.x, 2.8.x and the new Sesame 4.x release, for which a new Provider is available.

Figure 16.186. Fig. 1 Sesame Component Stack



If you need more information about how to set up your environment for working with the Sesame APIs, take a look at Chapter 2 of the Sesame User Guide, Setting up to use the Sesame libraries .

Setup

Required Files

This tutorial assumes you have Virtuoso server installed and that the database is accessible at "localhost:1111". In addition, the relevant version of the Virtuoso Sesame Provider, and Sesame java framework need to be installed.

You should download the Virtuoso Sesame Provider JAR archive for the version of Sesame being used, Virtuoso JDBC Driver, Sesame Framework and associated classes and sample programs from our download page . Note the version of the Sesame Provider (virt_sesameX.jar) can be determined with the command:

```
$ java -jar virt_sesame2.jar
OpenLink Virtuoso(TM) Provider for Sesame2(TM) Version 2.6.5 [Build 1.7]
$ java -jar virt_sesame4.jar
OpenLink Virtuoso(TM) Provider for Sesame4(TM) Version 4.0.0 [Build 0.1]
$
```

Sesame 2 Sample Program

Compilation

1. Ensure that full paths to the following files, or equivalents for your version of Sesame, are all included in the active CLASSPATH setting --

- openrdf-sesame-2.1.2-onejar.jar
- slf4j-api-1.5.0.jar
- slf4j-jdk14-1.5.0.jar
- commons-io-2.0.jar
- virtjdbc3.jar
- virt_sesame2.jar

2. Execute the following command:

```
javac VirtuosoTest.java
```

Note: we recommend adding the following to the connect string, to use utf-8 and row-auto-commit:

```
"/charset=UTF-8/log_enable=2"
-- i.e. in VirtuosoTest.java the line:
Repository repository = new VirtuosoRepository("jdbc:virtuoso://" + sa[0] + ":" + sa[1], sa[2], sa
-- should become:
Repository repository = new VirtuosoRepository("jdbc:virtuoso://" + sa[0] + ":" + sa[1]+ "/charset
```

Testing

1. Ensure that full paths to the following files are all included in the active CLASSPATH setting (note the addition of virtuoso_driver, here):

- openrdf-sesame-2.1.2-onejar.jar
- slf4j-api-1.5.0.jar
- slf4j-jdk14-1.5.0.jar
- commons-io-2.0.jar
- virtjdbc3.jar
- virt_sesame2.jar
- virtuoso_driver

2. Run the VirtuosoTest program to test the Sesame 2 Provider with the following command:

```
java VirtuosoTest <hostname> <port> <uid> <pwd>
```

3. The test run should look like this:

```
$ java VirtuosoTest localhost 1111 dba dba

== TEST 1:  : Start
   Loading data from URL: http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com/foaf.r
== TEST 1:  : End
PASSED: TEST 1

== TEST 2:  : Start
   Clearing triple store
== TEST 2:  : End
PASSED: TEST 2

== TEST 3:  : Start
   Loading data from file: virtuoso_driver/data.nt
== TEST 3:  : End
PASSED: TEST 3

== TEST 4:  : Start
   Loading UNICODE single triple
== TEST 4:  : End
PASSED: TEST 4

== TEST 5:  : Start
   Loading single triple
== TEST 5:  : End
PASSED: TEST 5

== TEST 6:  : Start
   Casted value type
== TEST 6:  : End
PASSED: TEST 6

== TEST 7:  : Start
```

```
   Selecting property
== TEST 7:  : End
PASSED: TEST 7

== TEST 8:  : Start
   Statement does not exists
== TEST 8:  : End
PASSED: TEST 8

== TEST 9:  : Start
   Statement exists (by resultset size)
== TEST 9:  : End
PASSED: TEST 9

== TEST 10:  : Start
   Statement exists (by hasStatement())
== TEST 10:  : End
PASSED: TEST 10

== TEST 11:  : Start
   Retrieving namespaces
== TEST 11:  : End
PASSED: TEST 11

== TEST 12:  : Start
   Retrieving statement (http://myopenlink.net/dataspace/person/kidehen http://myopenlink.net/foaf
== TEST 12:  : End
PASSED: TEST 12

== TEST 13:  : Start
   Writing the statements to file: (/Users/src/virtuoso-opensource/binsrc/sesame2/results.n3.txt)
== TEST 13:  : End
PASSED: TEST 13

== TEST 14:  : Start
   Retrieving graph ids
== TEST 14:  : End
PASSED: TEST 14

== TEST 15:  : Start
   Retrieving triple store size
== TEST 15:  : End
PASSED: TEST 15

== TEST 16:  : Start
   Sending ask query
== TEST 16:  : End
PASSED: TEST 16

== TEST 17:  : Start
   Sending construct query
== TEST 17:  : End
PASSED: TEST 17

== TEST 18:  : Start
   Sending describe query
== TEST 18:  : End
PASSED: TEST 18

============================
PASSED:18 FAILED:0
```

Sesame 4 Sample Program

Compilation

1. Ensure that full paths to the following files, or equivalents for your version of Sesame, are all included in the active CLASSPATH setting --

   - openrdf-sesame-4.0.0-onejar.jar
   - slf4j-api-1.7.10.jar
   - commons-io-2.4.jar

- virtjdbc4.jar
- virt_sesame4.jar

2. Execute the following command:

```
javac VirtuosoTest.java
```

Note: we recommend adding the following to the connect string, to use utf-8 and row-auto-commit:

```
"/charset=UTF-8/log_enable=2"
-- i.e. in VirtuosoTest.java the line:
Repository repository = new VirtuosoRepository("jdbc:virtuoso://" + sa[0] + ":" + sa[1], sa[2], sa
-- should become:
Repository repository = new VirtuosoRepository("jdbc:virtuoso://" + sa[0] + ":" + sa[1]+ "/charset
```

Testing

1. Ensure that full paths to the following files are all included in the active CLASSPATH setting (note the addition of virtuoso_driver, here):

- openrdf-sesame-4.0.0-onejar.jar
- slf4j-api-1.7.10.jar
- commons-io-2.4.jar
- virtjdbc4.jar
- virt_sesame4.jar
- virtuoso_driver

2. Run the VirtuosoTest program to test the Sesame 2 Provider with the following command:

```
java VirtuosoTest <hostname> <port> <uid> <pwd>
```

3. The test run should look like this:

```
$ java VirtuosoTest localhost 1111 dba dba

== TEST 1:  : Start
== TEST 1:  : End
PASSED: TEST 1

== TEST 2:  : Start
   Loading data from URL: http://dbpedia.org/data/Berlin.rdf
log4j:WARN No appenders could be found for logger (org.openrdf.rio.RDFParserRegistry).
log4j:WARN Please initialize the log4j system properly.
== TEST 2:  : End
PASSED: TEST 2

== TEST 3:  : Start
   Clearing triple store
== TEST 3:  : End
PASSED: TEST 3

== TEST 4:  : Start
   Loading data from file: virtuoso_driver/data.nt
== TEST 4:  : End
PASSED: TEST 4

== TEST 5:  : Start
   Loading UNICODE single triple
== TEST 5:  : End
PASSED: TEST 5

== TEST 6:  : Start
   Loading single triple
== TEST 6:  : End
PASSED: TEST 6

== TEST 7:  : Start
   Casted value type
== TEST 7:  : End
PASSED: TEST 7

== TEST 8:  : Start
```

```
   Selecting property
== TEST 8:  : End
PASSED: TEST 8

== TEST 9:  : Start
   Statement does not exists
== TEST 9:  : End
PASSED: TEST 9

== TEST 10:  : Start
   Statement exists (by resultset size)
== TEST 10:  : End
PASSED: TEST 10

== TEST 11:  : Start
   Statement exists (by hasStatement())
== TEST 11:  : End
PASSED: TEST 11

== TEST 12:  : Start
   Retrieving namespaces
== TEST 12:  : End
PASSED: TEST 12

== TEST 13:  : Start
   Retrieving statement (http://myopenlink.net/dataspace/person/kidehen http://myopenlink.net/foaf
== TEST 13:  : End
PASSED: TEST 13

== TEST 14:  : Start
   Writing the statements to file: (/Users/hwilliams/src/git/vos-7-develop/binsrc/sesame4/results.
== TEST 14:  : End
PASSED: TEST 14

== TEST 15:  : Start
   Retrieving graph ids
== TEST 15:  : End
PASSED: TEST 15

== TEST 16:  : Start
   Retrieving triple store size
== TEST 16:  : End
PASSED: TEST 16

== TEST 17:  : Start
   Sending ask query
== TEST 17:  : End
PASSED: TEST 17

== TEST 18:  : Start
   Sending construct query
== TEST 18:  : End
PASSED: TEST 18

== TEST 19:  : Start
   Sending describe query
== TEST 19:  : End
PASSED: TEST 19

============================
PASSED:19 FAILED:0
```

Getting Started

This section covers the essentials for connecting to and manipulating data stored in a Virtuoso repository using the Sesame API. More information on the Sesame Framework, including extended examples on how to use the API, can be found in Chapter 8 of the Sesame User's guide, the RepositoryConnection API .

The interfaces for the Repository API can be found in packages virtuoso.sesame2.driver and org.openrdf.repository. Several implementations for these interfaces exist in the Virtuoso Provider download package. The Javadoc reference for the Sesame API is available online and can also be found in the doc directory of the download.

Creating a Virtuoso Repository RDF object

The first step to connecting to Virtuoso through the Sesame API is to create a Repository for it. The Repository object operates on (stacks of) Sail object(s) for storage and retrieval of RDF data.

One of the simplest configurations is a repository that just stores RDF data in main memory, without applying any inference. This is also by far the fastest type of repository that can be used. The following code creates and initialize a non-inferencing main-memory repository:

```
import virtuoso.sesame2.driver.VirtuosoRepository;

Repository myRepository = VirtuosoRepository("jdbc:virtuoso://localhost:1111","dba","dba");

myRepository.initialize();
```

The constructor of the VirtuosoRepository class accepts the JDBC URL of the Virtuoso engine, and the username and password of an authorized user. Following this example, the repository needs to be initialized to prepare the Sail(s) that it operates on, which includes performing operations such as restoring previously stored data, setting up connections to a relational database, etc.

The repository that is created by the above code is volatile: its contents are lost when the object is garbage collected or when the program is shut down. This is fine for cases where, for example, the repository is used as a means for manipulating an RDF model in memory.

Creating a Virtuoso Repository Connection

Now that we have created a VirtuosoRepository object instance, we want to do something with it. This is achieved through the use of the VirtuosoRepositoryConnection class, which can be created by the VirtuosoRepository class.

A VirtuosoRepositoryConnection represents fi as the name suggests fi a connection to the actual Virtuoso quad store. We can issue operations over this connection, and close it when we are done to make sure we are not keeping resources unnecessarily occupied.

In the following sections, we will show some examples of basic operations using the Northwind dataset.

Adding RDF to Virtuoso

The Repository implements the Sesame Repository API, which offers various methods for adding data to a repository. Data can be added programmatically by specifying the location of a file that contains RDF data, and statements can be added individually or in collections.

We perform operations on the repository by requesting a RepositoryConnection from the repository, which returns a VirtuosoRepositoryConnection object. On this VirtuosoRepositoryConnection object we can perform the various operations, such as query evaluation; getting, adding, or removing statements; etc.

The following example code adds two files, one local and one located on the Web, to a repository:

```
import org.openrdf.repository.RepositoryException;

import org.openrdf.repository.Repository;

import org.openrdf.repository.RepositoryConnection;

import org.openrdf.rio.RDFFormat;

import java.io.File;

import java.net.URL;

File file = new File("/path/to/example.rdf");

String baseURI = "http://example.org/example/localRDF";

?
try {

   RepositoryConnection con = myRepository.getConnection();
```

```
        try {
            con.add(file, baseURI, RDFFormat.RDFXML);
            URL url = new URL("http://example.org/example/remoteRDF");
            con.add(url, url.toString(), RDFFormat.RDFXML);
        }
        finally {
            con.close();
        }
    }
    catch (RepositoryException rex) {
        // handle exception
    }
    catch (java.io.IOEXception e) {
        // handle io exception
    }
```

More information on other available methods can be found in the javadoc reference of the RepositoryConnection interface.

Querying Virtuoso

The Repository API has a number of methods for creating and evaluating queries. Three types of queries are distinguished: tuple queries, graph queries, and Boolean queries. The query types differ in the type of results that they produce.

Select Query: The result of a select query is a set of tuples (or variable bindings), where each tuple represents a solution of the query. This type of query is commonly used to get specific values (URIs, blank nodes, literals) from the stored RDF data. The method QueryFactory.executeQuery() returns a Value [ ][ ] for SPARQL "SELECT" queries. The method QueryFactory.executeQuery() also calls the QueryFactory.setResult() which populates a set of tuples for SPARQL "SELECT" queries. The graph can be retrieved using QueryFactory.getBooleanResult().

Graph Query: The result of a graph query is an RDF graph (or set of statements). This type of query is very useful for extracting sub-graphs from the stored RDF data, which can then be queried further, serialized to an RDF document, etc. The method QueryFactory.executeQuery() calls the QueryFactory.setGraphResult() which populates a graph for SPARQL "DESCRIBE" and "CONSTRUCT" queries. The graph can be retrieved using QueryFactory.getGraphResult().

Boolean Query: The result of a Boolean query is a simple Boolean value, i.e., TRUE or FALSE. This type of query can be used to check if a repository contains specific information. The method QueryFactory.executeQuery() calls the QueryFactory.setBooleanResult() which sets a Boolean value for SPARQL "ASK" queries. The value can be retrieved using QueryFactory.getBooleanResult().

Note: Although Sesame 2 currently supports two query languages: SeRQL and SPARQL, the Virtuoso provider only supports the W3C SPARQL specification at this time.

Evaluating a SELECT Query

To evaluate a tuple query we simply do the following:

```
    import java.util.List;
    import org.openrdf.OpenRDFException;
    import org.openrdf.repository.RepositoryConnection;
    import org.openrdf.query.TupleQuery;
```

```
import org.openrdf.query.TupleQueryResult;

import org.openrdf.query.BindingSet;

import org.openrdf.query.QueryLanguage;
?

try {

   RepositoryConnection con = myRepository.getConnection();

   try {

      String queryString = "SELECT x, y FROM  WHERE {x} p {y}";

      TupleQuery tupleQuery = con.prepareTupleQuery(QueryLanguage.SPARQL, queryString);

      TupleQueryResult result = tupleQuery.evaluate();

      try {

         ? // do something with the result

      }

      finally {

         result.close();

      }

   }

   finally {

      con.close();

   }

}

catch (RepositoryException e) {

   // handle exception

}
```

This evaluates a SPARQL query and returns a TupleQueryResult, which consists of a sequence of BindingSet objects. Each BindingSet contains a set of pairs called Binding objects. A Binding object represents a name/value pair for each variable in the query's projection.

We can use the TupleQueryResult to iterate over all results and get each individual result for x and y:

```
while (result.hasNext()) {

   BindingSet bindingSet = result.next();

   Value valueOfX = bindingSet.getValue("x");

   Value valueOfY = bindingSet.getValue("y");

   // do something interesting with the query variable values here?

}
```

As you can see, we retrieve values by name rather than by an index. The names used should be the names of variables as specified in your query. The TupleQueryResult.getBindingNames() method returns a list of binding names, in the order in which they were specified in the query. To process the bindings in each binding set in the order specified by the projection, you can do the following:

```
List bindingNames = result.getBindingNames();

while (result.hasNext()) {

   BindingSet bindingSet = result.next();

   Value firstValue = bindingSet.getValue(bindingNames.get(0));

   Value secondValue = bindingSet.getValue(bindingNames.get(1));

   // do something interesting with the values here?

}
```

It is important to invoke the close() operation on the TupleQueryResult, after we are done with it. A TupleQueryResult evaluates lazily and keeps resources (such as connections to the underlying database) open. Closing the TupleQueryResult frees up these resources. Do not forget that iterating over a result may cause exceptions! The best way to make sure no connections are kept open unnecessarily is to invoke close() in the finally clause.

An alternative to producing a TupleQueryResult is to supply an object that implements the TupleQueryResultHandler interface to the query's evaluate() method. The main difference is that when using a return object, the caller has control over when the next answer is retrieved, whereas with the use of a handler, the connection simply pushes answers to the handler object as soon as it has them available.

As an example we will use SPARQLResultsXMLWriter, which is a TupleQueryResultHandler implementation that writes SPARQL Results XML documents to an output stream or to a writer:

```
import org.openrdf.query.resultio.sparqlxml.SPARQLResultsXMLWriter;

?
FileOutputStream out = new FileOutputStream("/path/to/result.srx");

try {

   SPARQLResultsXMLWriter sparqlWriter = new SPARQLResultsXMLWriter(out);

   RepositoryConnection con = myRepository.getConnection();

   try {

      String queryString = "SELECT * FROM  WHERE {x} p {y}";

      TupleQuery tupleQuery = con.prepareTupleQuery(QueryLanguage.SPARQL, queryString);

      tupleQuery.evaluate(sparqlWriter);

   }

   finally {

      con.close();

   }

}

finally {

   out.close();

}
```

You can just as easily supply your own application-specific implementation of TupleQueryResultHandler, if desired.

Lastly, an important warning: as soon as you are done with the RepositoryConnection object, you should close it. Notice that during processing of the TupleQueryResult object (for example, when iterating over its contents), the RepositoryConnection should still be open. We can invoke con.close() after we have finished with the result.

Evaluating a CONSTRUCT query

The following code evaluates a graph query on a repository:

```
import org.openrdf.query.GraphQueryResult;

GraphQueryResult graphResult = con.prepareGraphQuery(

      QueryLanguage.SPARQL, "CONSTRUCT * FROM {x} p {y}").evaluate();
```

A GraphQueryResult is similar to TupleQueryResult in that it is an object that iterates over the query results. However, for graph queries the query results are RDF statements, so a GraphQueryResult iterates over Statement objects:

```
while (graphResult.hasNext()) {

   Statement st = graphResult.next();

   // ? do something with the resulting statement here.

}
```

The TupleQueryResultHandler equivalent for graph queries is org.openrdf.rio.RDFHandler. Again, this is a generic interface; each object implementing it can process the reported RDF statements in any way it wants.

All writers from Rio (such as the RDFXMLWriter, TurtleWriter, TriXWriter, etc.) implement the RDFHandler interface. This allows them to be used in combination with querying quite easily. In the following example, we use a TurtleWriter to write the result of a SPARQL graph query to standard output in Turtle format:

```
import org.openrdf.rio.turtle.TurtleWriter;

?
RepositoryConnection con = myRepository.getConnection();

try {

   TurtleWriter turtleWriter = new TurtleWriter(System.out);

   con.prepareGraphQuery(QueryLanguage.SPARQL, "CONSTRUCT * FROM  WHERE {x} p {y}").evaluate(turtleWriter

}

finally {

   con.close();

}
```

Again, note that as soon as we are done with the result of the query (either after iterating over the contents of the GraphQueryResult or after invoking the RDFHandler), we invoke con.close() to close the connection and free resources.

Javadoc API Documentation

Javadocs covers the complete set of classes, interfaces, and methods implemented by the provider:

- Javadoc API Documentation for the Sesame 2.6 Provider
- Javadoc API Documentation for the Sesame 2.7+ Provider
- Javadoc API Documentation for the Sesame 4.x Provider

Virtuoso Sesame HTTP Repository Configuration and Usage

What

Sesame is an open source Java framework for storing, querying and reasoning with RDF and RDF Schema. It can be used as a database for RDF and RDF Schema, or as a Java library for applications that need to work with RDF internally. The Sesame HTTP repository serves as a proxy for a RDF store hosted on a remote Sesame server, enabling the querying of the RDF store using the Sesame HTTP protocol.

Why

The Sesame HTTP repository endpoint provides users with the greater flexibility for manipulating the RDF store via a common interface. Sesame provides you with the necessary tools to parse, interpret, query and store all this information, embedded in your own application if you want, or, if you prefer, in a separate database or even on a remote server.

How

To create a new Sesame HTTP repository, the Console needs to create such an RDF document and submit it to the SYSTEM repository. The Console uses so called repository configuration templates to accomplish this. Repository configuration templates are simple Turtle RDF files that describe a repository configuration, where some of the parameters are replaced with variables. The Console parses these templates and asks the user to supply values for the variables. The variables are then substituted with the specified values, which produces the required configuration data.

Setup and Testing

This section details the steps required for configuring and testing a Virtuoso Sesame Repository, both using the HTTP and Console Sesame repositories.

Requirements

- Sesame 2.3.1 or higher
- Virtuoso Sesame 2.x (virt_sesame2.jar) or Virtuoso Sesame 4.x (virt_sesame4.jar) Provider
- Virtuoso JDBC Driver (virtjdbc4.jar)
- Virtuoso System Repository config file for Sesame 2.x or Sesame 4.x (create.xsl)
- Virtuoso Repository config file for Sesame 2.x or Sesame 4.x (create-virtuoso.xsl)
- Configuration Template file for a Virtuoso Repository for Sesame 2.x or Sesame 4.x
- Apache Tomcat version 5 or higher

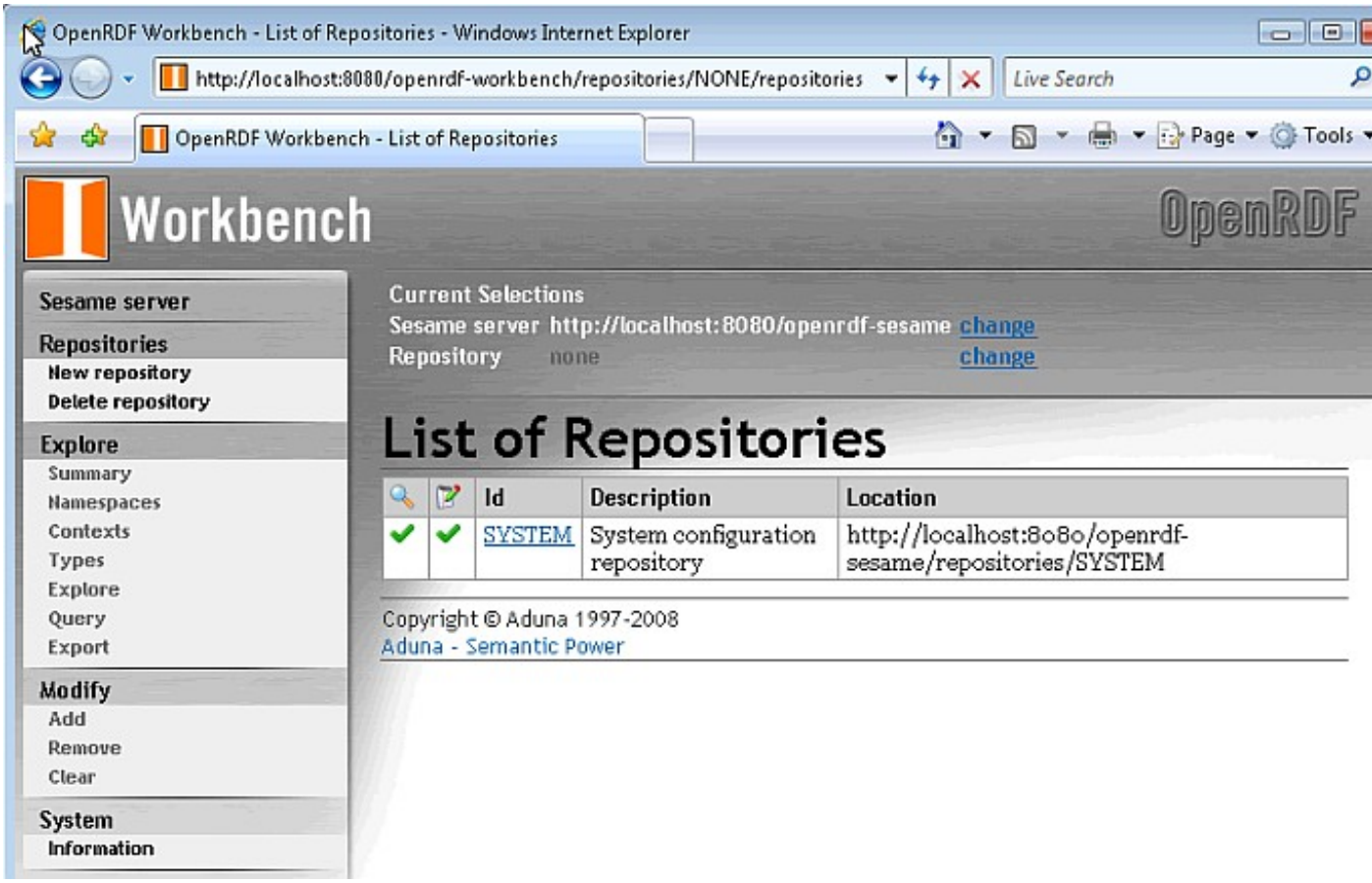
Setup Sesame HTTP Repository

This section details the steps required for configuring and testing a Virtuoso HTTP Sesame Repository.

1. Install Apache Tomcat web server
2. From the Sesame 2.3.1 or higher "lib" directory copy the "openrdf-sesame.war" and "openrdf-worbbench.war" files to the tomcat "webapps" directory where they will automatically be deployed creating two new sub directories "openrdf-sesame" and "openrdf-workbench".
3. Place the corresponding Virtuoso Sesame Provider "virt_sesame2.jar" or "virt_sesame4.jar" and JDBC Driver "virtjdbc4.jar" into the Tomcat ~/webapps/openrdf-sesame/WEB-INF/lib/ and ~/webapps/openrdf-workbench/WEB-INF/lib/ directories for use by the Sesame HTTP Repository for accessing the Virtuoso RDF repository.
4. Place the "create.xsl" and "create-virtuoso.xsl" files in the Tomcat ~/webapps/openrdf-workbench/transformations/ directory. Note "create.xsl" replaces the default provided with Sesame and contains the necessary entries required to reference the new "create-virtuoso.xsl" template file for Virtuoso repository configuration.
5. The Sesame HTTP Repository will now be accessible on the URLs

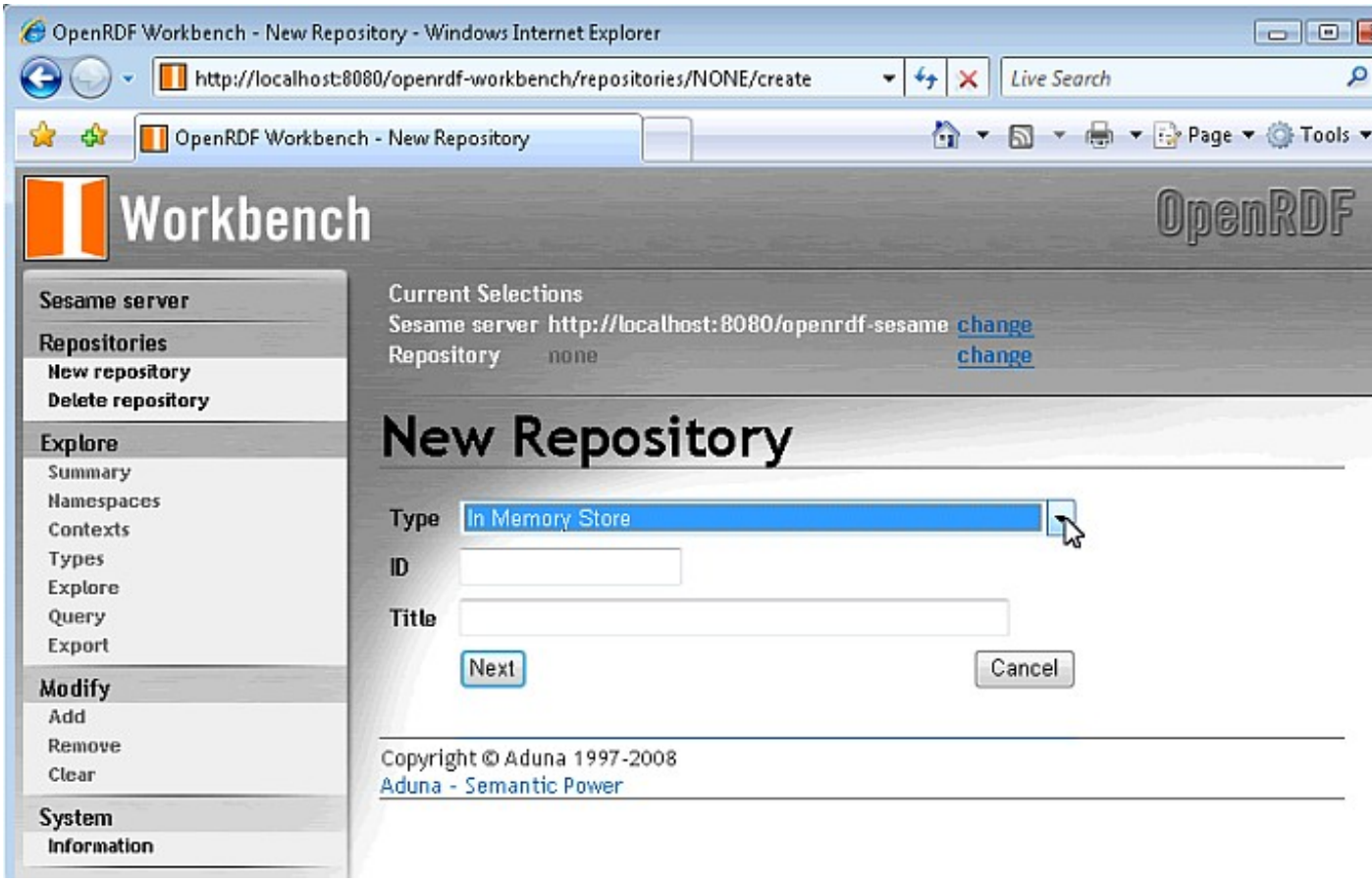
   ```
   http://example.com/openrdf-sesame
   http://example.com/openrdf-workbench
   ```
6. The Sesame OpenRDF Workbench is used for accessing the Sesame HTTP Repositories, loading " http://example.com/openrdf-workbench " will enable the default "SYSTEM" repository to be accessed.

Figure 16.187. Virtuoso Sesame HTTP Repository Configuration and Usage



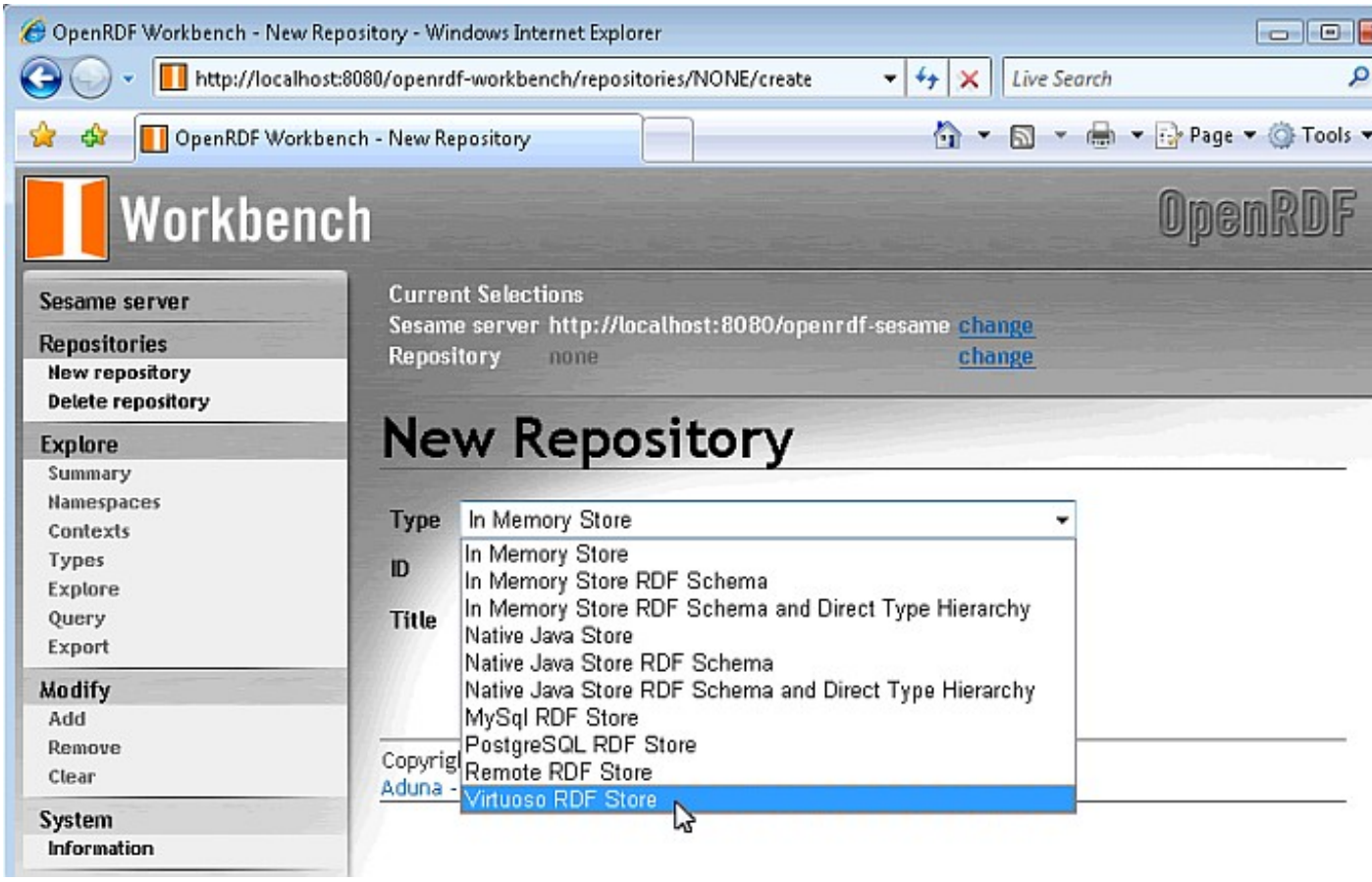
7. Click on the "New Repository" link in the left frame to create a new Sesame Repository.

Figure 16.188. Virtuoso Sesame HTTP Repository Configuration and Usage



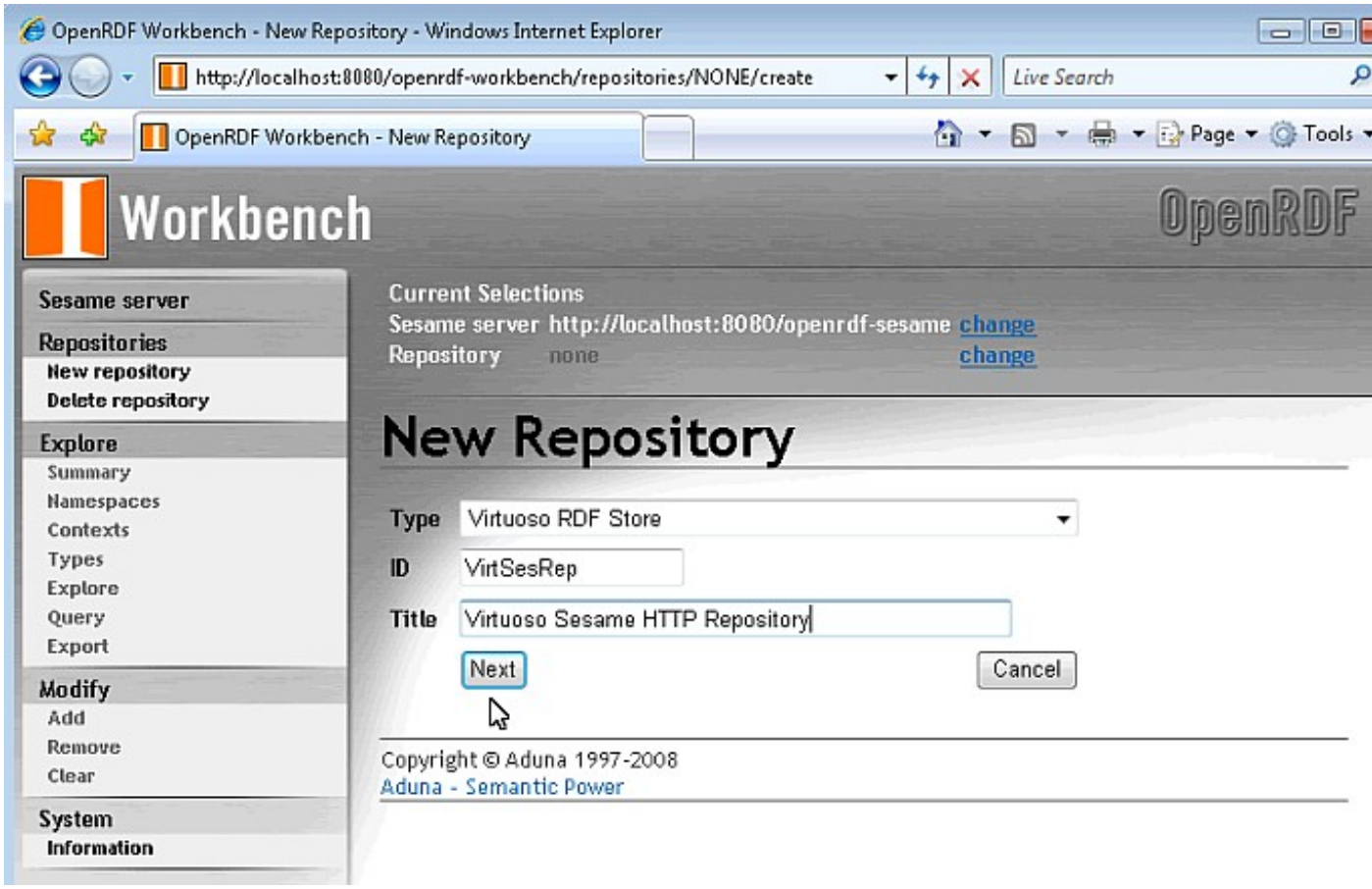
8. Select the "Virtuoso RDF Store" from the "Type" drop down list box presented.

Figure 16.189. Virtuoso Sesame HTTP Repository Configuration and Usage



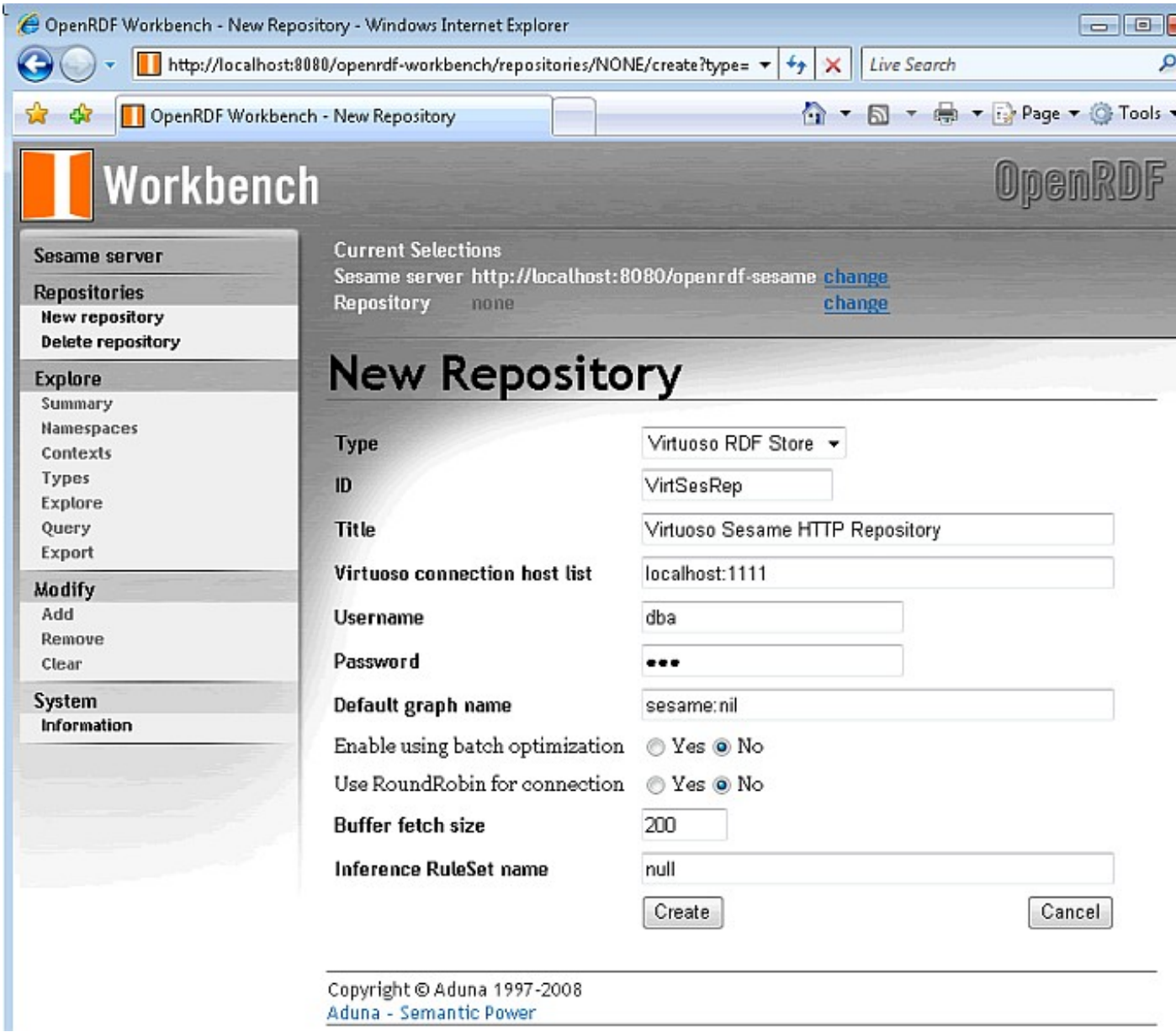
9. Choose suitable repository "ID" and "Title" for the Virtuoso repository to be created and click "Next".

Figure 16.190. Virtuoso Sesame HTTP Repository Configuration and Usage



10. Fill in the connection parameters for the target Virtuoso sever the repository is to be created for and click the "create" button. The minimum required are the hostname, port number, username and password of the Virtuoso Server.

Figure 16.191. Virtuoso Sesame HTTP Repository Configuration and Usage

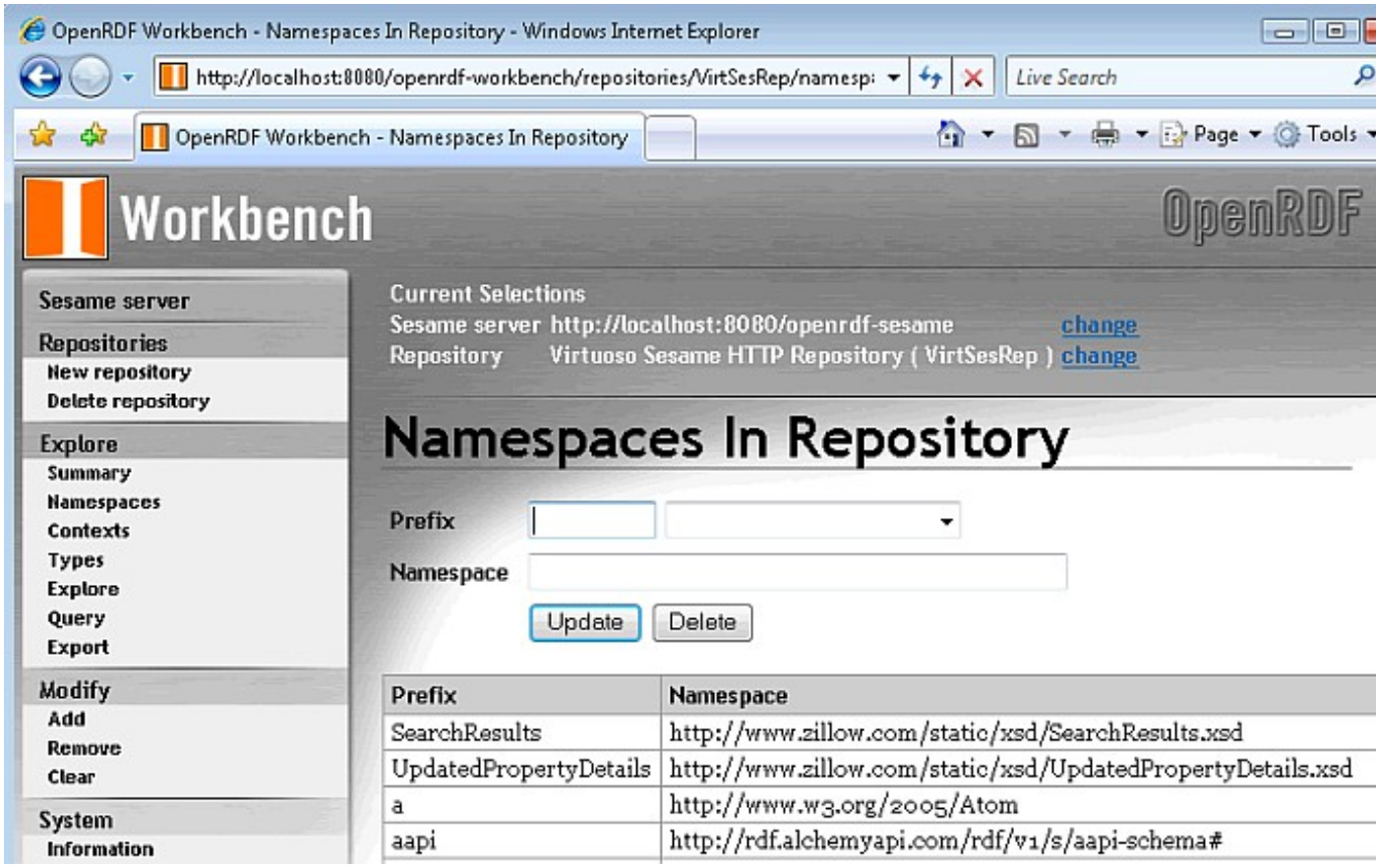


11. The new Virtuoso respository will be created and its summary page displayed.

Figure 16.192. Virtuoso Sesame HTTP Repository Configuration and Usage

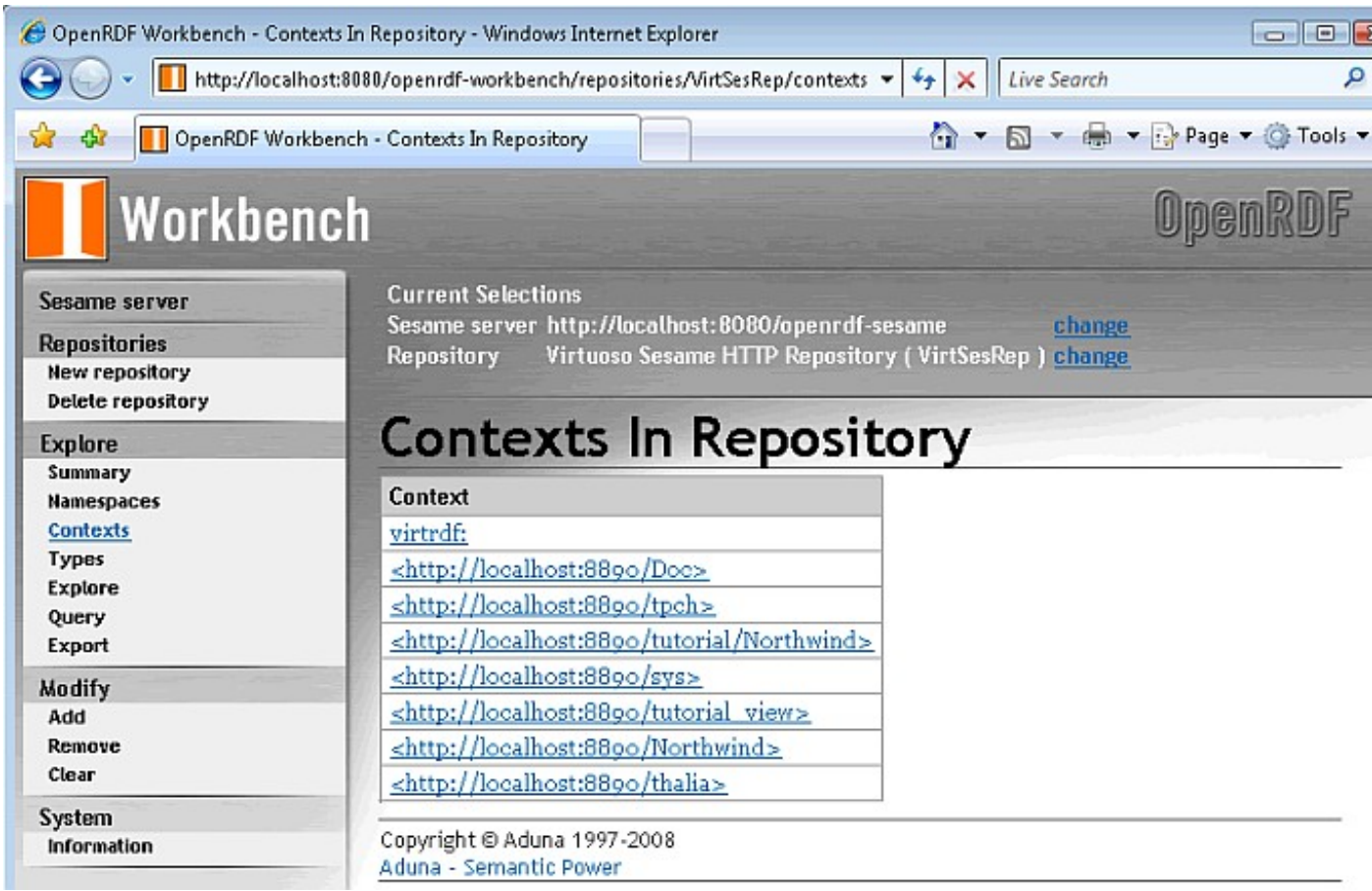
12. Click on the "Namespaces" link in the left frame to obtain a list of the available namespaces in the Virtuoso repository.

Figure 16.193. Virtuoso Sesame HTTP Repository Configuration and Usage



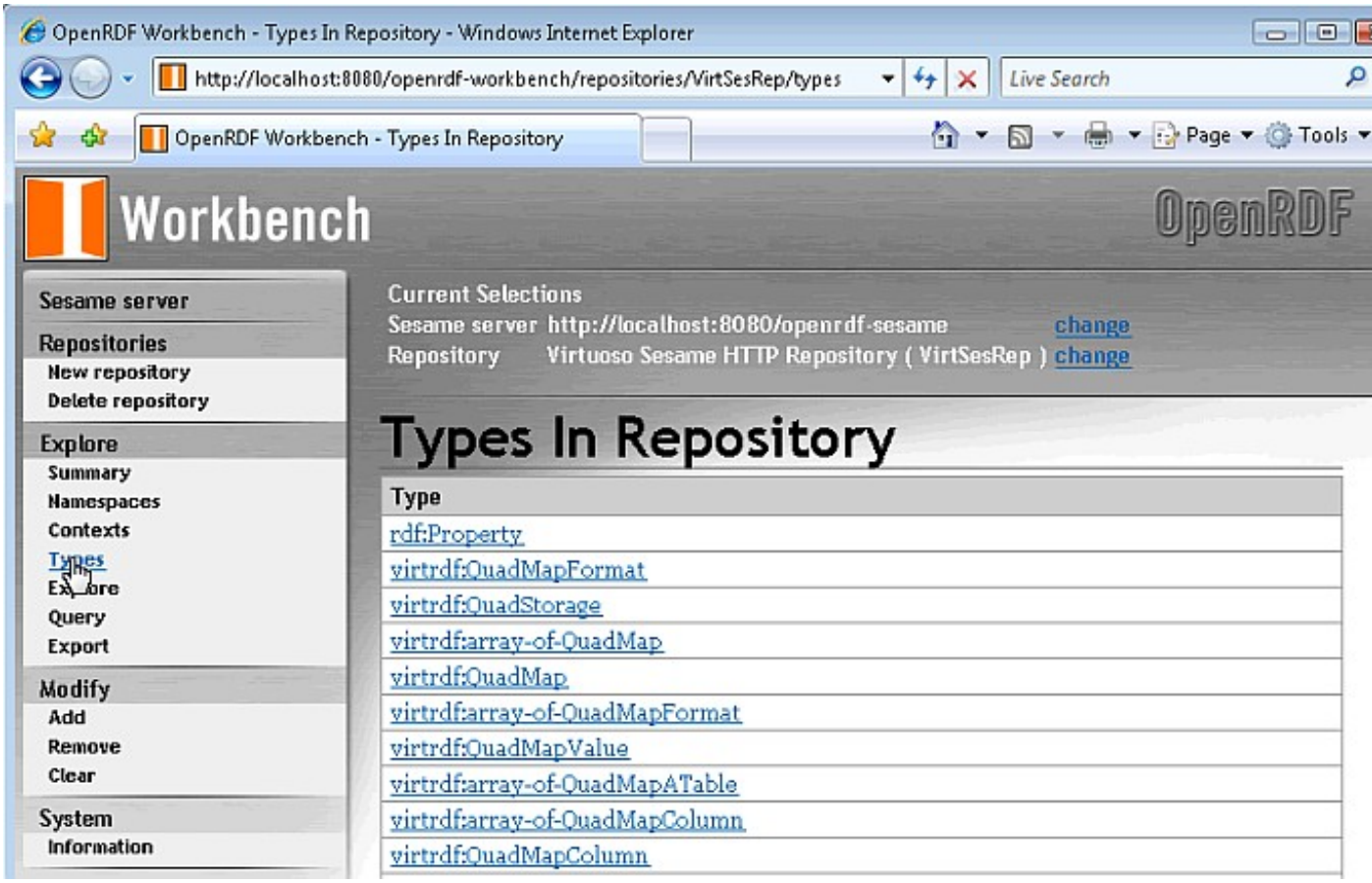
13. Click on the "Context" link in the left frame to obtain a list of the available contexts in the Virtuoso repository.

Figure 16.194. Virtuoso Sesame HTTP Repository Configuration and Usage



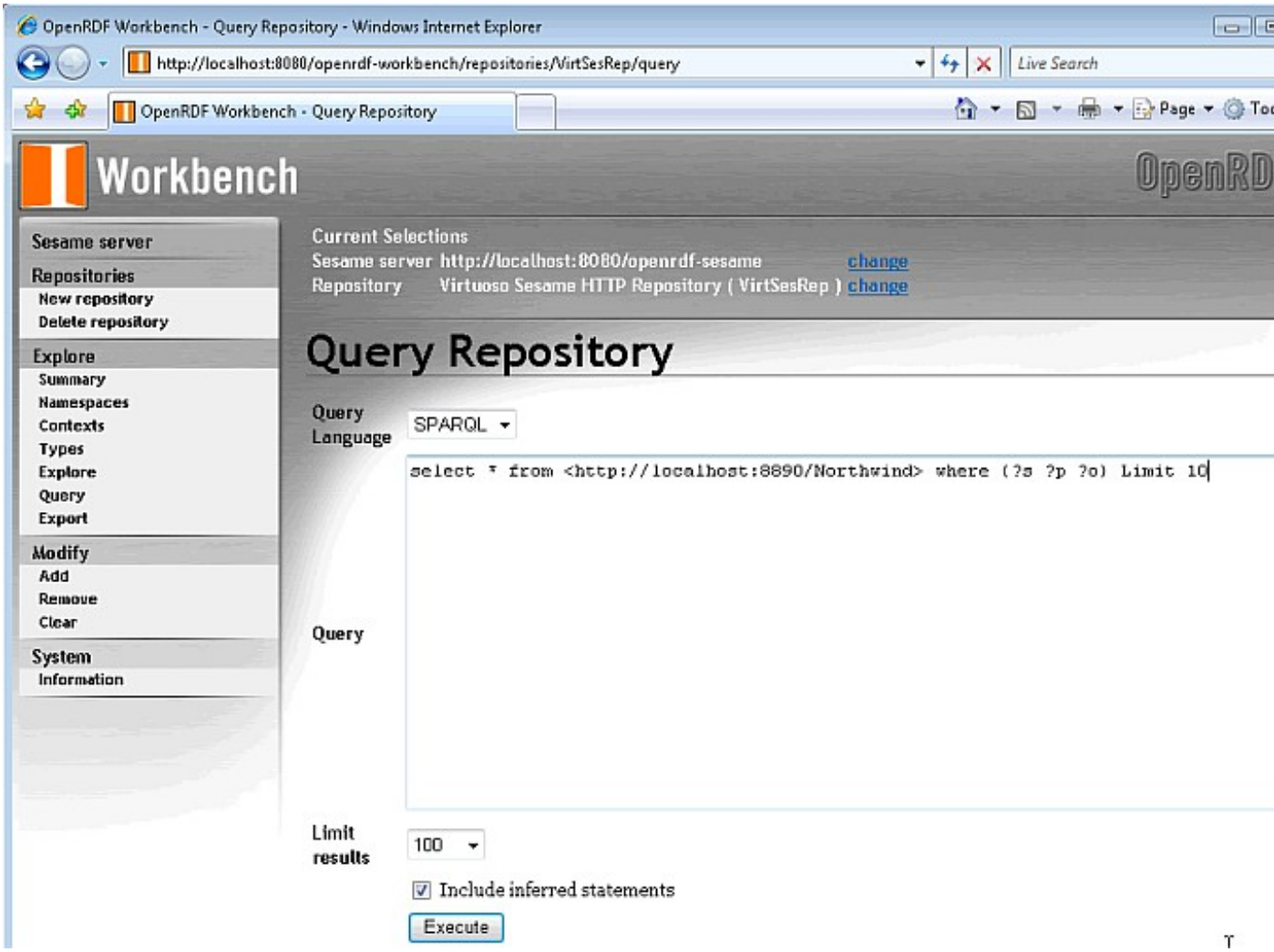
14. Click on the "Types" link in the left frame to obtain a list of the available types in the Virtuoso repository.

Figure 16.195. Virtuoso Sesame HTTP Repository Configuration and Usage



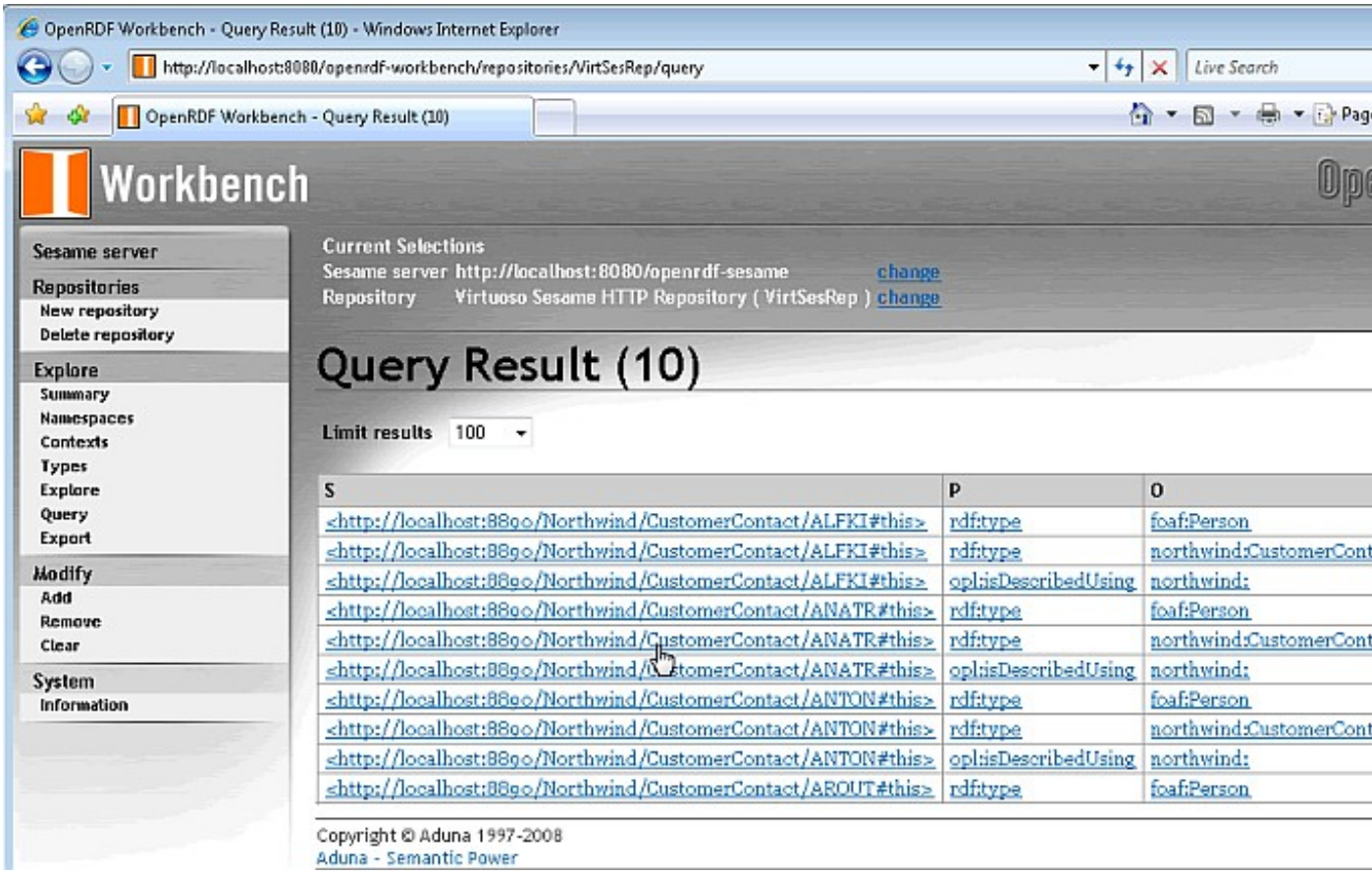
15. Click on the "Query" link in the left frame, enter a suitable SPARQL query to execute against the Virtuoso repository and click the "execute" button.

Figure 16.196. Virtuoso Sesame HTTP Repository Configuration and Usage



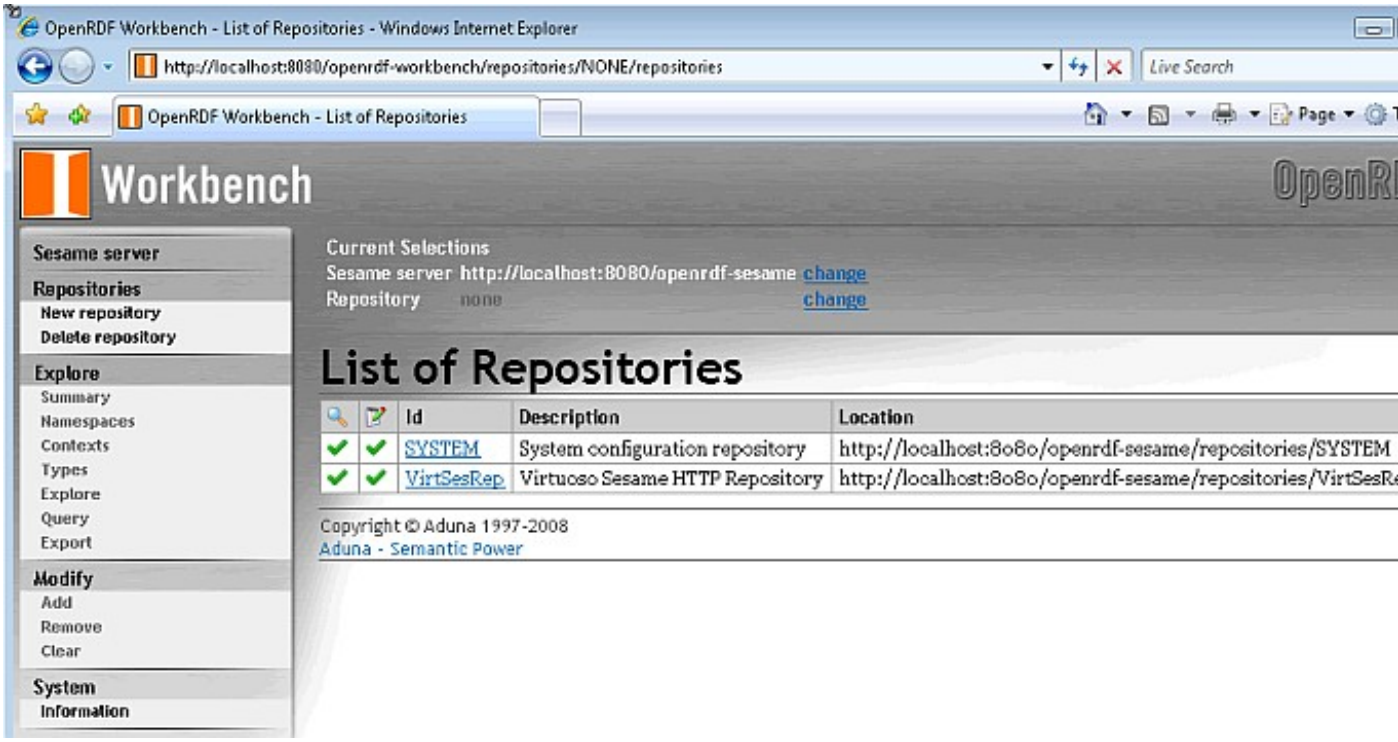
16. The results of the SPARQL query are returned.

Figure 16.197. Virtuoso Sesame HTTP Repository Configuration and Usage



17. Click on the "Repositories" link in the left frame and the newly created Virtuoso repository entry is displayed along side the default SYSTEM repository.

Figure 16.198. Virtuoso Sesame HTTP Repository Configuration and Usage



This section details the steps required for configuring and testing a Virtuoso Sesame Console Repository:

1. Extract Sesame 2.3.1 or higher archive to a location of choice and place the virt_sesame2.jar or virt_sesame4.jar and virtjdbc4.jar files to the sesame "lib" directory
2. Start the sesame console application by running the "console.bat" script in the sesame "bin" directory and then "exit." the program

```
$ sh console.sh
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/Users/myuser/openrdf-sesame-2.3.1/lib/logback-classic-0.9.18.ja
SLF4J: Found binding in [jar:file:/Users/myuser/openrdf-sesame-2.3.1/lib/slf4j-jdk14-1.5.10.jar!/o
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
10:32:38.317 [main] DEBUG info.aduna.platform.PlatformFactory - os.name <http://os.name> <http://o
10:32:38.351 [main] DEBUG info.aduna.platform.PlatformFactory - Detected Mac OS X platform
Connected to default data directory
Commands end with '.' at the end of a line
Type 'help.' for help
exit.
```

3. This will create the necessary sesame application data directories as detailed in the sesame data directory configuration documentation.

```
Windows - C:\Documents and Settings\LocalService\Application Data\Aduna\
Mac OS X - /Users/myuser/Library/Application Support/Aduna/
Linux - $HOME/.aduna/
```

4. If you do not want to use the default sesame data directory location the Sesame console application can be started by specifying a custom data directory location with the "-d" option. Note in this case the directory "OpenRDF Sesame console" always has to be manually appended to the directory as Sesame assumes the data file will reside in a sub directory of this name.

```
$ sh console.sh -d /Users/myuser/OpenRDF Sesame console
```

5. Add the virtuoso.ttl file to the ~/OpenRDF Sesame console/templates folder, to enable the Virtuoso repository default configuration parameters to be located.
6. Start the sesame console application with the required data directory location and create a Virtuoso repository as detailed in the steps below, the key parameters to be specified being the target Virtuoso server hostname, port number, username, password and a unique "Repository ID".

```
$ sh console.sh
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/Users/myuser/openrdf-sesame-2.3.1/lib/logback-classic-0.9.18.ja
SLF4J: Found binding in [jar:file:/Users/myuser/openrdf-sesame-2.3.1/lib/slf4j-jdk14-1.5.10.jar!/o
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
10:32:38.317 [main] DEBUG info.aduna.platform.PlatformFactory - os.name <http://os.name> <http://o
10:32:38.351 [main] DEBUG info.aduna.platform.PlatformFactory - Detected Mac OS X platform
Connected to default data directory
Commands end with '.' at the end of a line
Type 'help.' for help
create virtuoso .
Please specify values for the following variables:
Host list [localhost:1111]:
Username [dba]:
Password [dba]:
Default graph name [sesame:nil]:
Enable using batch optimization (false|true) [false]:
Use RoundRobin for connection (false|true) [false]:
Buffer fetch size [200]:
Inference RuleSet name [null]:
Repository ID [virtuoso]: myvirt
Repository title [Virtuoso repository]:
Repository created
show r .
+----------
|SYSTEM
|myvirt ("Virtuoso repository")
+----------
open myvirt .
Opened repository 'myvirt'
myvirt> show n .
```

```
+----------
|bif  bif:
|dawgt  http://www.w3.org/2001/sw/DataAccess/tests/test-dawg#
|dbpedia  http://dbpedia.org/resource/
|dbpprop  http://dbpedia.org/property/
|dc  http://purl.org/dc/elements/1.1/
|foaf  http://xmlns.com/foaf/0.1/
|geo  http://www.w3.org/2003/01/geo/wgs84_pos#
|go  http://purl.org/obo/owl/GO#
|math  http://www.w3.org/2000/10/swap/math#
|mesh  http://purl.org/commons/record/mesh/
|mf  http://www.w3.org/2001/sw/DataAccess/tests/test-manifest#
|nci  http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#
|obo  http://www.geneontology.org/formats/oboInOwl#
|owl  http://www.w3.org/2002/07/owl#
|protseq  http://purl.org/science/protein/bysequence/
|rdf  http://www.w3.org/1999/02/22-rdf-syntax-ns#
|rdfdf  http://www.openlinksw.com/virtrdf-data-formats#
|rdfs  http://www.w3.org/2000/01/rdf-schema#
|sc  http://purl.org/science/owl/sciencecommons/
|scovo  http://purl.org/NET/scovo#
|skos  http://www.w3.org/2004/02/skos/core#
|sql  sql:
|vcard  http://www.w3.org/2001/vcard-rdf/3.0#
|virtrdf  http://www.openlinksw.com/schemas/virtrdf#
|void  http://rdfs.org/ns/void#
|xf  http://www.w3.org/2004/07/xpath-functions
|xml  http://www.w3.org/XML/1998/namespace
|xsd  http://www.w3.org/2001/XMLSchema#
|xsl10  http://www.w3.org/XSL/Transform/1.0
|xsl1999  http://www.w3.org/1999/XSL/Transform
|xslwd  http://www.w3.org/TR/WD-xsl
|yago  http://dbpedia.org/class/yago/
+----------
exit.
```

Connection to Sesame HTTP repository from Console repository

The Sesame Console repository can connect to a Sesame HTTP repository and vice-versa, enabling access to remote Sesame HTTP repositories from a local server.

1. The Sesame Console repository can connect to a Sesame HTTP repository and query it as if local using the "connect" command.

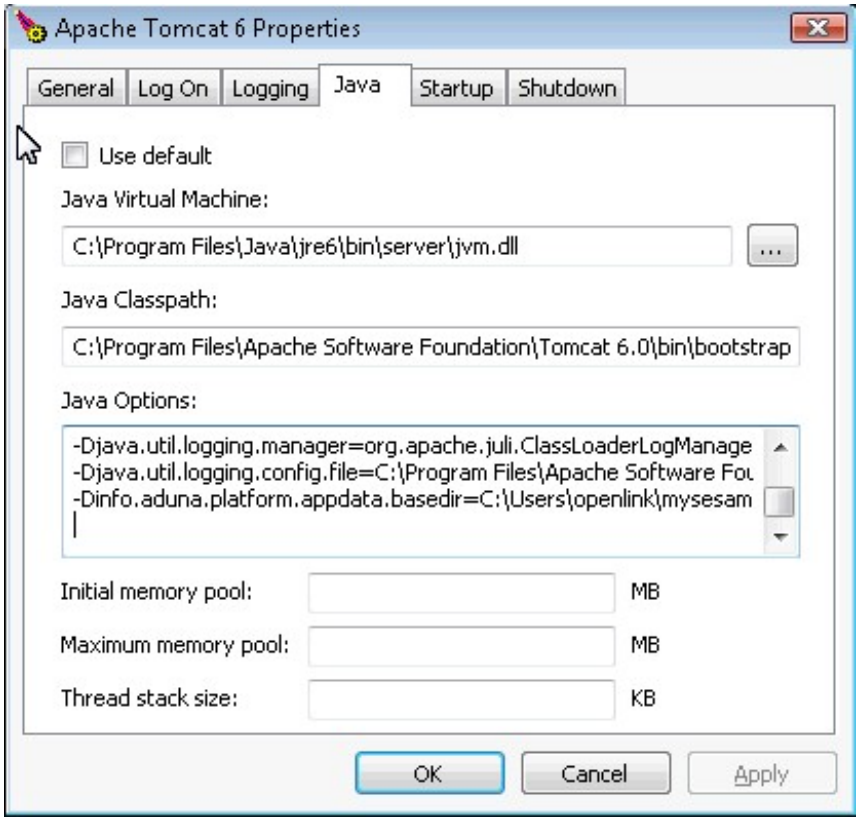
```
$ sh console.sh
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/Users/myuser/openrdf-sesame-2.3.1/lib/logback-classic-0.9.18.ja
SLF4J: Found binding in [jar:file:/Users/myuser/openrdf-sesame-2.3.1/lib/slf4j-jdk14-1.5.10.jar!/c
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
10:32:38.317 [main] DEBUG info.aduna.platform.PlatformFactory - os.name <http://os.name> <http://c
10:32:38.351 [main] DEBUG info.aduna.platform.PlatformFactory - Detected Mac OS X platform
Connected to default data directory
Commands end with '.' at the end of a line
Type 'help.' for help
> connect http://localhost:8080/openrdf-sesame.
Connected to http://localhost:8080/openrdf-sesame
> show r.
+----------
|SYSTEM ("System configuration repository")
|VirtSesRep ("Virtuoso Sesame HTTP Repository")
+----------
> open VirtSesRep.
Opened repository 'VirtSesRep'
VirtSesRep> sparql select * from <http://localhost:8890/Northwind> where {?s ?p ?o} Limit 10.
Evaluating query...
+------------------------+------------------------+------------------------+
| s                      | p                      | o                      |
+------------------------+------------------------+------------------------+
| <http://localhost:8890/Northwind/CustomerContact/ALFKI#this>| rdf:type  | foaf:Person
| <http://localhost:8890/Northwind/CustomerContact/ALFKI#this>| rdf:type  | northwind:CustomerCont
| <http://localhost:8890/Northwind/CustomerContact/ALFKI#this>| opl:isDescribedUsing   | northwind
| <http://localhost:8890/Northwind/CustomerContact/ANATR#this>| rdf:type | foaf:Person
```

```
| <http://localhost:8890/Northwind/CustomerContact/ANATR#this>| rdf:type | northwind:CustomerConta
| <http://localhost:8890/Northwind/CustomerContact/ANATR#this>| opl:isDescribedUsing   | northwind
| <http://localhost:8890/Northwind/CustomerContact/ANTON#this>| rdf:type  | foaf:Person
| <http://localhost:8890/Northwind/CustomerContact/ANTON#this>| rdf:type  | northwind:CustomerCont
| <http://localhost:8890/Northwind/CustomerContact/ANTON#this>| opl:isDescribedUsing   | northwind
| <http://localhost:8890/Northwind/CustomerContact/AROUT#this>| rdf:type  | foaf:Person
+-----------------------+-----------------------+-----------------------+
10 result(s) (530 ms)
VirtSesRep> show n.
+----------
|SearchResults  http://www.zillow.com/static/xsd/SearchResults.xsd
|UpdatedPropertyDetails  http://www.zillow.com/static/xsd/UpdatedPropertyDetails.xsd
|a  http://www.w3.org/2005/Atom
|aapi  http://rdf.alchemyapi.com/rdf/v1/s/aapi-schema#
|address  http://schemas.talis.com/2005/address/schema#
|admin  http://webns.net/mvcb/
|amz  http://webservices.amazon.com/AWSECommerceService/2005-10-05
|atom  http://atomowl.org/ontologies/atomrdf#
|audio  http://purl.org/media/audio#
|awol  http://bblfish.net/work/atom-owl/2006-06-06/#
|aws  http://soap.amazon.com/
|b3s  http://b3s.openlinksw.com/
```

2. Conversely the Sesame HTTP repository can be configured to access the repository created by the Sesame console. To do this the location of the data directory for both needs to be reconfigured using the Java system property info.aduna.platform.appdata.basedir (does not include "OpenRDF Sesame console directory) to point to the same location. When you are using Tomcat as the servlet container then you can set this property using the JAVA_OPTS parameter. Note, if you are using Apache Tomcat as a Windows Service you should use the Windows Services configuration tool to set this property. Other users can either edit the Tomcat startup script or set the property some other way.

```
* set JAVA_OPTS=-Dinfo.aduna.platform.appdata.basedir=\path\to\other\dir\ (on Windows)
* export JAVA_OPTS='-Dinfo.aduna.platform.appdata.basedir=/path/to/other/dir/' (on Linux/UNIX/Ma
```

Figure 16.199. Virtuoso Sesame HTTP Repository Configuration and Usage



Remote Sesame HTTP Repository Connections

There are two methods of making a remote Sesame HTTP connection to a Virtuoso repository:

- *Sesame Remote Repository Manager*

  class - A manager for Repositorys that reside on a remote server, allowing one to access repositories over HTTP similar to how local repositories are accessed using the LocalRepositoryManager?. A connection via this method is made via the Virtuoso Sesame provider which will be faster and provide better support for transactions, than making a direct connection to the SPARQL endpoint using the HTTPRepository class. This is analogous to the ODBC Driver Manager in the ODBC realm for generic access to a data store via a suitable ODBC Driver.
- *Sesame HTTP Repository*

  class - A repository that serves as a proxy for a remote repository on a Sesame server or directly to a SPARQL endpoint. This method does not make use of the Virtuoso Sesame Provider, using instead the HTTP protocol to make requests directly Sesame server or other SPARQL endpoint.

Using the Sesame Remote Repository Manager class

If a Sesame HTTP Repository endpoint, as detailed above already exists, the Remote Repository Manager class can be used to make a remote connection as follows by specify the URL to the remote Sesame HTTP Server and the RepositoryID for the repository the connection is to be made to:

```
//Initialize Remote Repository Manager
RepositoryManager repositoryManager = new RemoteRepositoryManager( "http://hostname:portno/openrdf-sesame
repositoryManager.initialize();

//Set Virtuoso (or any other) repositoryID on http://hostname:portno/openrdf-sesame
Repository repository = repositoryManager.getRepository("RepositoryID");

// Open a connection to this repository
con = repository.getConnection();

// ... do something
```

Using the Sesame HTTP Repository class

A direct connection to the default built-in Virtuoso SPARQL Endpoint can be made using the Sesame HTTP Repository class by simply specifying the URL to the Virtuoso SPARQL Endpoint i.e. http://{hostname}:{port}/sparql as follows:

```
// Initialize Direct SPARQL Endpoint HTTP Repository connection
String endpointURL = "http://hostname:portno/sparql";
HTTPRepository sparqlEndpoint = new HTTPRepository(endpointURL, "");
sparqlEndpoint.initialize();

// Open a connection to this repository
con = sparqlEndpoint.getConnection();

// ... do something
```

Javadoc API Documentation

Sesame Provider Javadoc API Documentation is available enabling the complete set of classes, interfaces and methods implemented for the provider to be viewed.

16.18.3. Virtuoso Redland Provider

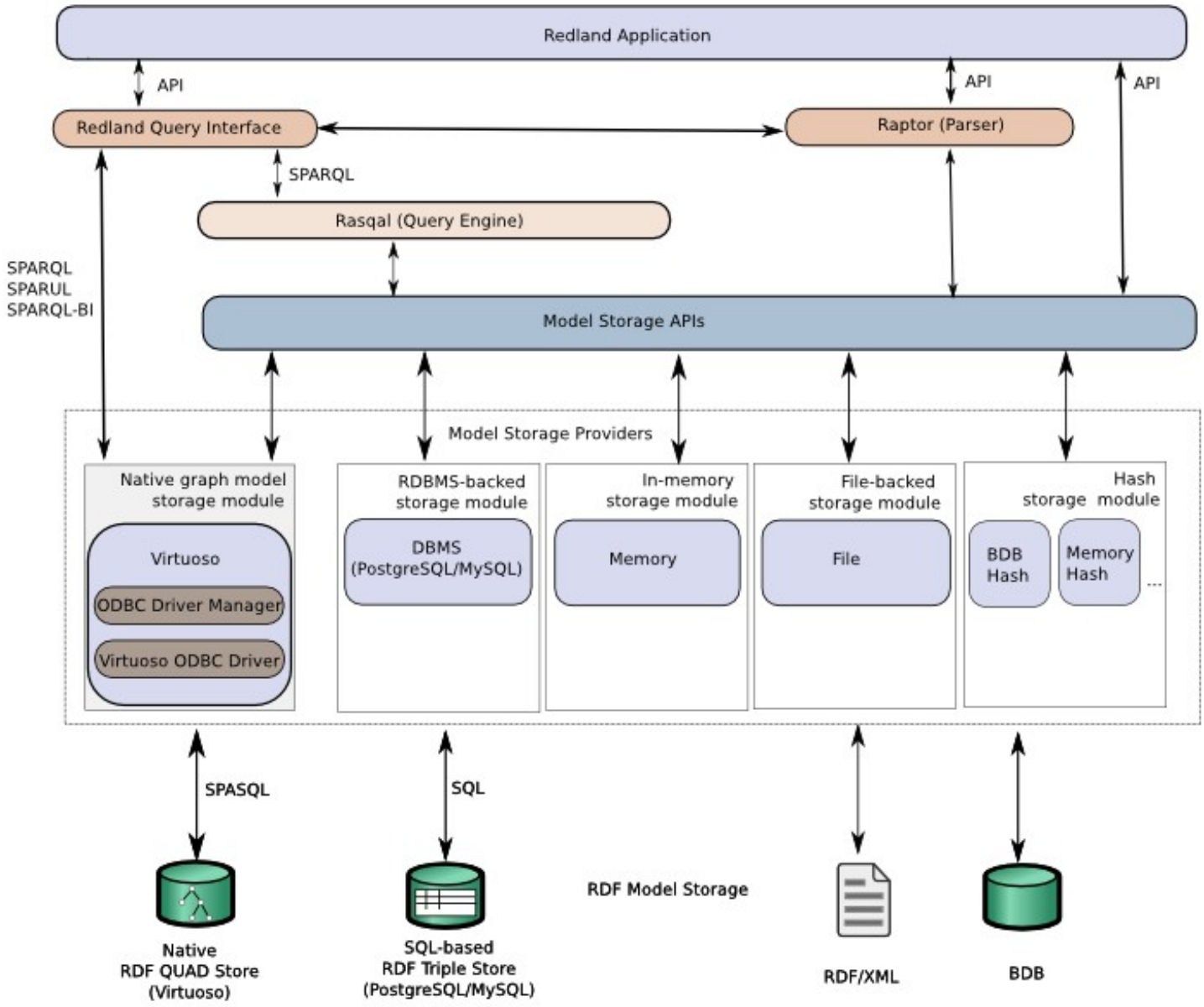
What is Redland

Redland is a set of free software 'C' libraries that provide support for the Resource Description Framework (RDF), providing modular, object based libraries and APIs for manipulating the RDF graph, triples, URIs and Literals. Redland includes several high-level language APIs providing RDF manipulation and storage and requires the Raptor RDF parser and Rasqal RDF syntax and query library for its use.

What is the Virtuoso Redland Provider

The Virtuoso Redland RDF Provider is an implementation of the Storage API, Model and Query interfaces of the Redland framework for RDF. This provider enables the execution of queries via the Redland Rasqal query engine or via Virtuoso query engine directly against the Virtuoso Quad store. The Virtuoso Redland Provider uses ODBC as the data access mechanism for communicating the Virtuoso Quad Store and requires the Virtuoso ODBC Driver be installed on the Redland client and a suitable ODBC DSN be configured for connecting to the target Virtuoso Quad Store instance. The provider has been tested against the Redland 1.0.8 version currently available for download.

Figure 16.200. Redland Component Stack



As indicated in the above diagram the Virtuoso Provider can be used to execute RDF queries either directly against the Virtuoso graph storage module supporting the SPARQL , SPARQL SPARUL, SPARQL-BI query languages or via the Rasqal query engine built into Redland which supports the SPARQL query language. This is done by simply changing the syntax of the query using the "vsparql" rather then default "sparql" construct when executing a query as indicated in the sample queries below:

```
rdfproc -r xml -t "user='dba',password='dba',dsn='Demo'" gr query sparql - "SELECT * WHERE { ?s ?p ?o }"

rdfproc -r xml -t "user='dba',password='dba',dsn='Demo'" gr query vsparql - "SELECT * WHERE { ?s ?p ?o }"
```

The Virtuoso Provider uses the SPASQL query language for querying the remote Virtuoso QUAD store.

Setup

Required Files

The Virtuoso Redland Provider has been integrated into the Redland RDF Framework and submitted to the open source project to become part of the standard distribution available for download . Until this submission has been accepted and committed into the available Redland release a tar ball created by OpenLink Software and a diff for application to a Redland 1.0.8 tree can be obtained from:

- ftp://download.openlinksw.com/support/vos/redland-vos-1.0.8.tar.gz Redland 1.0.8 tar ball with Virtuoso storage support
- Redland 1.0.8 Diff file of changes made for Virtuoso storage support

Compiling Redland with Virtuoso storage support

- Download Redland , extract and apply diff above or download the tar ball above with diff already applied and extract to a location of choice.
- The following additional configure options are available for enabling the Virtuoso storage support:

```
  --with-virtuoso(=yes|no) Enable Virtuoso RDF store (default=auto)
  --with-iodbc(=DIR)        Select iODBC support
                            DIR is the iODBC base install directory
                            (default=/usr/local)
  --with-unixodbc(=DIR)    Select UnixODBC support
                            DIR is the UnixODBC base install directory
                            (default=/usr/local)
  --with-datadirect(=DIR) Select DataDirect support
                            DIR is the DataDirect base install directory
                            (default=/usr/local)
  --with-odbc-inc=DIR      Specify custom ODBC include directory
                            (default=/usr/local/include)
  --with-odbc-lib=DIR      Specify custom ODBC lib directory
                            (default=/usr/local/lib)
```

- The "--with-virtuoso" option default to being auto enable if a valid ODBC Driver Manager (iODBC, UnixODBC? or DataDirect?) or include and lib directories for required ODBC header files and libraries are located with the suitable setting for one or more of the other ODBC related options above. Assuming iODBC is installed the following option can be used to enable Virtuoso storage support to be configured for compilation into your Redland build:

```
./configure --with-iodbc=/usr/local/iODBC
```

- Run "make" to compile the Redland libraries and "sudo make install" to install in the default "/usr/local" location
- Test compilation with test utility utils/rdfproc:

```
rdfproc test parse http://planetrdf.com/guide/rss.rdf
rdfproc test print
rdfproc test serialize ntriples
```

  This test will use the default 'hashes' storage.
- Ensure you have the Virtuoso ODBC Driver installed and a valid ODBC DSN called "Local Virtuoso" configured for your target Virtuoso Server
- Set the following environment variable:

```
export RDFPROC_STORAGE_TYPE=virtuoso                                    ;; Enable Virtuoso Storage
export ODBCINI=<path_to_odbcini_directory>/odbc.ini                       ;; Enable ODBC DSN to be
export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH                      ;; May be required to e
```

- Test Virtuoso storage with the provided test program utils/vtest:

```
$ utils/vtest
  1: Remove all triples in <http://red> context
**PASSED**: removed context triples from the graph
  2: Add triples to <http://red> context
**PASSED**: add triple to context
  3: Print all triples in <http://red> context
[[
  {[aa], [bb], [cc]} with context [http://red]
  {[aa], [bb1], [cc]} with context [http://red]
  {[aa], [a2], "cc"} with context [http://red]
```

```
  {[aa], [a2], (cc)} with context [http://red]
  {[mm], [nn], "Some long literal with language@en"} with context [http://red]
  {[oo], [pp], "12345^^<http://www.w3.org/2001/XMLSchema#int>"} with context [http://red]
]]
**PASSED**:
  4: Count of triples in <http://red> context
**PASSED**: graph has 6 triples
  5: Exec:  ARC  aa bb
Matched node: [cc]
**PASSED**:
  6: Exec:  ARCS  aa cc
Matched node: [bb] with context [http://red]
Matched node: [bb1] with context [http://red]
: matching nodes: 2
**PASSED**:
  7: Exec:  ARCS-IN  cc
Matched arc: [bb] with context [http://red]
Matched arc: [bb1] with context [http://red]
**PASSED**: matching arcs: 2
  8: Exec:  ARCS-OUT  aa
Matched arc: [bb] with context [http://red]
Matched arc: [bb1] with context [http://red]
Matched arc: [a2] with context [http://red]
Matched arc: [a2] with context [http://red]
**PASSED**: matching arcs: 4
  9: Exec:  CONTAINS aa bb1 cc
**PASSED**: the graph contains the triple
 10: Exec:  FIND aa - -
Matched triple: {[aa], [bb], [cc]} with context [http://red]
Matched triple: {[aa], [bb1], [cc]} with context [http://red]
Matched triple: {[aa], [a2], "cc"} with context [http://red]
Matched triple: {[aa], [a2], (cc)} with context [http://red]
**PASSED**: matching triples: 4
 11: Exec:  HAS-ARC-IN cc bb
**PASSED**: the graph contains the arc
 12: Exec:  HAS-ARC-OUT aa bb
**PASSED**: the graph contains the arc
 13: Exec:  SOURCE  aa cc
Matched node: [aa]
**PASSED**:
 14: Exec:  SOURCES  bb cc
Matched node: [aa] with context [http://red]
: matching nodes: 1
**PASSED**:
 15: Exec:  TARGET  aa bb
Matched node: [cc]
**PASSED**:
 16: Exec:  TARGETS  aa bb
Matched node: [cc] with context [http://red]
: matching nodes: 1
**PASSED**:
 17: Exec:  REMOVE aa bb1 cc
**PASSED**: removed triple from the graph
 18: Exec:  QUERY "CONSTRUCT {?s ?p ?o} FROM <http://red> WHERE {?s ?p ?o}"
Matched triple: {[aa], [a2], "cc"}
Matched triple: {[oo], [pp], "12345^^<http://www.w3.org/2001/XMLSchema#int>"}
Matched triple: {[aa], [a2], (cc)}
Matched triple: {[aa], [bb], [cc]}
Matched triple: {[mm], [nn], "Some long literal with language@en"}
**PASSED**: matching triples: 5
 19: Exec1:  QUERY_AS_BINDINGS "SELECT * WHERE {graph <http://red> { ?s ?p ?o }}"
**: Formatting query result as 'xml':
<?xml version="1.0" encoding="utf-8"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head>
    <variable name="s"/>
    <variable name="p"/>
    <variable name="o"/>
  </head>
  <results>
    <result>
      <binding name="s"><uri>aa</uri></binding>
      <binding name="p"><uri>bb</uri></binding>
```

```
      <binding name="o"><uri>cc</uri></binding>
    </result>
    <result>
      <binding name="s"><uri>aa</uri></binding>
      <binding name="p"><uri>a2</uri></binding>
      <binding name="o"><literal>cc</literal></binding>
    </result>
    <result>
      <binding name="s"><uri>aa</uri></binding>
      <binding name="p"><uri>a2</uri></binding>
      <binding name="o"><bnode>cc</bnode></binding>
    </result>
    <result>
      <binding name="s"><uri>mm</uri></binding>
      <binding name="p"><uri>nn</uri></binding>
      <binding name="o"><literal>Some long literal with language@en</literal></binding>
    </result>
    <result>
      <binding name="s"><uri>oo</uri></binding>
      <binding name="p"><uri>pp</uri></binding>
      <binding name="o"><literal>12345^^<http://www.w3.org/2001/XMLSchema#int></literal></binding>
    </result>
  </results>
</sparql>
**PASSED**:
 20: Exec2:  QUERY_AS_BINDINGS "SELECT * WHERE {graph <http://red> { ?s ?p ?o }}"
: Query returned bindings results:
result: [s=[aa], p=[bb], o=[cc]]
result: [s=[aa], p=[a2], o=cc]
result: [s=[aa], p=[a2], o=(cc)]
result: [s=[mm], p=[nn], o=Some long literal with language@en]
result: [s=[oo], p=[pp], o=12345^^<http://www.w3.org/2001/XMLSchema#int>]
: Query returned 5 results
**PASSED**:
==============================================
PASSED: 20  FAILED:  0
```

Connection Parameters

The Virtuoso provider has the following connection parameters available fro use:

- *dsn*

  - ODBC datasource name
- *user*

  - user name of database server
- *password*

  - password of database server
- *host*

  - hostname:portno of the database server
- *charset*

  - database charset to use

NOTE: Take care exposing the password as for example, program arguments or environment variables. The rdfproc utility can help this by reading the password from standard input. Inside programs, one way to prevent storing the password in a string is to construct a Redland hash of the storage options such as via librdf hash_from_string and use librdf_new_storage_with_options to create a storage. The rdfproc utility source code demonstrates this.

The storage name parameter given to the storage constructor librdf new_storage is used inside the virtuoso store to allow multiple stores inside one Virtuoso database instance as parameterized with the above options.

This store always provides contexts; the boolean storage option contexts is not checked.

Examples:

```
/* A new Virtuoso store */
storage=librdf_new_storage(world, "virtuoso", "db1",
    "dsn='Local Virtuoso',user='demo',password='demo'");

/* A different, existing Virtuoso store in the same database as above */
storage=librdf_new_storage(world, "virtuoso", "db2",
    "dsn='Local Virtuoso',user='demo',password='demo'");

/* An existing Virtuoso store on a different database server */
storage=librdf_new_storage(world, "virtuoso", "http://red3",
    "dsn='Remote Virtuoso',user='demo',password='demo'");

/* Opening with an options hash */
options=librdf_new_hash(world, NULL);
librdf_hash_from_string(options,
    "dsn='Local Virtuoso',user='demo'");
librdf_hash_put_strings(options, "password", user_password);
storage=librdf_new_storage_with_options(world, "virtuoso", "http://red3", options);
```

References

- RedLand Triple Store
- RedLand Storage Modules

16.19. RDF Graph Replication

The following section demonstrates how to replicate graphs from one Virtuoso instance to (an)other Virtuoso instance(s), using the RDF Replication Feature.

Terms used in this section:

- *Host Virtuoso Instance*

  , aka the publisher: the instance where we will insert RDF data into a Named Graph; then create a publication of this graph.
- *Destination Virtuoso Instance*

  , aka the subscriber: the instance which will subscribe to the publication from the Host Virtuoso Instance.

See Also:

`DB.DBA.RDF_REPL_START()`

`DB.DBA.RDF_REPL_GRAPH_INS()`

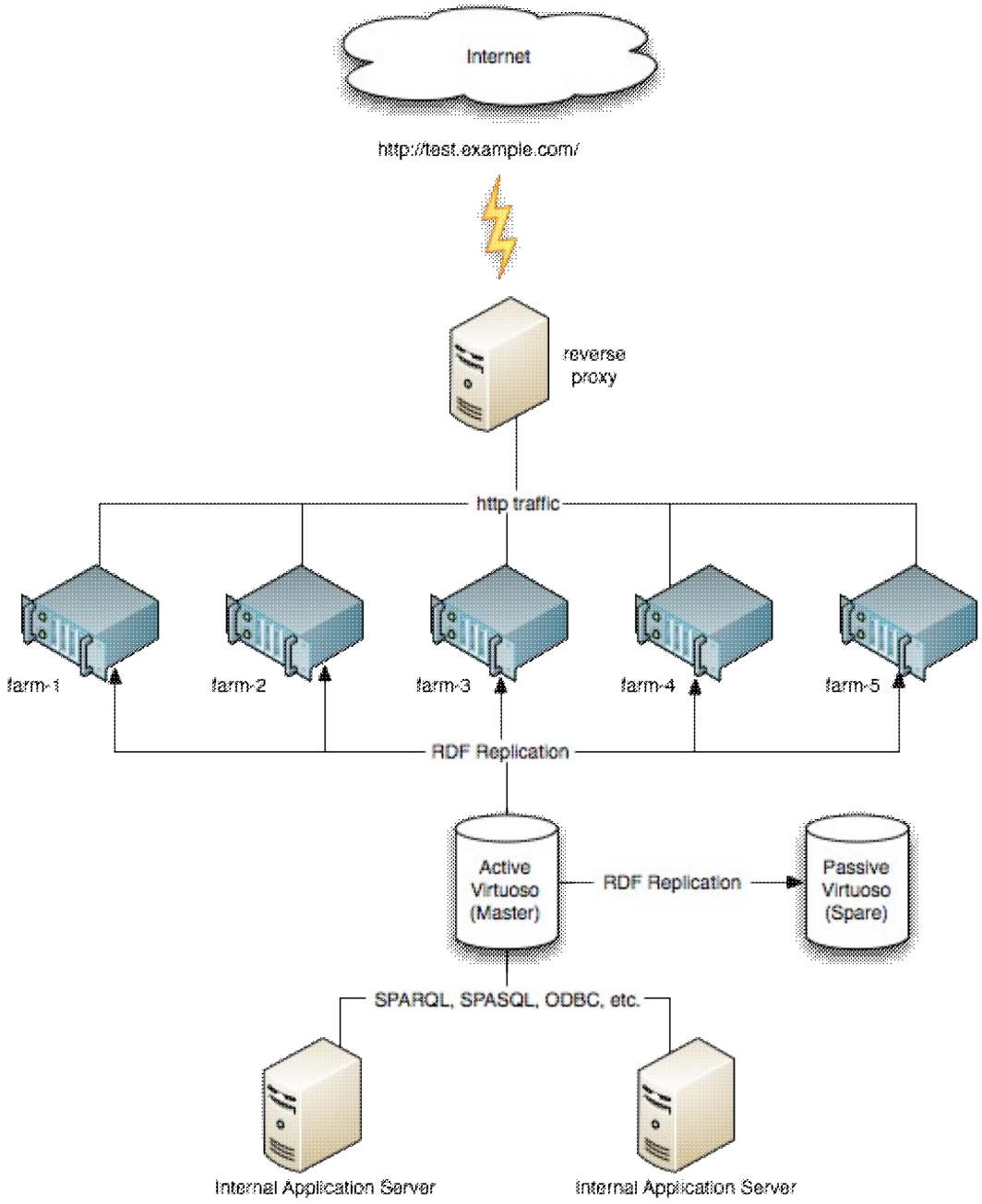
`DB.DBA.RDF_RDF_REPL_GRAPH_DEL()`

The basic outline:

- First, use the Virtuoso Conductor on a Host Virtuoso Instance to publish a named graph.
- Then, use the Virtuoso Conductor on a Destination Virtuoso Instance to subscribe to deltas from the published graph.
- Finally, see how a change in the publisher's graph will appear in the subscriber's graph.

16.19.1. Replication Scenarios

Figure 16.201.



Introduction

In this section we will examine a proposed setup for a back-end server called MASTER which publishes a number of graphs to a set of front-end machines called FARM-1 .. FARM-n and discuss a couple of common scenarios like adding an extra machine to the farm, or replacing a broken instance of MASTER.

In this example we will assume each virtuoso instance running on its own machine, so they can use the same port numbers for both the main server (default 1111) as well as the http port (default 8890) as each machine has an unique IP addresses. In the example we use MASTER-IP and FARM-x-IP which should be replaced by either the real IP address or the DNS name of the machine in question.

Since there will be a reverse-proxy service in front of the farm, all virtuoso instances should have the URIQA Default host set to the outside name for this service. In this example we will use http://test.example.com as the web service we are trying to setup.

Setup

Installing Virtuoso

All machines in this setup should be installed with similar installation paths like:

- /opt/virtuoso
- /dbs/virtuoso
- /virtuoso
- ...

The partition should be big enough to have room for the Virtuoso binaries and libraries, the transaction logs, backups and, if you do not want to use the striping feature of Virtuoso, it will need to have room for the main database files as well.

Here are the quick installation steps:

1. Login as root.
2. Create local user called virtuoso using the chosen installation path as home direcotory.
3. Login as virtuoso.
4. Extract virtuoso-universal-server-6.1.tar in home directory.
5. Run sh install.sh to install Virtuoso.
6. Remove the file install.sh virtuoso-universal-server-6.1.tar virtuoso-server.taz if not otherwise needed.
7. Run bin/virtuoso-stop.sh to shutdown this Virtuoso instance.
8. Install virtuoso.lic for this system in $HOME/bin directory.

As the replication process needs to make an ODBC connection to the MASTER machine, all machines should have the following information in the $HOME/bin/odbc.ini:

```
[ODBC Data Sources]
..
MASTER_DSN = OpenLink Virtuoso

..
[MASTER_DSN]
Driver = OpenLink Virtuoso
Address = MASTER_IP:1111
```

Setting up MASTER

The MASTER machine is the back-end server machine. Various applications feed SPARQL data into this machine it publishes a set of graphs using RDF Replication.

The MASTER machine should ideally be equipped with multiple redundant disks in RAID-1 or RAID-6 mode to minimize the risk that a single bad disk takes down the system. From a Virtuoso point of view we will use a combination of online backups combined with checkpoint audit trail to backup the content of the database in a safe way. The online backups, the checkpoint audit trail as well as the replication logs can also be copied to secondary storage using the rsync command and can be easily scripted as a cron job.

Changes to database/virtuoso.ini:

```
...
[Parameters]
SchedulerInterval    = 1     ; run the internal scheduler every minute
CheckpointAuditTrail = 1     ; enable audit trail on transaction logs
CheckpointInterval   = 60    ; perform an automated checkpoint every 60 minutes
...
[URIQA]
DefaultHost = test.example.com
...
[Replication]
ServerName   = MASTER
ServerEnable = 1
QueueMax     = 5000000
...
```

Once the MASTER is started using the bin/virtuoso-start.sh script we must enable RDF replication before we start add data to the graphs we wish to replicate, so every record is accounted for by the replication process. If there is existing data in the graphs to be

published, then this data would need to be added to a subscriber manually since the replication process creates a delta set of changes since publishing was enabled.

To enable publishing of the graph we use the isql program to connect to the MASTER instance:

```
$ isql MASTER-IP:1111

-- and run the following commands:

-- enable this instance as a publisher
rdf_repl_start();

-- add graphs to replication list
rdf_repl_graph_ins('http://test.example.com');
```

Next we create a backup directory inside the database directory and setup the online backup, again using the isql program:

```
$ cd database
$ mkdir backup
$ isql MASTER_IP:1111

-- and run the following commands:

-- clear any previous context
backup_context_clear();

-- start the backup
backup_online ('bkup-#', 1000000, 0, vector ('backup'));
```

The following files can now be backed up using rsync or similar tool to another machine:

Table 16.22. Files that can be backed up using rsync or similar tool to another machine

Files	Description
database/backup/*.bp	the incremental backup files
database/virtuoso.trx	the main transaction log containing the most recent updates to the database that have not been checkpointed into the database
database/virtuosoTIMESTAMP.trx	all the previous transaction logs which can be used to reconstruct the database
database/__rdf_repl*.log	all the replication logs containing the changes to the published graph

NOTE: Since the database is constantly modified during operation, it is of NO use to backup the virtuoso.db using an rsync script unless the virtuoso instance was shutdown beforehand, or certain extra precautions are taken which we will explain later on.

Setup SPARE master

The SPARE machine is a replica of the MASTER machine. This machine subscribes to the publication of the MASTER to keep an exact match of the RDF graphs, but also publishes this data without any initial subscribers.

The SPARE machine should ideally be equipped similar to the MASTER machine, with multiple redundant disks in RAID-1 or RAID-6 mode to minimize the risk that a single bad disk takes down the system. From a Virtuoso point of view we will use a combination of online backups combined with checkpoint audit trail to backup the content of the database in a safe way. The online backups, the checkpoint audit trail as well as the replication logs can also be copied to secondary storage using the rsync command and can be easily scripted as a cron job.

Changes to database/virtuoso.ini:

```
...
[Parameters]
SchedulerInterval    = 1     ; run the internal scheduler every minute
CheckpointAuditTrail = 1     ; enable audit trail on transaction logs
CheckpointInterval   = 60    ; perform an automated checkpoint every 60 minutes
...
[URIQA]
```

```
DefaultHost = test.example.com
...
[Replication]
ServerName   = SPARE
ServerEnable = 1
QueueMax     = 5000000
...
```

We must enable RDF replication before we start add data to the graphs we wish to replicate, so every record is accounted for by the replication process. If there is existing data in the graphs to be published, then this data would need to be added to a subscriber manually since the replication process creates a delta set of changes since publishing was enabled.

To enable publishing of the graph, as well as subscribing to the MASTER, we first start up this Virtuoso instance with bin/virtuoso-start.sh and then use the isql program to connect to the SPARE instance:

```
$ bin/virtuoso-start.sh
$ isql SPARE-IP:1111

-- and run the following commands:

-- enable this instance as a publisher
rdf_repl_start();

-- add graphs to replication list
rdf_repl_graph_ins('http://test.example.com');

-- connect to master
repl_server ('MASTER', 'MASTER_DSN');

-- start subscribing to __rdf_repl
repl_subscribe ('MASTER', '__rdf_repl', 'dav', 'dav', 'dba', 'dba');

-- start initial replication
repl_sync_all ();

-- add subscription to scheduler
DB.DBA.SUB_SCHEDULE ('MASTER', '__rdf_repl', 1);
```

Next we create a backup directory inside the database directory and setup the online backup, again using the isql program:

```
$ cd database
$ mkdir backup
$ isql SPARE_IP:1111

--and run the following commands:

-- clear any previous context
backup_context_clear();

-- start the backup
backup_online ('bkup-#', 1000000, 0, vector ('backup'));
```

The following files can now be backed up using rsync or similar tool to another machine:

Table 16.23. Files that can be backed up using rsync or similar tool to another machine

Files	Description
database/backup/*.bp	the incremental backup files
database/virtuoso.trx	the main transaction log containing the most recent updates to the database that have not been checkpointed into the database
database/virtuosoTIMESTAMP.trx	all the previous transaction logs which can be used to reconstruct the database
database/__rdf_repl*.log	all the replication logs containing the changes to the published graph

Note: Since the database is constantly modified during operation, it is of NO use to backup the virtuoso.db using an rsync script unless the virtuoso instance was shutdown beforehand, or certain extra precautions are taken which we will explain later on.

Setup FARM-1

The FARM-1 machine is the first front-end server machine. It subscribes to the publication of the MASTER instance to keep up-to-date.

The FARM-1 machine can be run on simpler hardware than the MASTER instance.It does not require the same level of redundancy in terms of hard disks etc, as there are a number of these machines running in parallel each capable of returning results to the proxy. If one FARM machine dies, it can simply be taken from the reverse-proxy list, repaired or replaced with a fresh machine before it is added to the list of servers in the reverse proxy. As such it does not need to be backed up separately, although we could make a backup of this installation to quickly install the rest of the identical FARM boxes.

Change the database/virtuoso.ini file:

```
...
[Parameters]
SchedulerInterval    = 1     ; run the internal scheduler every minute
CheckpointAuditTrail = 0     ; disable audit trail on transaction logs
CheckpointInterval   = 60    ; perform an automated checkpoint every 60 minutes
...
[URIQA]
DefaultHost = test.example.com
...
[Replication]
ServerName   = FARM-1        ; each FARM machine needs to have a unique replication name
ServerEnable = 1
QueueMax     = 5000000
...
```

Next we start up the Virtuoso instance using the bin/virtuoso-start.sh command and use the isql program to subscribe to the MASTER:

```
$ bin/virtuoso-start.sh
$ isql FARM-1-IP:1111

-- connect to master
repl_server ('MASTER', 'MASTER_DSN');

-- start subscribing to __rdf_repl
repl_subscribe ('MASTER', '__rdf_repl', 'dav', 'dav', 'dba', 'dba');

-- start initial replication
repl_sync_all ();

-- add subscription to scheduler
DB.DBA.SUB_SCHEDULE ('MASTER', '__rdf_repl', 1);
```

At this point we can shutdown this Virtuoso instance using the bin/virtuoso-stop.sh command and make a copy of the whole virtuoso installation as a blueprint to copy to another FARM-x machine.

Setup FARM-2 from scratch

We can repeat the same steps we did for the FARM-1 machine, and just make sure we use FARM-2 as the replication name in the database/virtuoso.ini file and use FARM-2-IP:1111 as an argument to the isql program.

Change bin/virtuoso.ini:

```
[Replication]
ServerName = FARM-2
```

Setup FARM-3 using blueprint from FARM-1 installation

Extract the tarred/zipped copy of the installation made at the end of the setup of FARM-1.

Before starting up the instance, we only need to give this instance a unique name for replication:

Change bin/virtuoso.ini:

```
[Replication]
ServerName = FARM-3
```

Next we start up the Virtuoso instance using the bin/virtuoso-start.sh command and since the subscription records and schedule are already performed in the previous step, we just use the isql program to perform a sync against the MASTER:

```
$ bin/virtuoso-start.sh
$ isql FARM-3-IP:1111

-- change replication name
DB.DBA.REPL_SERVER_RENAME ('FARM-1', 'FARM-3')

-- sync against master
repl_sync_all();
```

Setup FARM-4 using clone of FARM-1

If the system has been running for some time, it may not be practical to do a replication from start, so there is an alternative way to setup a new FARM-4 machine.

We can either restore the blue-print backup we make at the end of FARM-1 installation, or we do a fresh installation of virtuoso on the FARM-4 machine.

In both cases we shutdown the virtuoso instance and remove the database, as we are going to replace this.

```
$ bin/virtuoso-stop.sh
$ cd database
$ rm virtuoso.db virtuoso.trx virtuoso.log virtuoso.pxa
```

Change the database/virtuoso.ini file:

```
...
[Parameters]
SchedulerInterval    = 1     ; run the internal scheduler every minute
CheckpointAuditTrail = 0     ; disable audit trail on transaction logs
CheckpointInterval   = 60    ; perform an automated checkpoint every 60 minutes
...
[URIQA]
DefaultHost = test.example.com
...
[Replication]
ServerName   = FARM-4        ; each FARM machine needs to have a unique replication name
ServerEnable = 1
QueueMax     = 5000000
...
```

Next we are going to temporarily disable checkpointing on FARM-1 machine so we can copy its database without risking corruption:

```
$ isql FARM-1-IP:1111

-- disable automatic checkpointing
checkpoint_interval (-1);

-- and do an explicit checkpoint
checkpoint;
```

It is now safe to copy the database across using the rsync command:

```
$ rsync -avz virtuoso@FARM-1-IP:/path/to/virtuoso/database/virtuoso.db database/virtuoso.db
```

Next we re-enable checkpoint interval on FARM-1:

```
$ isql FARM-1-IP:1111

-- re-enable checkpointing
checkpoint_interval(60);
```

The last step is to start the database:

```
$ bin/virtuoso-start.sh
$ isql FARM-4-IP:1111

-- change replication name
DB.DBA.REPL_SERVER_RENAME ('FARM-1', 'FARM-4')

-- sync against master
repl_sync_all();
```

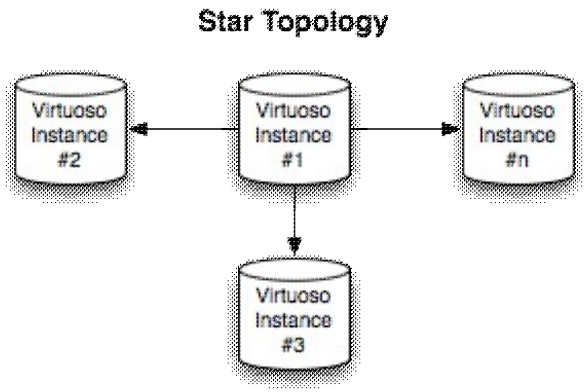
16.19.2. Replication Topologies

Typical replication topologies are Chains, Stars and Bi-directional. They can be achieved with Virtuoso, by repeating the "Publish" and/or "Subscribe" steps on each relevant node.

Star Replication Topology

In a Star, there is one Publisher, and many Subscribers.

Figure 16.202. Star Replication Topology



To set up a Star, follow the scenario:

1. Configure Instance #1 to Publish.
2. Configure Instance #2 to Subscribe to #1.
3. Repeat as necessary.

Star Replication Topology Example

The following How-To walks you through setting up Virtuoso RDF Graph Replication in a Star Topology.

Prerequisites

Database INI Parameters

Suppose there are 3 Virtuoso instances respectively with the following ini parameters values:

1. virtuoso1.ini:

```
...
[Database]
DatabaseFile    = virtuoso1.db
TransactionFile = virtuoso1.trx
ErrorLogFile     = virtuoso1.log
...
[Parameters]
ServerPort              = 1111
SchedulerInterval       = 1
...
```

```
[HTTPServer]
ServerPort                  = 8891
...
[URIQA]
DefaultHost = example.com:8891
...
[Replication]
ServerName   = db1
...
```

2. virtuoso2.ini:

```
...
[Database]
DatabaseFile    = virtuoso2.db
TransactionFile = virtuoso2.trx
ErrorLogFile     = virtuoso2.log
...
[Parameters]
ServerPort               = 1112
SchedulerInterval        = 1
...
[HTTPServer]
ServerPort                  = 8892
...
[URIQA]
DefaultHost = localhost:8892
...
[Replication]
ServerName   = db2
...
```

3. virtuoso3.ini:

```
...
[Database]
DatabaseFile    = virtuoso3.db
TransactionFile = virtuoso3.trx
ErrorLogFile     = virtuoso3.log
...
[Parameters]
ServerPort               = 1113
SchedulerInterval        = 1
...
[HTTPServer]
ServerPort                  = 8893
...
[URIQA]
DefaultHost = example.com:8893
...
[Replication]
ServerName   = db3
...
```

Database DSNs

Use the ODBC Administrator on your Virtuoso host (e.g., on Windows, Start menu -> Control Panel -> Administrative Tools -> Data Sources (ODBC); on Mac OS X, /Applications/Utilities/OpenLink ODBC Administrator.app) to create a System DSN for each of db1, db2, db3, with names db1, db2 and db3, respectively.

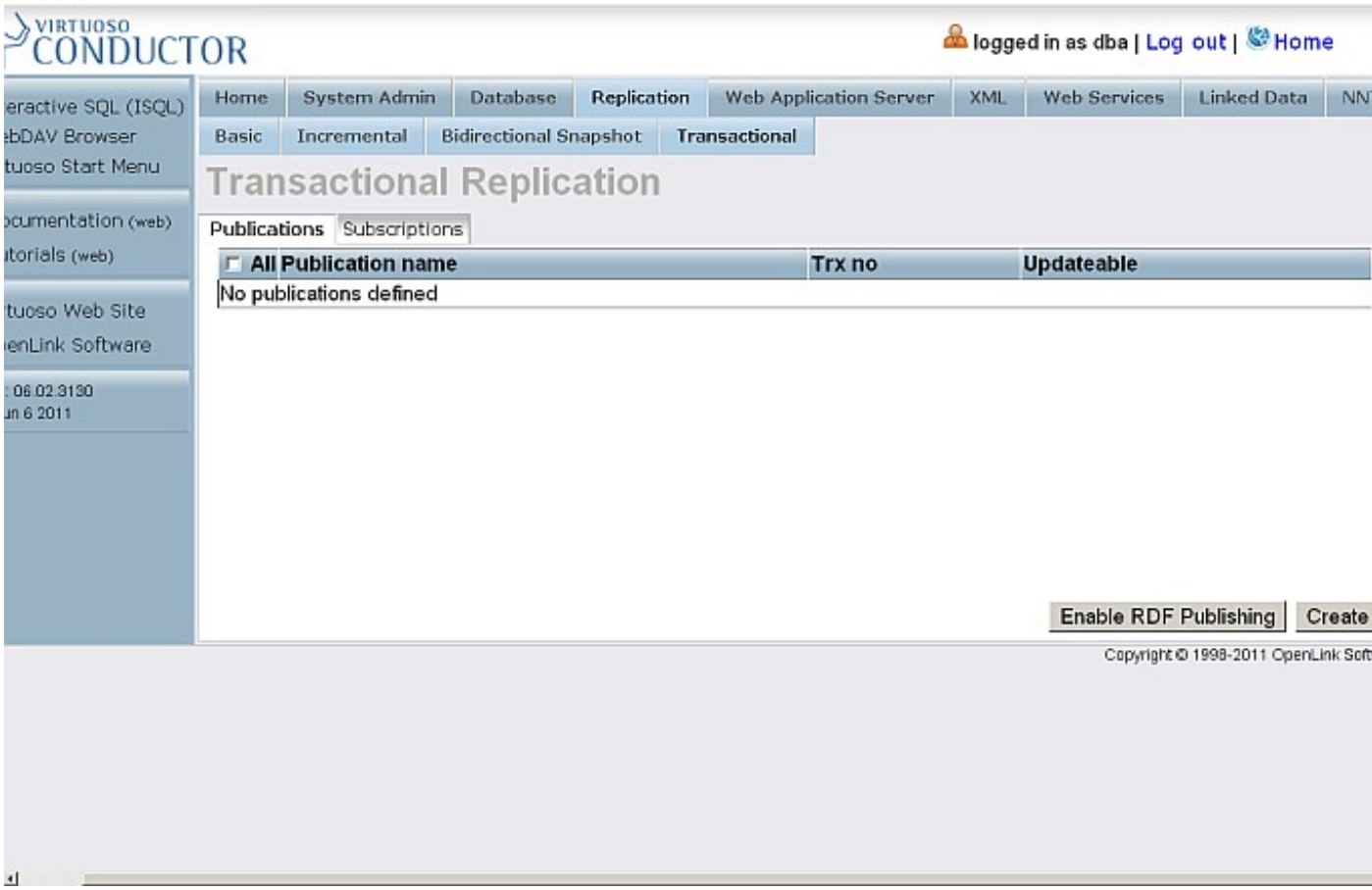
Install Conductor package

On each of the 3 Virtuoso instances install the conductor_dav.vad package.

Create a Publication on the Host Virtuoso Instance db1

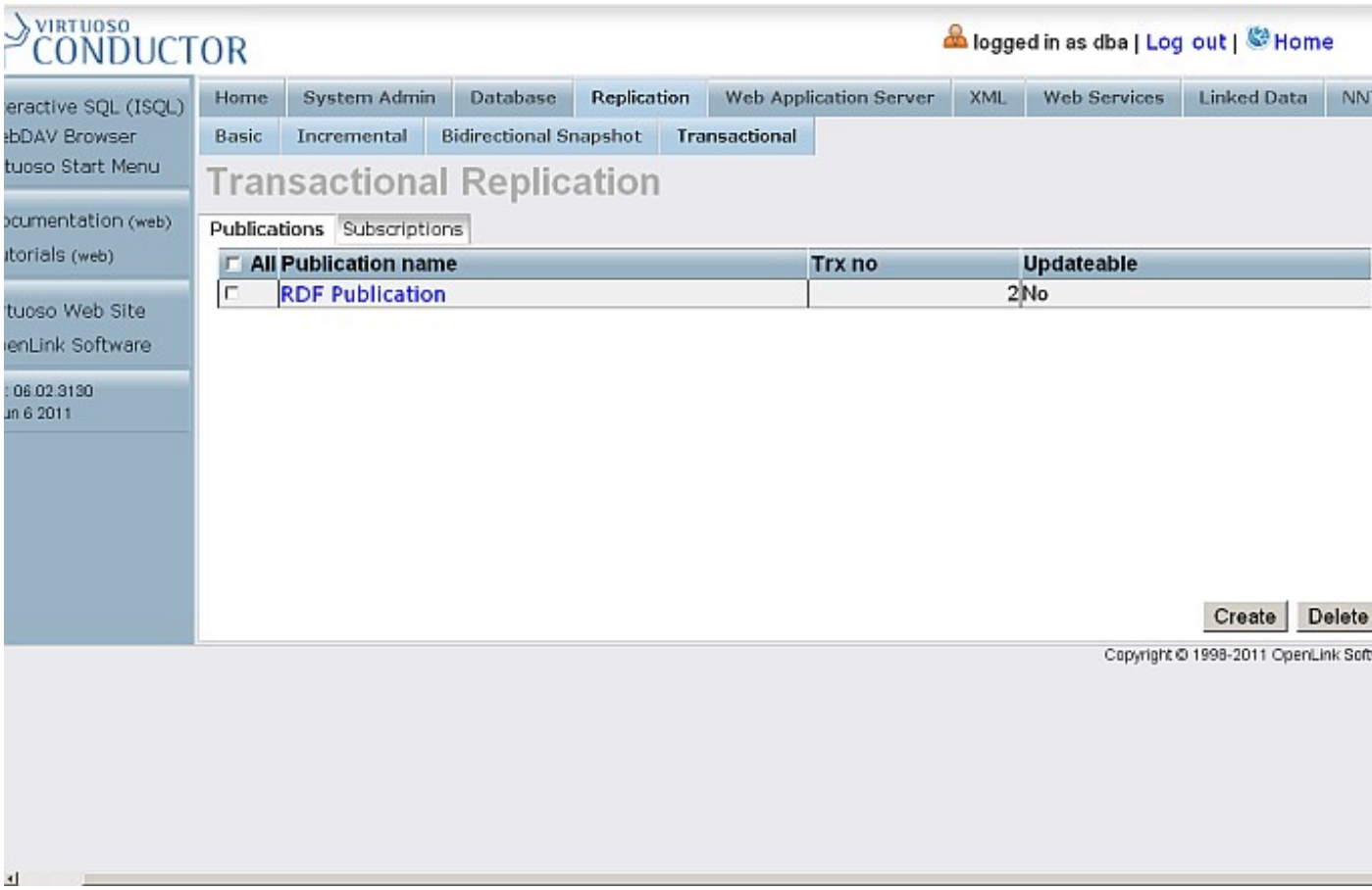
1. Go to Conductor -> Replication -> Transactional -> Publications

Figure 16.203. Star Replication Topology



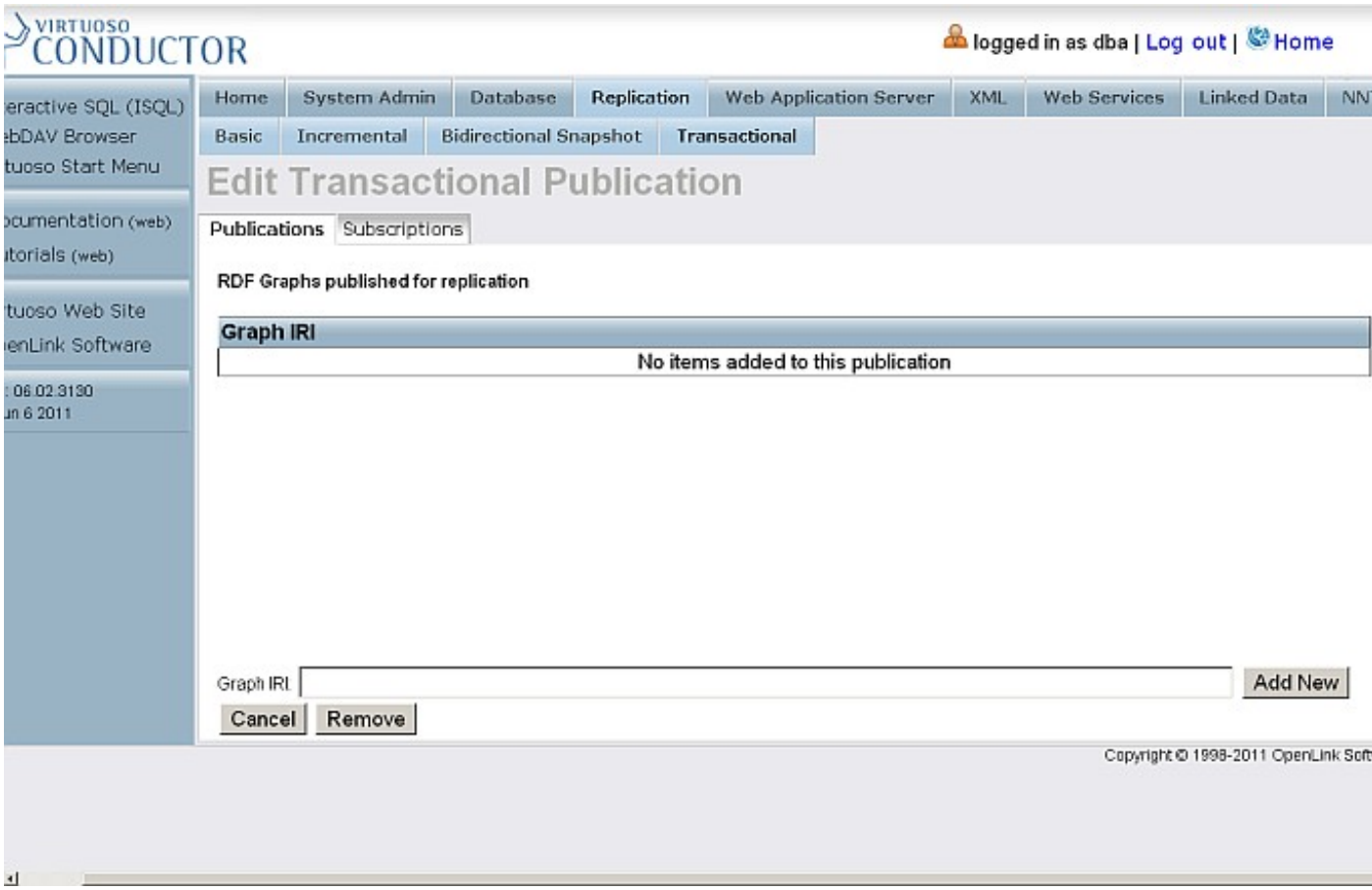
2. Click Enable RDF Publishing
3. A publication with the name RDF Publication should be created:

Figure 16.204. Star Replication Topology



4. Click the link which is the publication name.
5. You will be shown the publication items page:

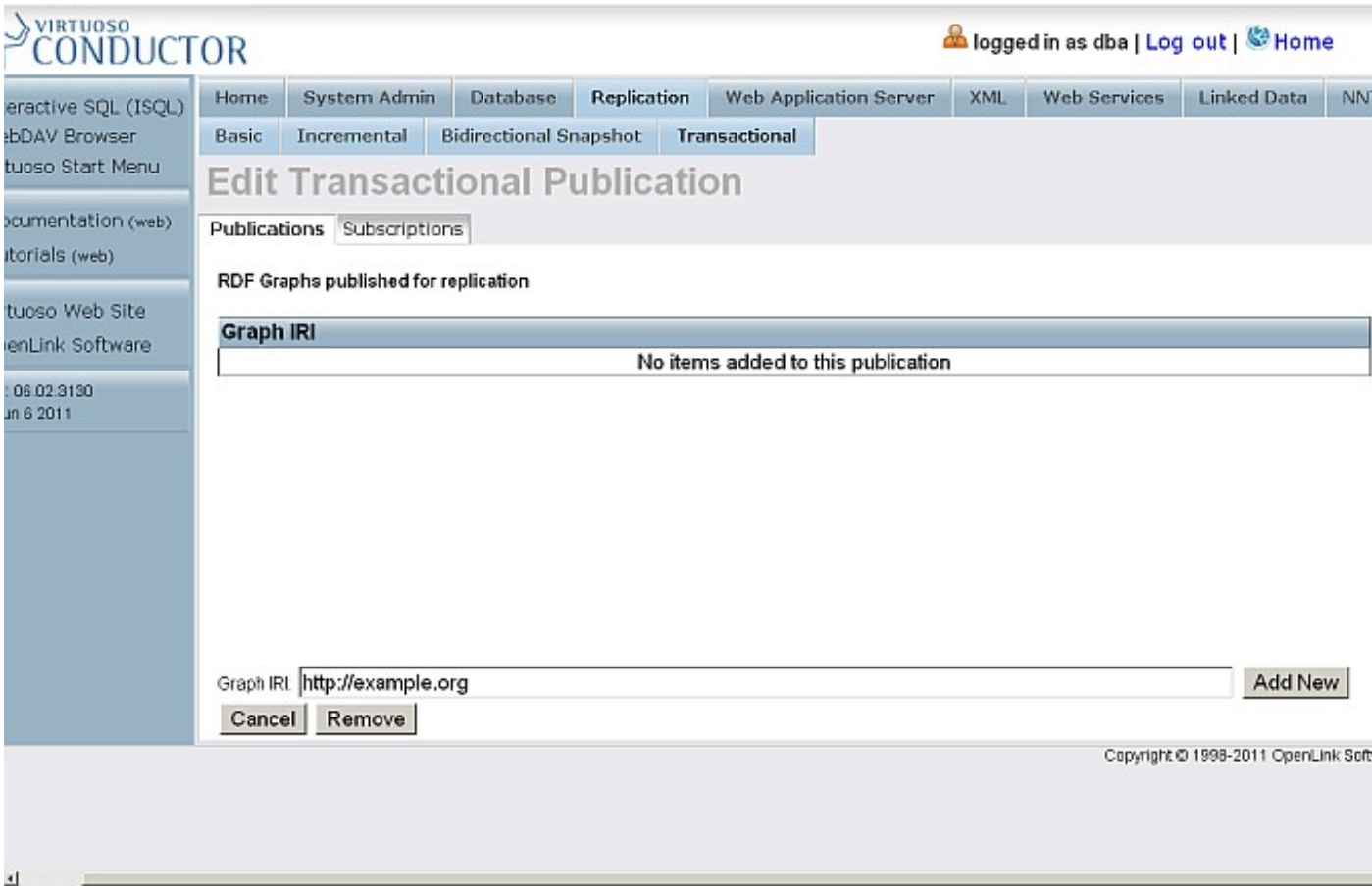
Figure 16.205. Star Replication Topology



6. Enter for Graph IRI:

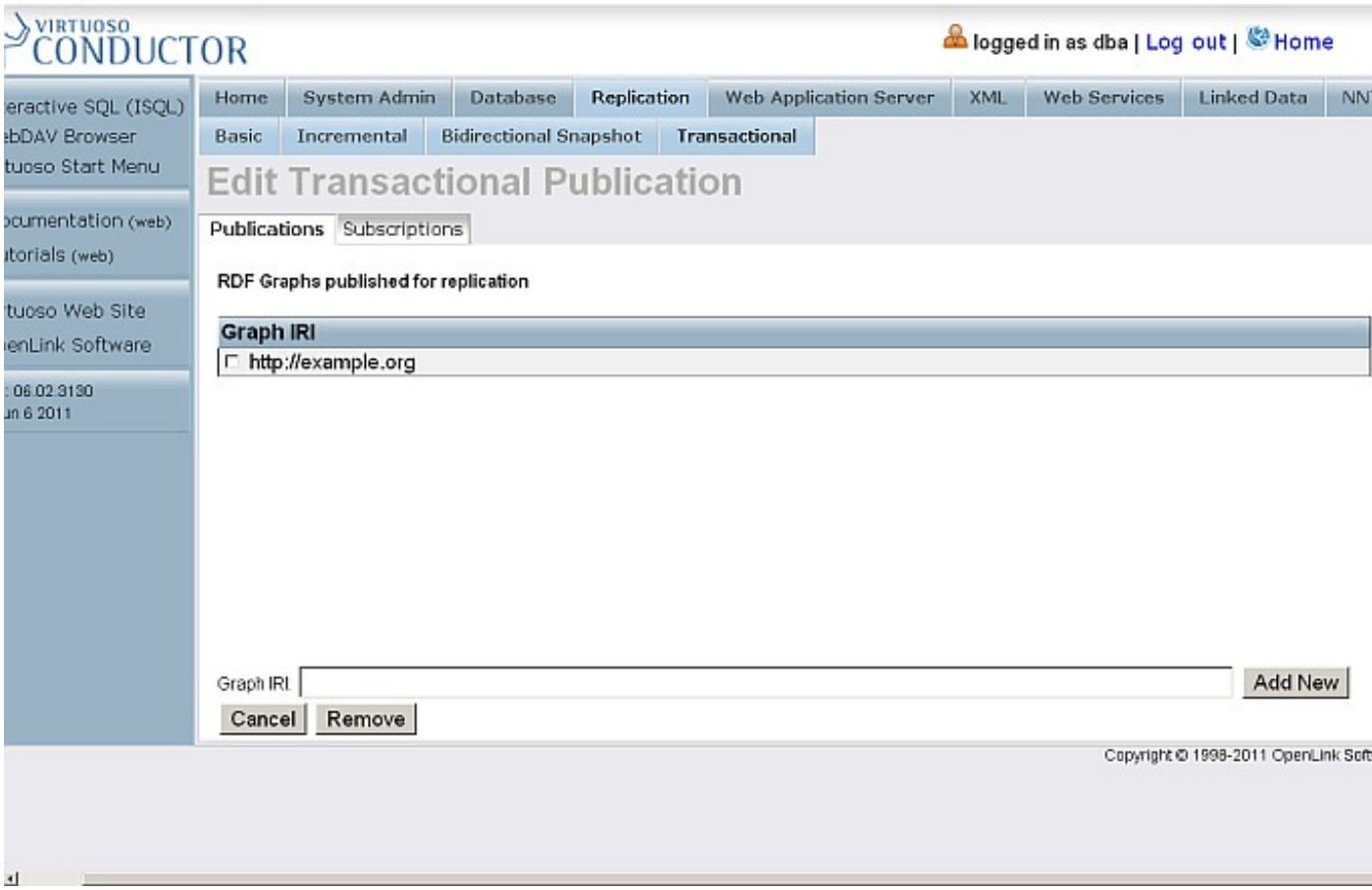
```
http://example.org
```

Figure 16.206. Star Replication Topology



7. Click Add New
8. The item will be created and shown in the list of items for the currently viewed publication.

Figure 16.207. Star Replication Topology

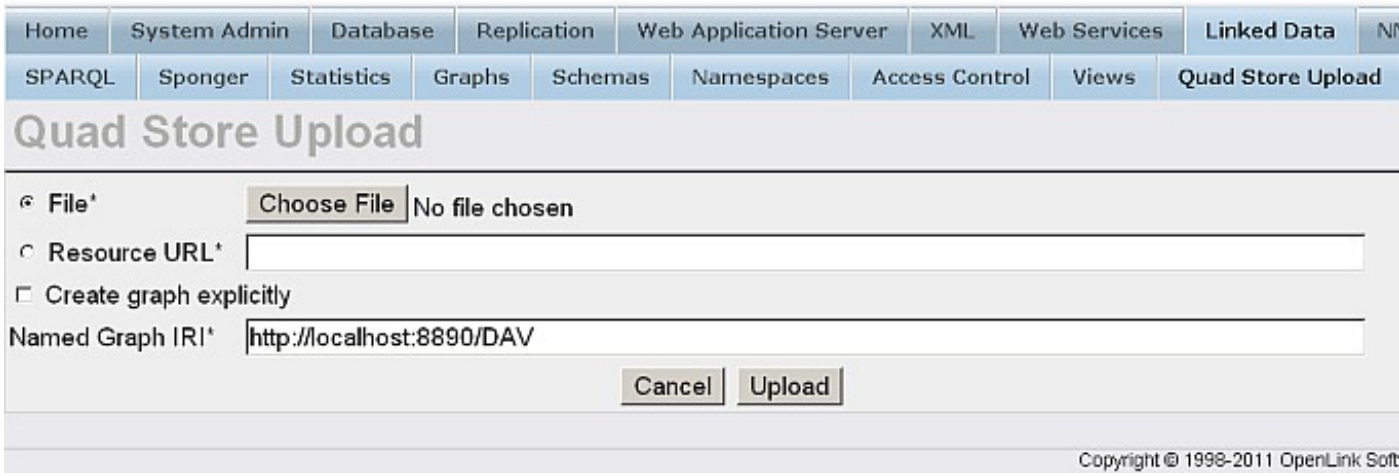


Insert Data into a Named Graph on the Host Virtuoso Instance

There are several ways to insert data into a Virtuoso Named Graph. In this example, we will use the Virtuoso Conductor's Import RDF feature:

1. In the Virtuoso Conductor, go to Linked Data -> Quad Store Upload:

   Figure 16.208. Replication Topology



2. In the form:

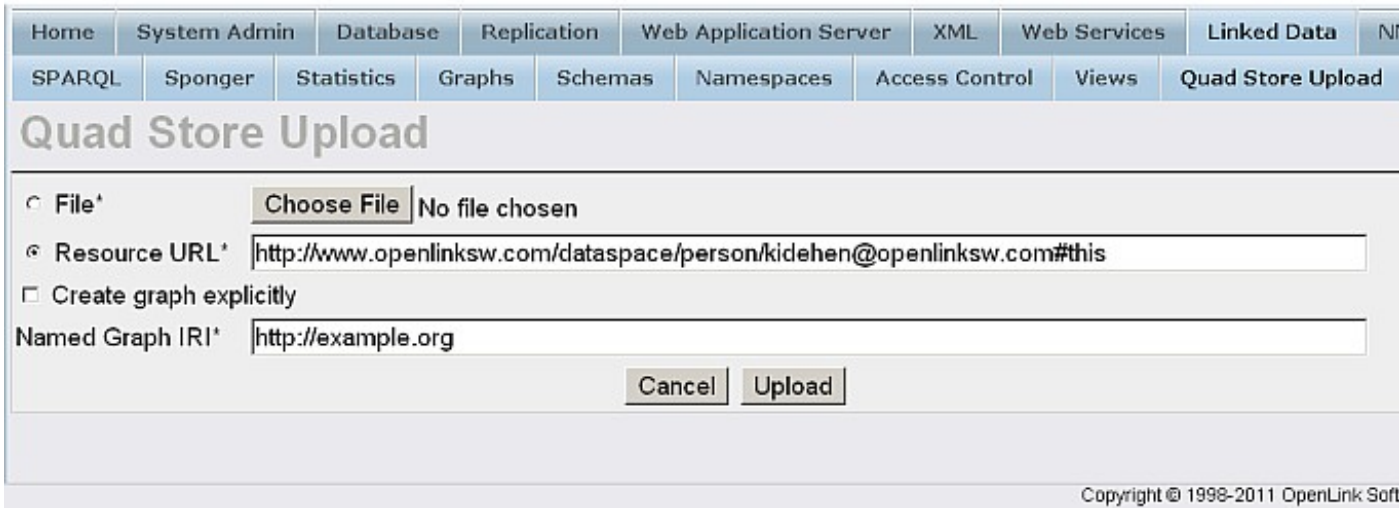
   - Tick the box for Resource URL and enter your resource URL, for e.g.:

     ```
     http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#this
     ```

   - Enter for Named Graph IRI:

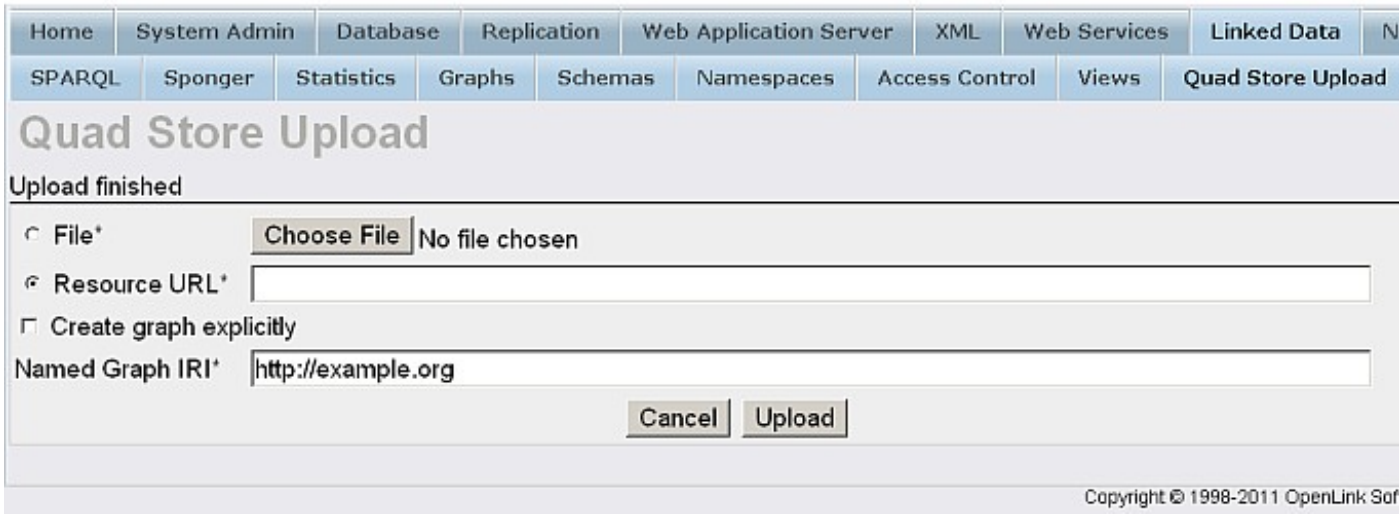
```
http://example.org
```

Figure 16.209. Replication Topology



3. Click Upload
4. A successful upload will result in this message:

Figure 16.210. Star Replication Topology



5. Check the inserted triples by executing a query like the following against the SPARQL endpoint, http://cname:port/sparql:

```
SELECT *
  FROM <http://example.org>
 WHERE { ?s ?p ?o }
```

Figure 16.211. Star Replication Topology

http://www.openlinksw.com/dataspace/kidehen@openlinksw.com#this
http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#based_near
http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#based_near
http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#based_near
http://www.openlinksw.com/dataspace/organization/dav#this
http://www.openlinksw.com/dataspace/organization/dav#this
http://www.openlinksw.com/dataspace/organization/dav#this
http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#org
http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#org
http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#org
http://www.openlinksw.com/dataspace/organization/uda#this
http://www.openlinksw.com/dataspace/organization/uda#this
http://www.openlinksw.com/dataspace/organization/uda#this
http://www.openlinksw.com/dataspace/organization/vdb#this
http://www.openlinksw.com/dataspace/organization/vdb#this
http://www.openlinksw.com/dataspace/organization/vdb#this
http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#addr
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%20%5B127%
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%20%5B127%

6. See how many triples have been inserted in your graph:

```
SELECT COUNT(*)
  FROM <http://example.org>
 WHERE { ?s ?p ?o }
```

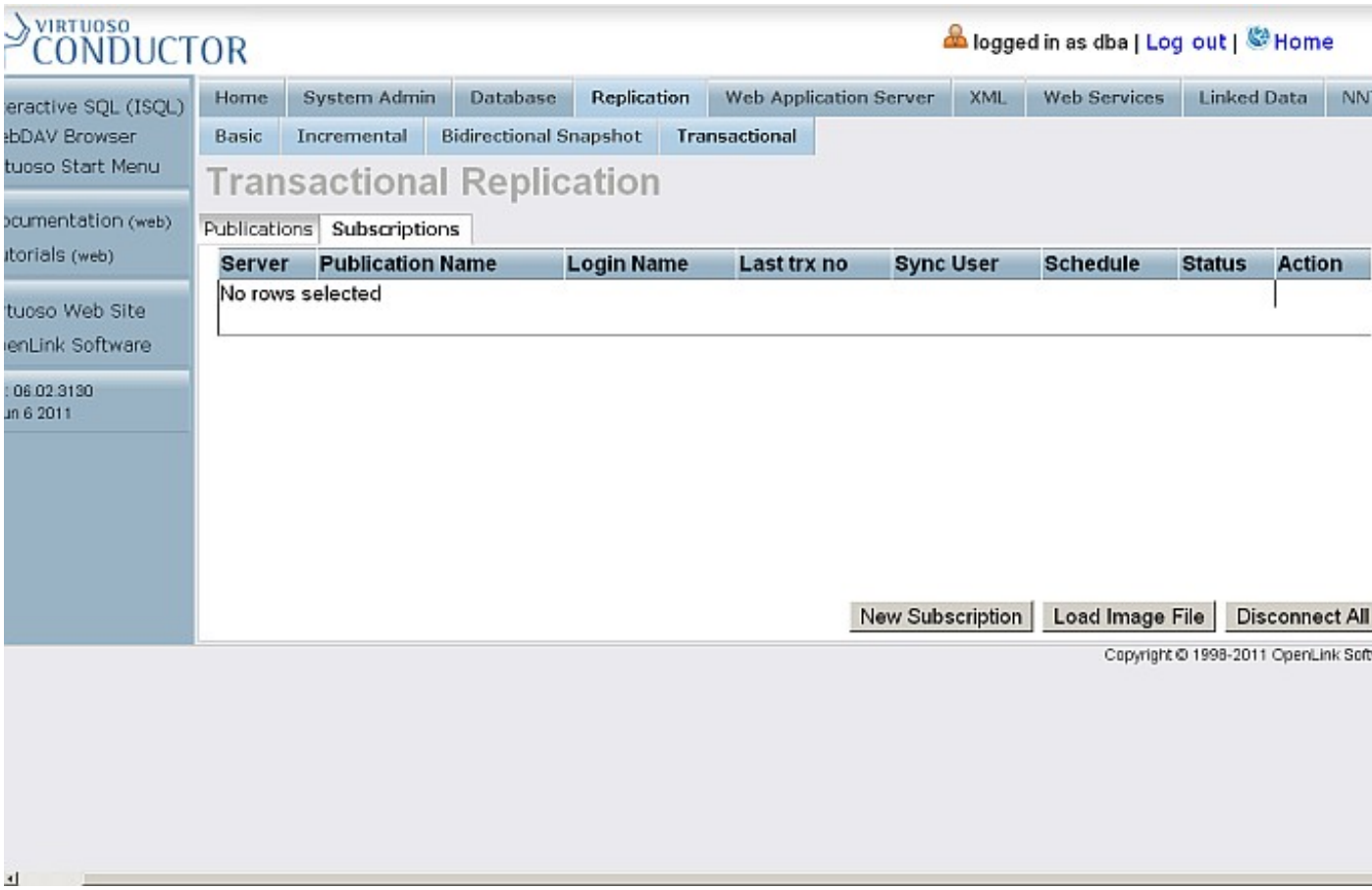
Figure 16.212. Star Replication Topology

callret-0
57

Subscribe to the Publication on the a Destination Virtuoso Instance db2, db3, etc.

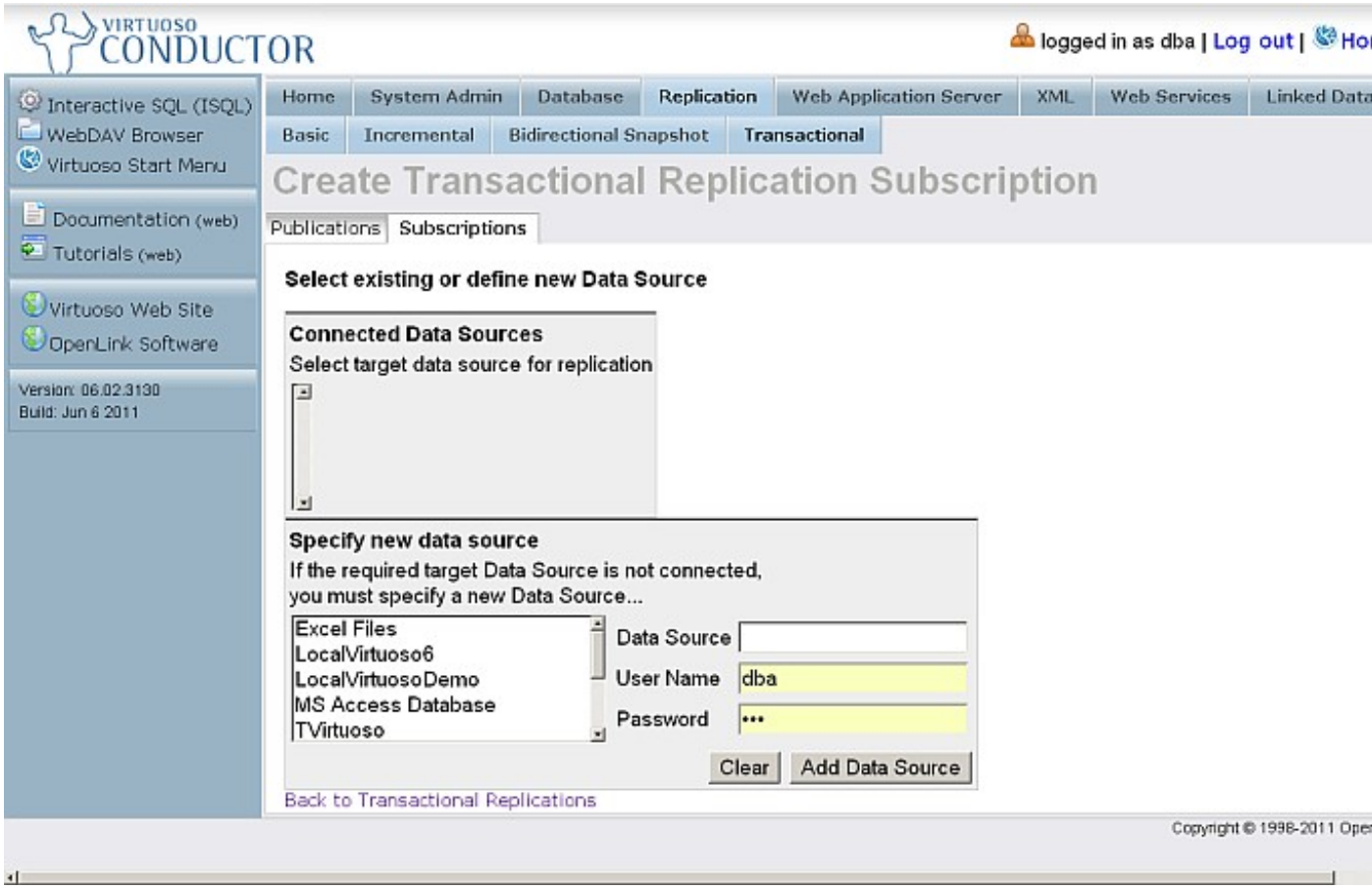
1. Go to Conductor -> Replication -> Transactional -> Subscriptions

Figure 16.213. Star Replication Topology



2. Click New Subscription

Figure 16.214. Star Replication Topology



3. Specify a new Data Source Enter or selected target data source from the available connected Data Sources:

Figure 16.215. Star Replication Topology

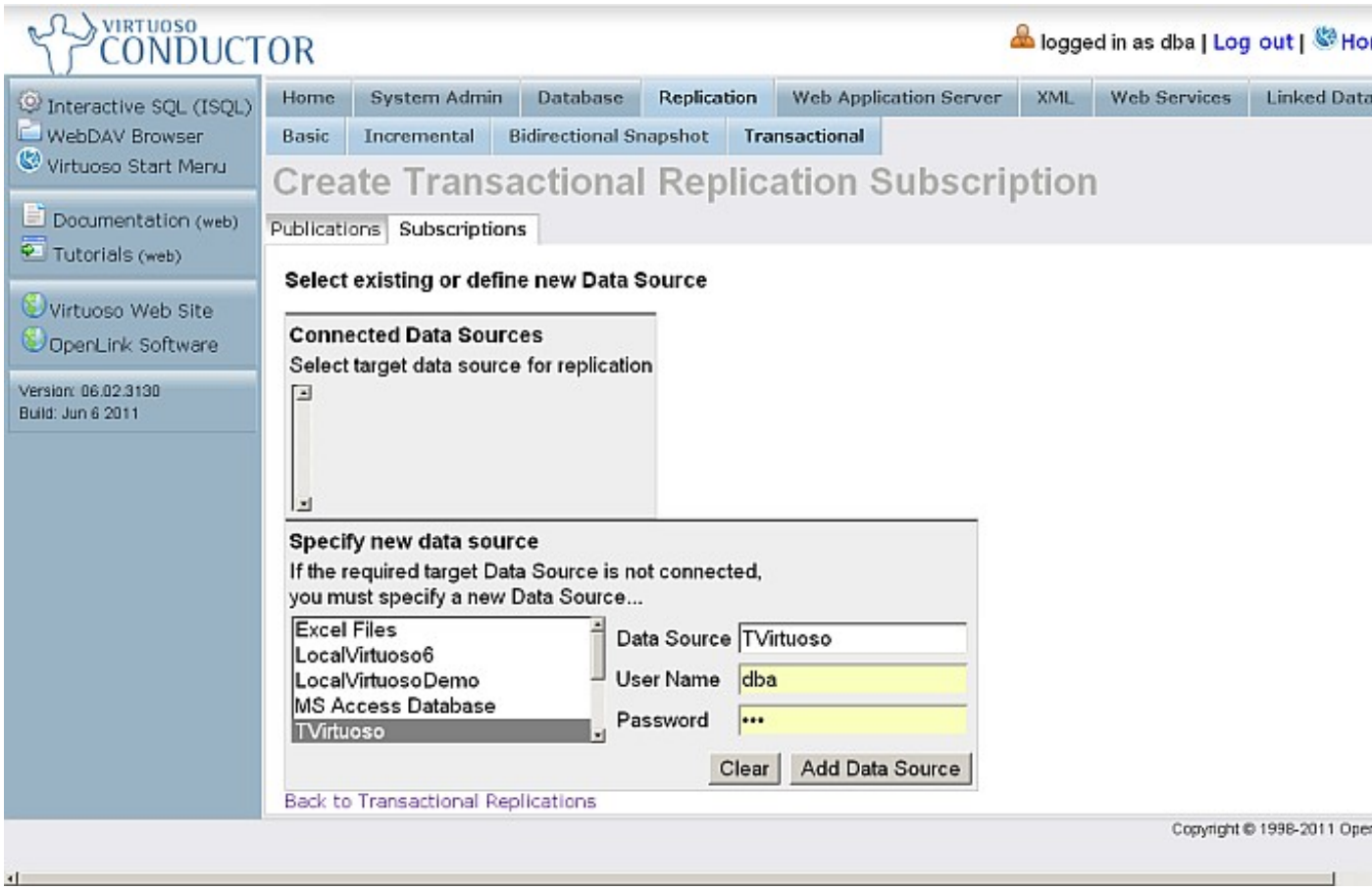
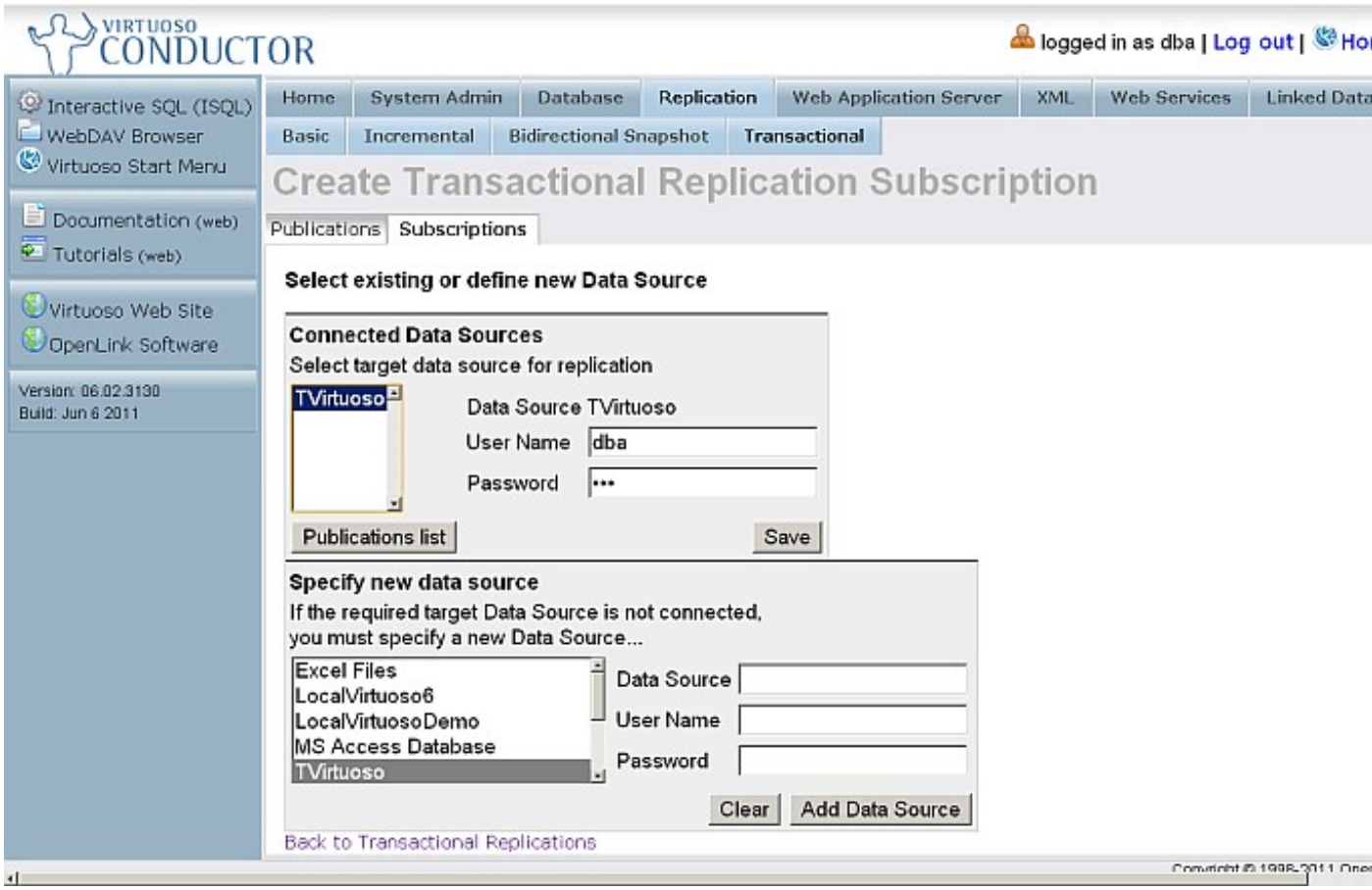


Figure 16.216. Star Replication Topology

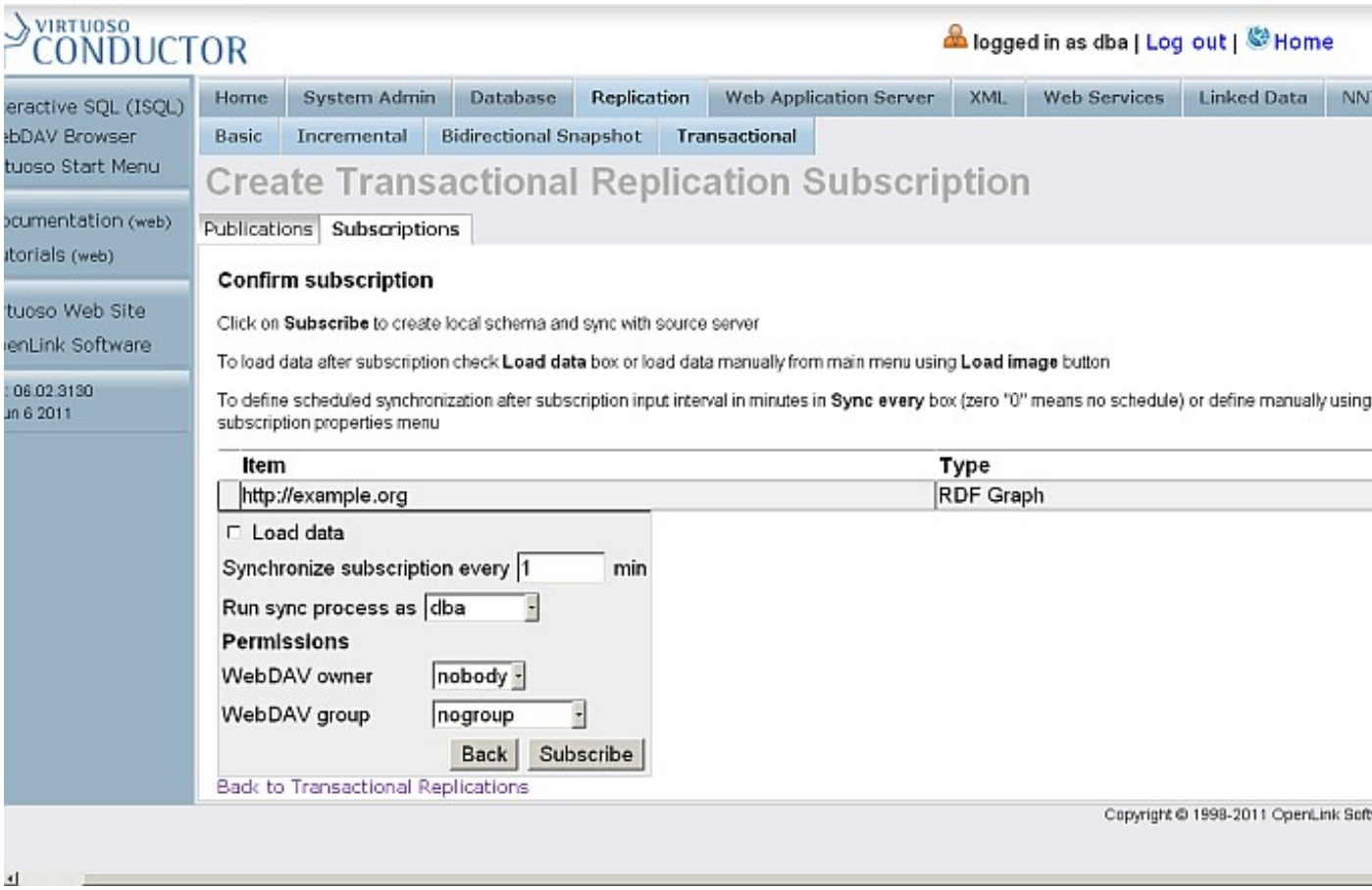


4. Click Publications list

Figure 16.217. Star Replication Topology

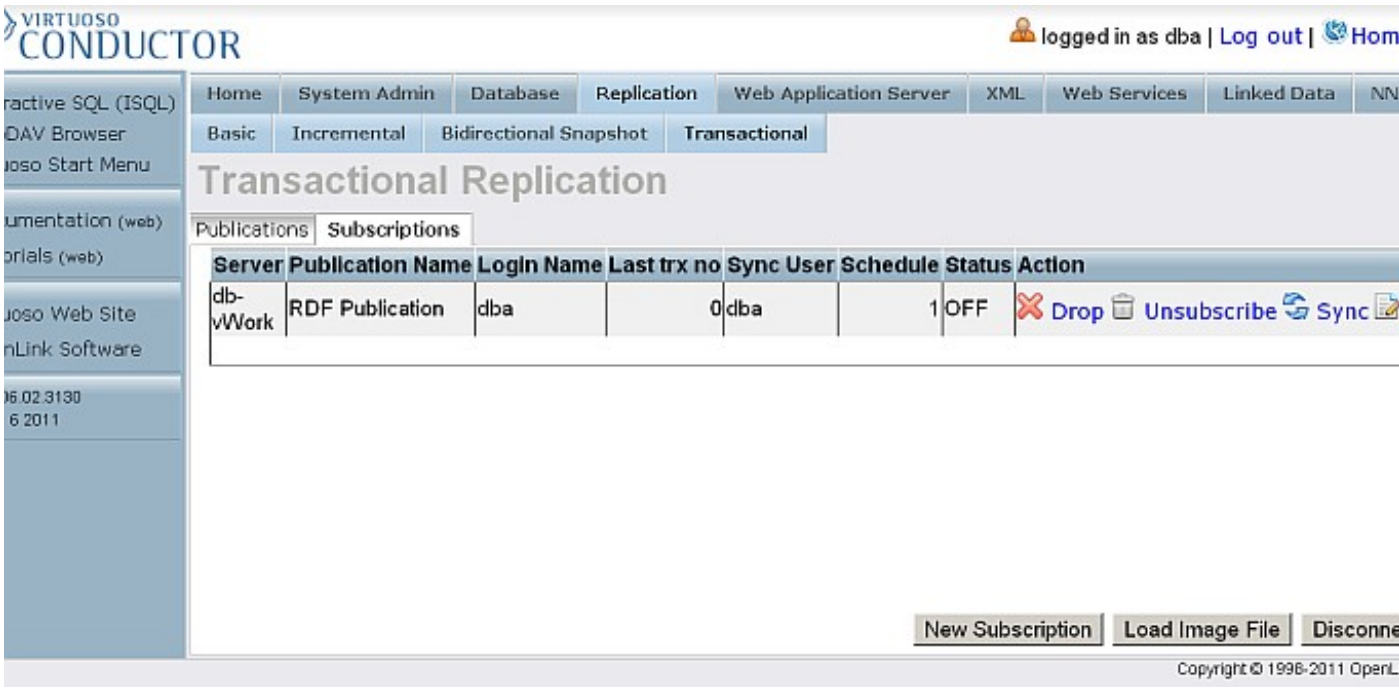
5. Select the RDF Publication and click List Items

Figure 16.218. Star Replication Topology



6. Click Subscribe
7. The subscription will be created

Figure 16.219. Star Replication Topology



8. Click Sync
9. Check the retrieved triples by executing the following query

```
SELECT *
  FROM <http://example.org>
```

```
WHERE {?s ?p ?o}
```

Figure 16.220. Star Replication Topology

s	p	o
http://www.openlinksw.com/dataspace/organization/dav#this	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://xmlns.com/foaf/0.1/Organization
http://www.openlinksw.com/dataspace/organization/dav#this	http://xmlns.com/foaf/0.1/nick	dav
http://www.openlinksw.com/dataspace/organization/dav#this	http://xmlns.com/foaf/0.1/name	OpenLink Software
http://www.openlinksw.com/dataspace/organization/vdb#this	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://xmlns.com/foaf/0.1/Organization
http://www.openlinksw.com/dataspace/organization/vdb#this	http://xmlns.com/foaf/0.1/nick	vdb
http://www.openlinksw.com/dataspace/organization/vdb#this	http://xmlns.com/foaf/0.1/name	Virtuso Data Space Bot
http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#addr	http://www.w3.org/2001/vcard-rdf/3.0#Country	United States
http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#org	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://xmlns.com/foaf/0.1/Organization
http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#org	http://purl.org/dc/elements/1.1/title	OpenLink Software

10. See how many triples have been inserted into your graph by executing the following query:

```
SELECT COUNT(*)
  FROM <http://example.org>
 WHERE {?s ?p ?o}
```

Figure 16.221. Star Replication Topology

callret-0
57

These steps may be repeated for any number of Subscriber.

Insert Triples into the Host Virtuoso Instance Graph and check availability at Destination Virtuoso Instance Graph

1. To check the starting count, on the Destination Virtuoso Instance SPARQL Endpoint, execute:

```
SELECT COUNT(*)
  FROM <http://example.org>
 WHERE { ?s ?p ?o }
```

2. On the Host Virtuoso Instance go to Conductor -> Database -> Interactive SQL and execute the following statement:

```
SPARQL INSERT INTO GRAPH <http://example.org>
  {
     <http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#this>
     <http://xmlns.com/foaf/0.1/interest>
     <http://dbpedia.org/resource/Web_Services>
  } ;
SPARQL INSERT INTO GRAPH <http://example.org>
  {
    <http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#this>
```

```
    <http://xmlns.com/foaf/0.1/interest>
    <http://dbpedia.org/resource/Web_Clients>
  } ;
SPARQL INSERT INTO GRAPH <http://example.org>
  {
    <http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#this>
    <http://xmlns.com/foaf/0.1/interest>
    <http://dbpedia.org/resource/SPARQL>
  } ;
```

Figure 16.222. Star Replication Topology

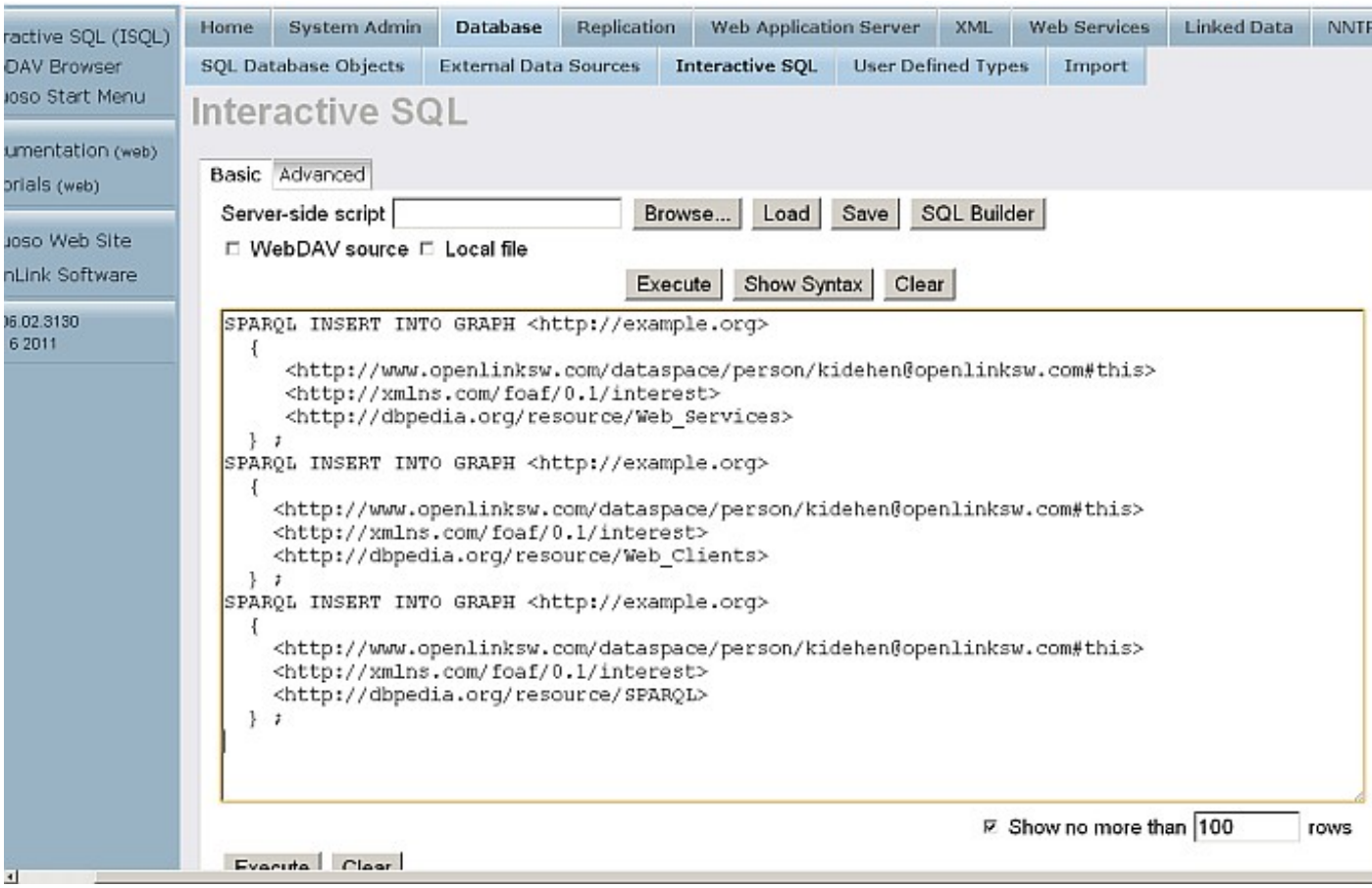
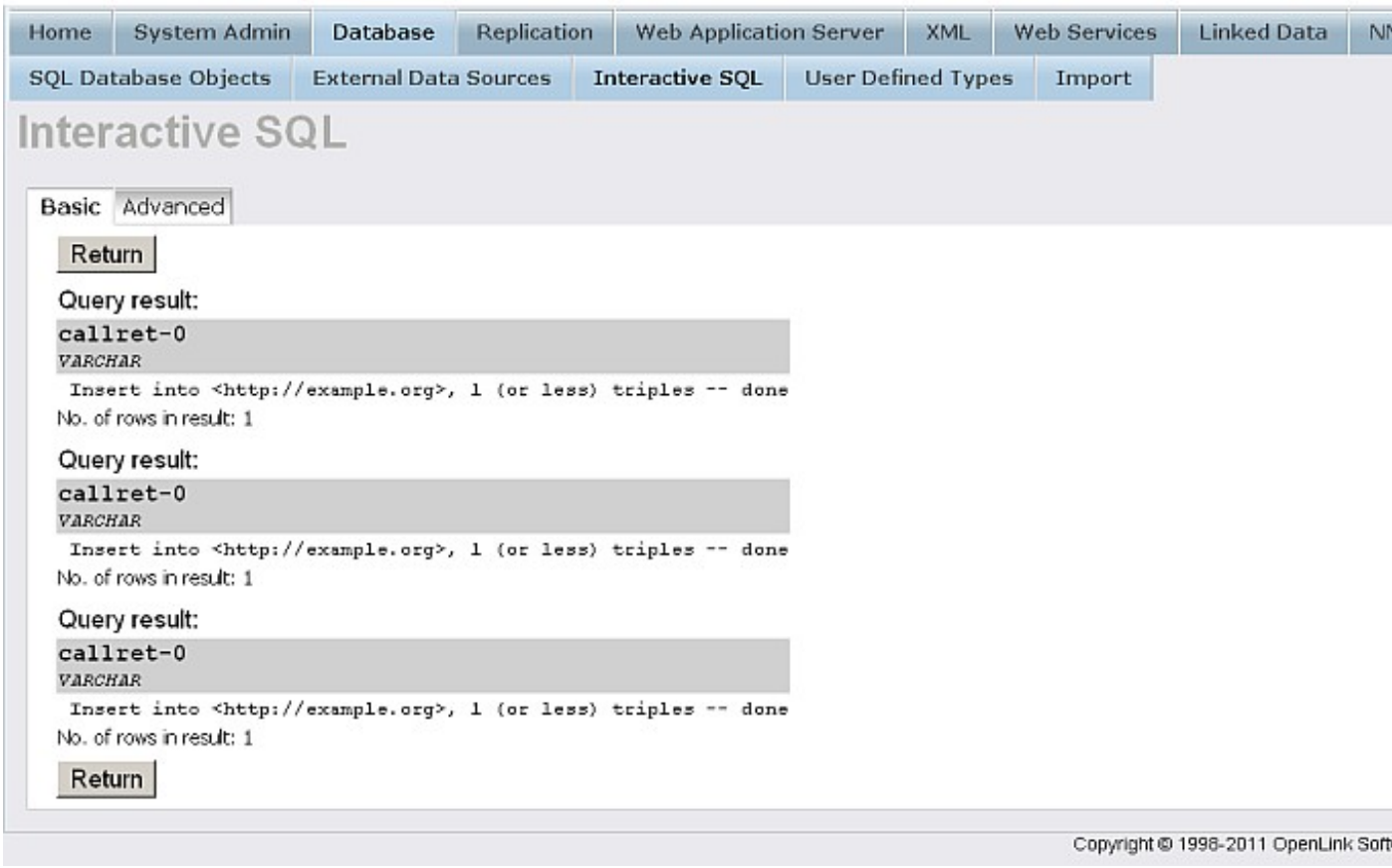


Figure 16.223. Star Replication Topology



3. To confirm that the triple count has increased by the number of inserted triples, execute the following on the Destination Virtuoso Instance SPARQL Endpoint:

```
SELECT COUNT(*)
  FROM <http://example.org>
 WHERE { ?s ?p ?o }
```

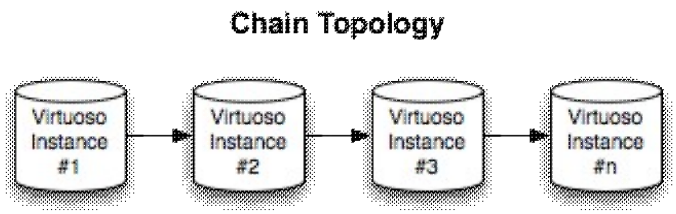
Figure 16.224. Star Replication Topology

callret-0
60

Chain Replication Topology

In a Chain, there is one original Publisher, to which there is only one Subscriber. That Subscriber may also serve as a Publisher, again with only one Subscriber. The chain ends with a Subscriber which does not Publish.

Figure 16.225. Chain Replication Topology



To set up a Chain, follow the scenario:

1. Configure Instance #1 to Publish.
2. Configure Instance #2 to Subscribe to #1.

3. Configure Instance #2 to Publish.
4. Configure Instance #3 to Subscribe to #2.
5. Repeat as necessary.

Chain Replication Topology Example

The following How-To walks you through setting up Virtuoso RDF Graph Replication in a Chain Topology.

Prerequisites

Database INI Parameters

Suppose there are 3 Virtuoso instances respectively with the following ini parameters values:

1. virtuoso1.ini:

```
...
[Database]
DatabaseFile    = virtuoso1.db
TransactionFile = virtuoso1.trx
ErrorLogFile     = virtuoso1.log
...
[Parameters]
ServerPort               = 1111
SchedulerInterval        = 1
...
[HTTPServer]
ServerPort                  = 8891
...
[URIQA]
DefaultHost = example.com:8891
...
[Replication]
ServerName   = db1
...
```

2. virtuoso2.ini:

```
...
[Database]
DatabaseFile    = virtuoso2.db
TransactionFile = virtuoso2.trx
ErrorLogFile     = virtuoso2.log
...
[Parameters]
ServerPort               = 1112
SchedulerInterval        = 1
...
[HTTPServer]
ServerPort                  = 8892
...
[URIQA]
DefaultHost = localhost:8892
...
[Replication]
ServerName   = db2
...
```

3. virtuoso3.ini:

```
...
[Database]
DatabaseFile    = virtuoso3.db
TransactionFile = virtuoso3.trx
ErrorLogFile     = virtuoso3.log
...
[Parameters]
ServerPort               = 1113
SchedulerInterval        = 1
...
[HTTPServer]
```

```
ServerPort                  = 8893
...
[URIQA]
DefaultHost = example.com:8893
...
[Replication]
ServerName   = db3
...
```

Database DSNs

Use the ODBC Administrator on your Virtuoso host (e.g., on Windows, Start menu -> Control Panel -> Administrative Tools -> Data Sources (ODBC); on Mac OS X, /Applications/Utilities/OpenLink ODBC Administrator.app) to create a System DSN for each of db1, db2, db3, with names db1, db2 and db3, respectively.

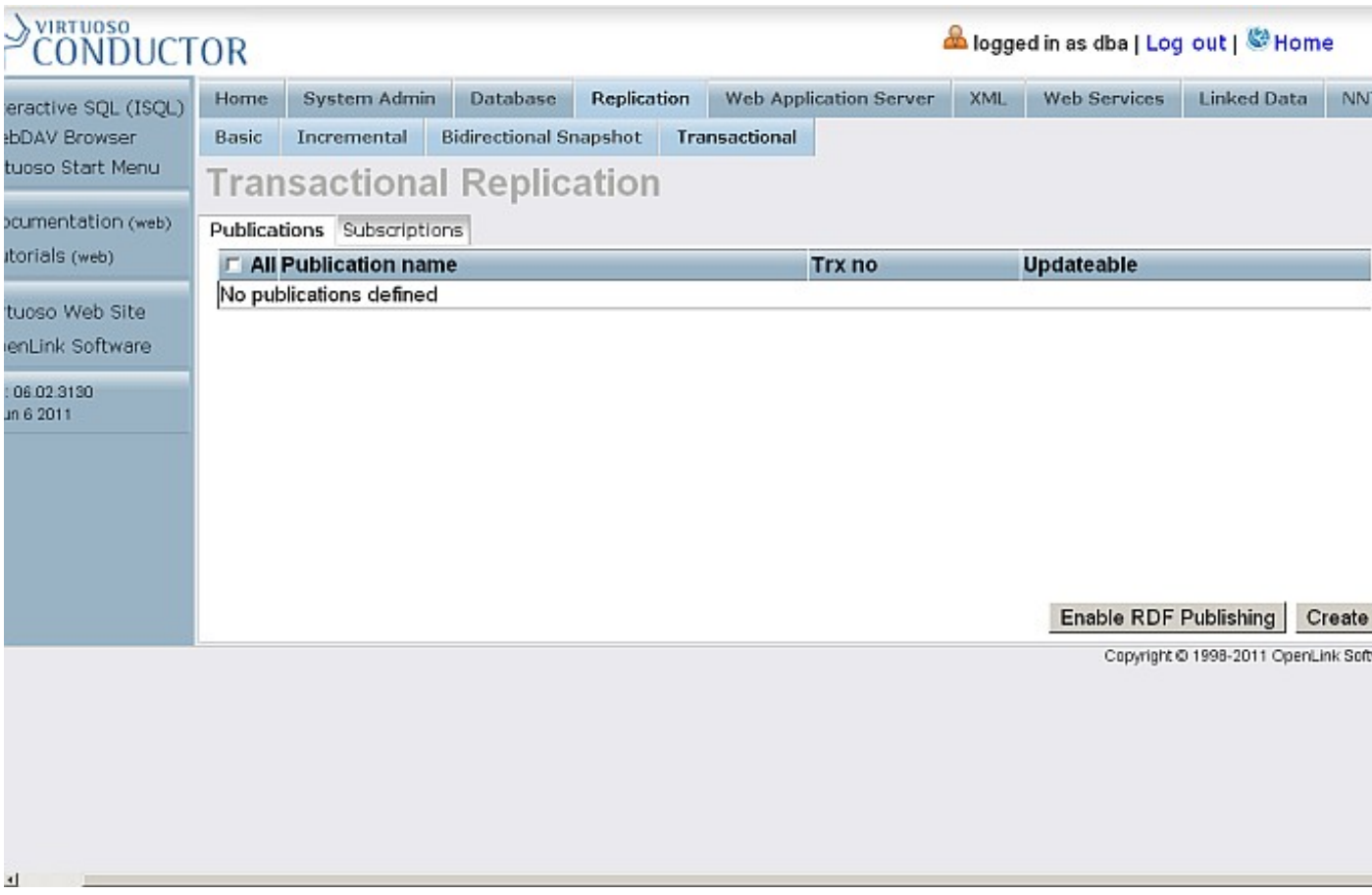
Install Conductor package

On each of the 3 Virtuoso instances install the conductor_dav.vad package.

Create Publication on db1

1. Go to http://example.com:8891/conductor and log in as dba
2. Go to Conductor - > Replication - > Transactional - > Publications

Figure 16.226. Chain Replication Topology



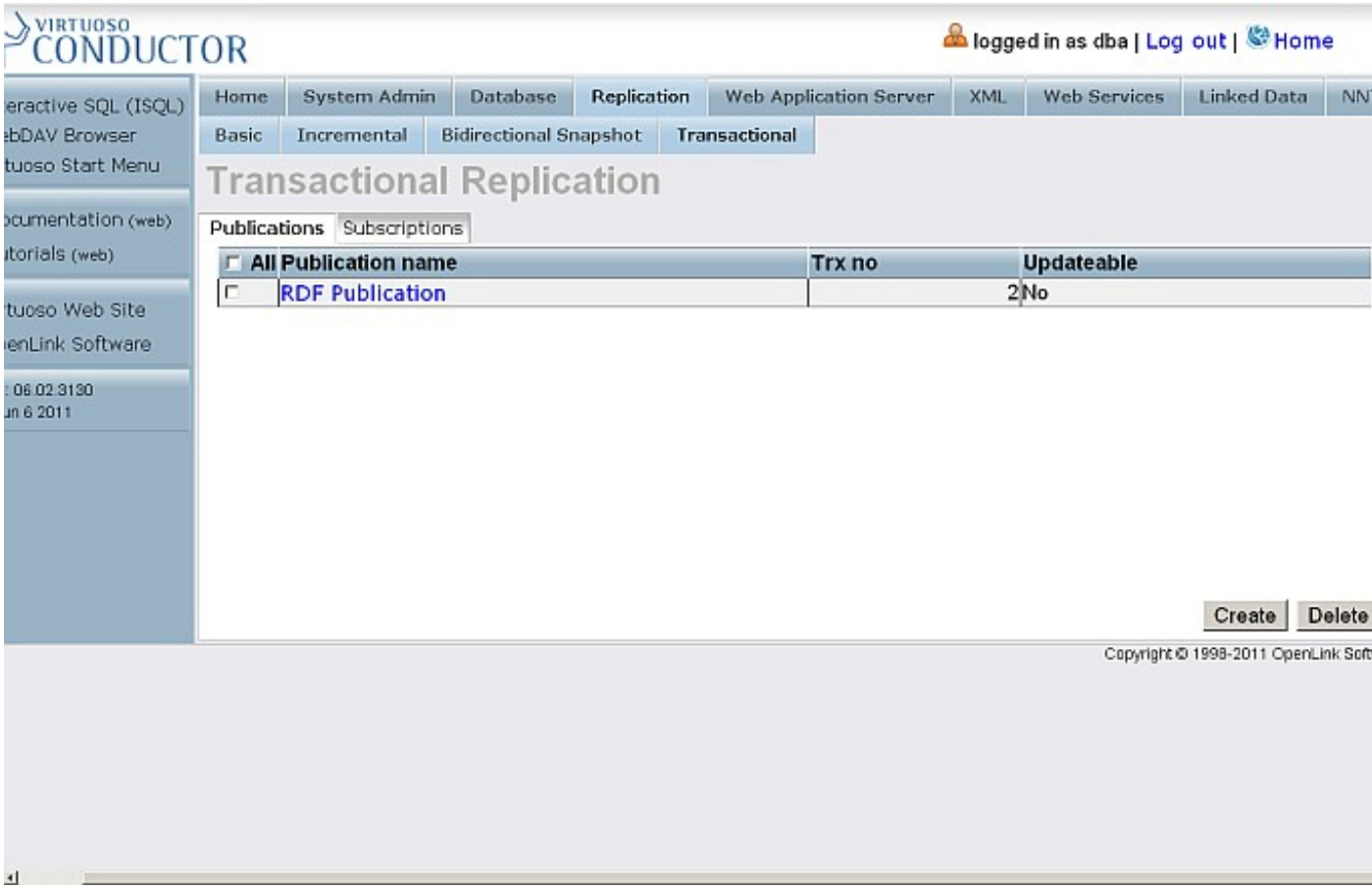
3. Click

   Enable RDF Publishing
4. As result publication with the name

   RDF Publication

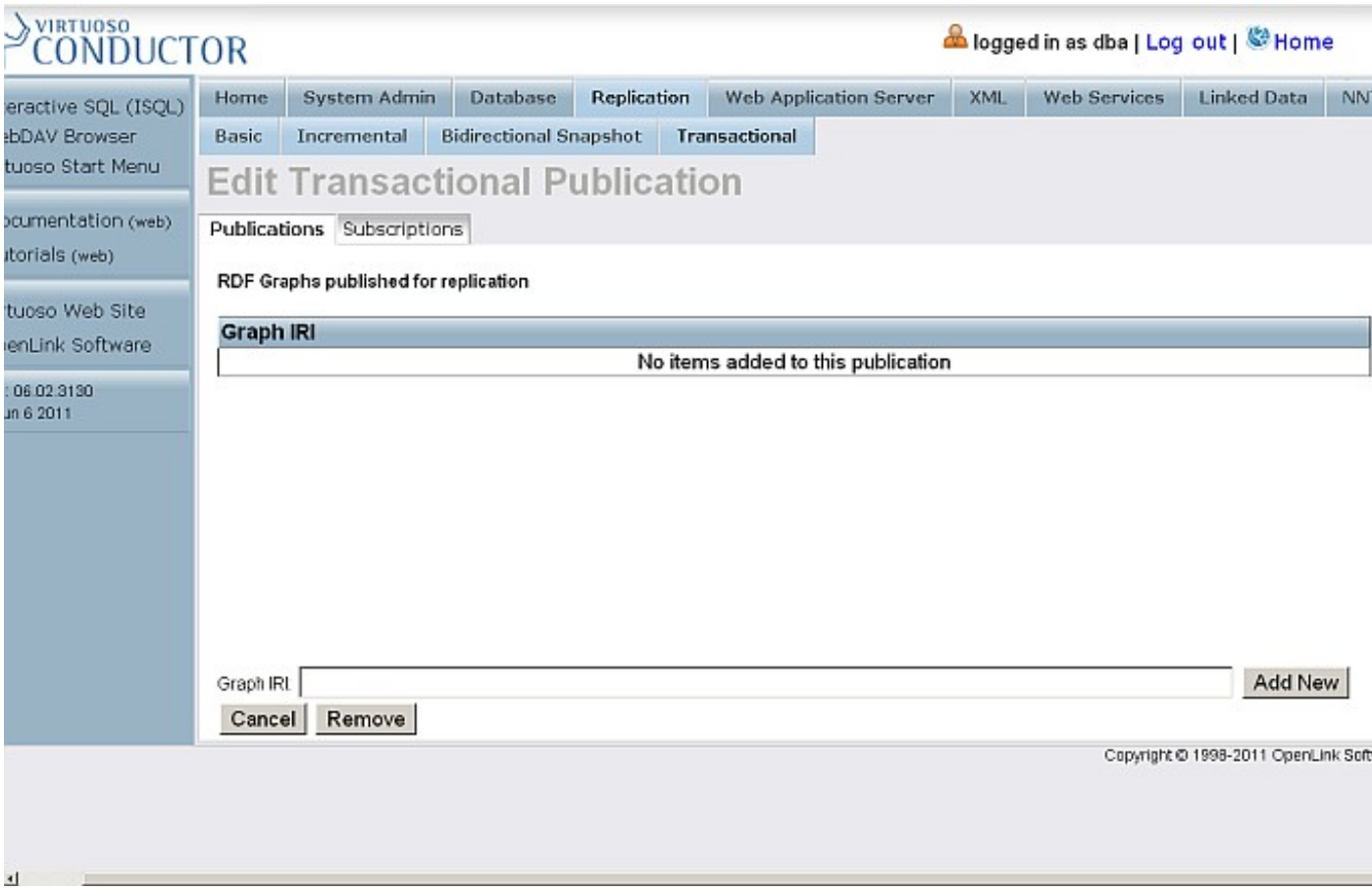
   should be created

Figure 16.227. Chain Replication Topology



5. Click the link which is the publication name.
6. You will be shown the publication items page

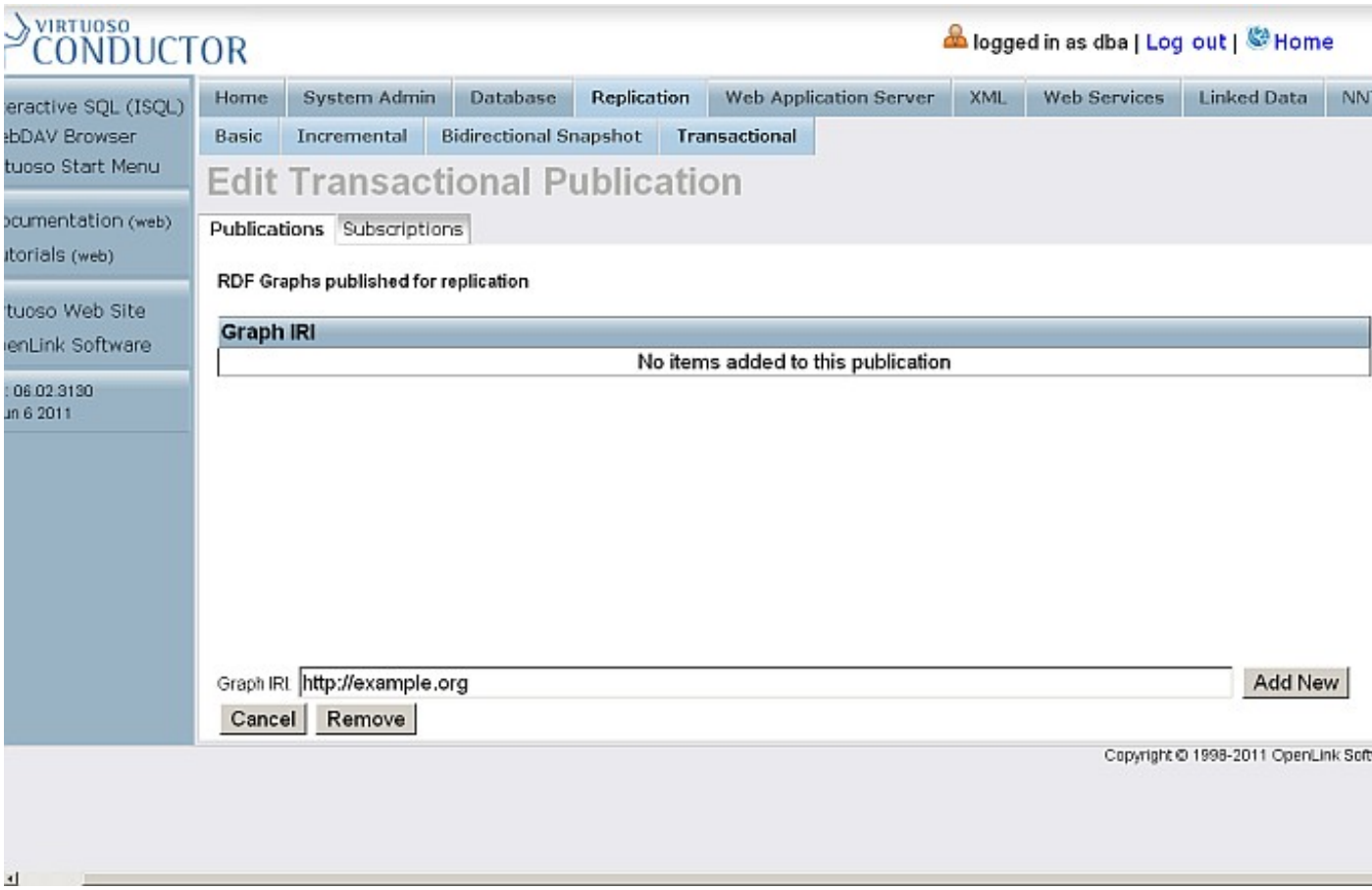
Figure 16.228. Chain Replication Topology



7. Enter for Graph IRI:

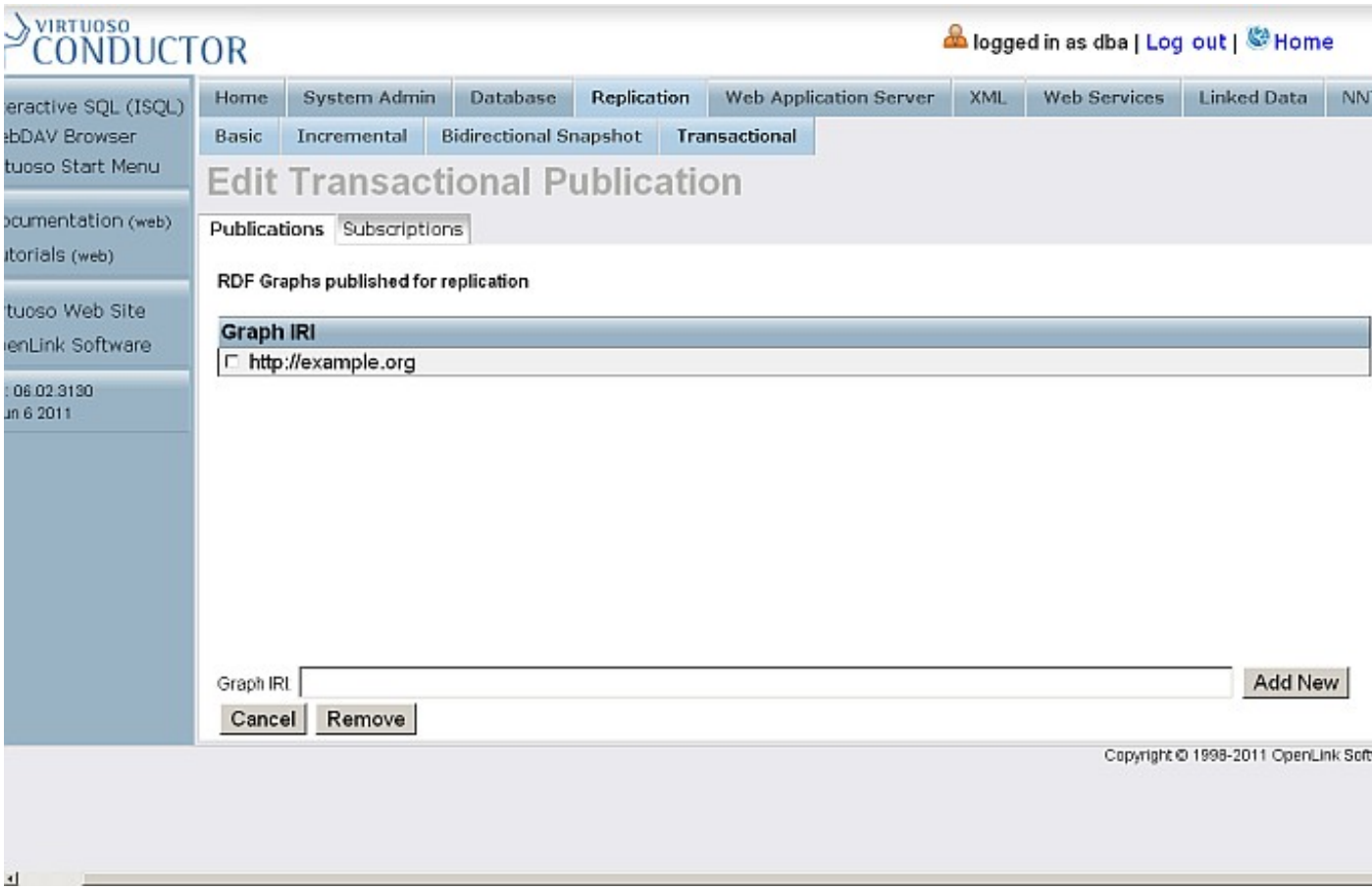
```
http://example.org
```

Figure 16.229. Chain Replication Topology



8. Click Add New
9. The item will be created and shown in the list of items for the currently viewed publication.

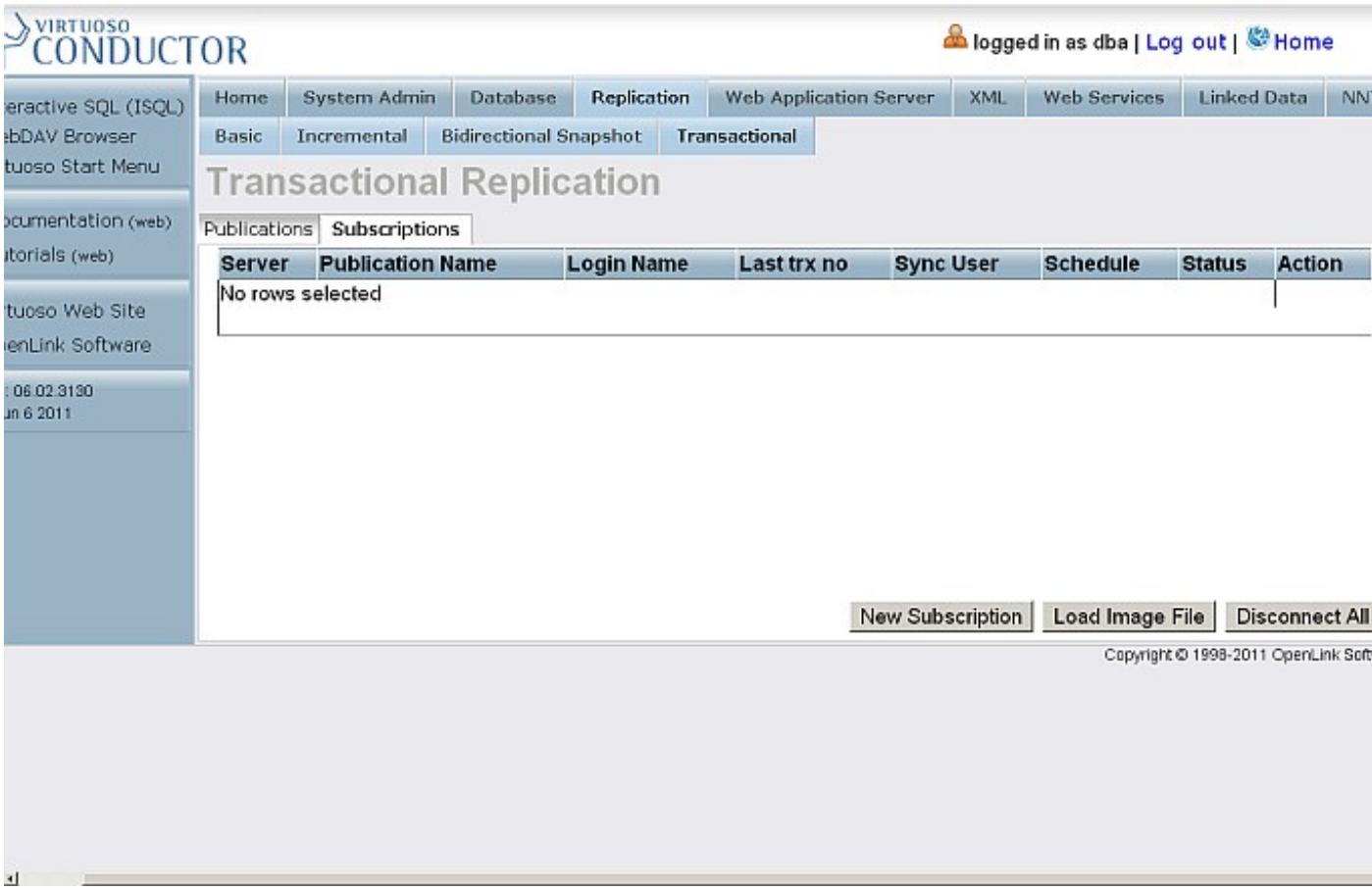
Figure 16.230. Chain Replication Topology



Create subscription from db2 to db1's Publication

1. Log in at http://example.com:8892/conductor
2. Go to Replication - > Transactional - > Subscriptions

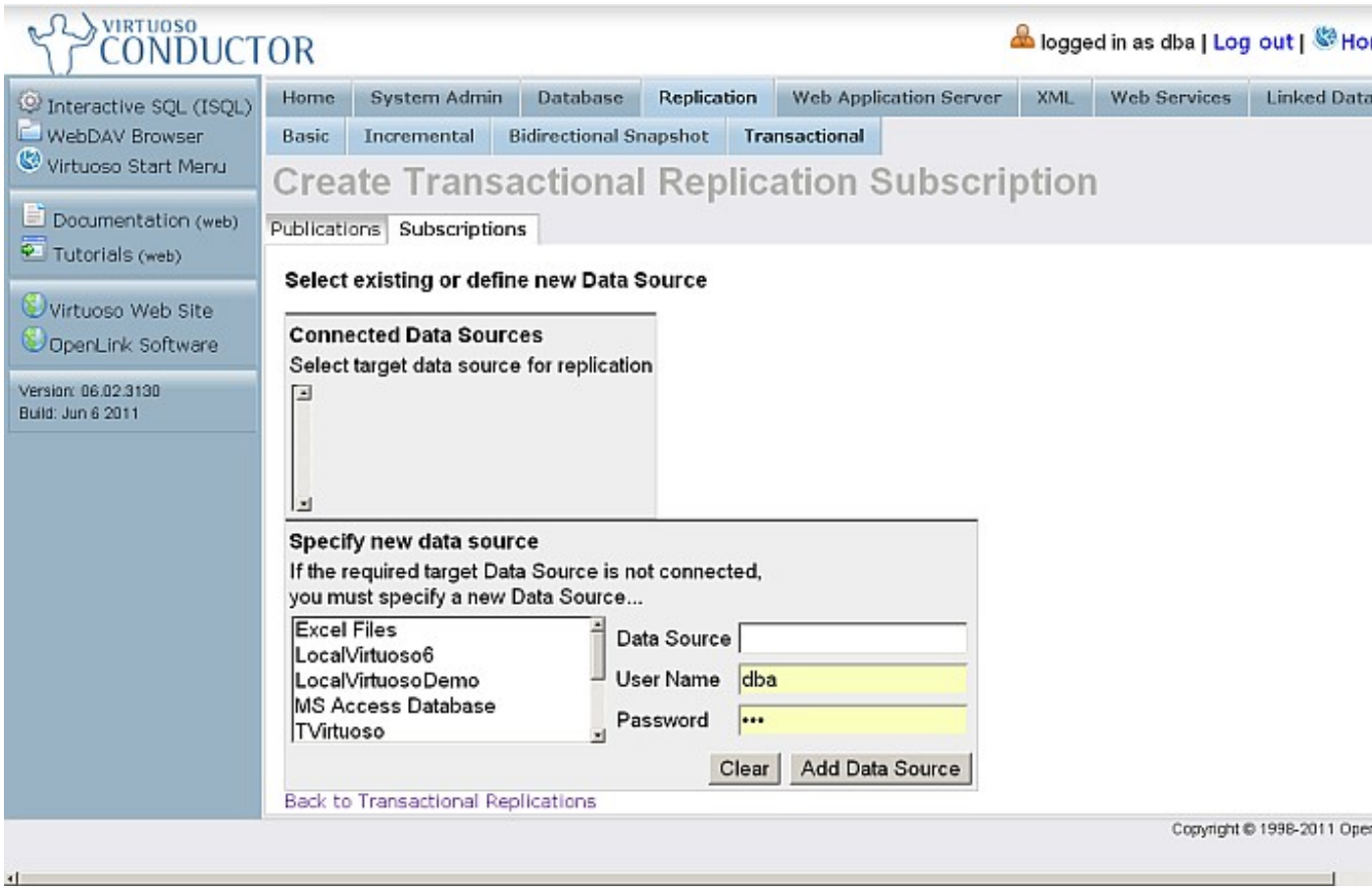
Figure 16.231. Chain Replication Topology



3. Click

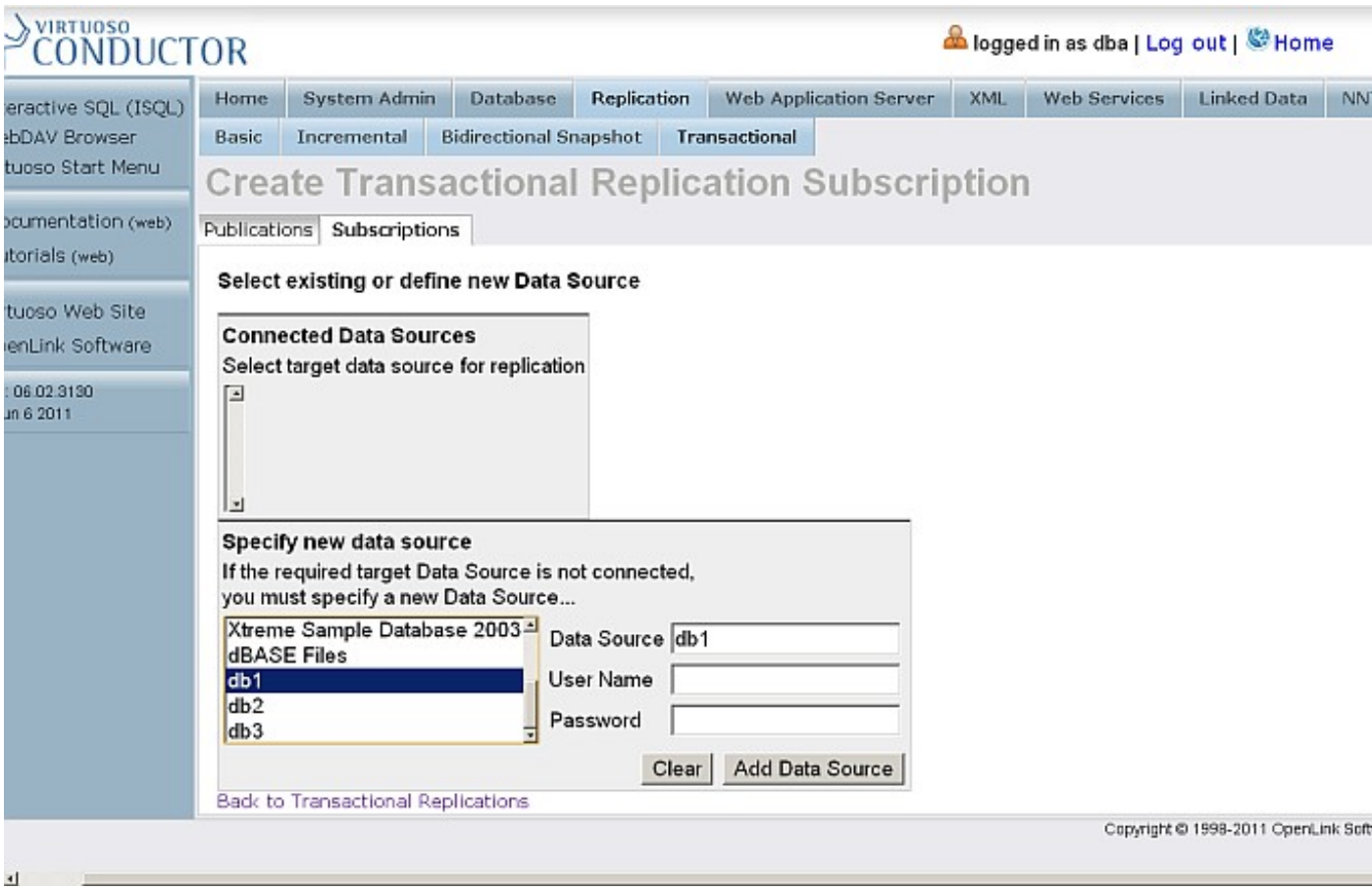
   New Subscription

Figure 16.232. Chain Replication Topology



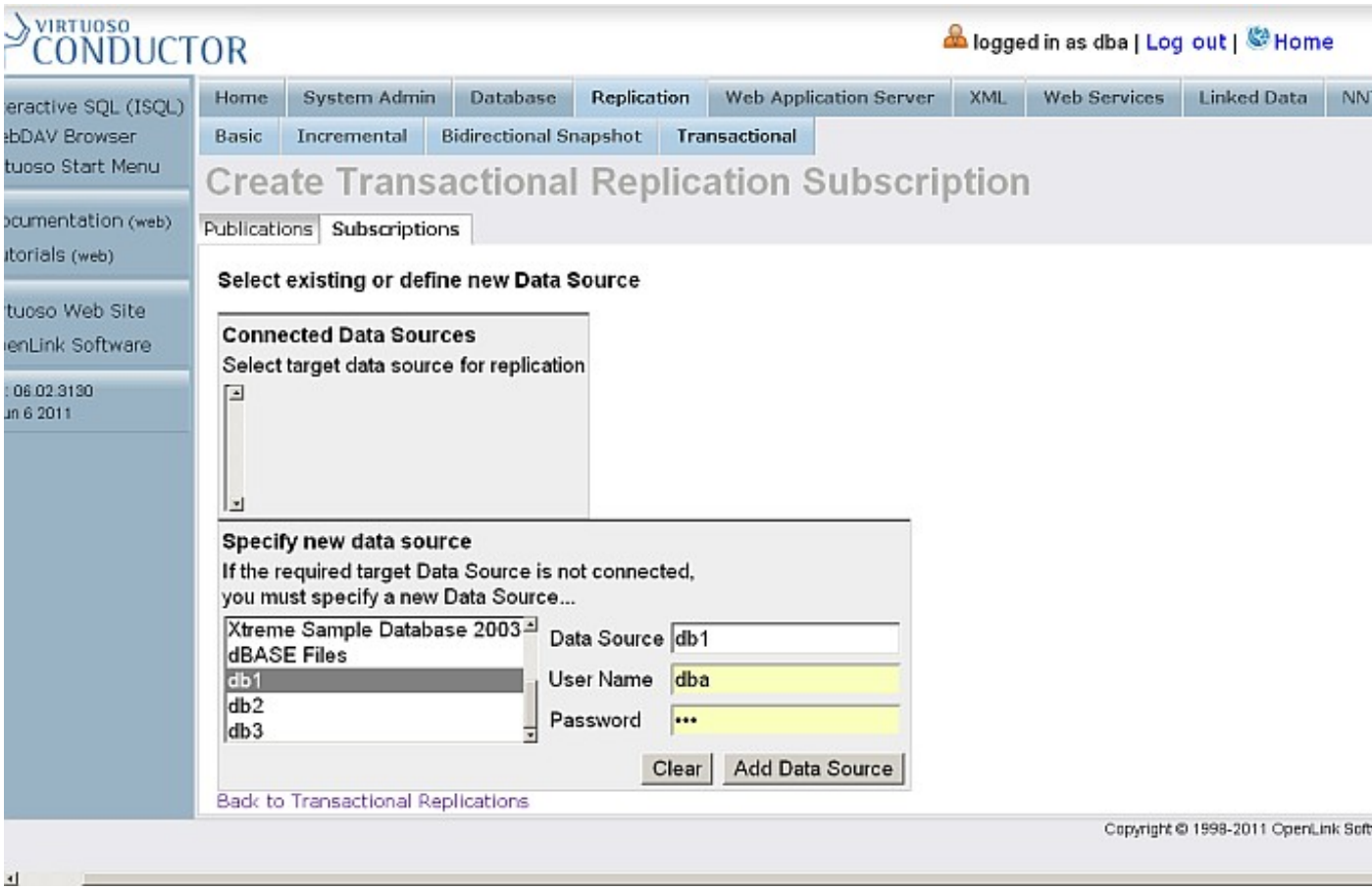
4. From the list of "Specify new data source" select Data Source db1

Figure 16.233. Chain Replication Topology



5. Enter for db1 dba user credentials

Figure 16.234. Chain Replication Topology

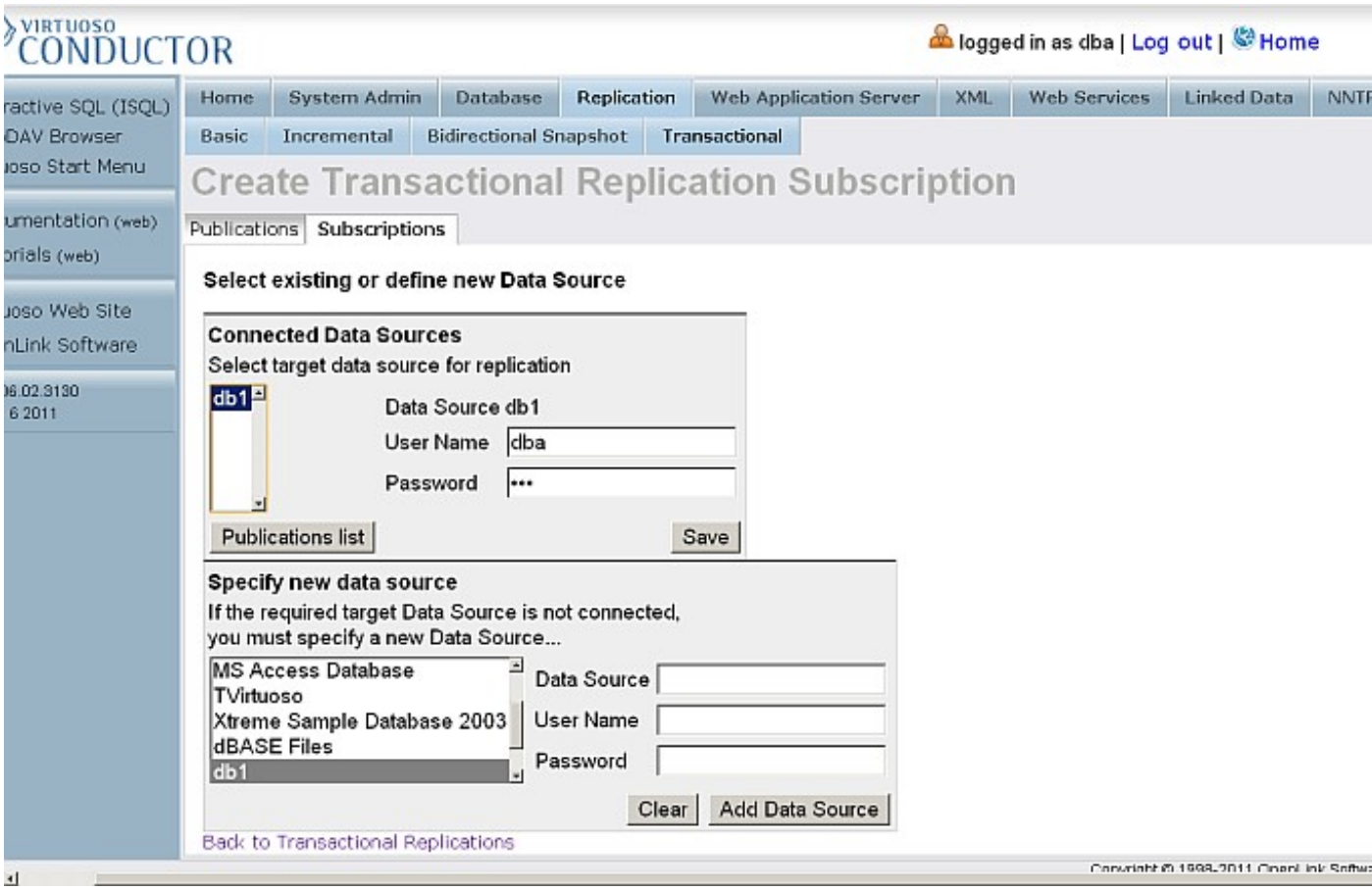


6. Click "Add Data Source"
7. As result

   db1

   will be shown in the "Connected Data Sources" list.

Figure 16.235. Chain Replication Topology

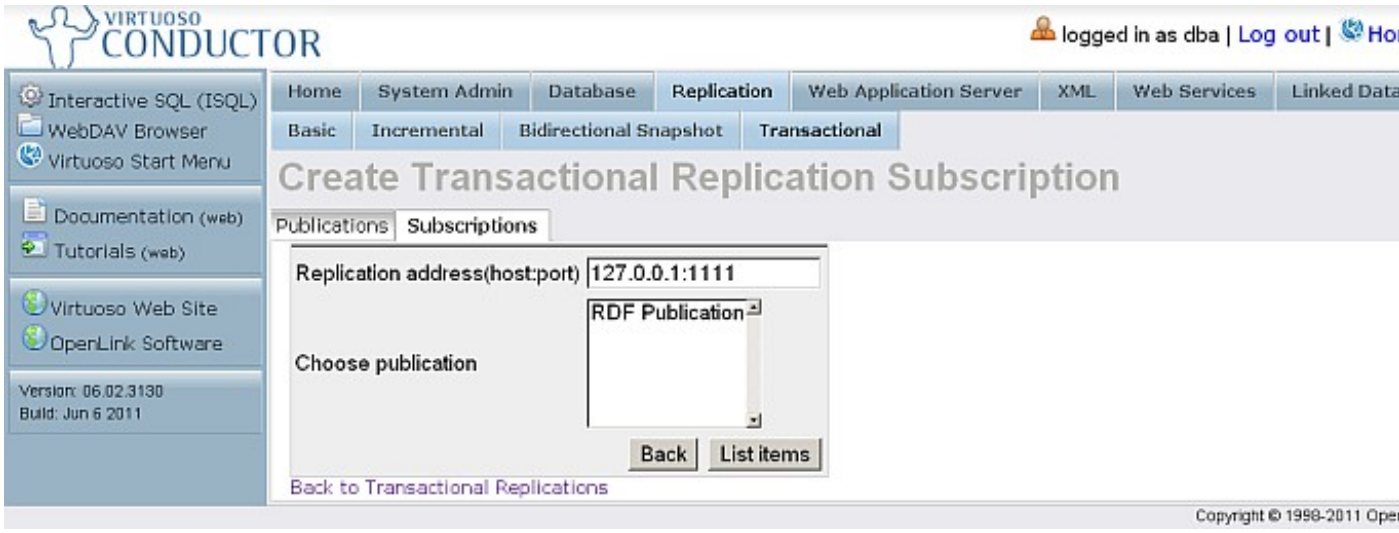


8. Select

   db1

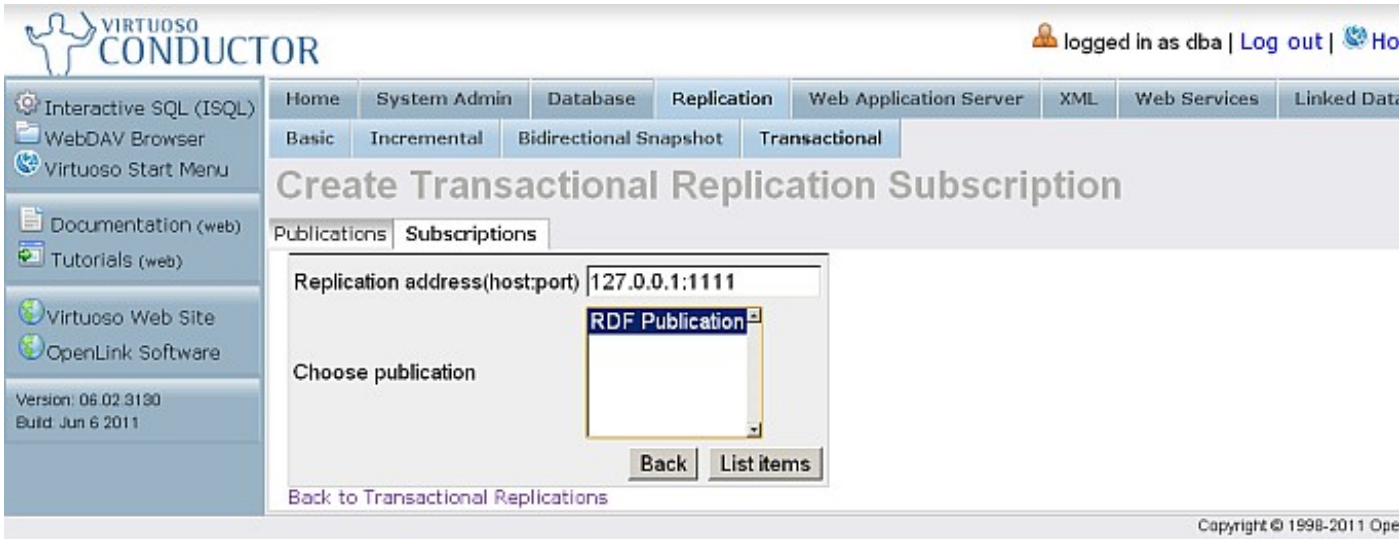
   the "Connected Data Sources" list and click "Publications list"

Figure 16.236. Chain Replication Topology



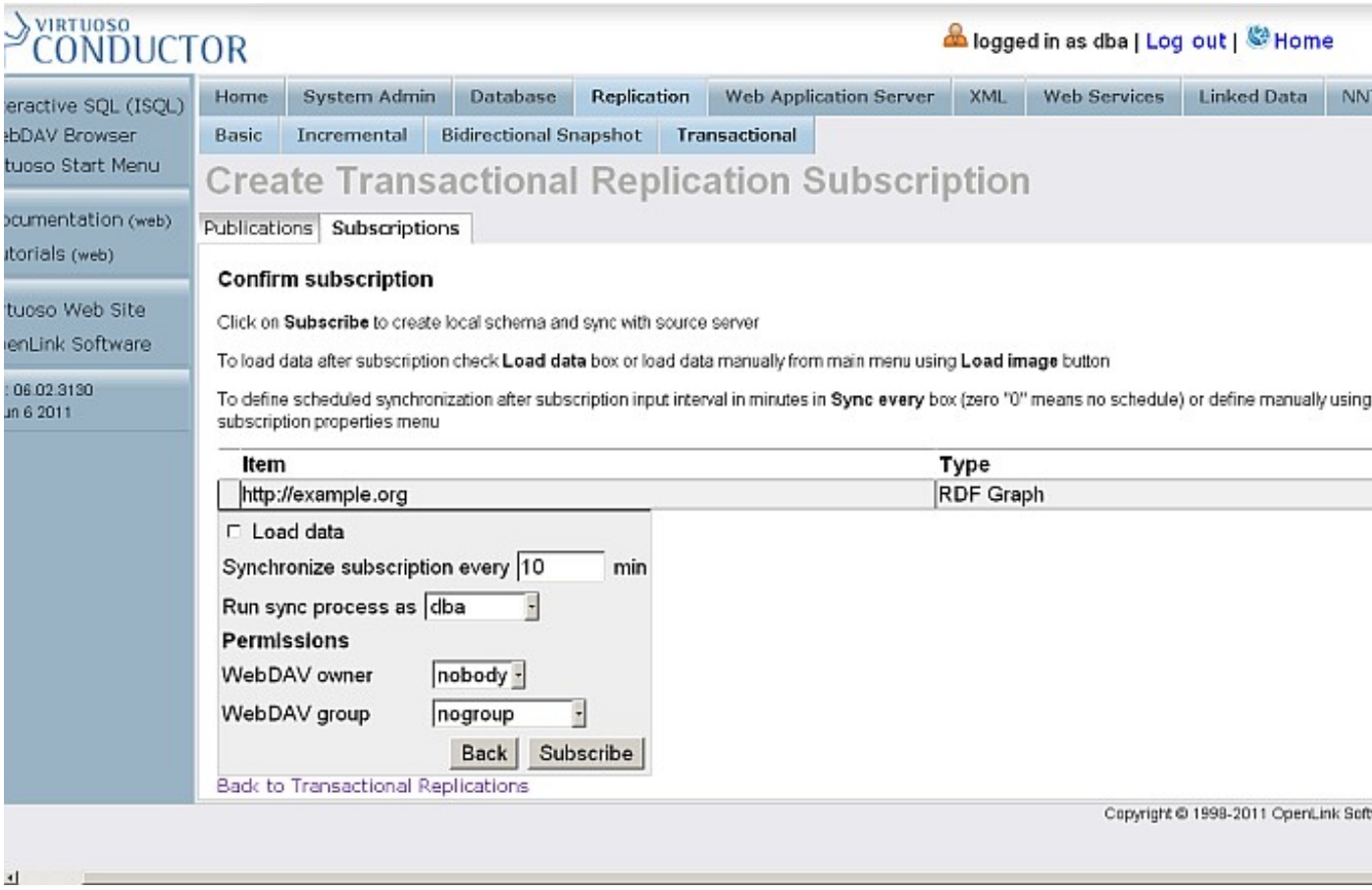
9. As result will be shown the list of available publications for the selected data source. Select the one with name "RDF Publication" and click "List Items".

Figure 16.237. Chain Replication Topology



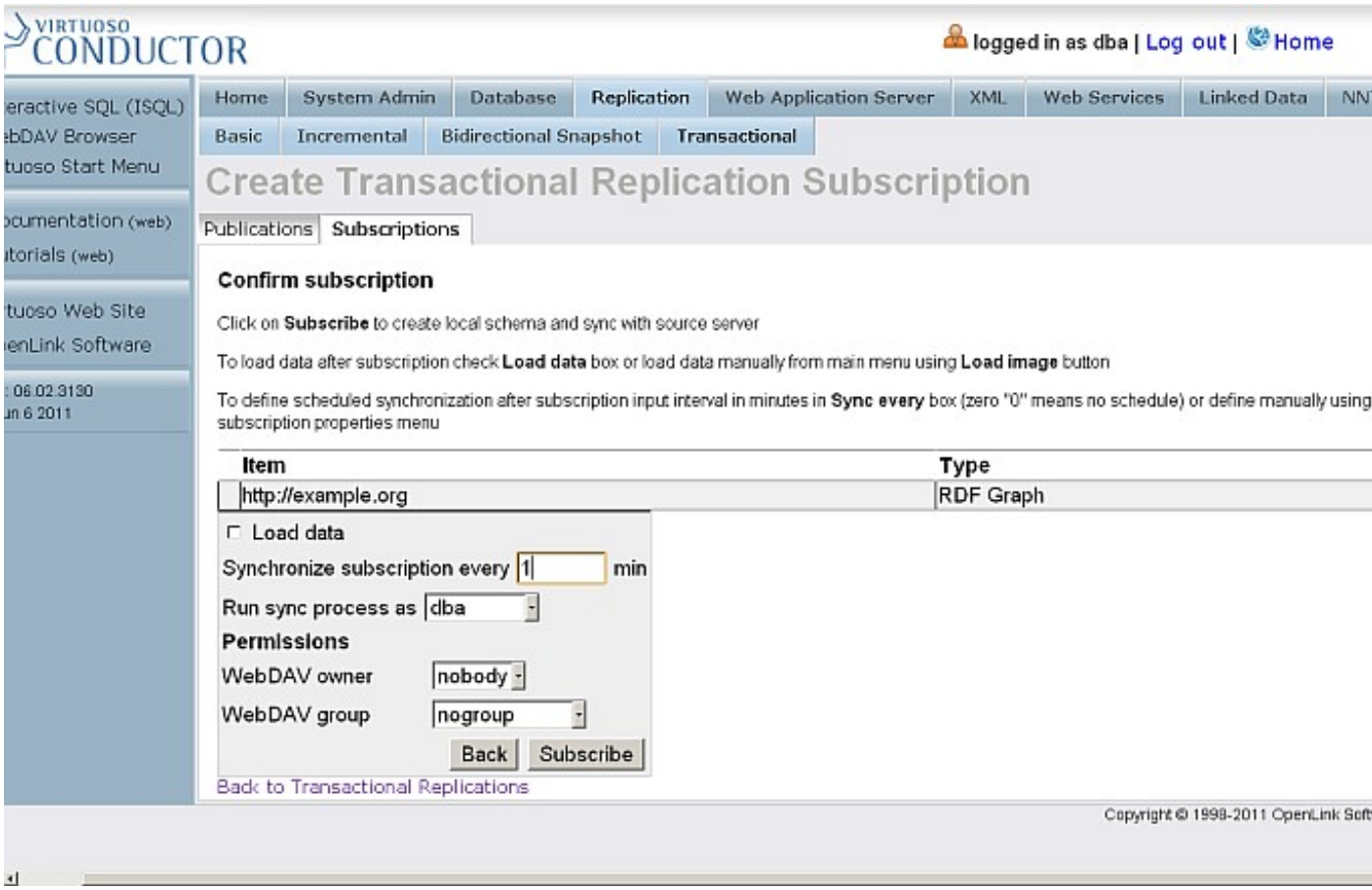
10. As result will be shown the "Confirm subscription" page.

Figure 16.238. Chain Replication Topology



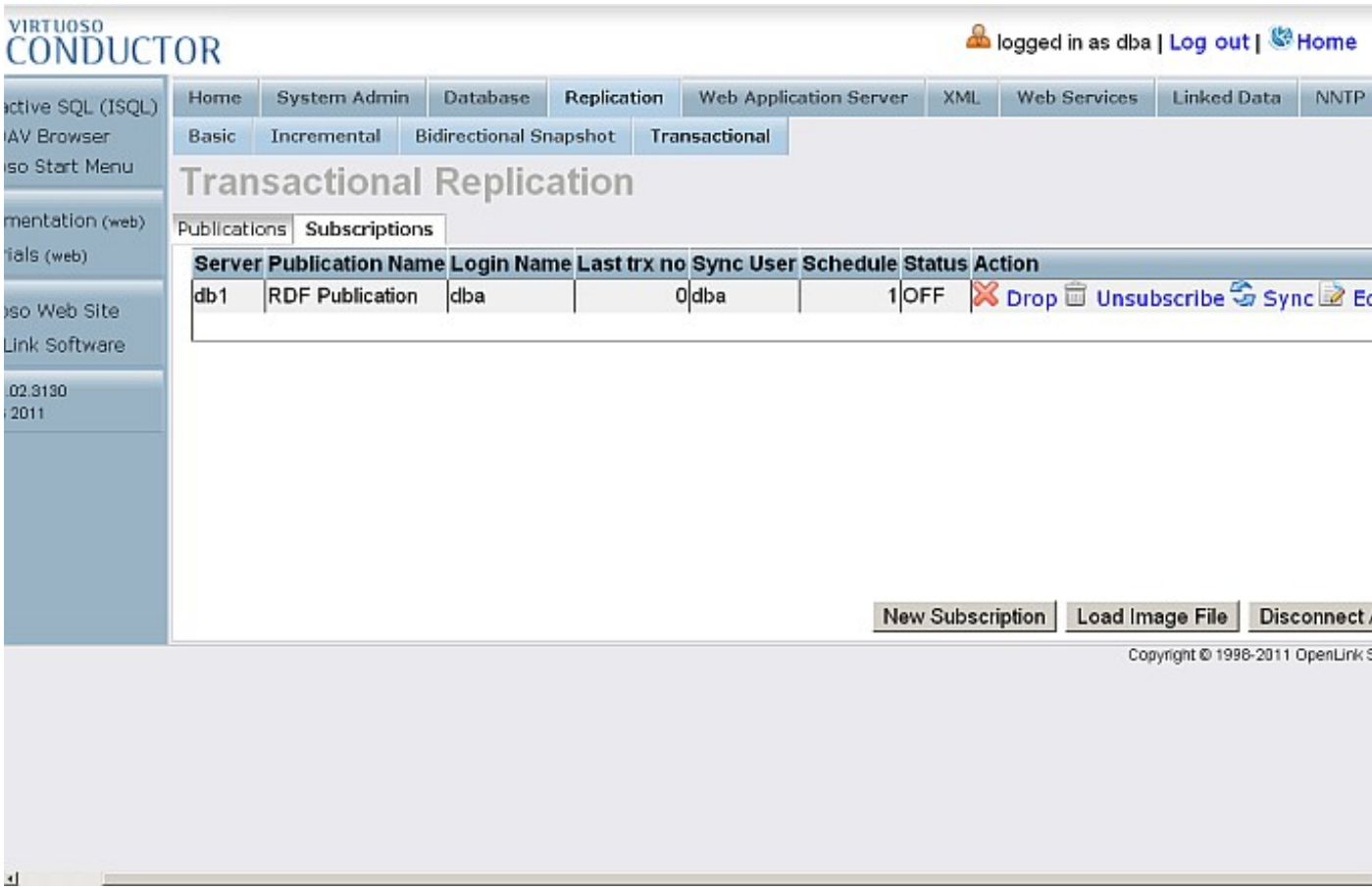
11. The sync interval by default is 10 minutes. For the testing purposes, we will change it to 1 minute.

Figure 16.239. Chain Replication Topology



12. Click "Subscribe"
13. The subscription will be created.

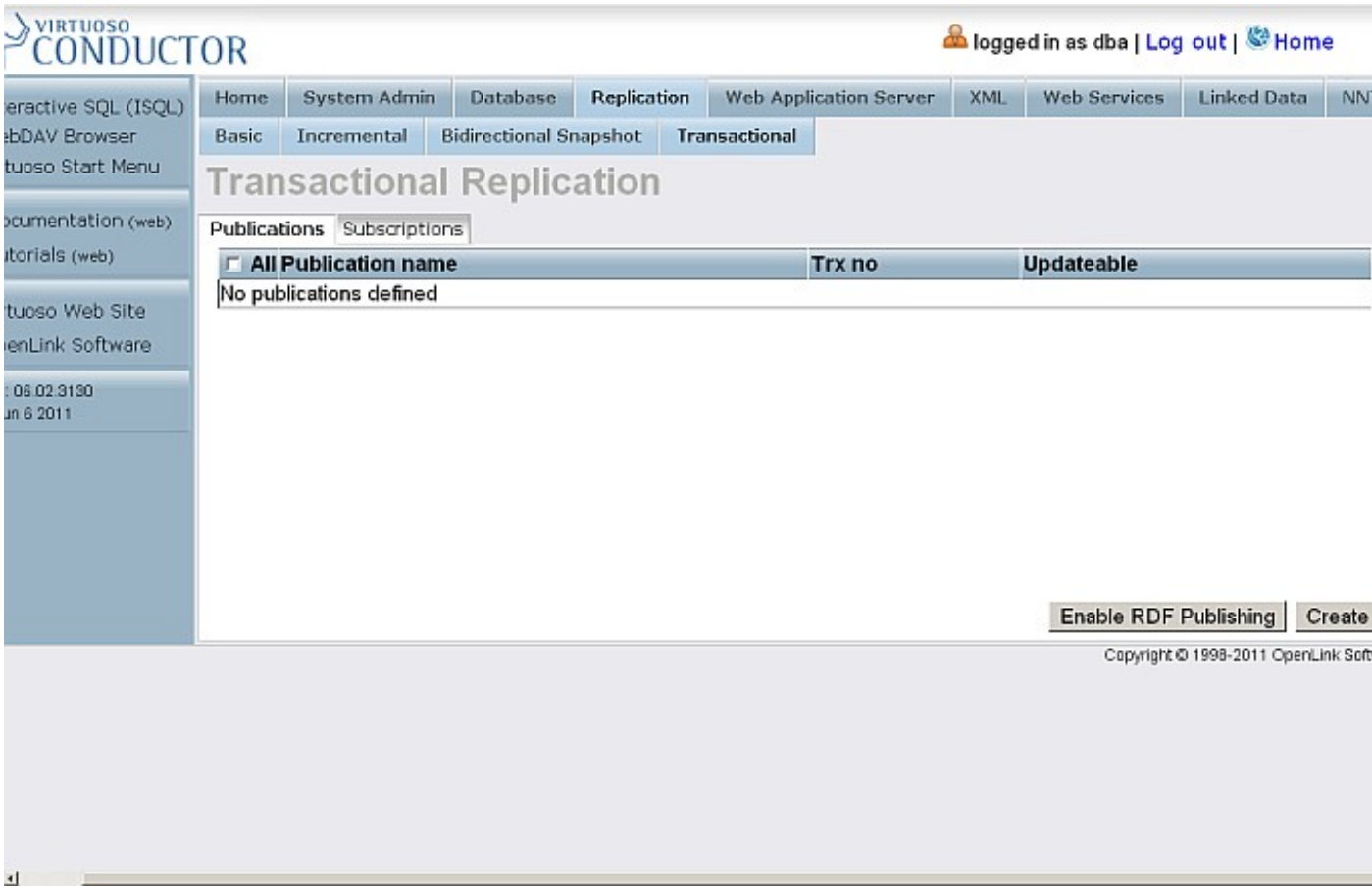
Figure 16.240. Chain Replication Topology



Create Publication on db2

1. Go to http://example.com:8892/conductor and log in as dba
2. Go to Conductor - > Replication - > Transactional - > Publications

Figure 16.241. Chain Replication Topology



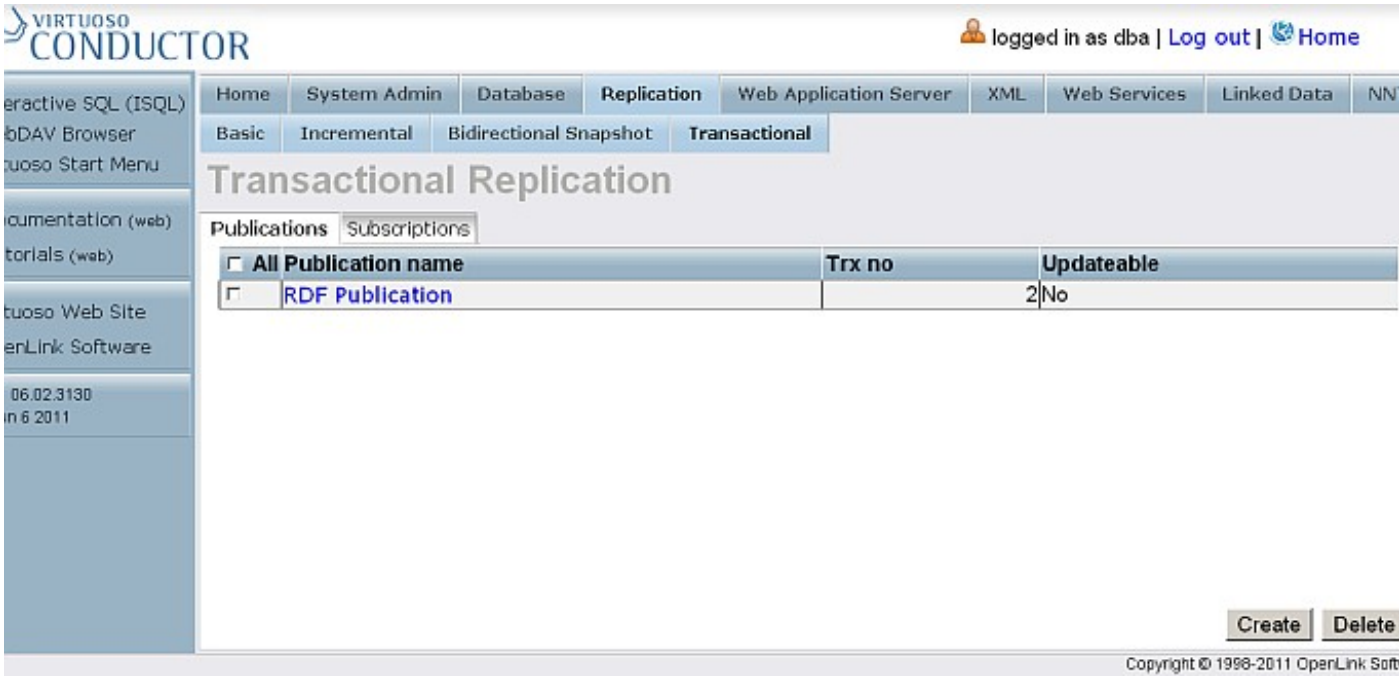
3. Click

   Enable RDF Publishing
4. As result publication with the name

   RDF Publication

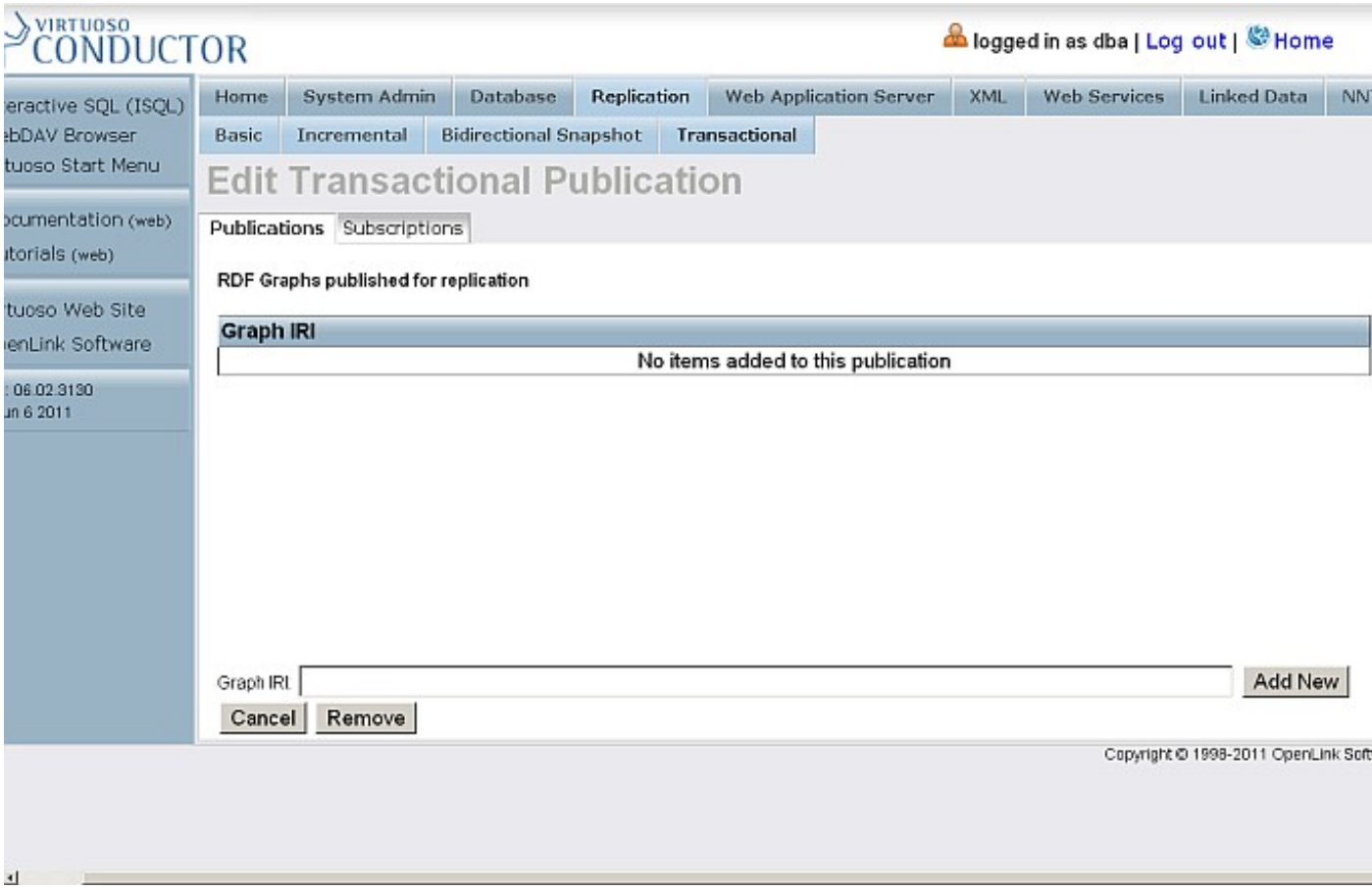
   should be created

Figure 16.242. Chain Replication Topology



5. Click the link which is the publication name.
6. You will be shown the publication items page

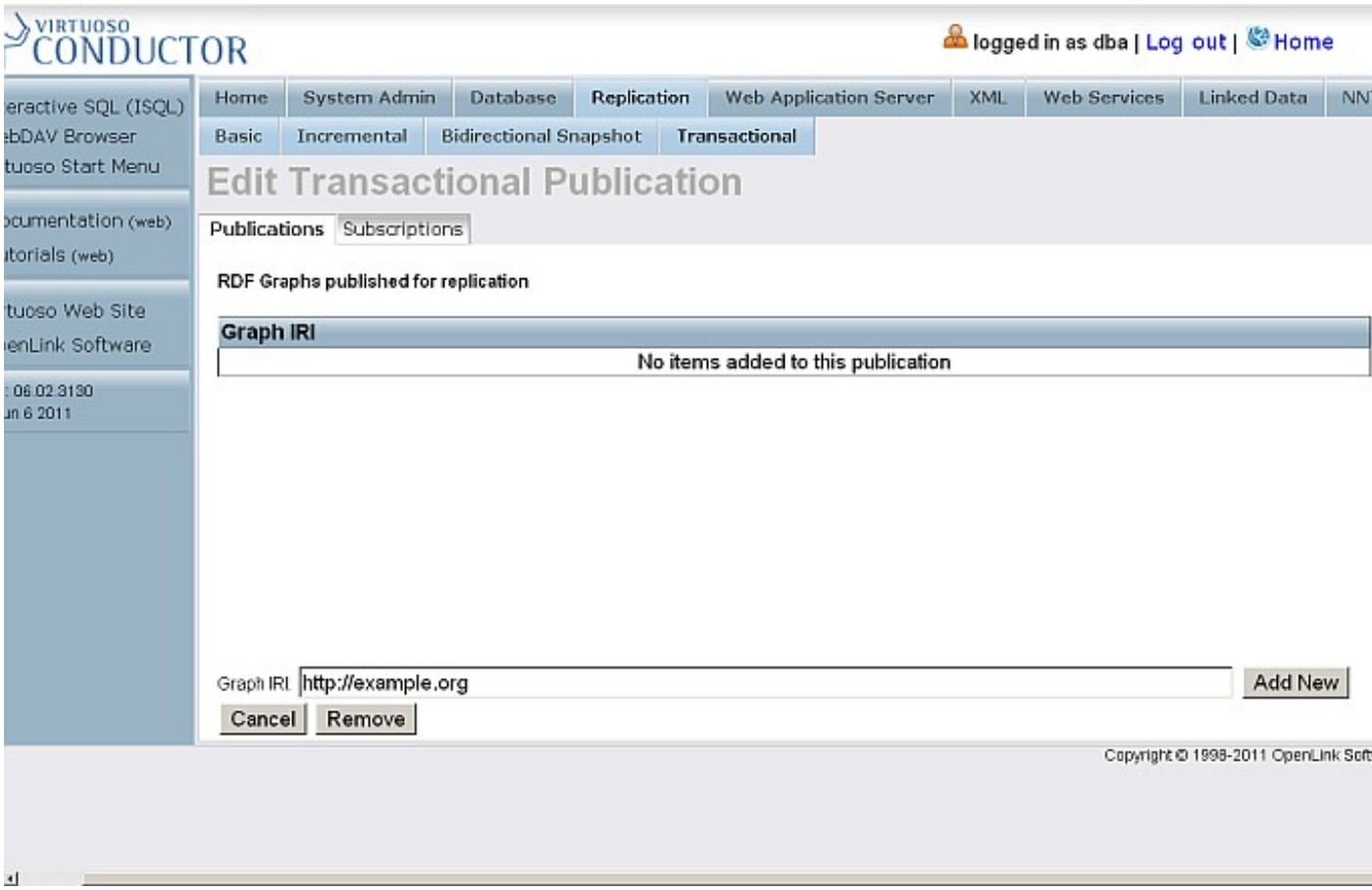
Figure 16.243. Chain Replication Topology



7. Enter for Graph IRI:

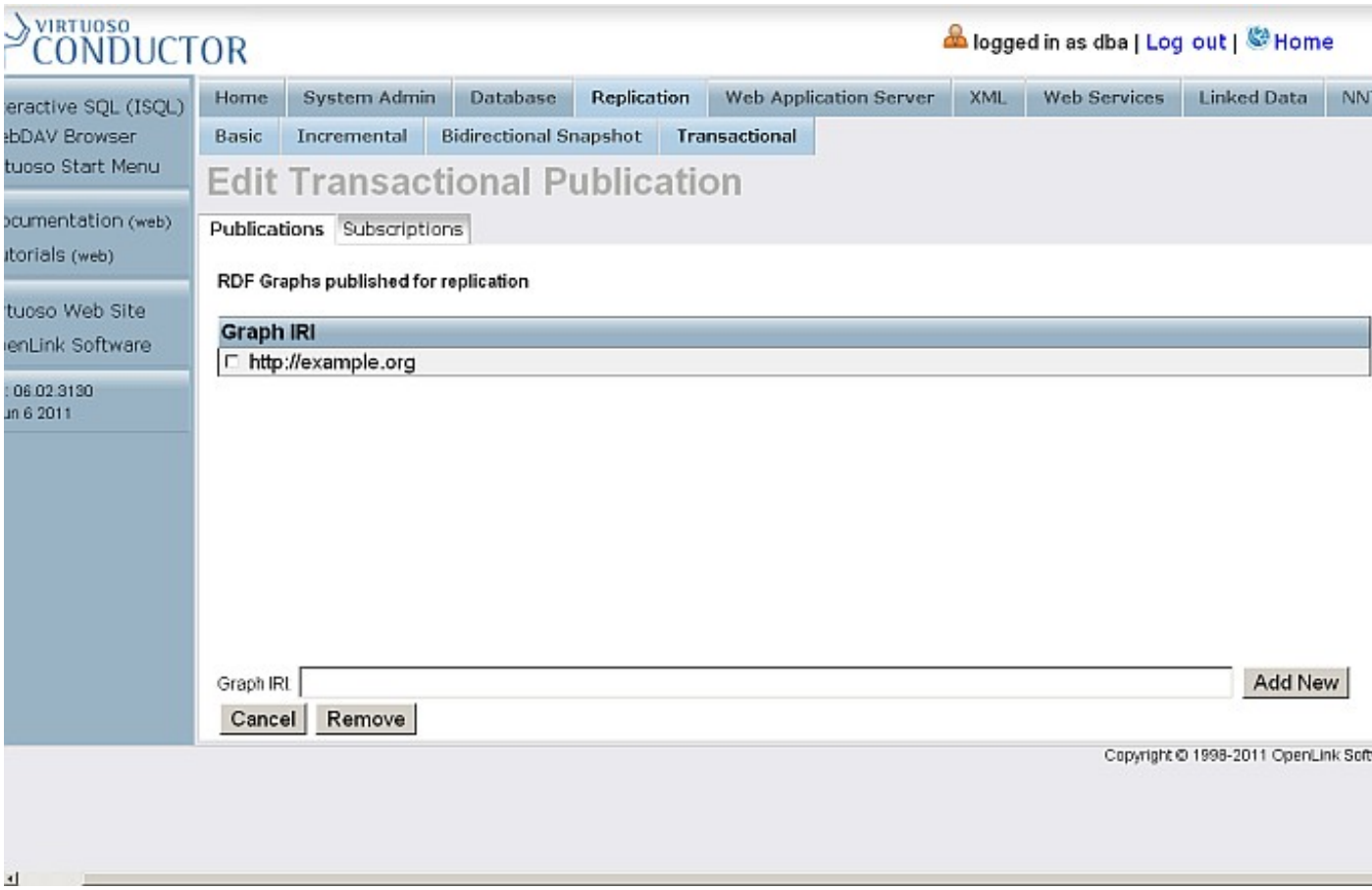
```
http://example.org
```

Figure 16.244. Chain Replication Topology



8. Click Add New
9. The item will be created and shown in the list of items for the currently viewed publication.

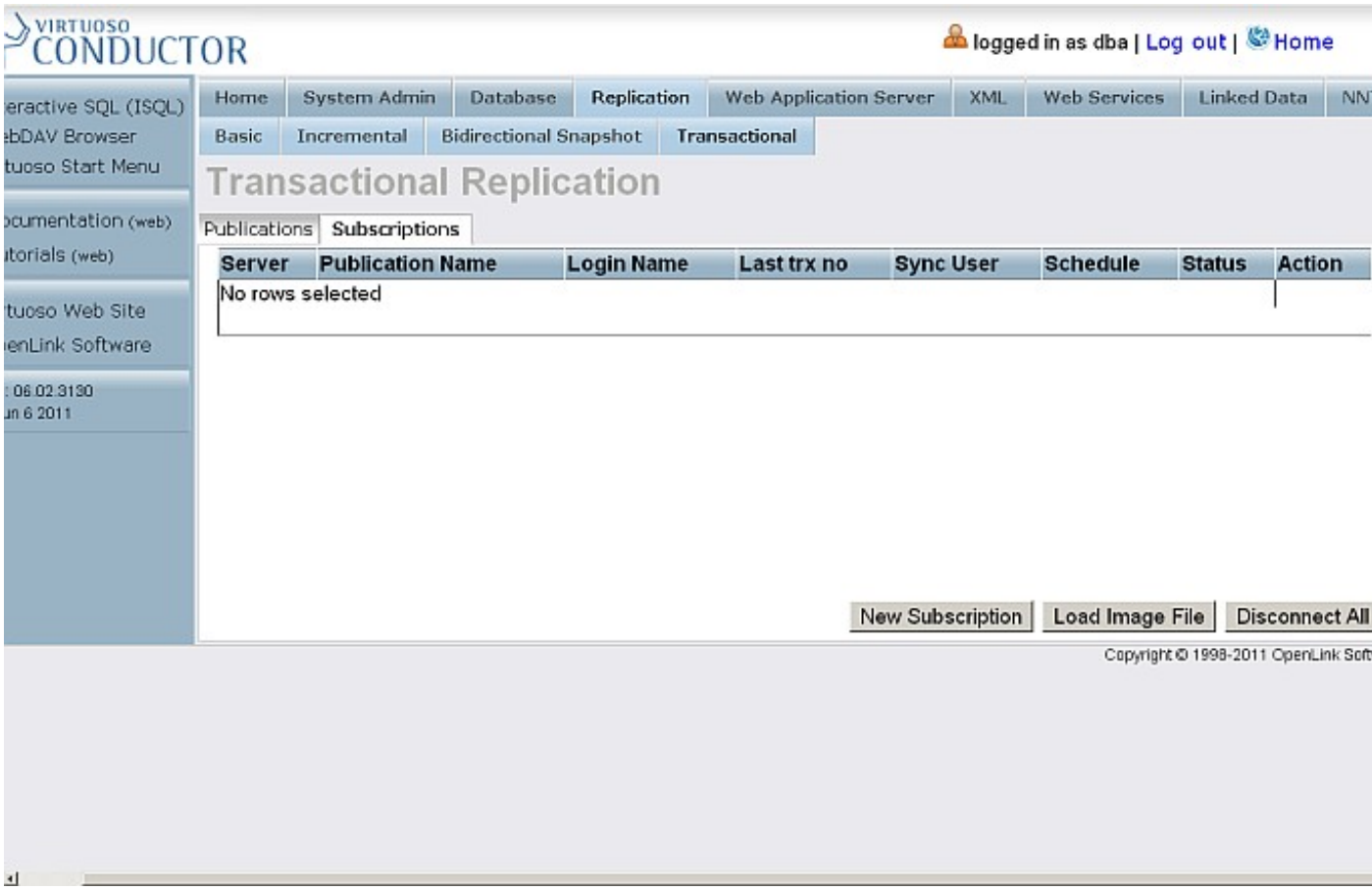
Figure 16.245. Chain Replication Topology



Create subscription from db3 to db2's Publication

1. Log in at http://example.com:8893/conductor
2. Go to Replication - > Transactional - > Subscriptions

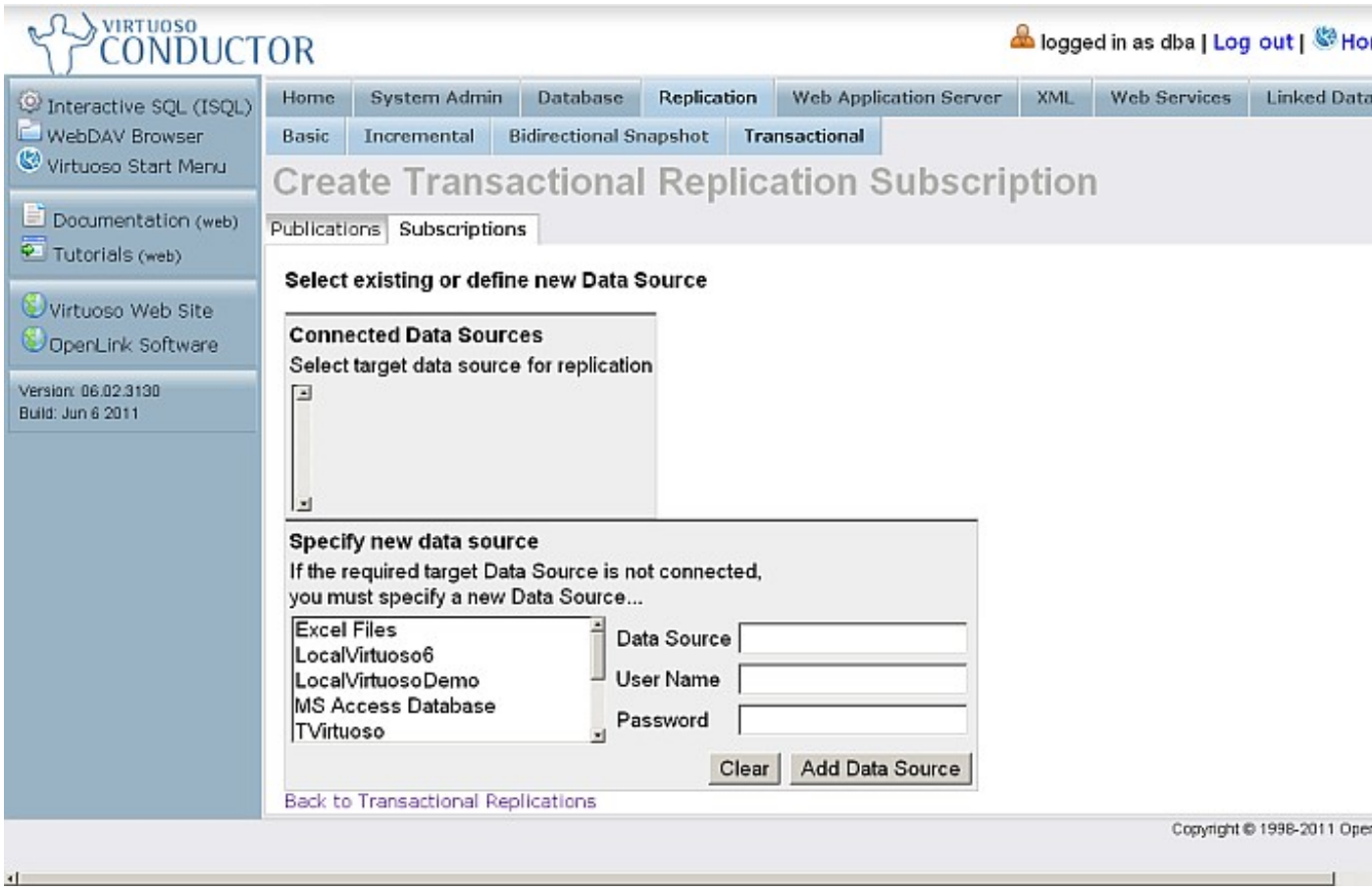
Figure 16.246. Chain Replication Topology



3. Click

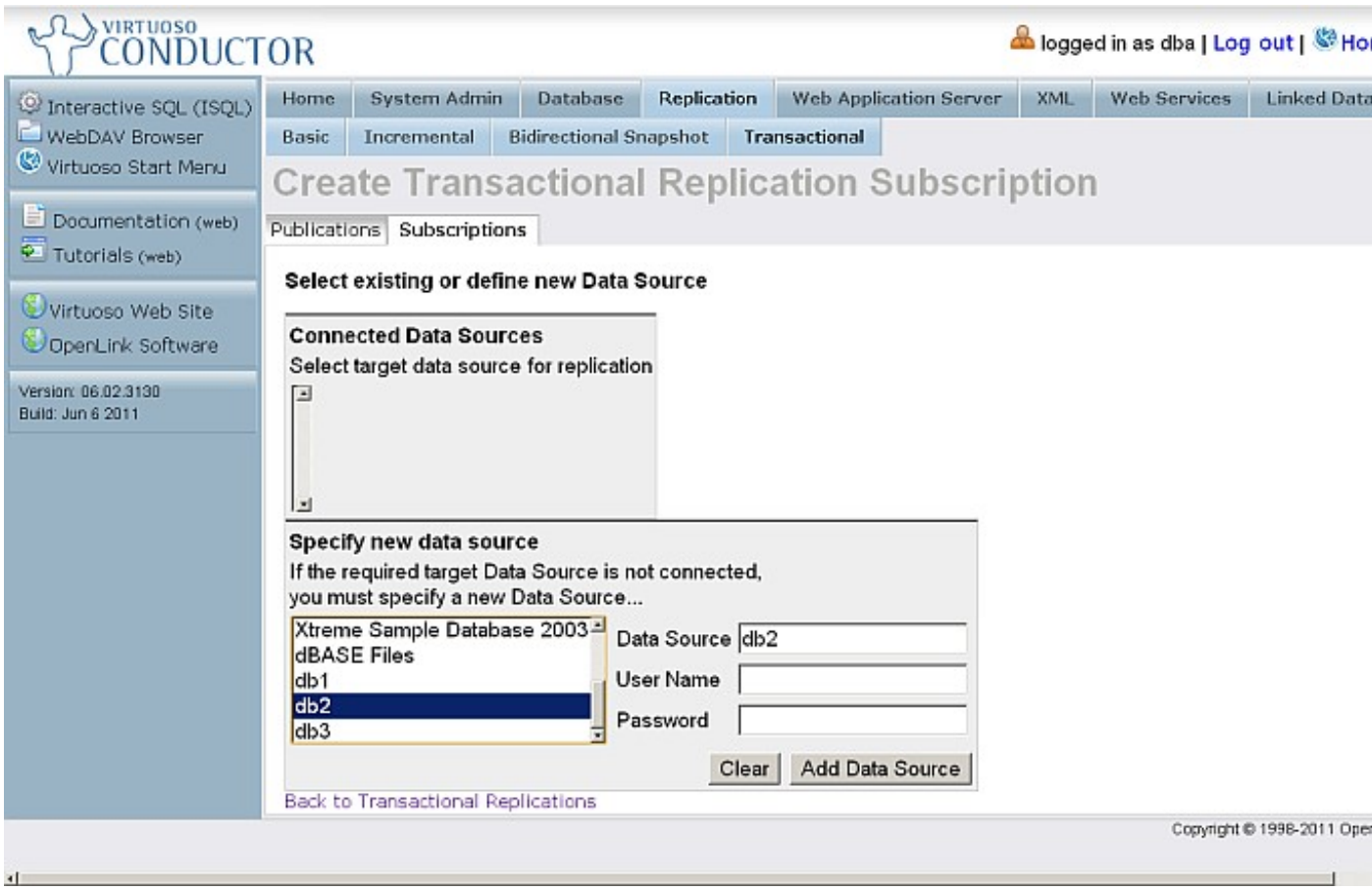
   New Subscription

Figure 16.247. Chain Replication Topology



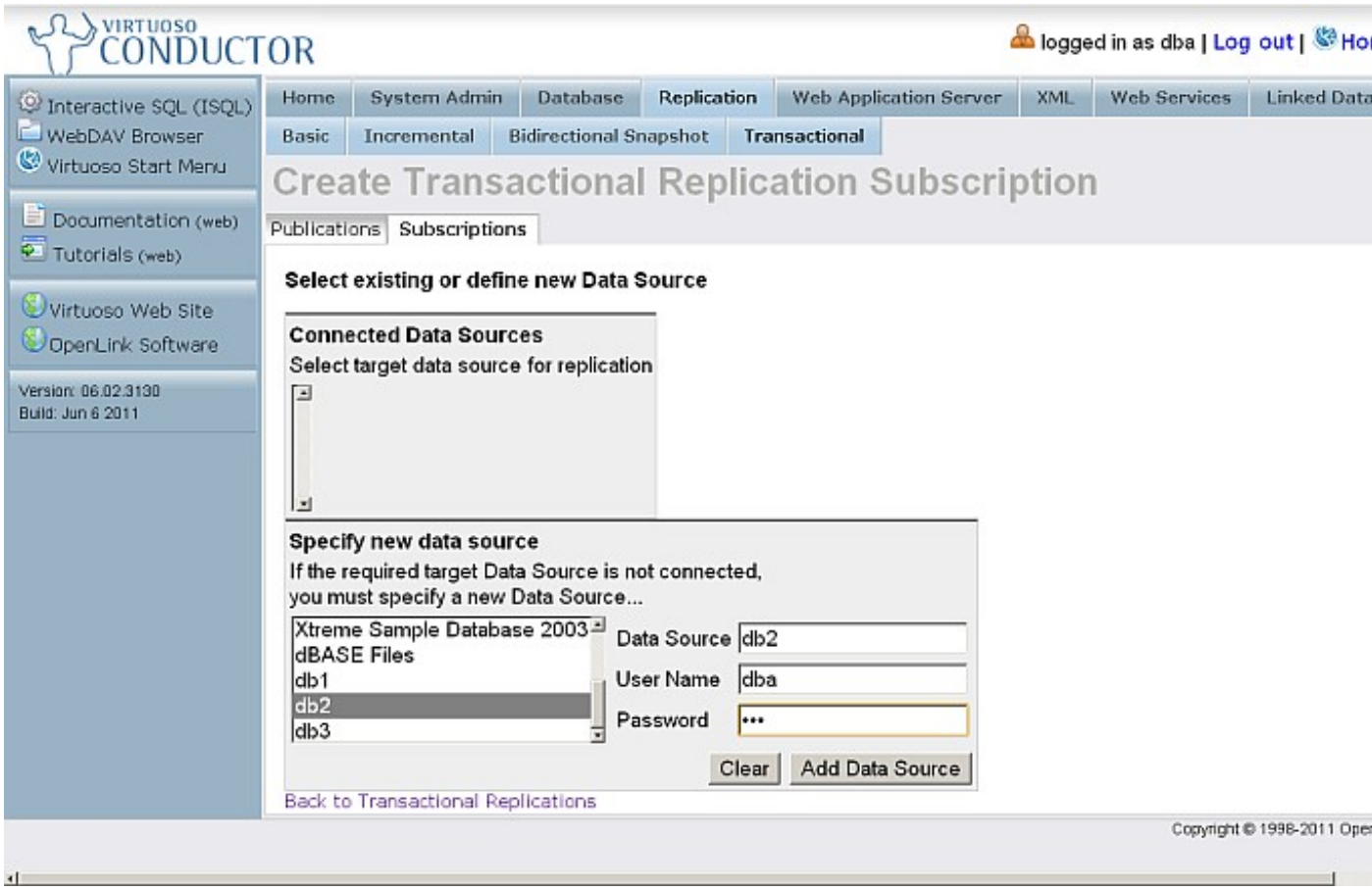
4. From the list of "Specify new data source" select Data Source db2

Figure 16.248. Chain Replication Topology



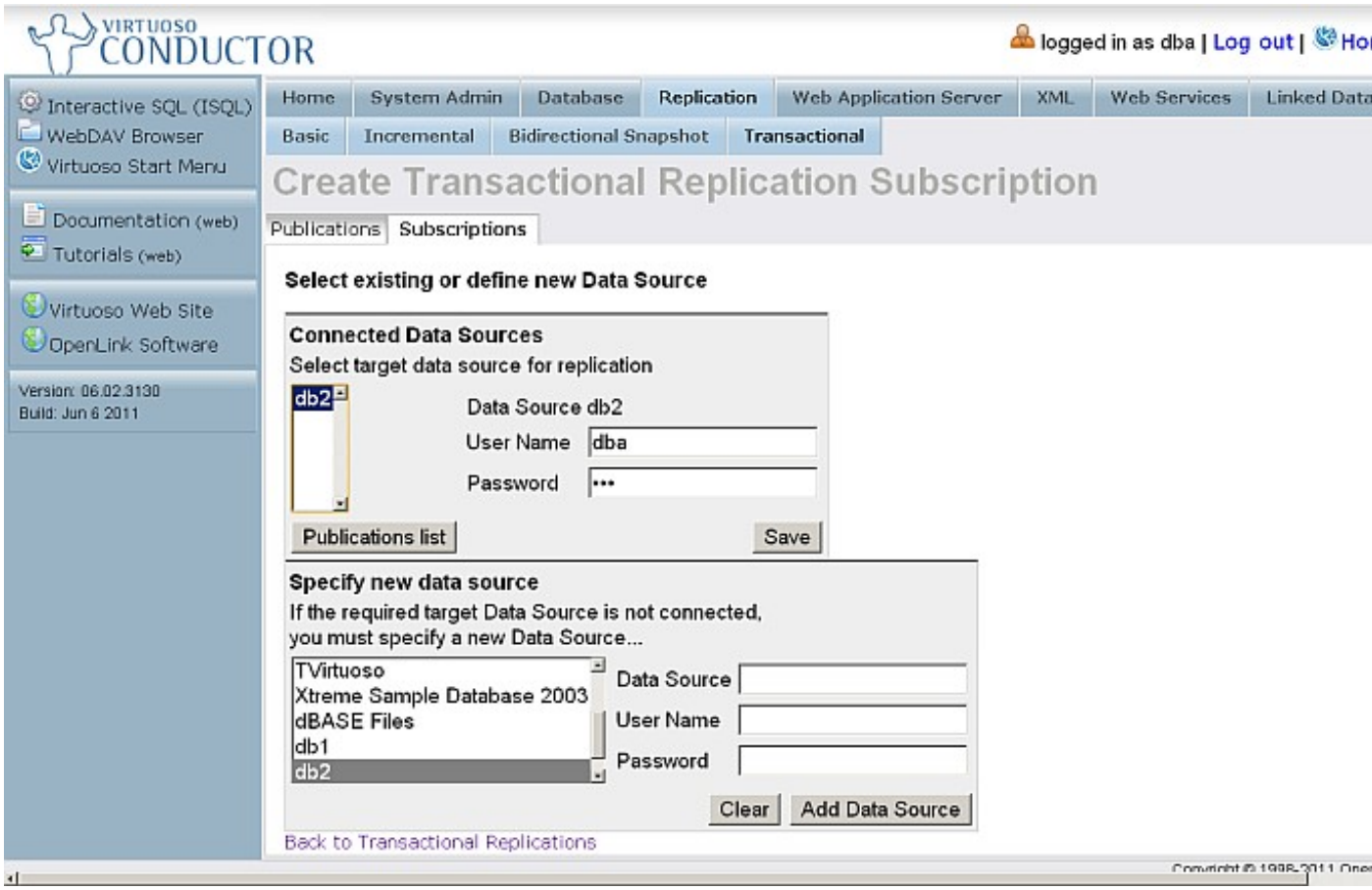
5. Enter for db2 dba user credentials

Figure 16.249. Chain Replication Topology



6. Click "Add Data Source"

Figure 16.250. Chain Replication Topology

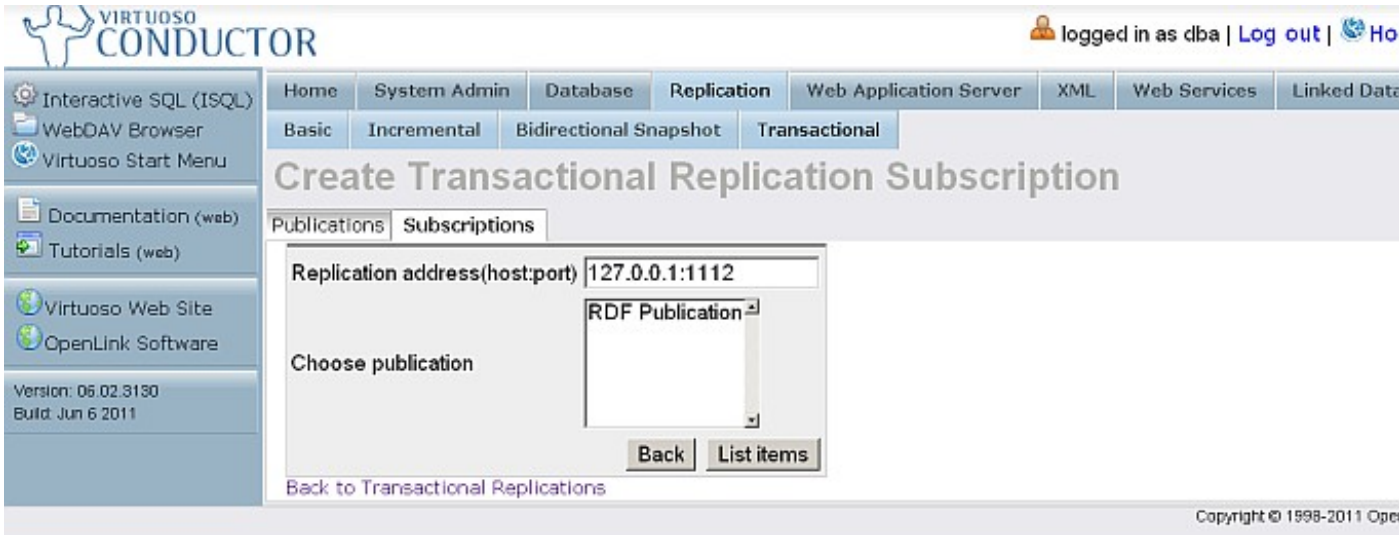


7. As result

   db2

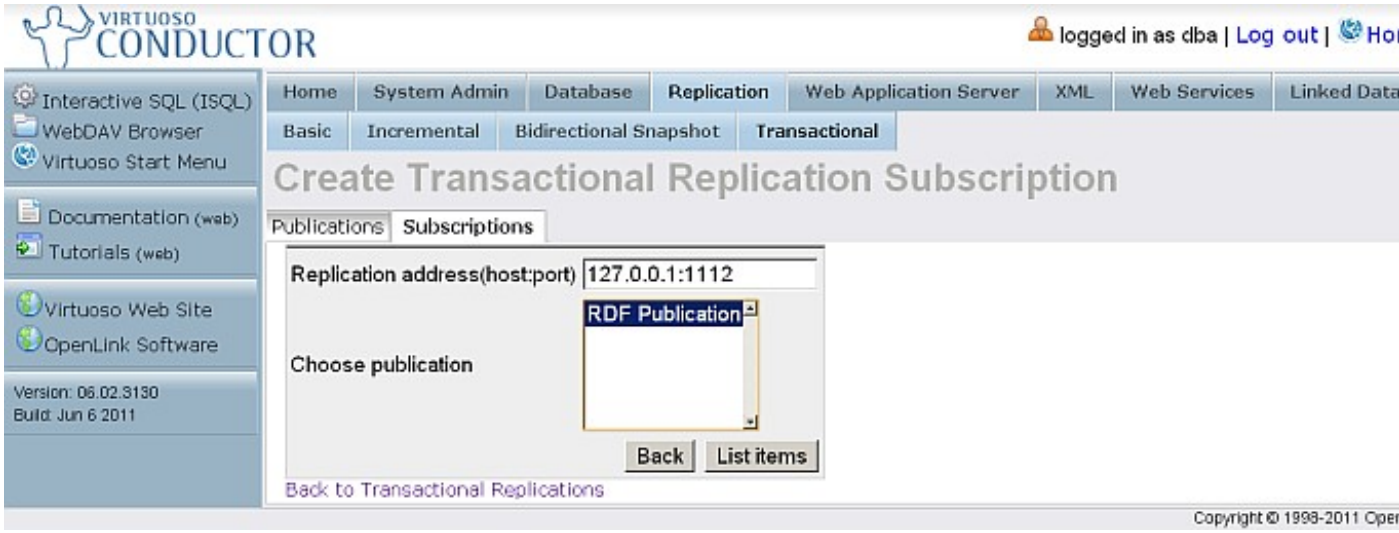
   will be shown in the "Connected Data Sources" list. Select it and click "Publications list"

Figure 16.251. Chain Replication Topology



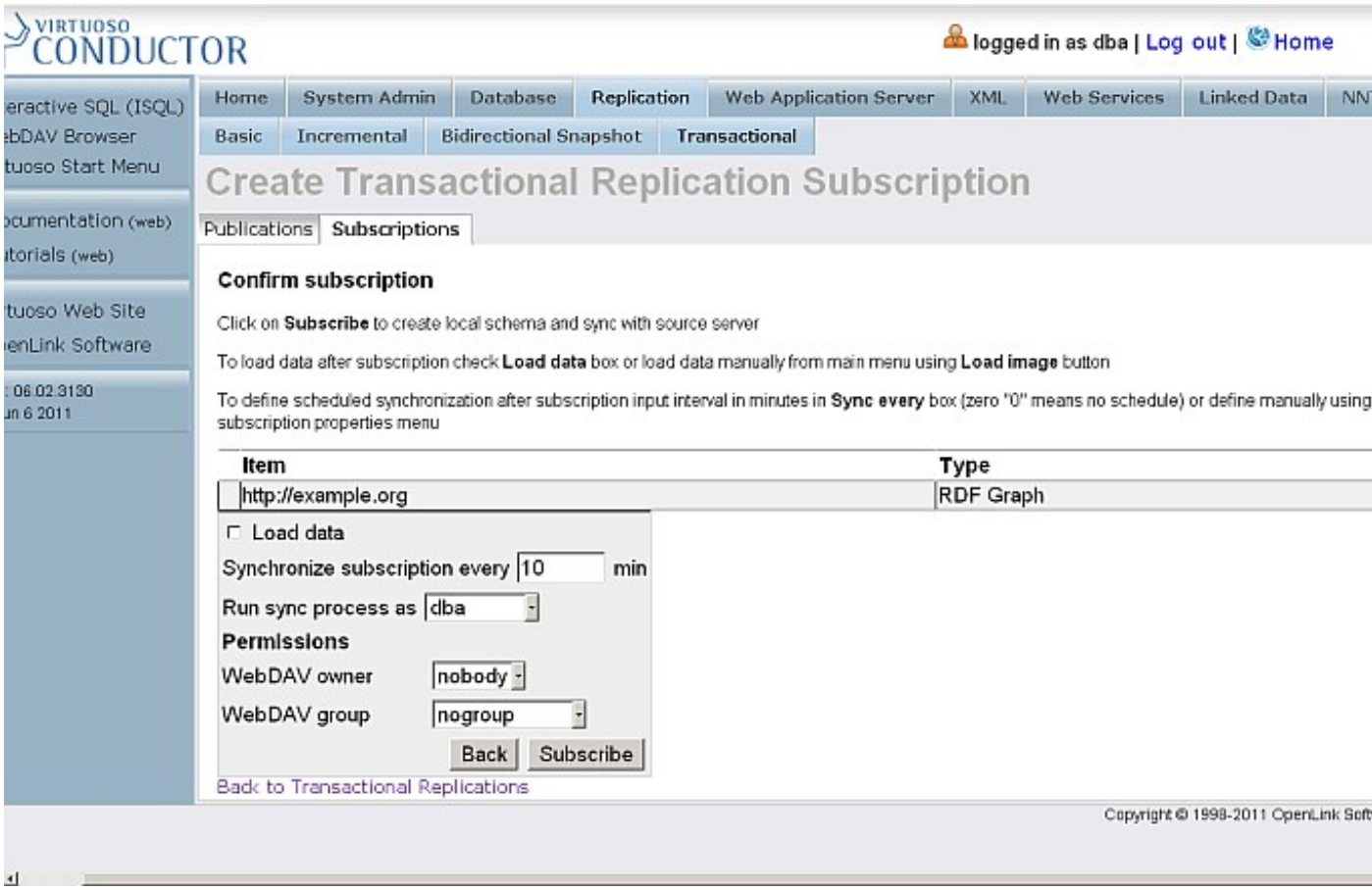
8. As result will be shown the list of available publications for the selected data source. Select the one with name "RDF Publication" and click "List Items".

Figure 16.252. Chain Replication Topology



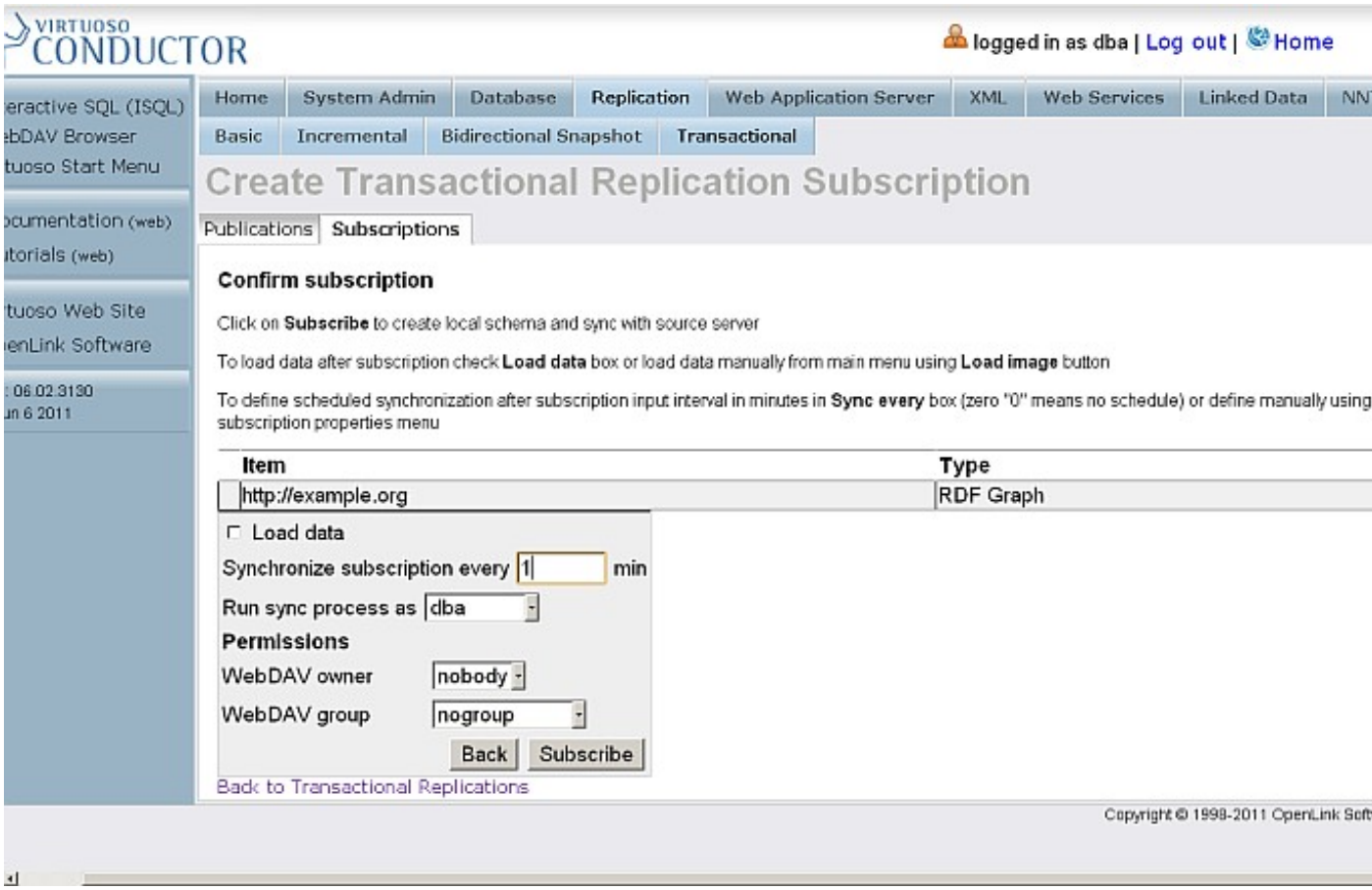
9. As result will be shown the "Confirm subscription" page.

Figure 16.253. Chain Replication Topology



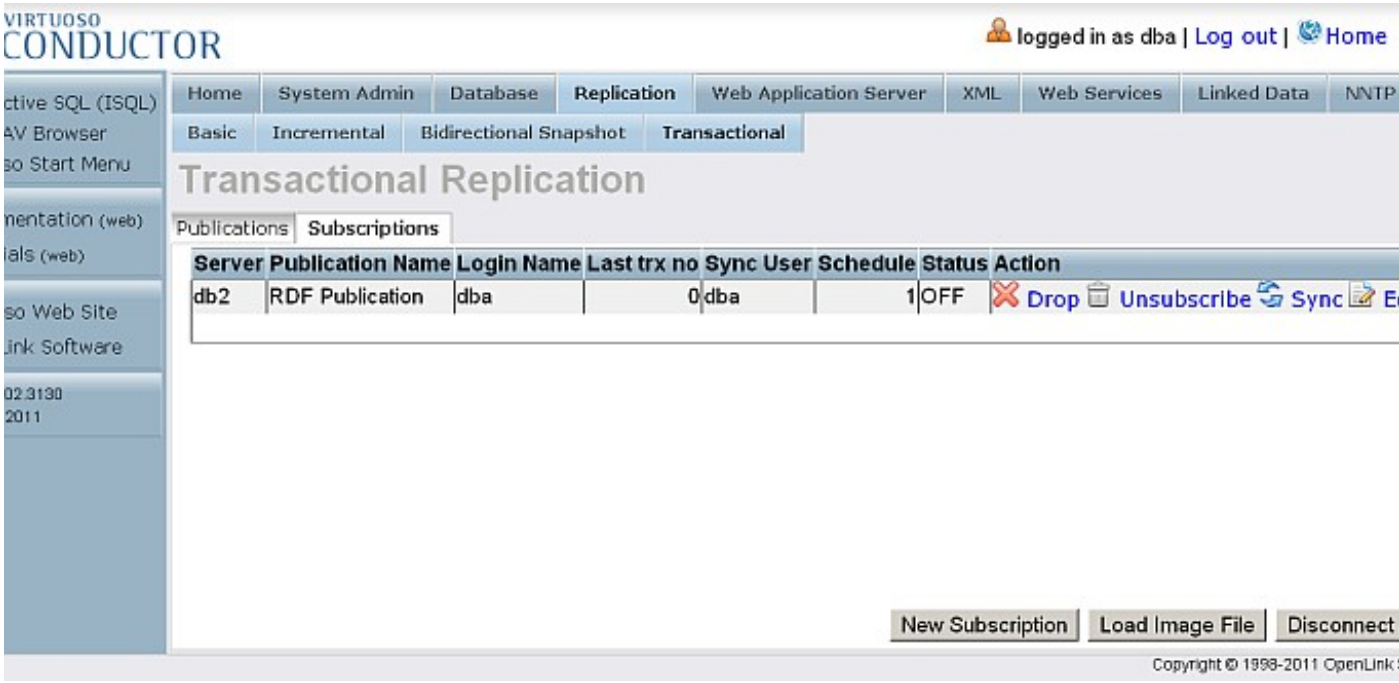
10. The sync interval by default is 10 minutes. For the testing purposes, we will change it to 1 minute.

Figure 16.254. Chain Replication Topology



11. Click "Subscribe"
12. The subscription will be created.

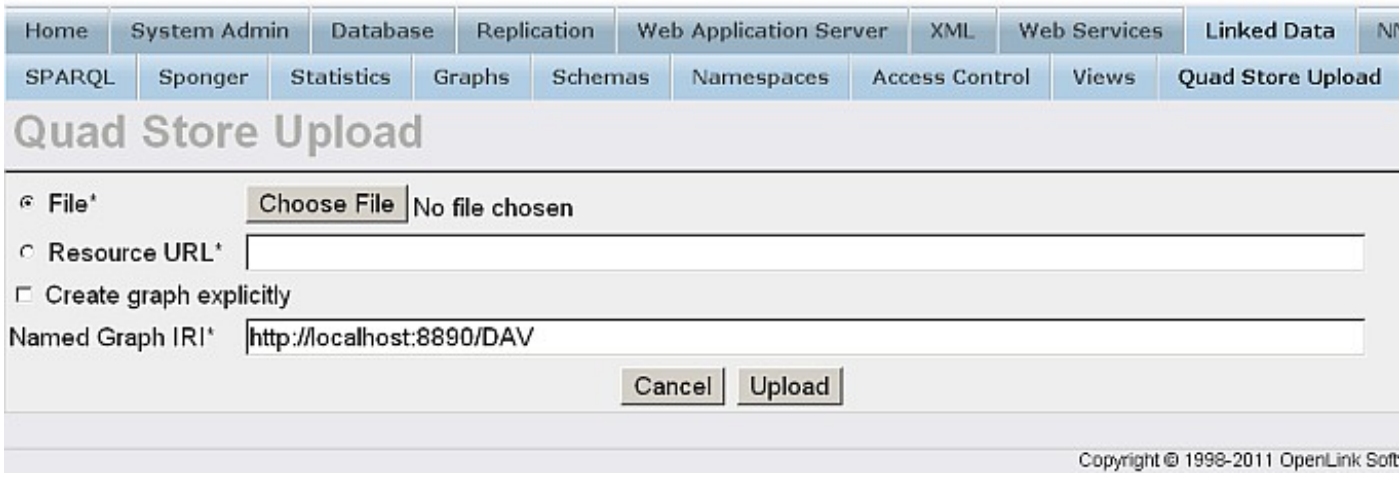
Figure 16.255. Chain Replication Topology



Insert Data into a Named Graph on the db1 Virtuoso Instance

1. Log in at http://example.com:8891/conductor
2. Go to Linked Data -> Quad Store Upload:

Figure 16.256. Chain Replication Topology



3. In the shown form:

   a. Tick the box for

   Resource URL

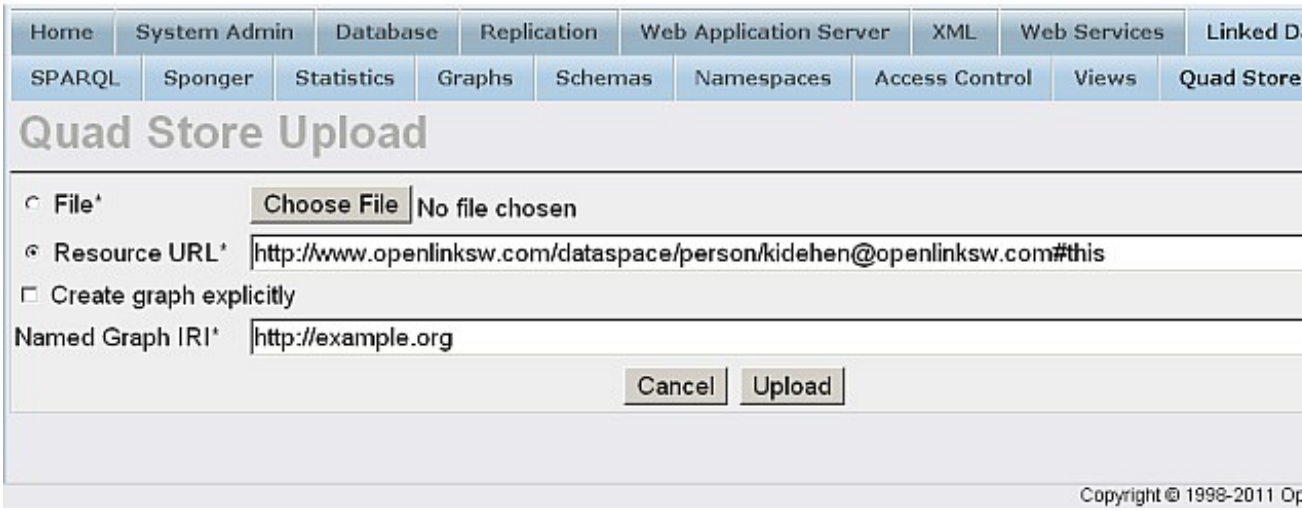
   and enter your resource URL, e.g.:

   ```
   http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#this
   ```

   b. Enter for Named Graph IRI:

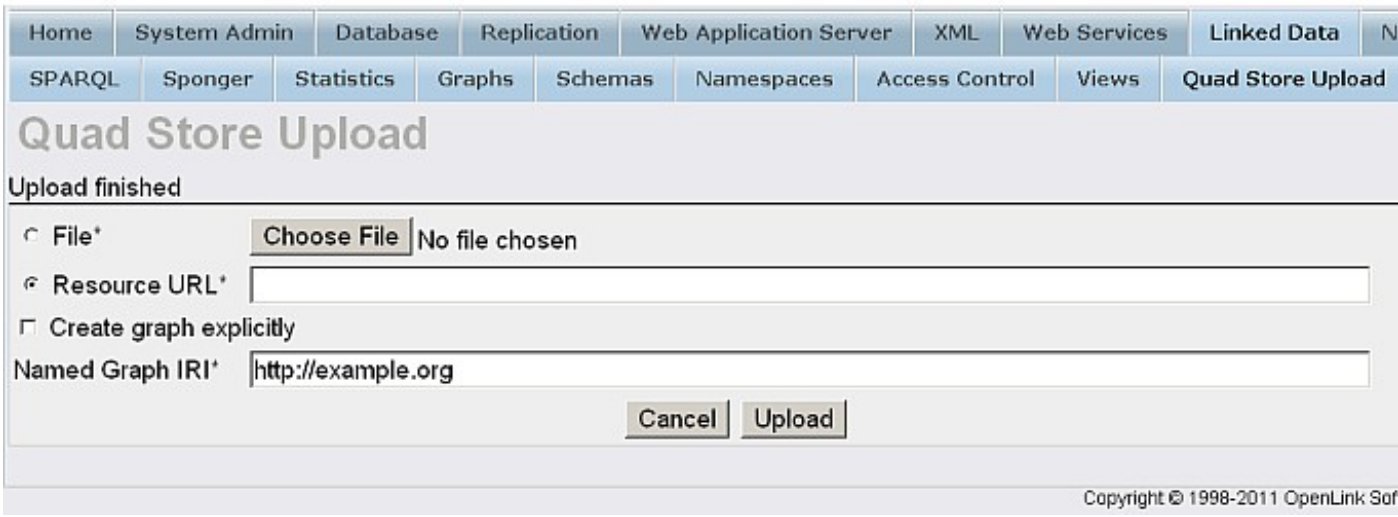
   ```
   http://example.org
   ```

   Figure 16.257. Chain Replication Topology



4. Click Upload
5. A successful upload will result in a shown message.

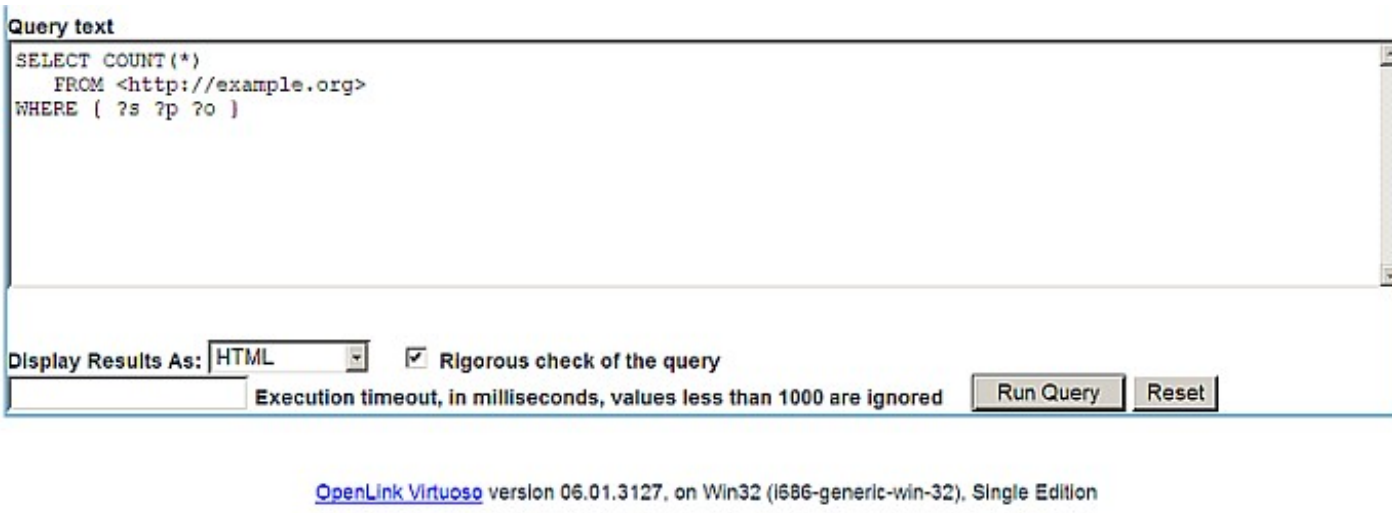
Figure 16.258. Chain Replication Topology



6. Check the count of the inserted triples by executing a query like the following against the SPARQL endpoint, http://example.com:8891/sparql:

```
SELECT COUNT(*)
   FROM <http://example.org>
WHERE { ?s ?p ?o }
```

Figure 16.259. Chain Replication Topology



7. Should return

   57

   as total.

Figure 16.260. Chain Replication Topology

callret-0
57

Check data on the Destination instances db2 and db3

1. To check the starting count, on each of the Destination Virtuoso Instances db2 and db3 from SPARQL Endpoint execute:

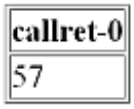
```
SELECT COUNT(*)
   FROM <http://example.org>
WHERE { ?s ?p ?o }
```

2. Should return

   57

   as total.

Figure 16.261. Chain Replication Topology

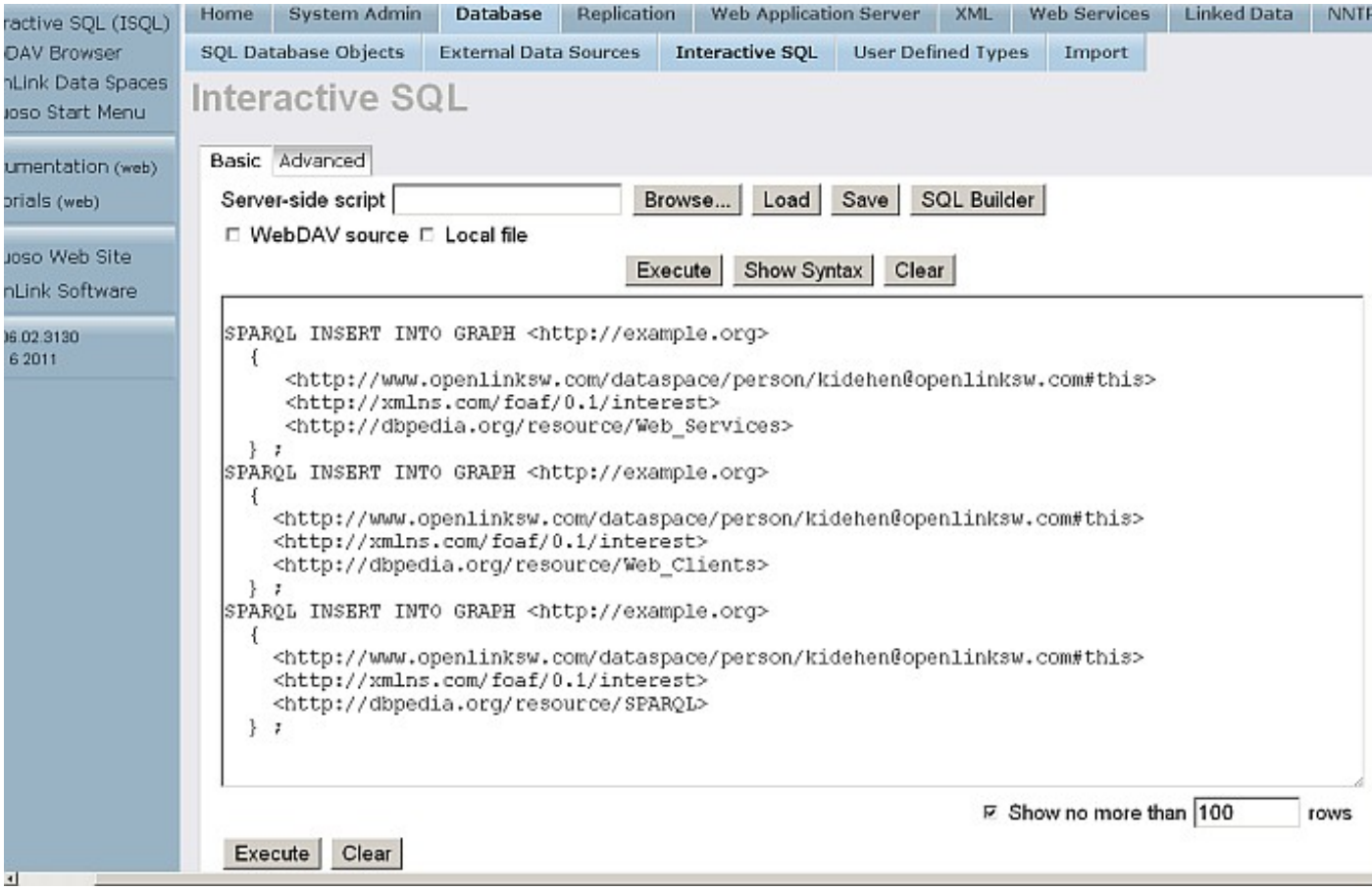


Add new data on db1

1. Disconnect db2 and db3.
2. On the Host Virtuoso Instance db1 go to Conductor - > Database - > Interactive SQL enter the following statement:

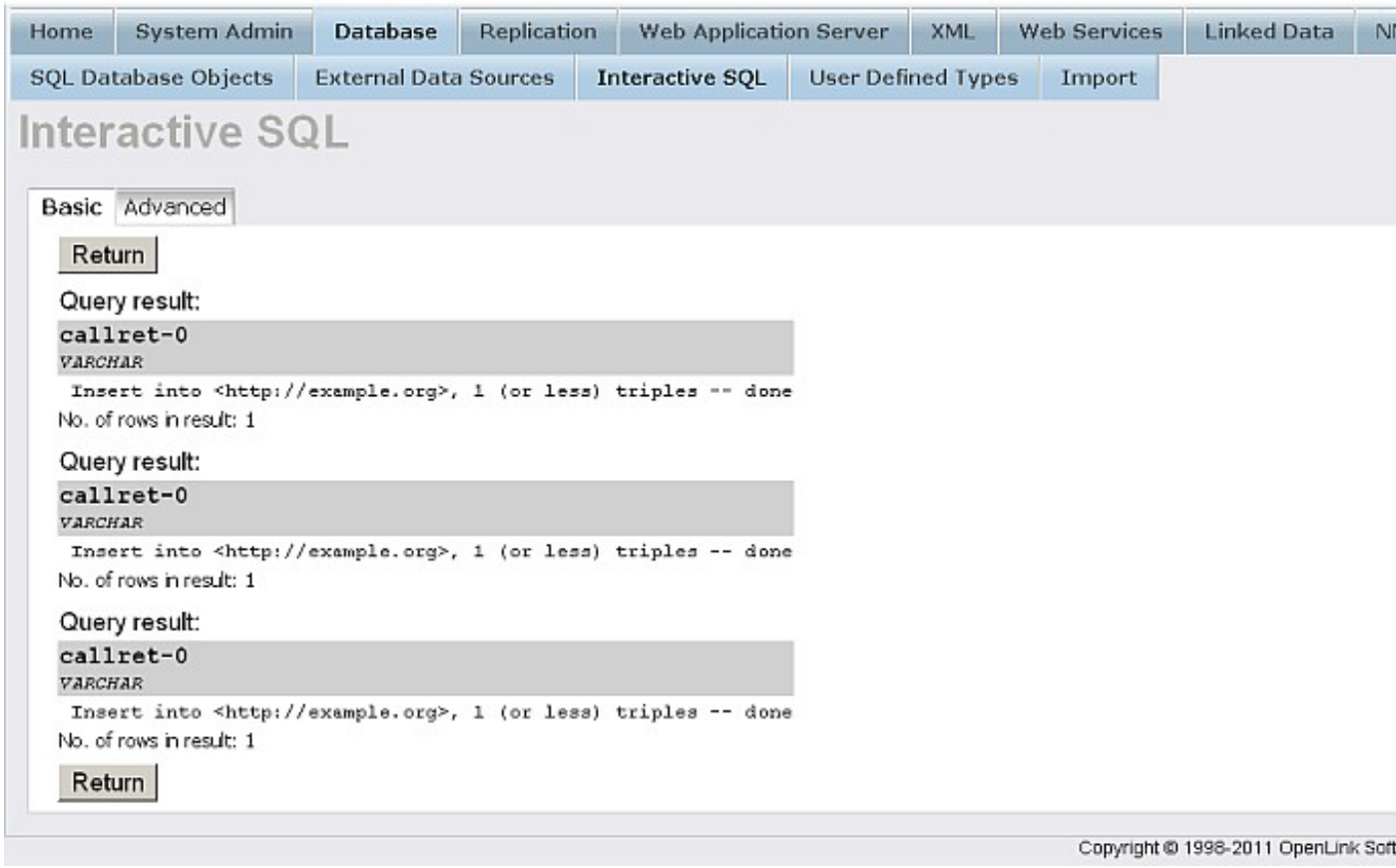
```
SPARQL INSERT INTO GRAPH <http://example.org>
  {
     <http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#this>
     <http://xmlns.com/foaf/0.1/interest>
     <http://dbpedia.org/resource/Web_Services>
  } ;
SPARQL INSERT INTO GRAPH <http://example.org>
  {
    <http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#this>
    <http://xmlns.com/foaf/0.1/interest>
    <http://dbpedia.org/resource/Web_Clients>
  } ;
SPARQL INSERT INTO GRAPH <http://example.org>
  {
    <http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#this>
    <http://xmlns.com/foaf/0.1/interest>
    <http://dbpedia.org/resource/SPARQL>
  } ;
```

Figure 16.262. Chain Replication Topology



3. Click "Execute"
4. As result the triples will be inserted

Figure 16.263. Chain Replication Topology



5. Check the count of the destination instance graph's triples by executing the following query like against the SPARQL endpoint, http://example.com:8891/sparql:

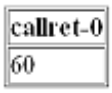
```
SELECT COUNT(*)
   FROM <http://example.org>
WHERE { ?s ?p ?o }
```

6. Should return

   60

   as total.

Figure 16.264. Chain Replication Topology



Check data on the Destination instances db2 and db3

1. Start instances db2 and db3
2. To confirm that the triple count has increased by the number of inserted triples, execute the following on the Destination Virtuoso Instance db2 and db3 SPARQL Endpoint:

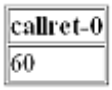
```
SELECT COUNT(*)
   FROM <http://example.org>
WHERE { ?s ?p ?o }
```

3. Should return

   60

   as total.

Figure 16.265. Chain Replication Topology



Bi-directional Replication Topology

Bi-directional Replication Topology Example

The following How-To walks you through setting up Virtuoso RDF Graph Replication in a Bi-directional Topology.

```
db1 <---- db2
db1 ---->

 db2
```

Prerequisites

Database INI Parameters

Suppose there are 2 Virtuoso instances respectively with the following ini parameters values:

1. virtuoso1.ini:

```
...
[Database]
DatabaseFile    = virtuoso1.db
TransactionFile = virtuoso1.trx
```

```
ErrorLogFile     = virtuoso1.log
...
[Parameters]
ServerPort               = 1111
SchedulerInterval        = 1
...
[HTTPServer]
ServerPort                  = 8891
...
[URIQA]
DefaultHost = example.com:8891
...
[Replication]
ServerName   = db1
...
```

2. virtuoso2.ini:

```
...
[Database]
DatabaseFile    = virtuoso2.db
TransactionFile = virtuoso2.trx
ErrorLogFile     = virtuoso2.log
...
[Parameters]
ServerPort               = 1112
SchedulerInterval        = 1
...
[HTTPServer]
ServerPort                  = 8892
...
[URIQA]
DefaultHost = localhost:8892
...
[Replication]
ServerName   = db2
...
```

Database DSNs

Use the ODBC Administrator on your Virtuoso host (e.g., on Windows, Start menu -> Control Panel -> Administrative Tools -> Data Sources (ODBC); on Mac OS X, /Applications/Utilities/OpenLink ODBC Administrator.app) to create a System DSN for db1 and db2 with names db1 and db2 respectively.

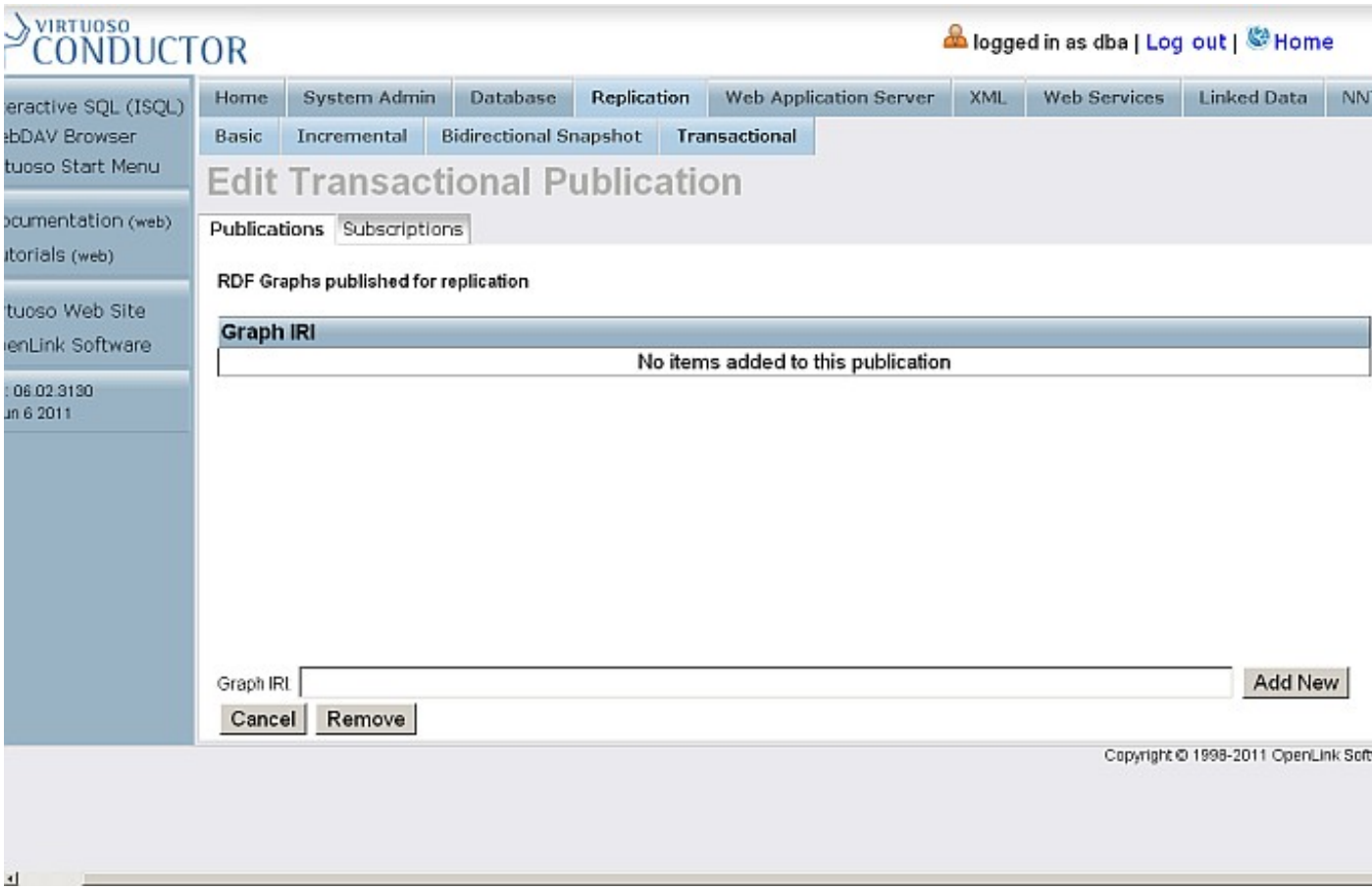
Install Conductor package

On each of the 2 Virtuoso instances install the conductor_dav.vad package.

Create Publication on db2

1. Go to http://example.com:8892/conductor and log in as dba
2. Go to Conductor -> Replication -> Transactional -> Publications

Figure 16.266. Bi-directional Replication Topology



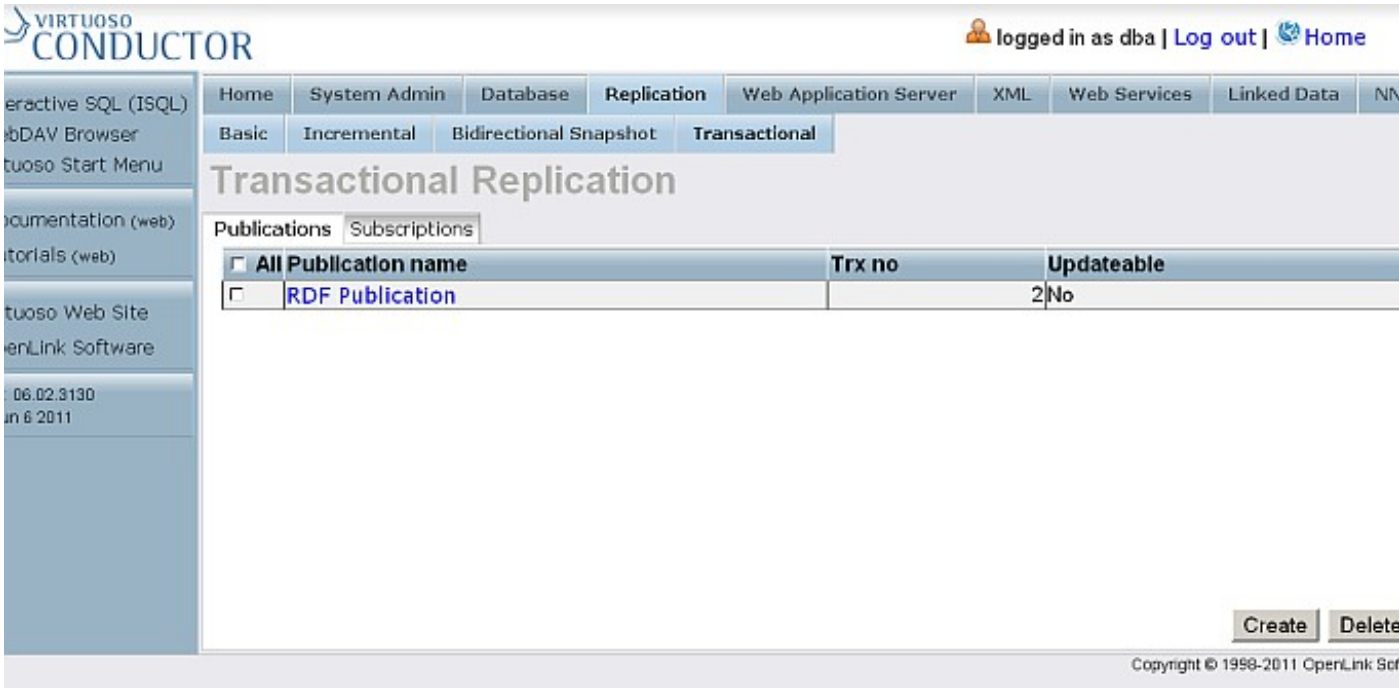
3. Click

   Enable RDF Publishing
4. As result publication with the name

   RDF Publication

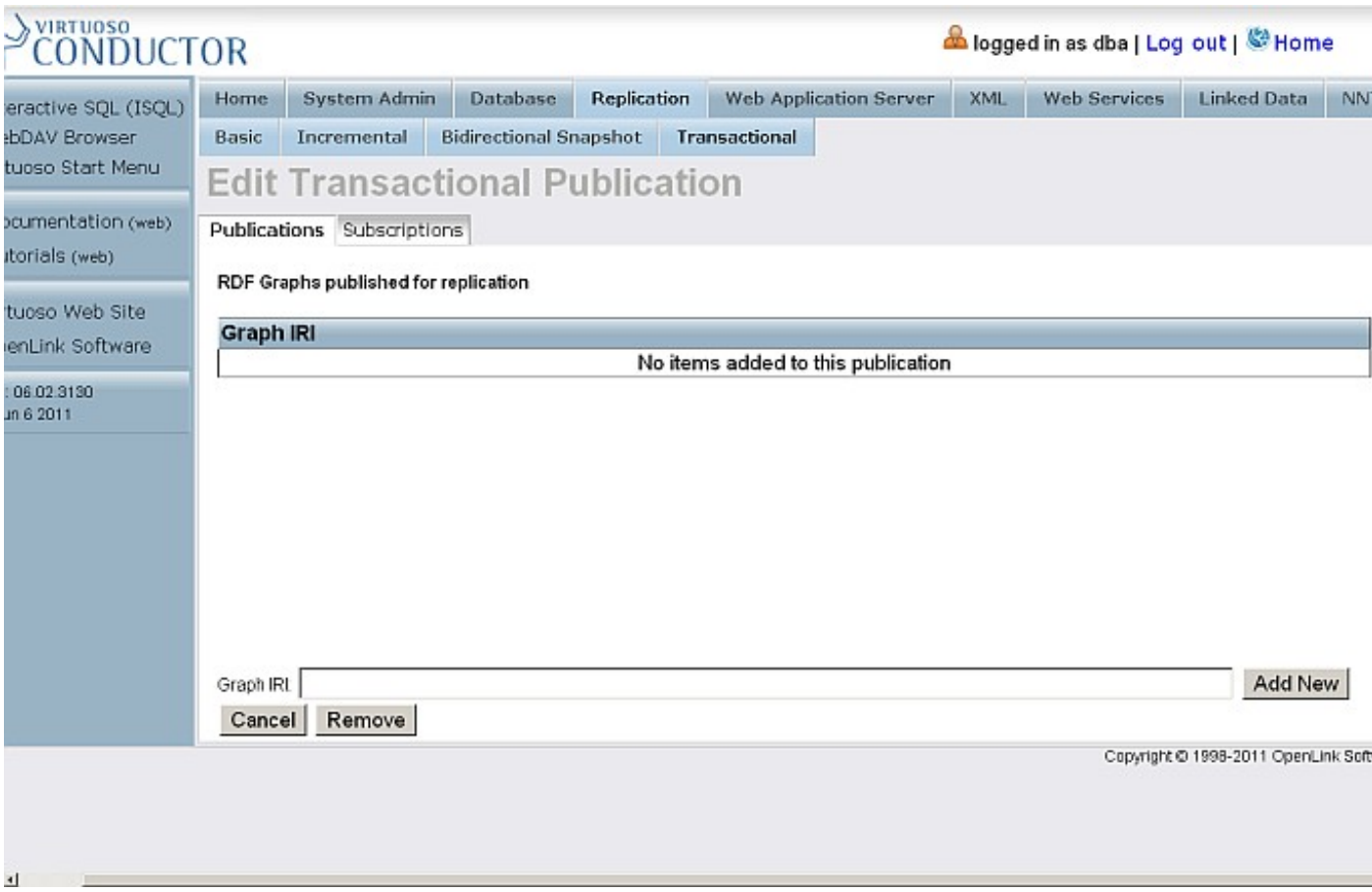
   should be created

Figure 16.267. Bi-directional Replication Topology



5. Click the link which is the publication name.
6. You will be shown the publication items page

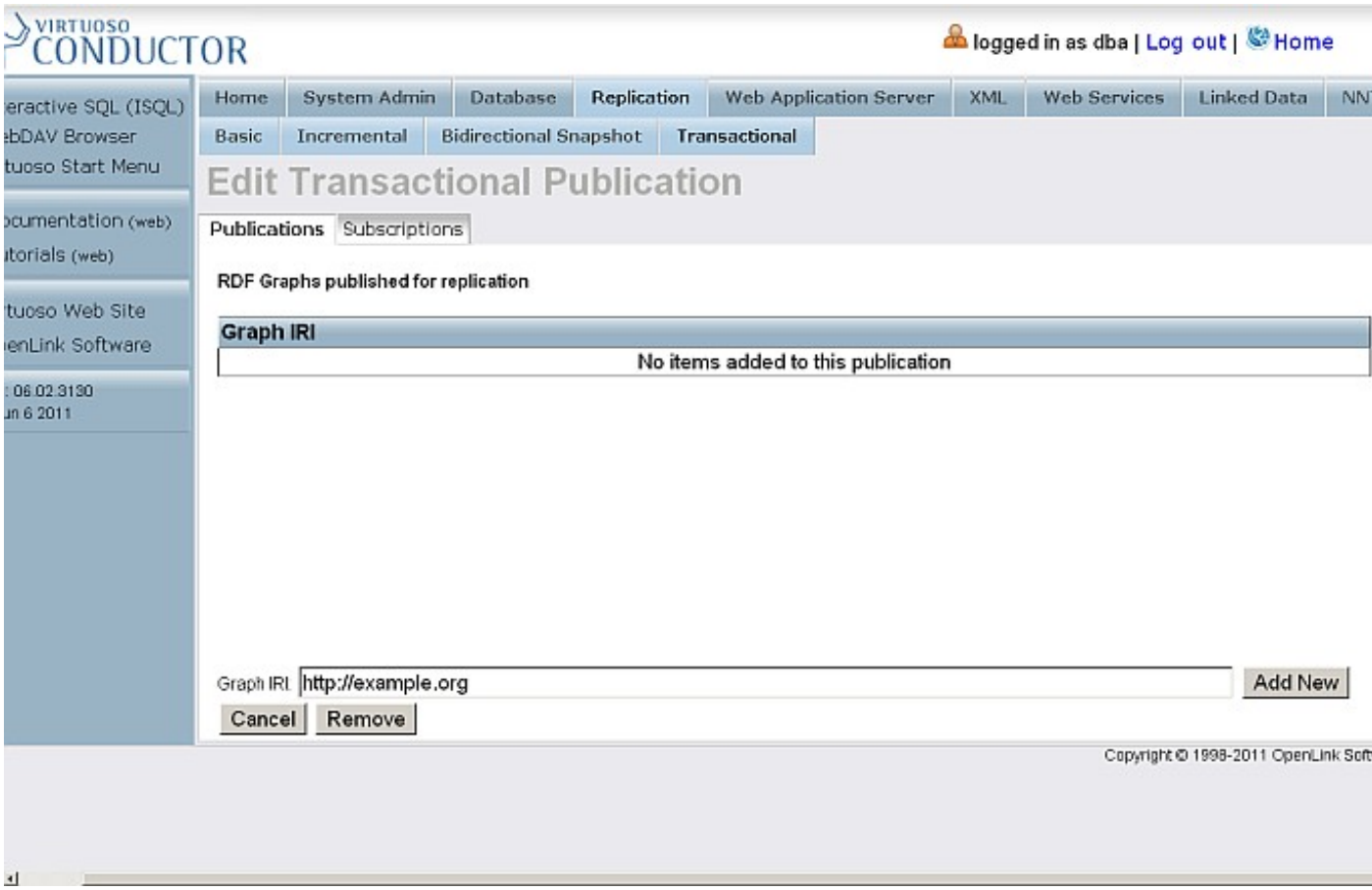
Figure 16.268. Bi-directional Replication Topology



7. Enter for Graph IRI:

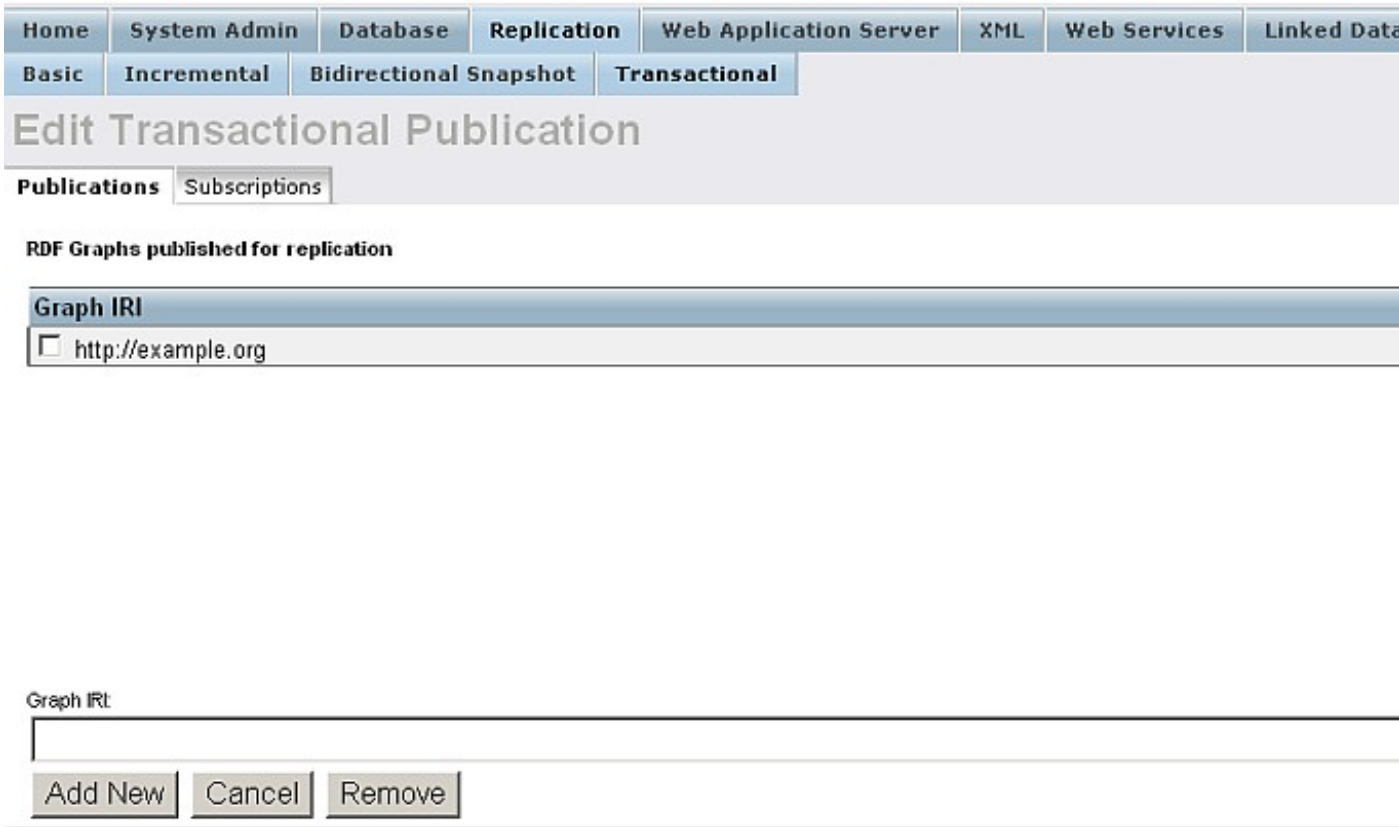
```
http://example.org
```

Figure 16.269. Bi-directional Replication Topology



8. Click Add New
9. The item will be created and shown in the list of items for the currently viewed publication.

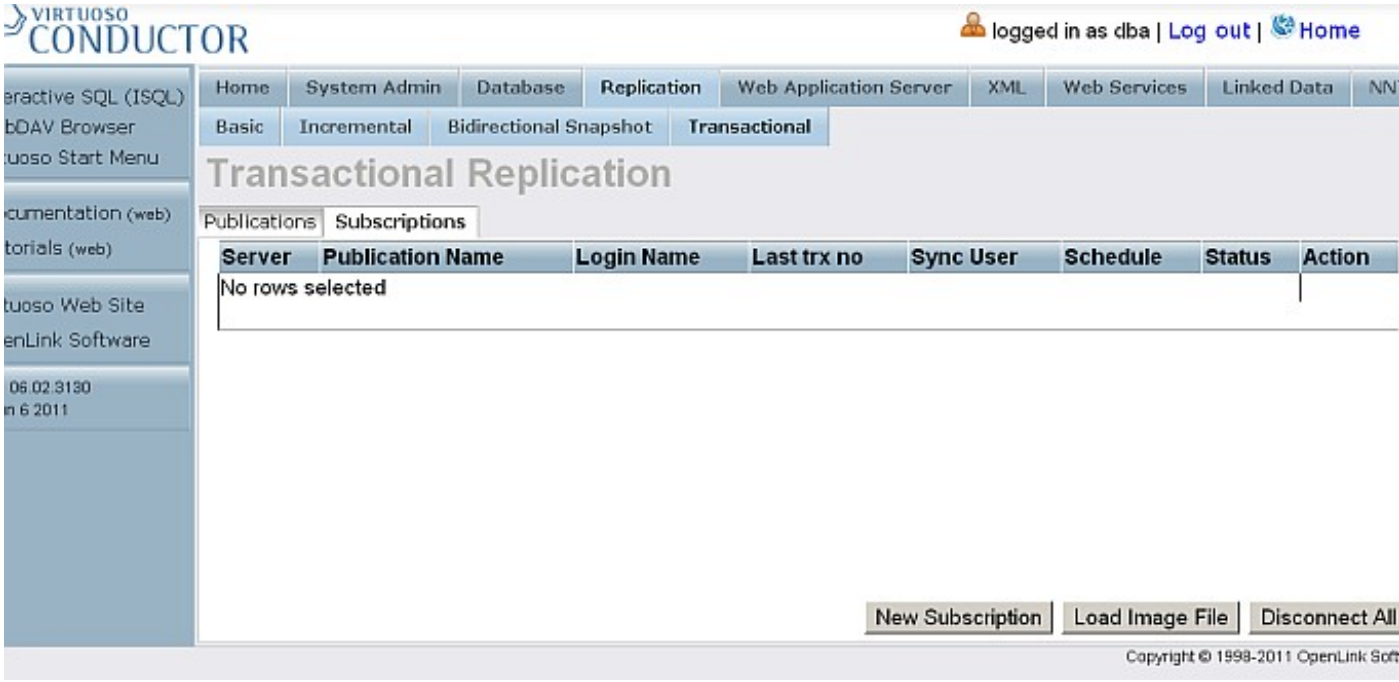
Figure 16.270. Bi-directional Replication Topology



Create subscription from db1 to db2's Publication

1. Log in at http://example.com:8891/conductor
2. Go to Replication -> Transactional -> Subscriptions

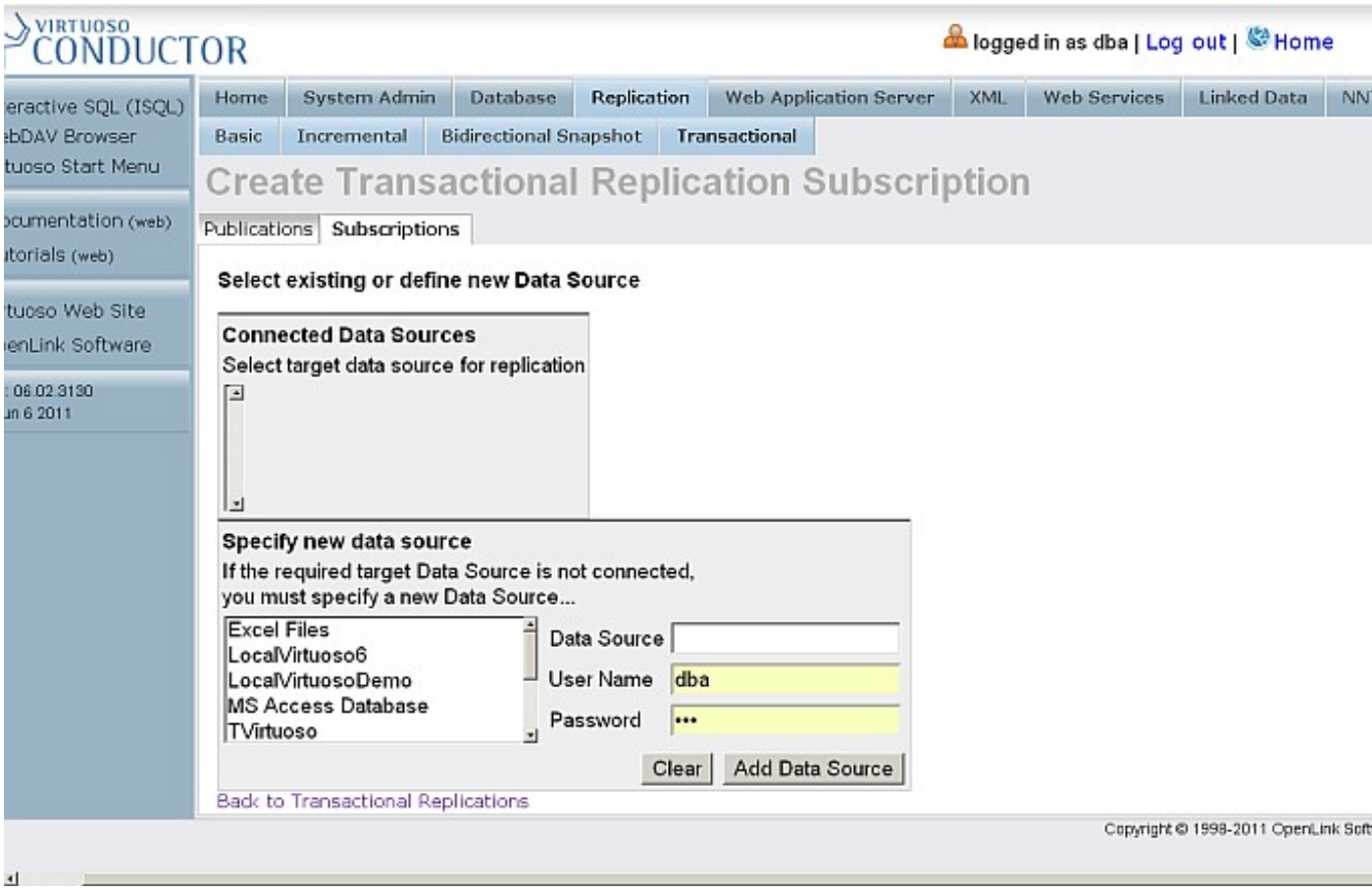
Figure 16.271. Bi-directional Replication Topology



3. Click

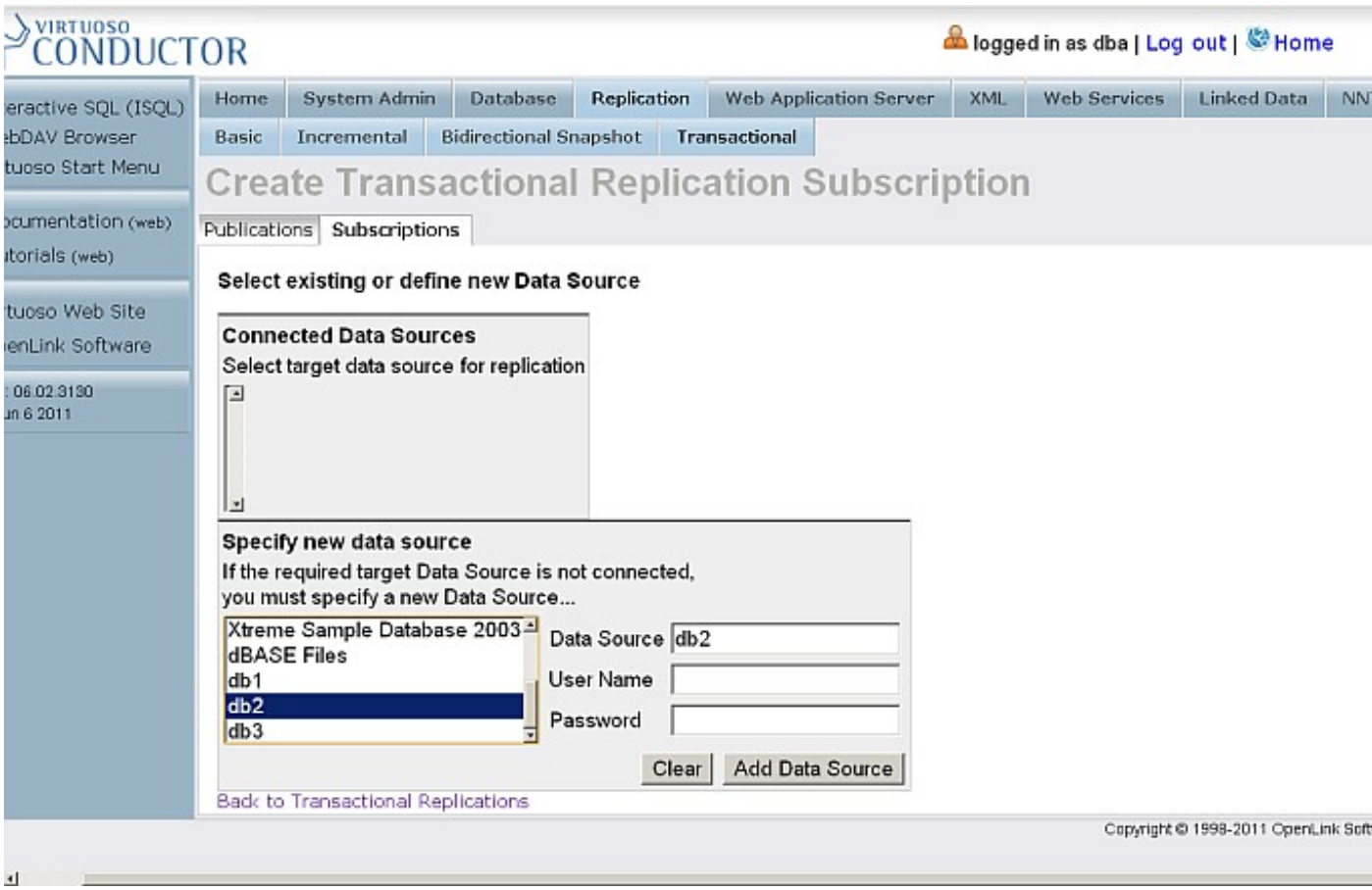
New Subscription

Figure 16.272. Bi-directional Replication Topology



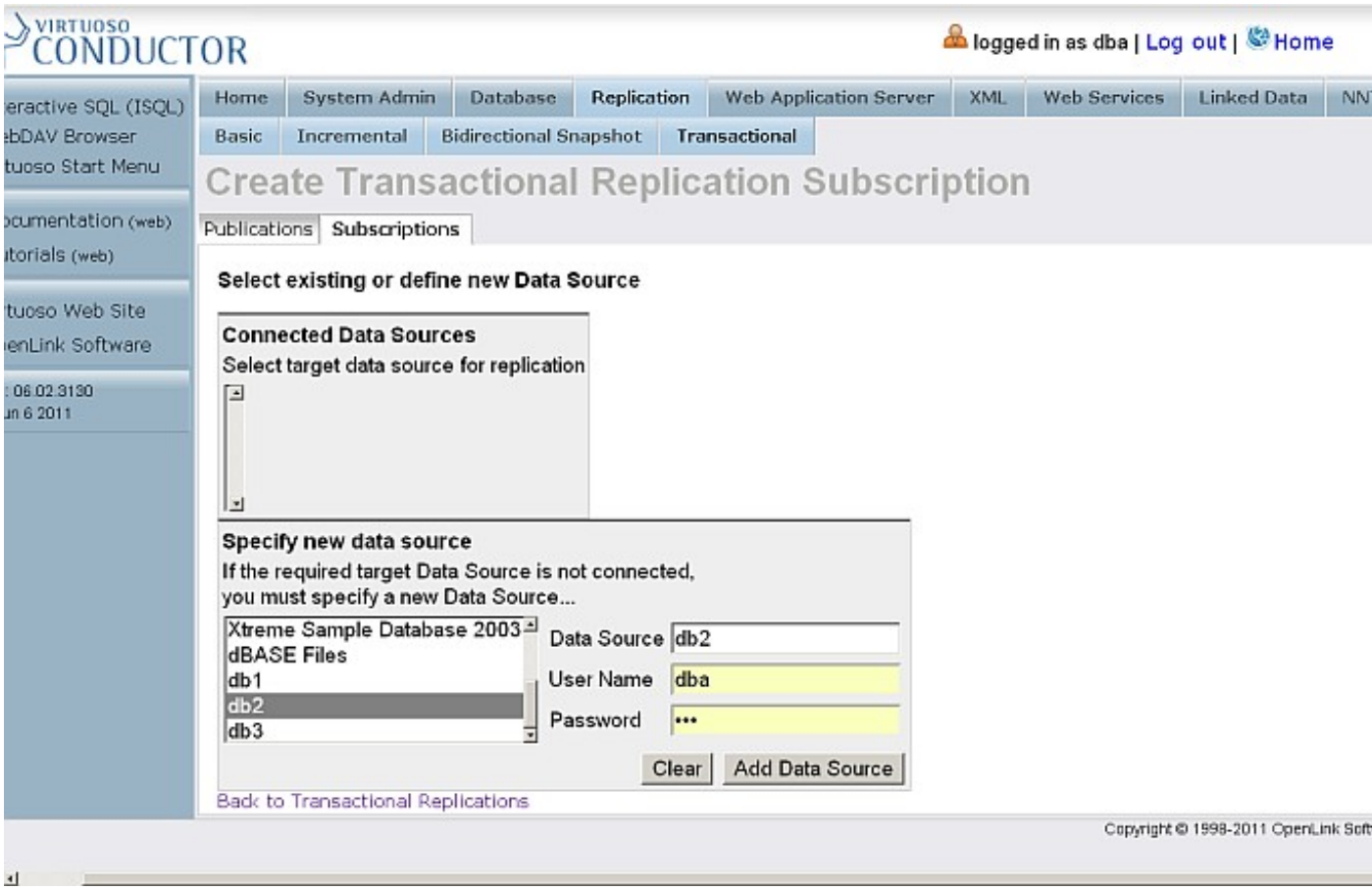
4. From the list of "Specify new data source" select Data Source db2

Figure 16.273. Bi-directional Replication Topology



5. Enter for db2 dba user credentials

Figure 16.274. Bi-directional Replication Topology

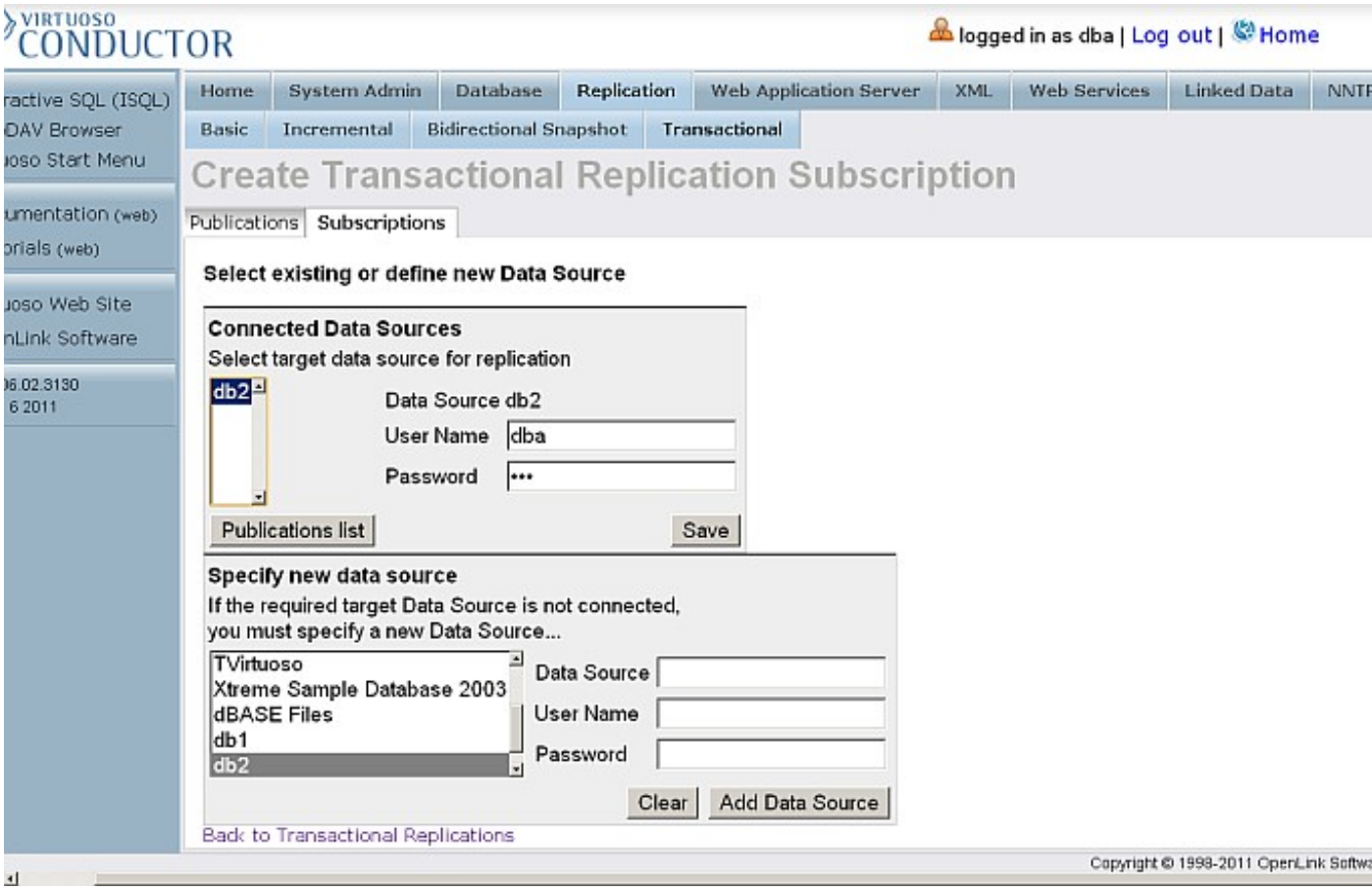


6. Click "Add Data Source"
7. As result

   db2

   will be shown in the "Connected Data Sources" list.

Figure 16.275. Bi-directional Replication Topology

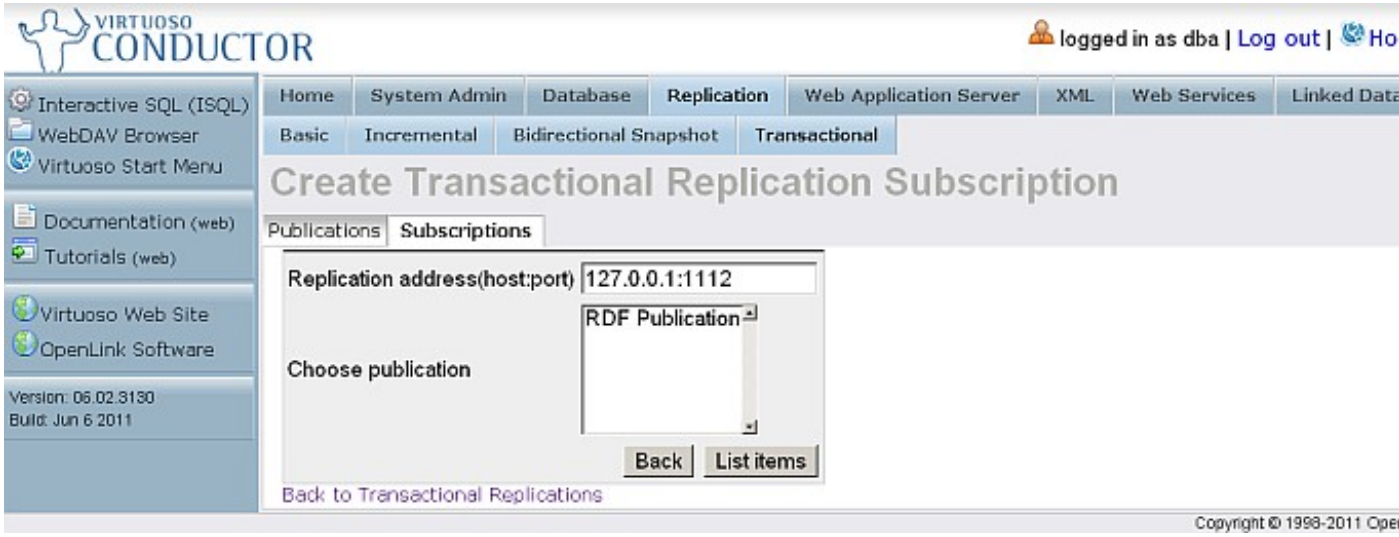


8. Select

   db2

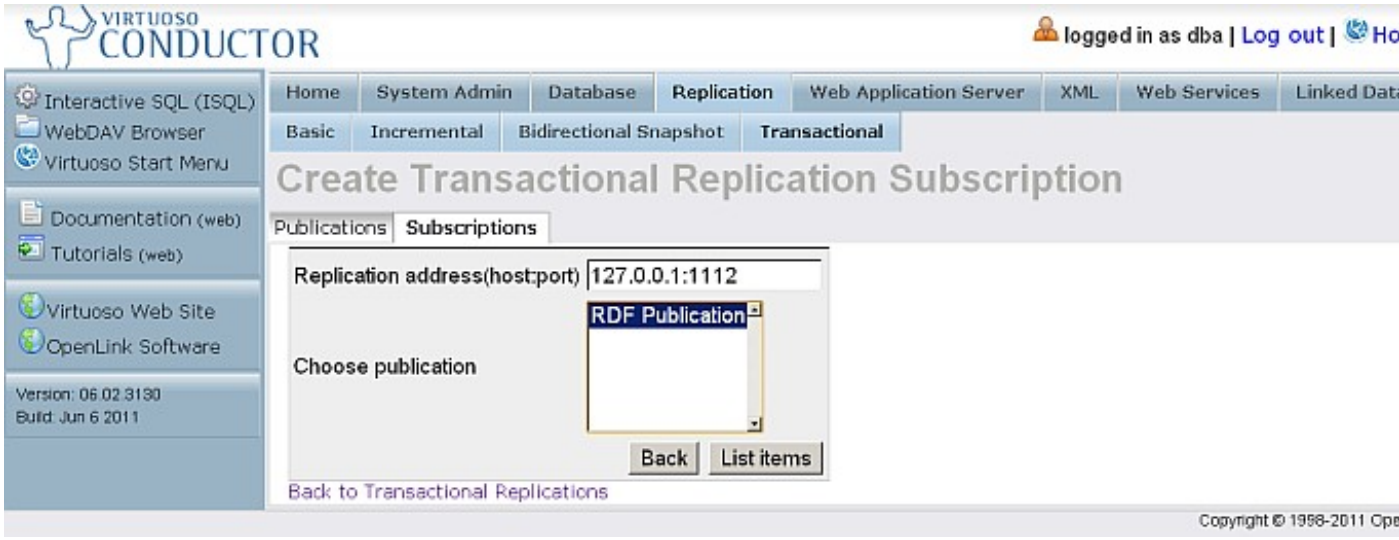
   the "Connected Data Sources" list and click "Publications list"

Figure 16.276. Bi-directional Replication Topology



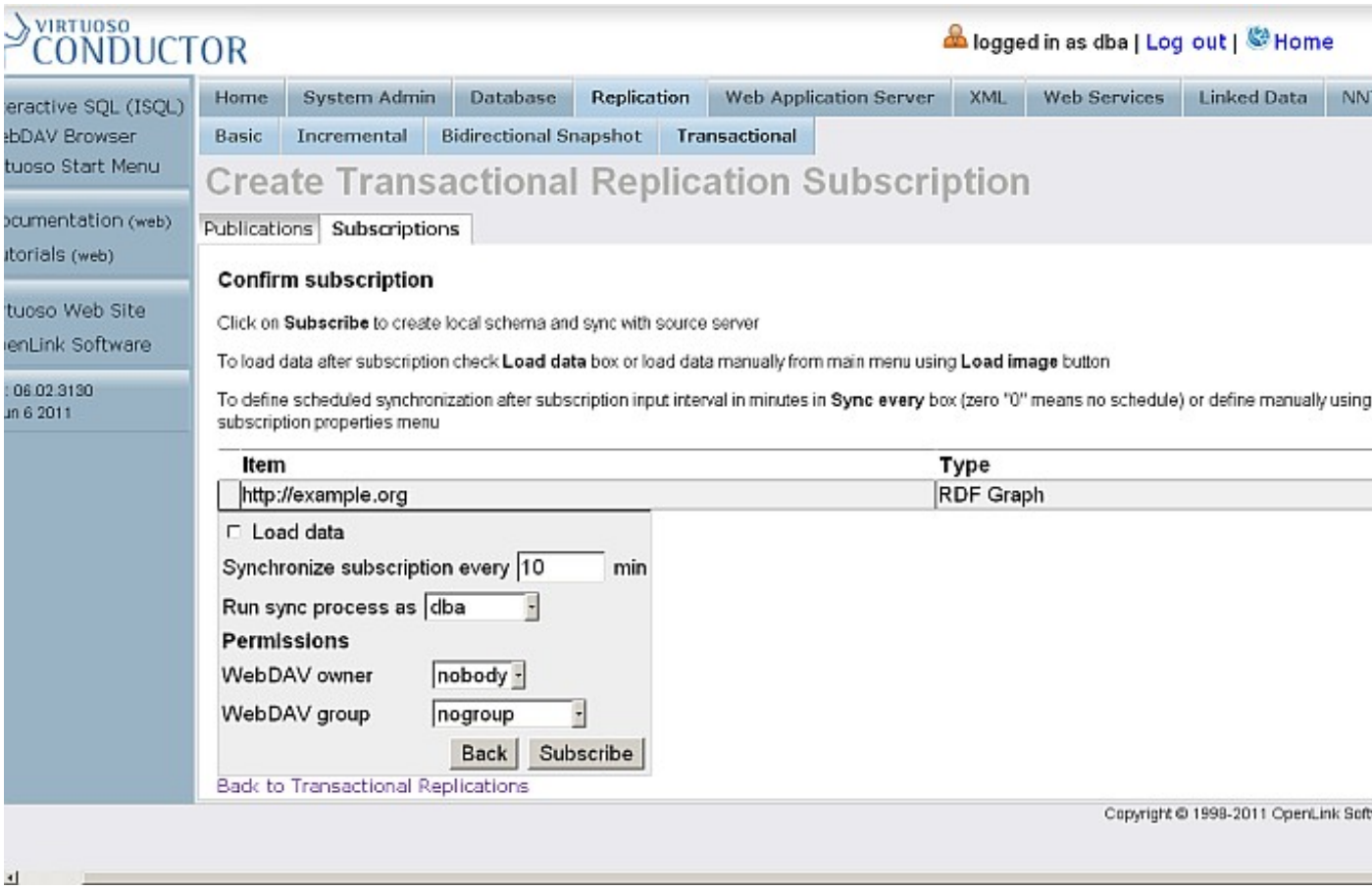
9. As result will be shown the list of available publications for the selected data source. Select the one with name "RDF Publication" and click "List Items".

Figure 16.277. Bi-directional Replication Topology



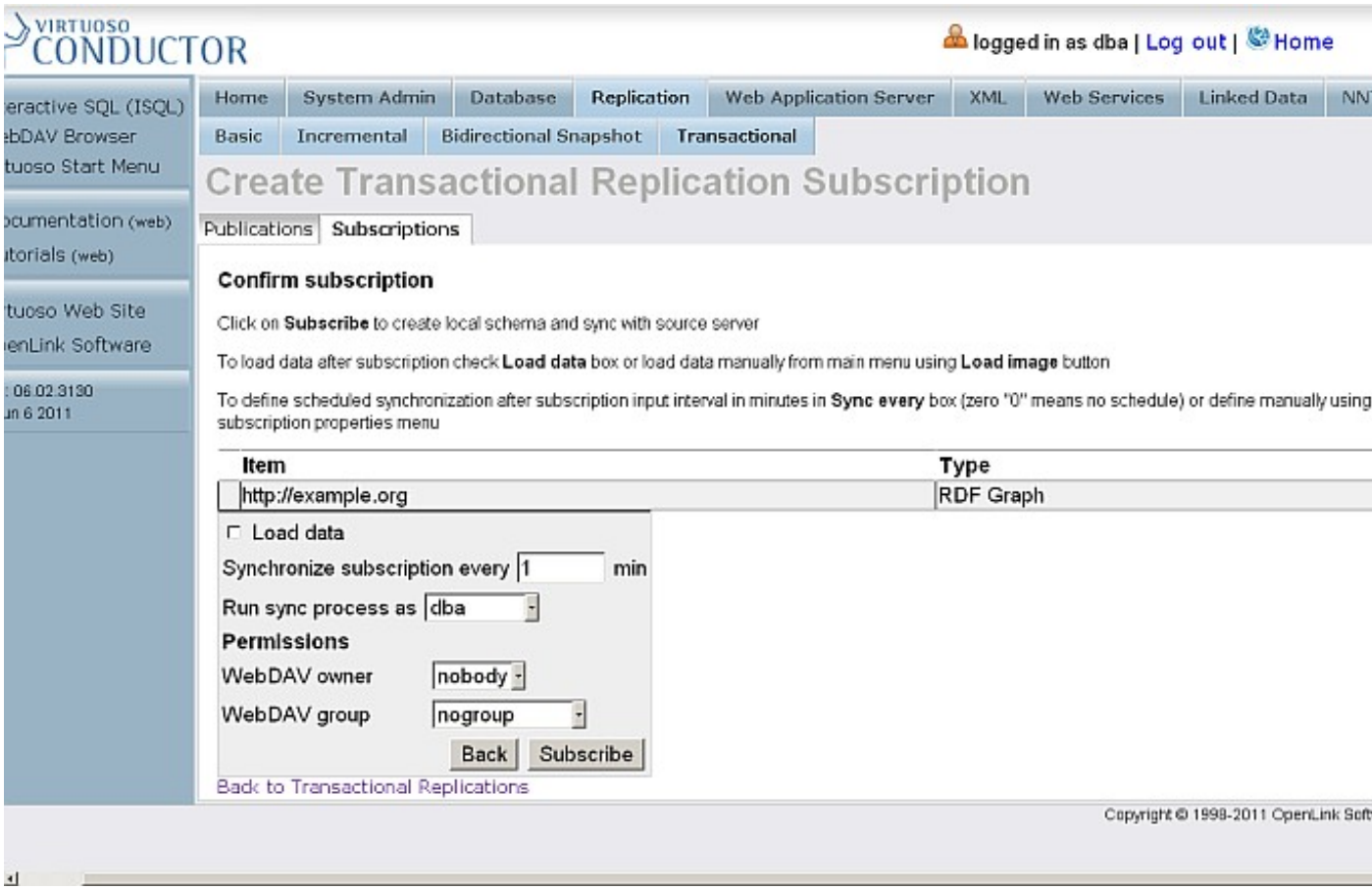
10. As result will be shown the "Confirm subscription" page.

Figure 16.278. Bi-directional Replication Topology



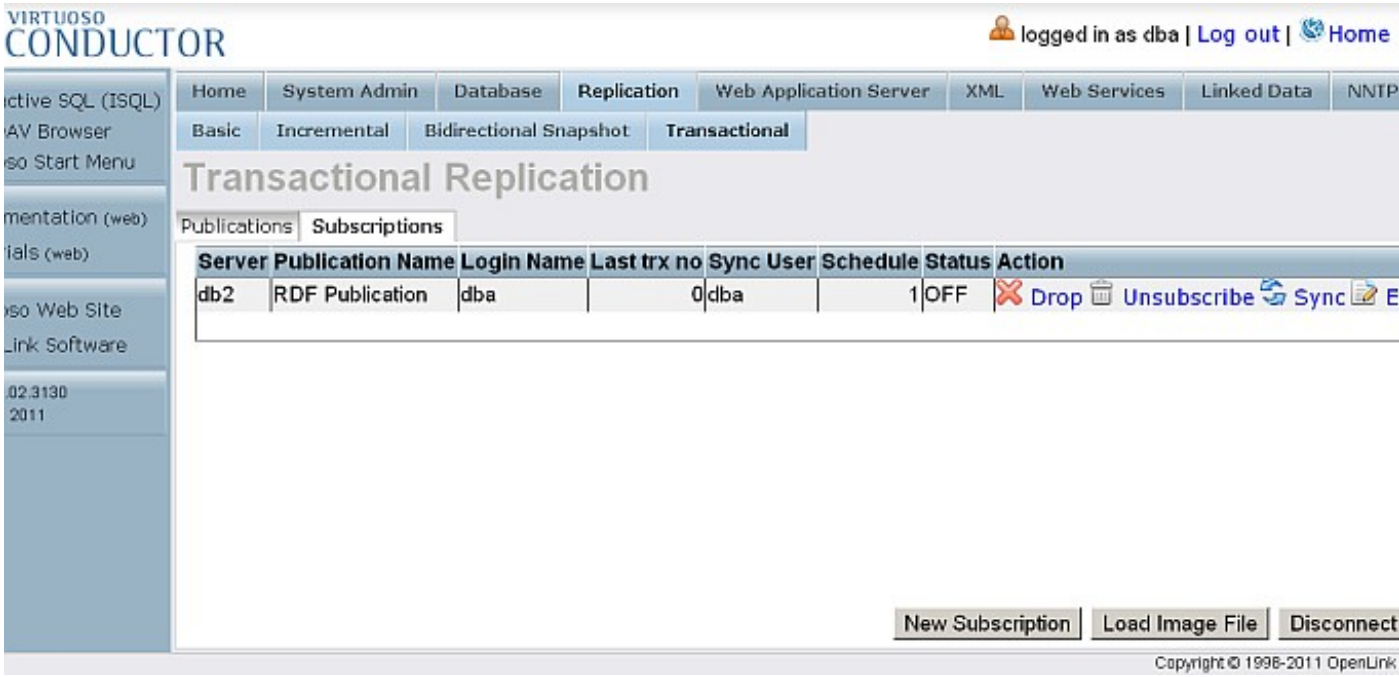
11. The sync interval by default is 10 minutes. For the testing purposes, we will change it to 1 minute.

Figure 16.279. Bi-directional Replication Topology



12. Click "Subscribe"
13. The subscription will be created.

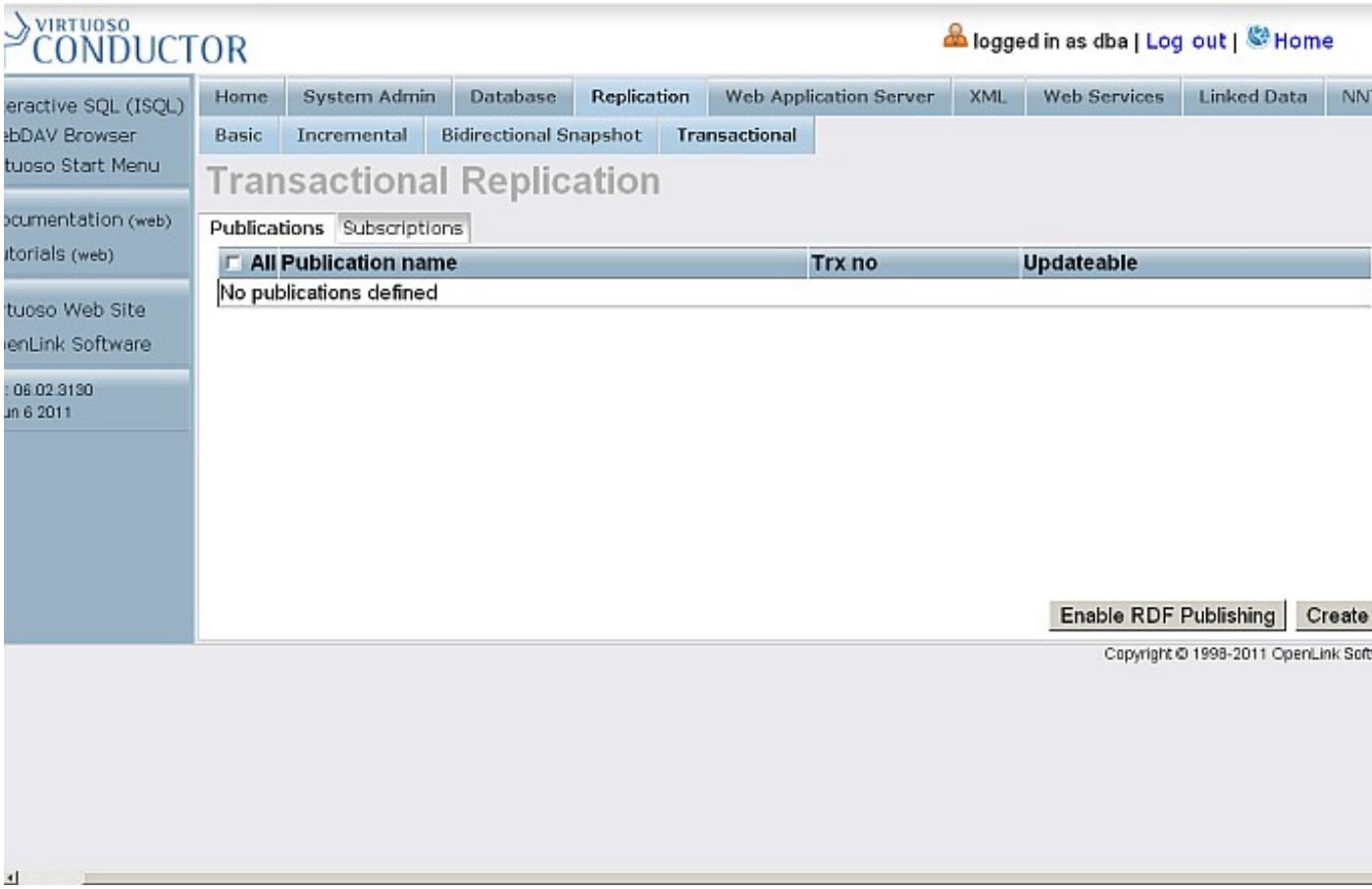
Figure 16.280. Bi-directional Replication Topology



Create Publication on db1

1. Go to http://example.com:8891/conductor and log in as dba
2. Go to Conductor -> Replication -> Transactional -> Publications

Figure 16.281. Bi-directional Replication Topology



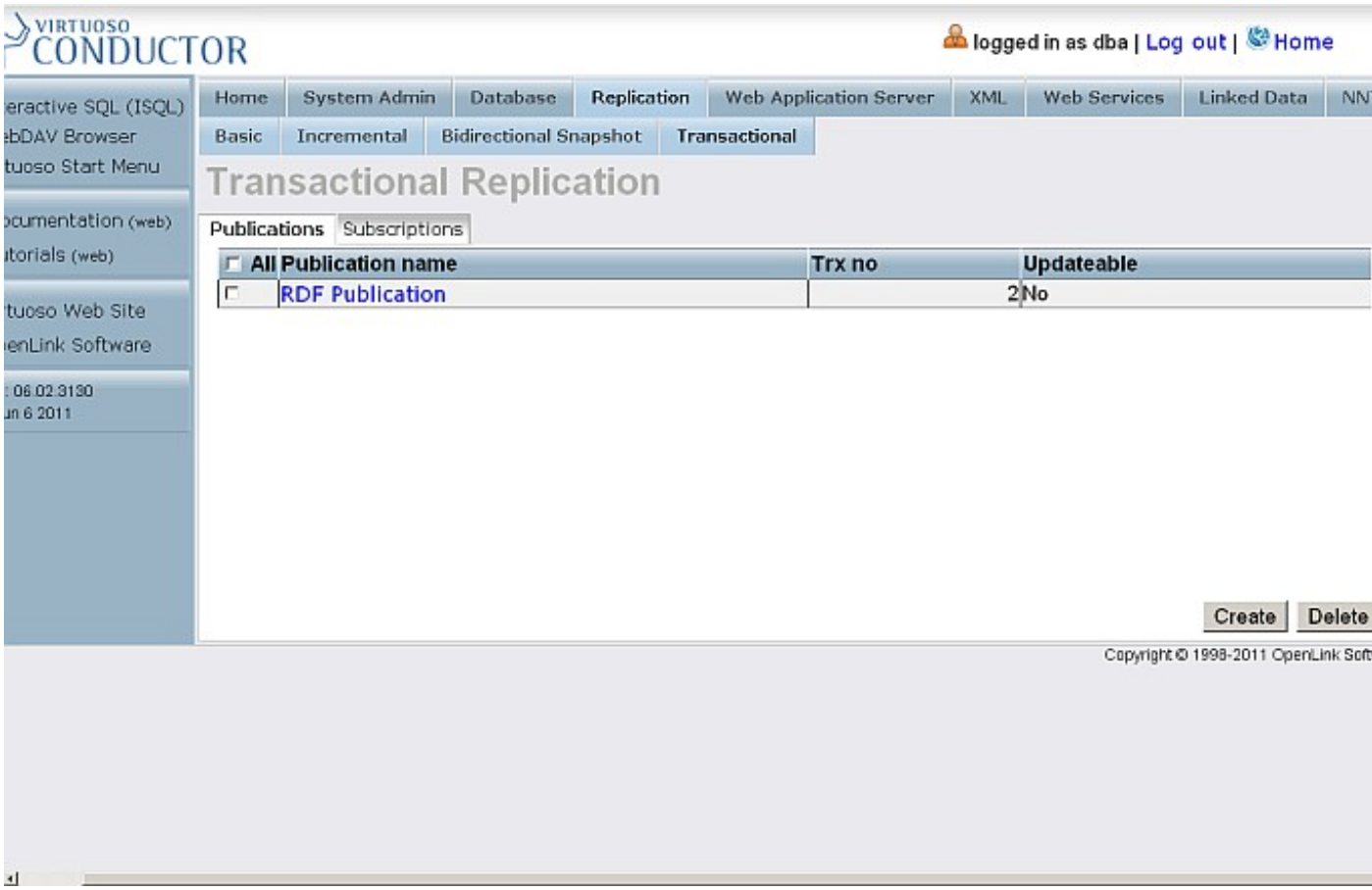
3. Click

   Enable RDF Publishing
4. As result publication with the name

   RDF Publication

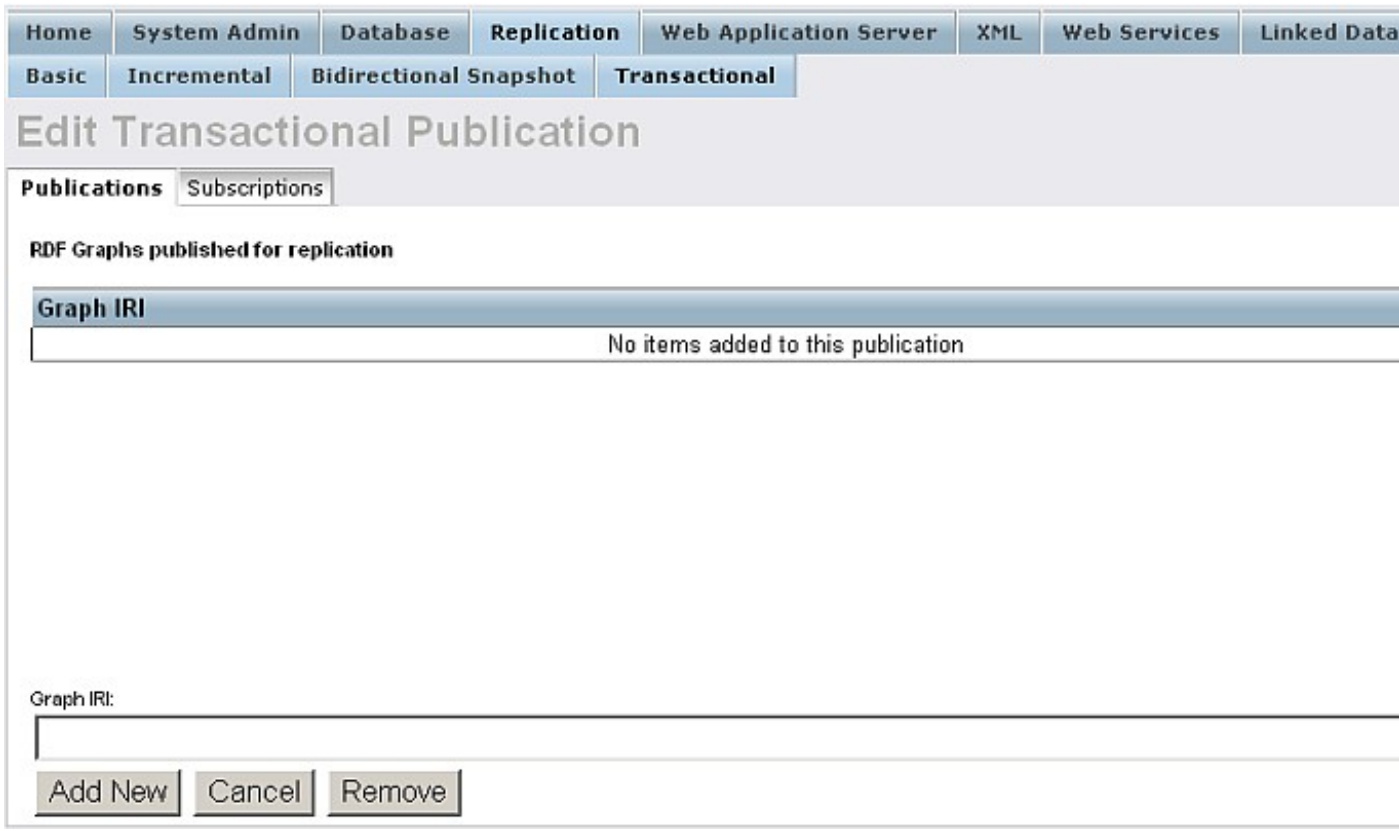
   should be created

Figure 16.282. Bi-directional Replication Topology



5. Click the link which is the publication name.
6. You will be shown the publication items page

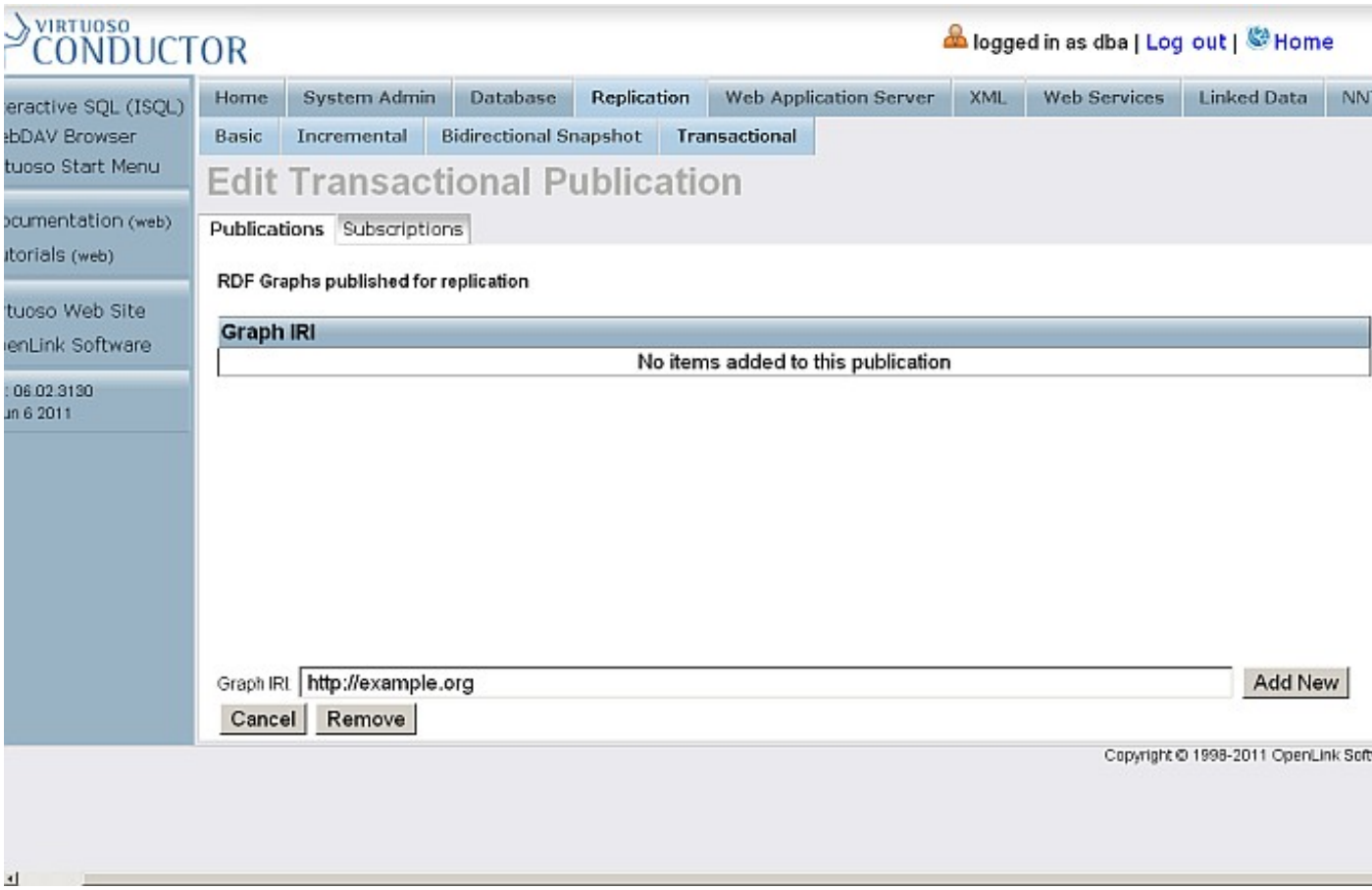
Figure 16.283. Bi-directional Replication Topology



7. Enter for Graph IRI:

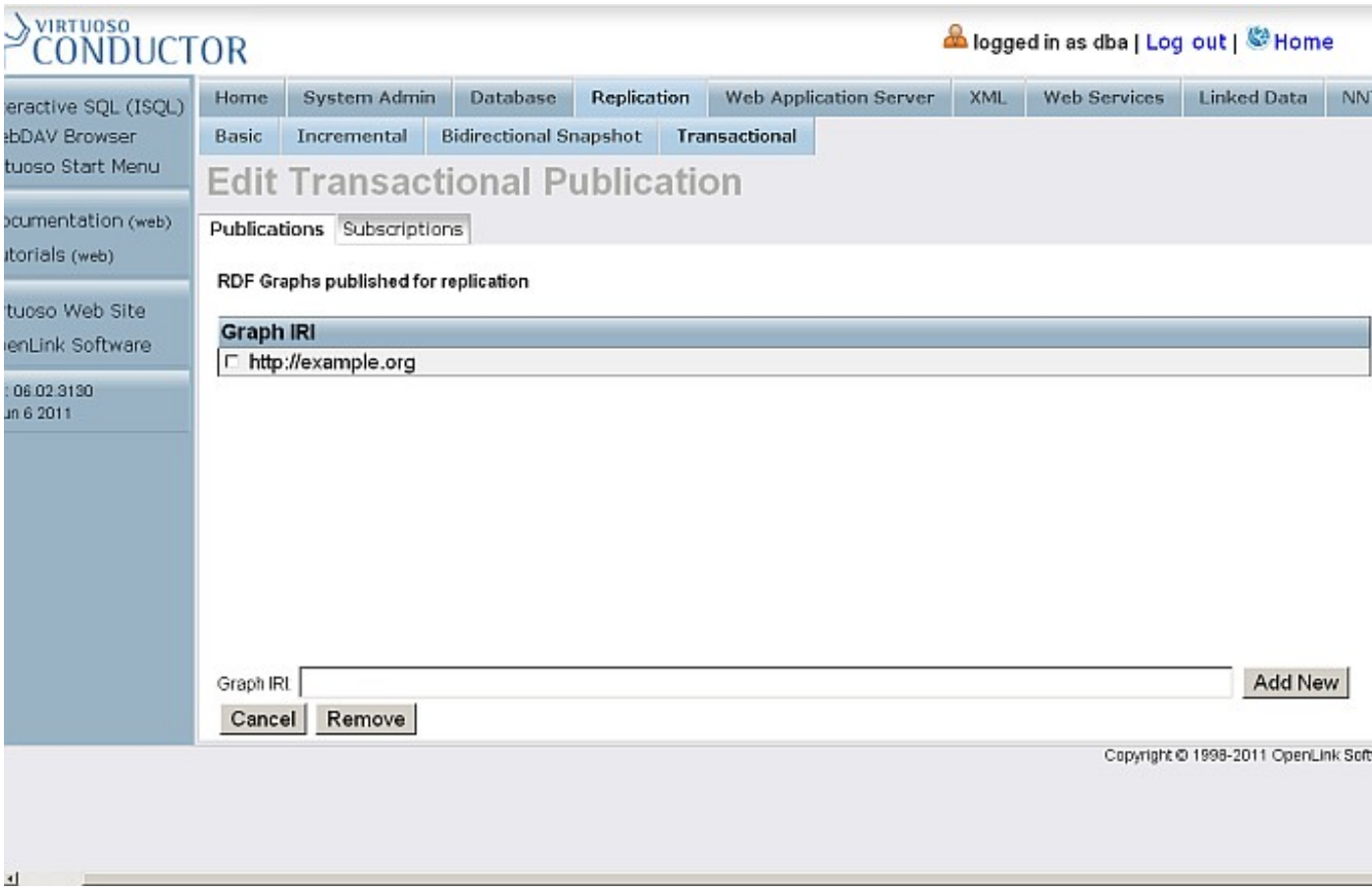
```
http://example.org
```

Figure 16.284. Bi-directional Replication Topology



8. Click Add New
9. The item will be created and shown in the list of items for the currently viewed publication.

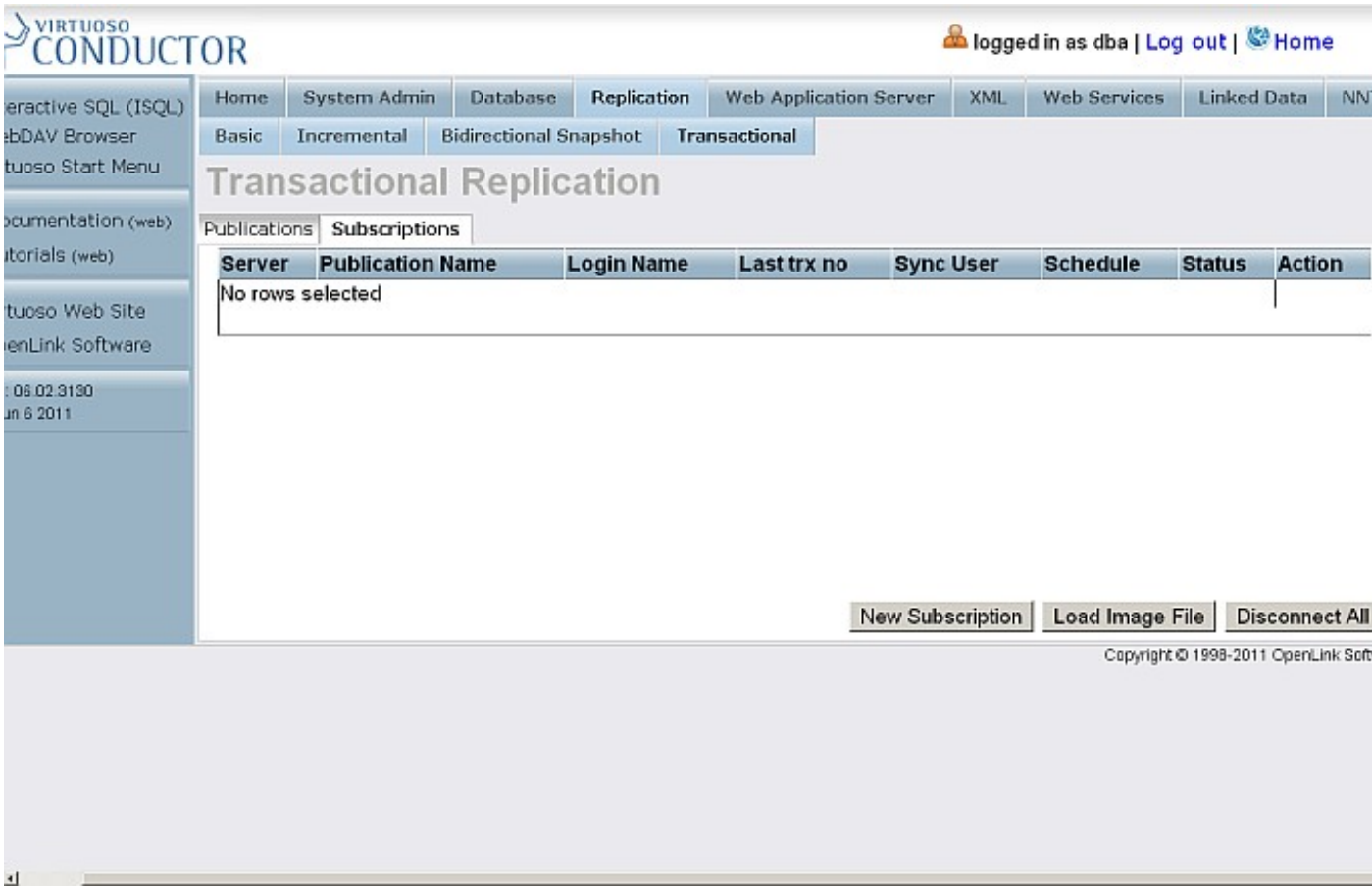
Figure 16.285. Bi-directional Replication Topology



Create subscription from db2 to db1's Publication

1. Log in at http://example.com:8892/conductor
2. Go to Replication -> Transactional -> Subscriptions

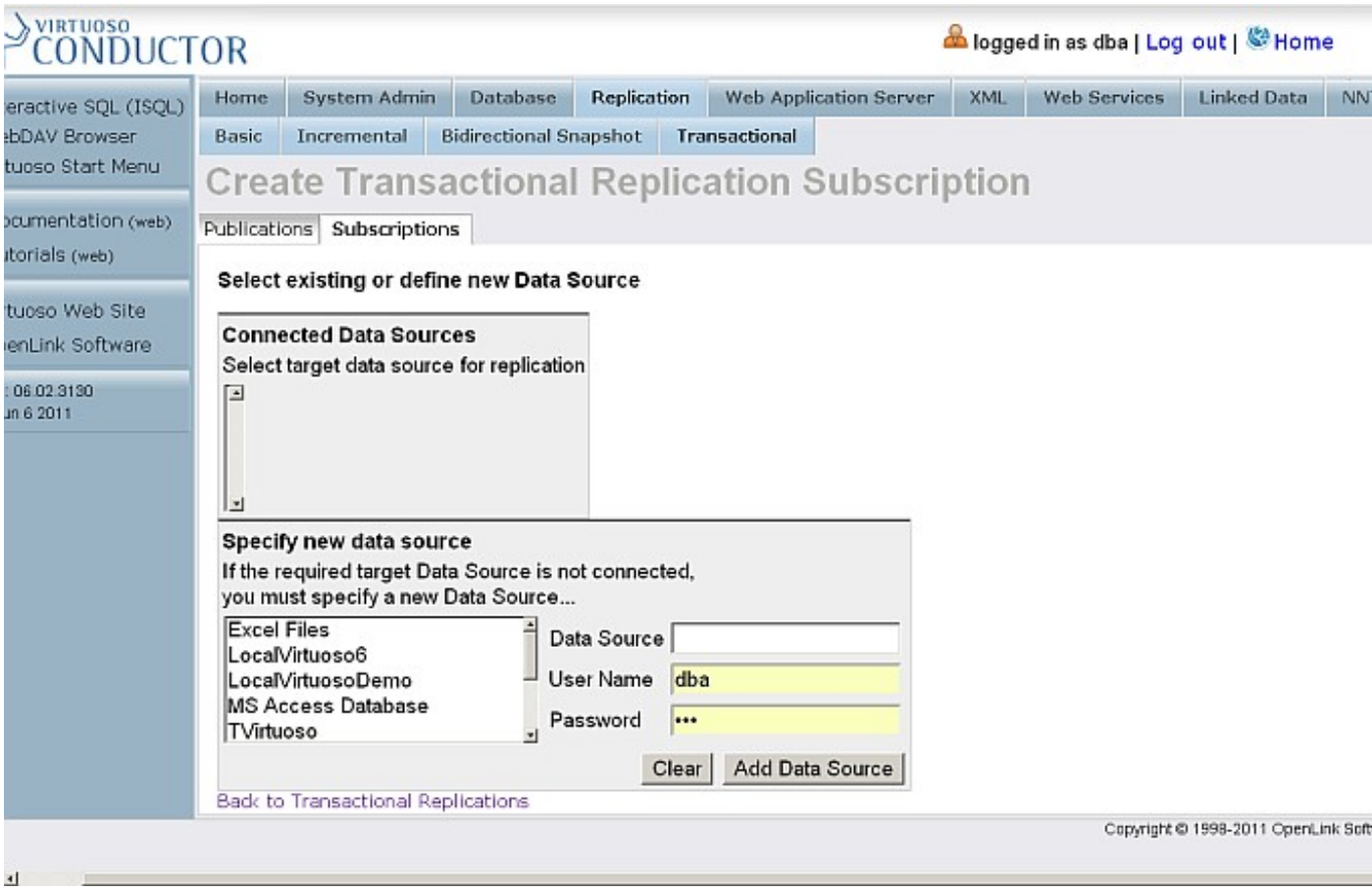
Figure 16.286. Bi-directional Replication Topology



3. Click

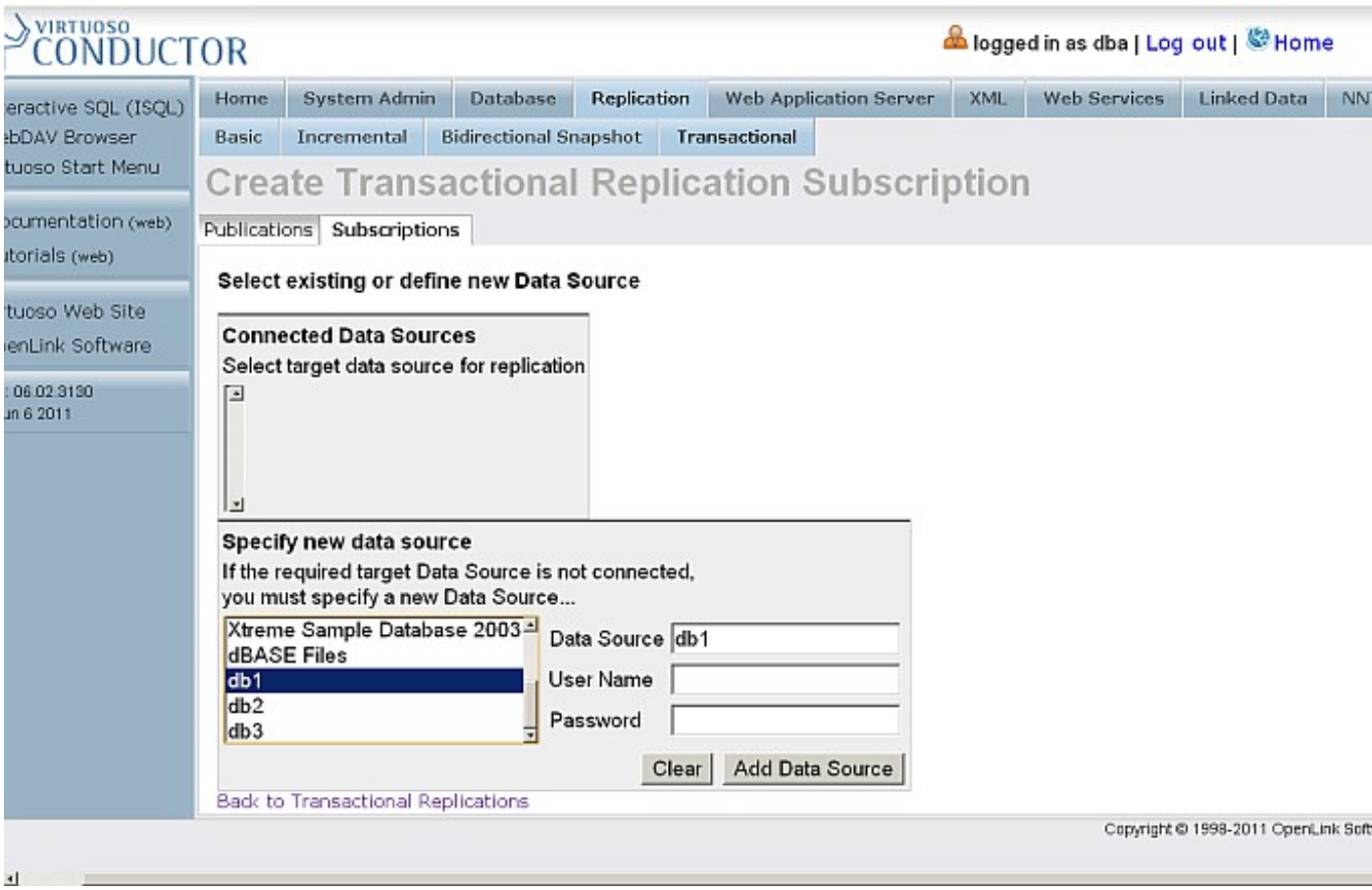
   New Subscription

Figure 16.287. Bi-directional Replication Topology



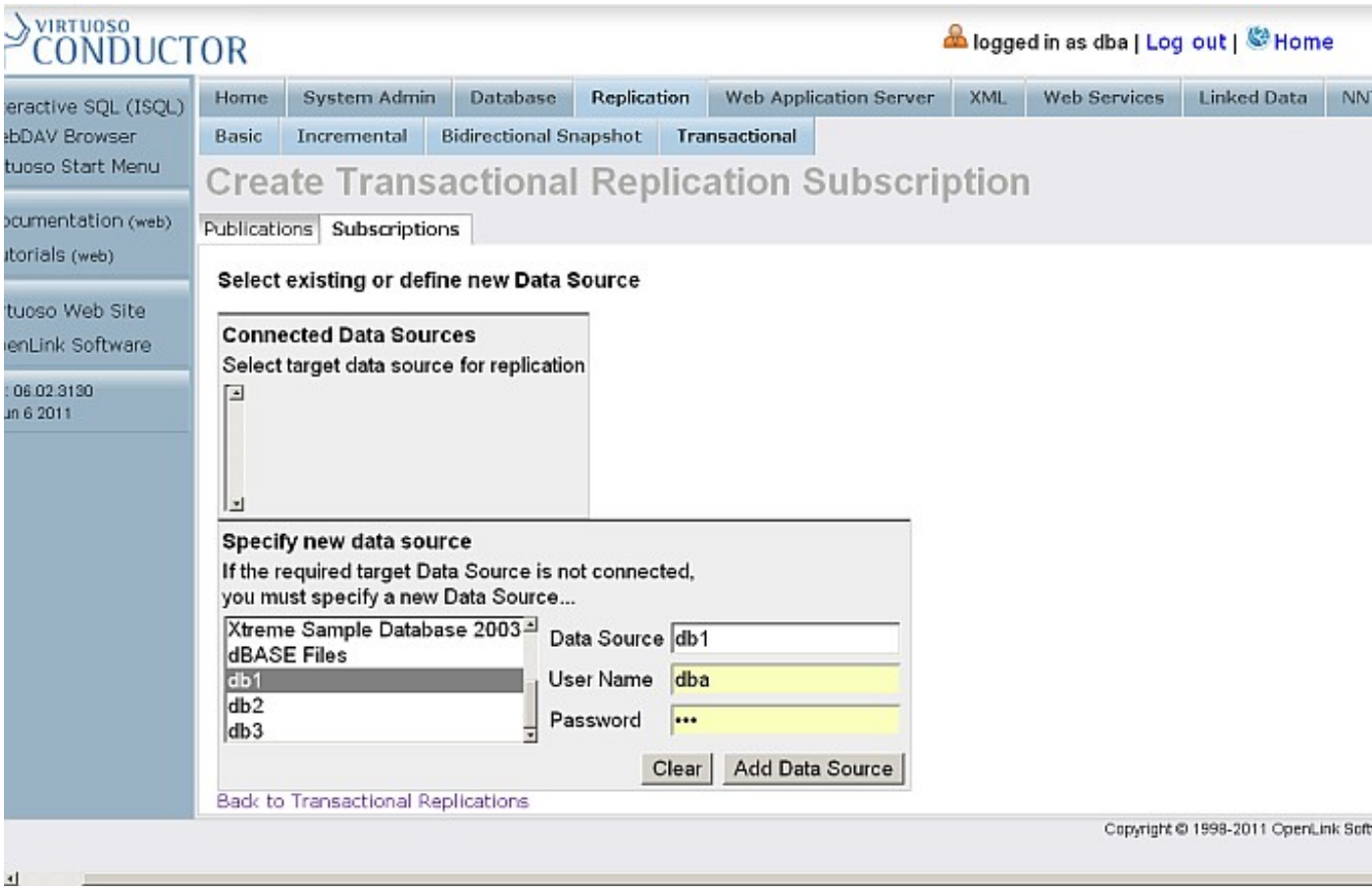
4. From the list of "Specify new data source" select Data Source db1

Figure 16.288. Bi-directional Replication Topology



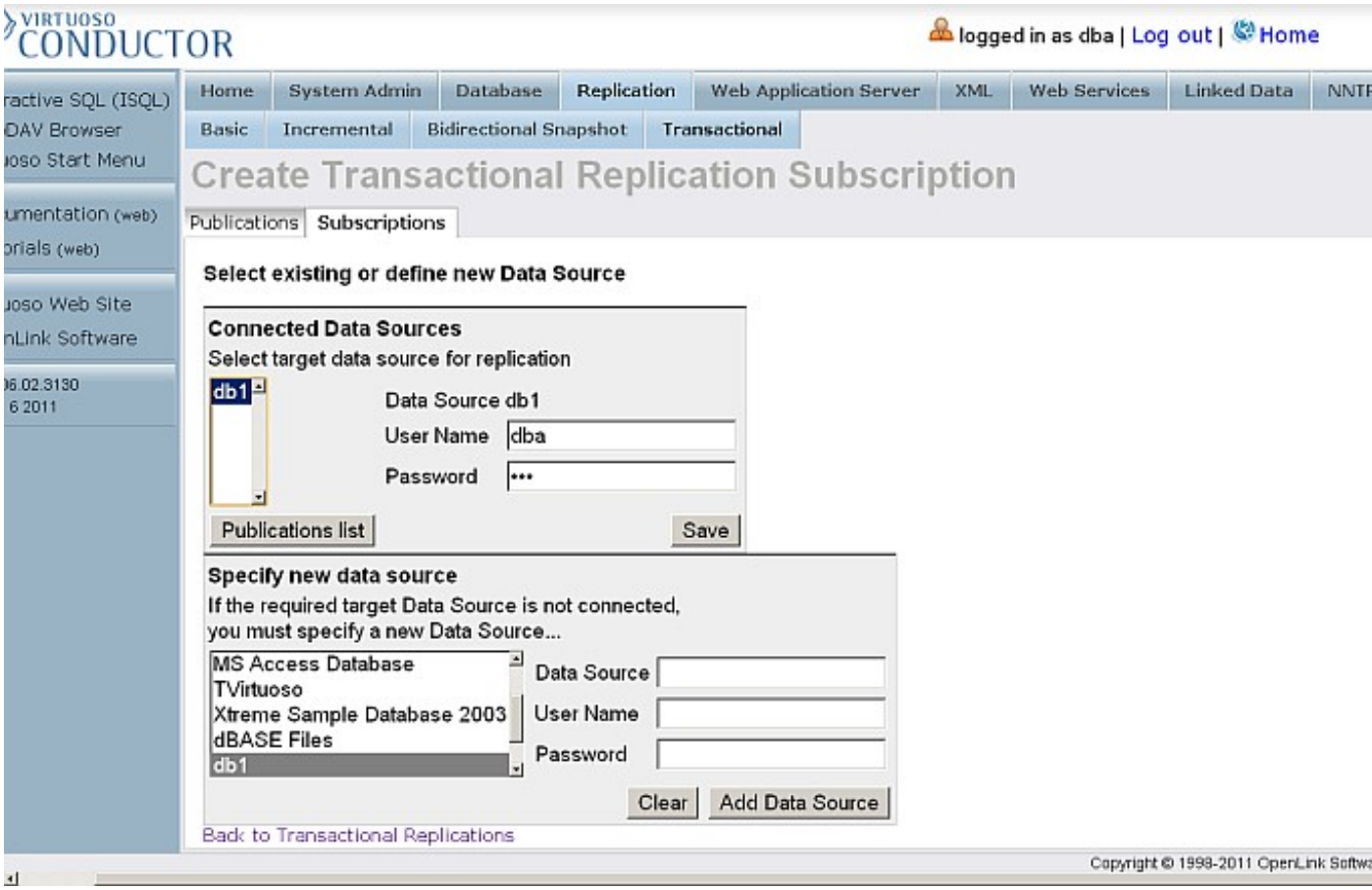
5. Enter for db1 dba user credentials

Figure 16.289. Bi-directional Replication Topology



6. Click "Add Data Source"

Figure 16.290. Bi-directional Replication Topology

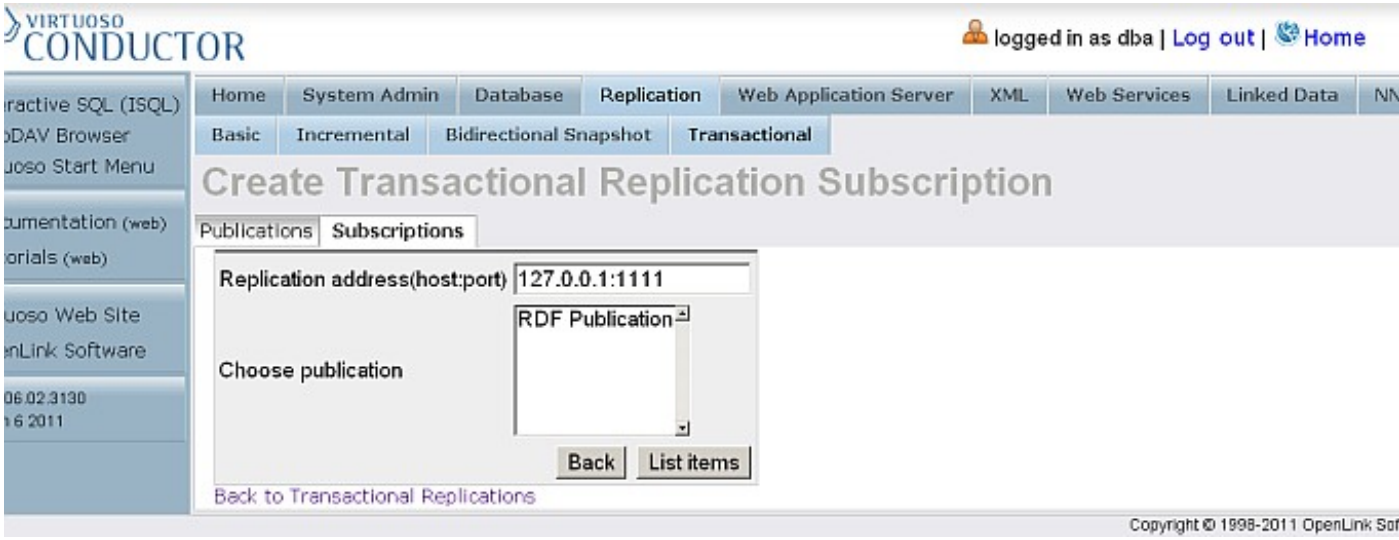


7. As result

   db1

   will be shown in the "Connected Data Sources" list. Select it and click "Publications list"

Figure 16.291. Bi-directional Replication Topology

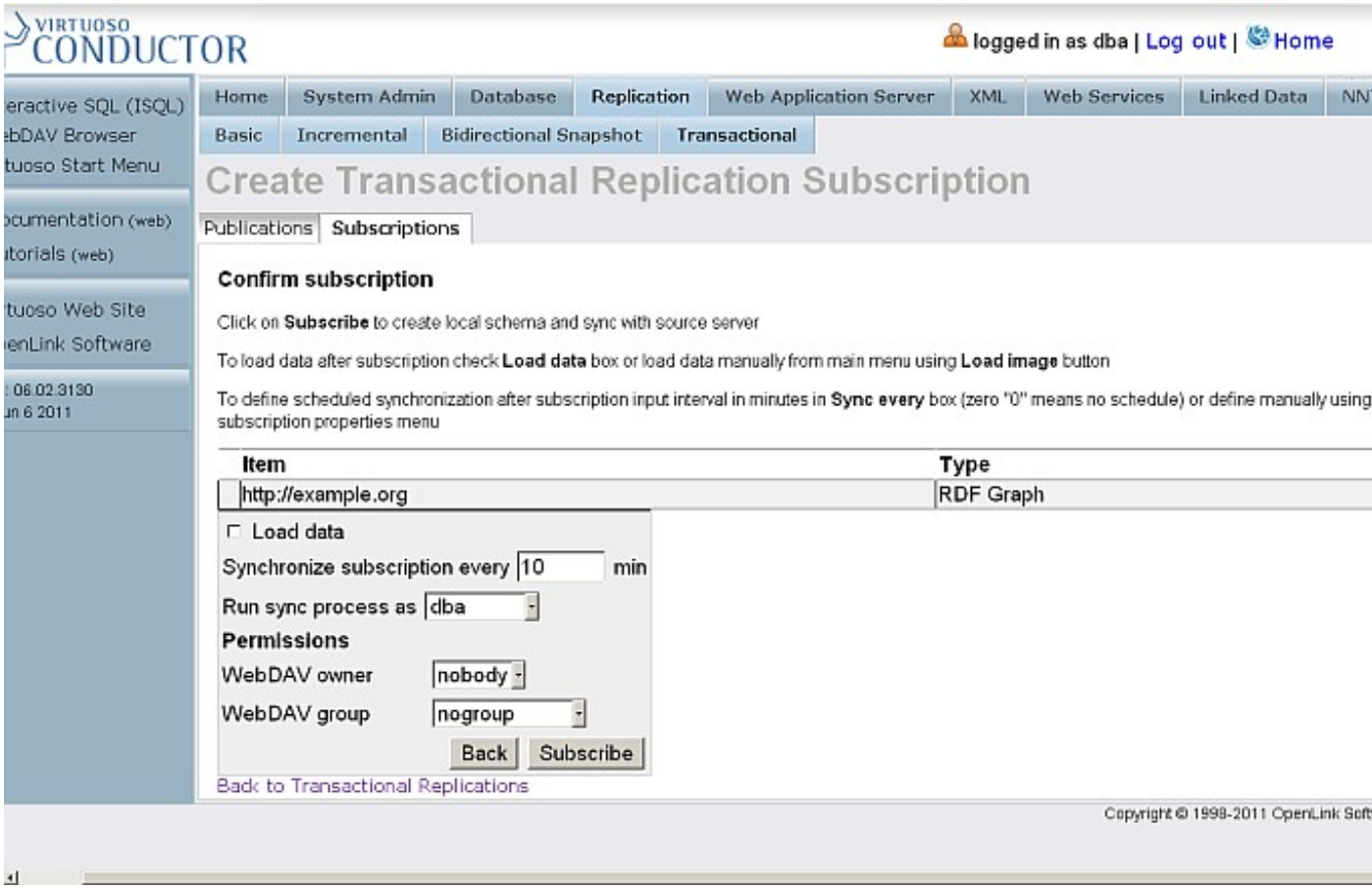


8. As result will be shown the list of available publications for the selected data source. Select the one with name "RDF Publication" and click "List Items".

Figure 16.292. Bi-directional Replication Topology

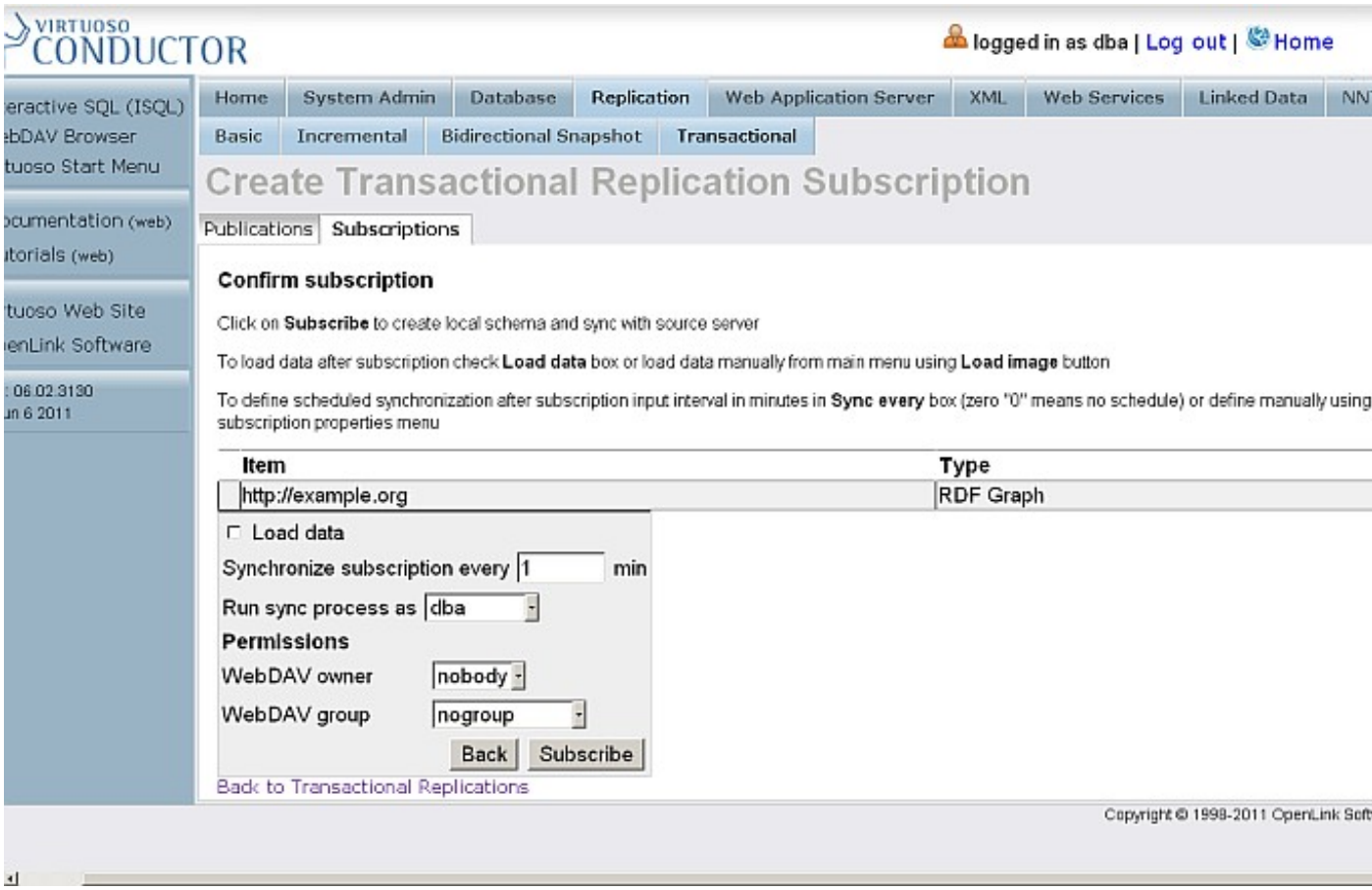
9. As result will be shown the "Confirm subscription" page.

Figure 16.293. Bi-directional Replication Topology



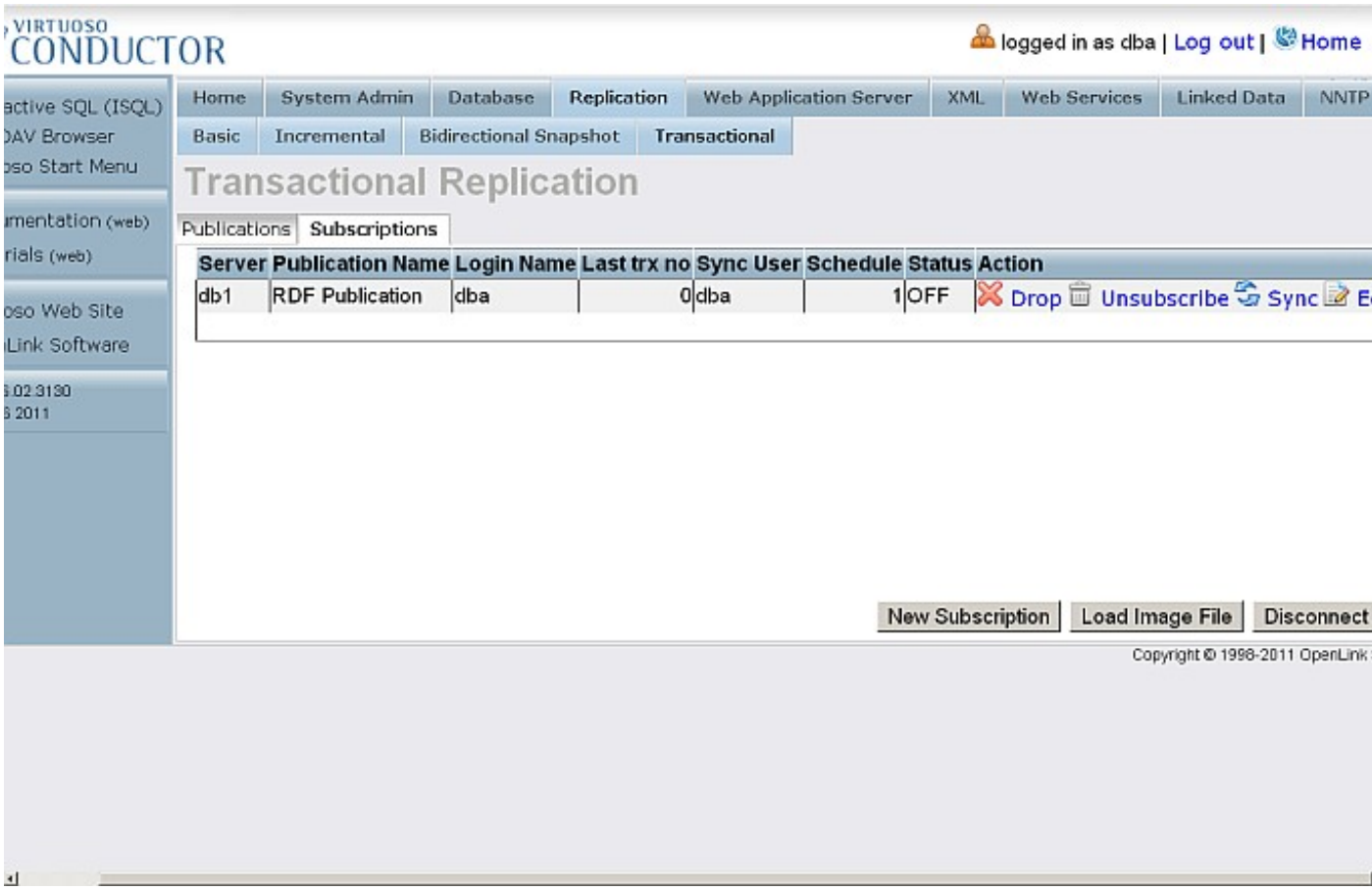
10. The sync interval by default is 10 minutes. For the testing purposes, we will change it to 1 minute.

Figure 16.294. Bi-directional Replication Topology



11. Click "Subscribe"
12. The subscription will be created.

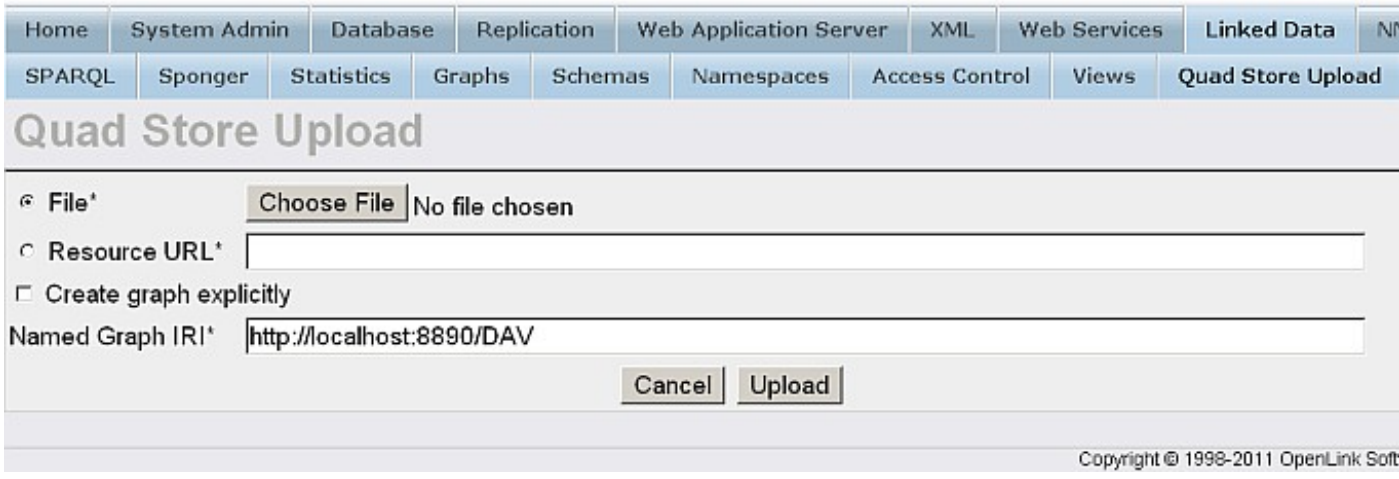
Figure 16.295. Bi-directional Replication Topology



Insert Data into a Named Graph on the db2 Virtuoso Instance

1. Log in at http://example.com:8892/conductor
2. Go to Linked Data -> Quad Store Upload:

Figure 16.296. Bi-directional Replication Topology



3. In the shown form:
4. Tick the box for

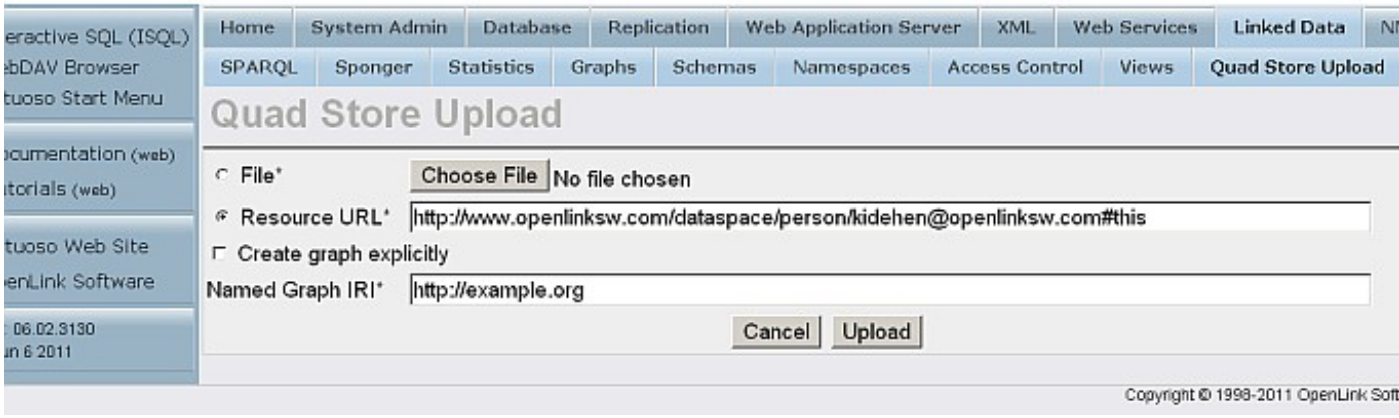
   Resource URL

   and enter your resource URL, e.g.:

   ```
   http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#this
   ```
5. Enter for Named Graph IRI:

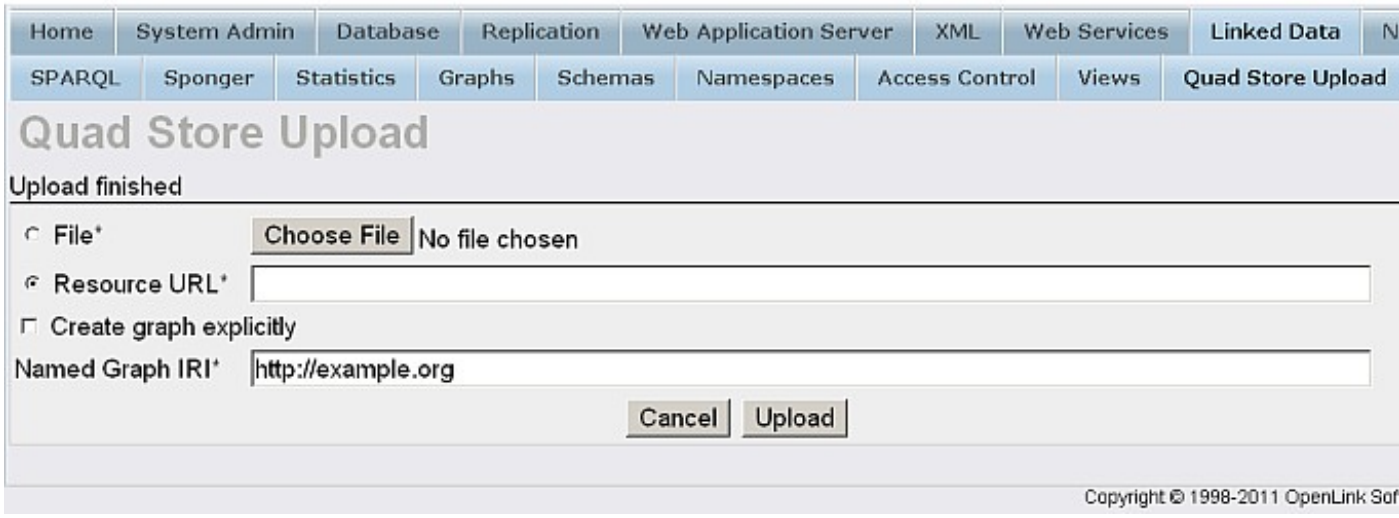
```
http://example.org
```

Figure 16.297. Bi-directional Replication Topology



6. Click Upload
7. A successful upload will result in a shown message.

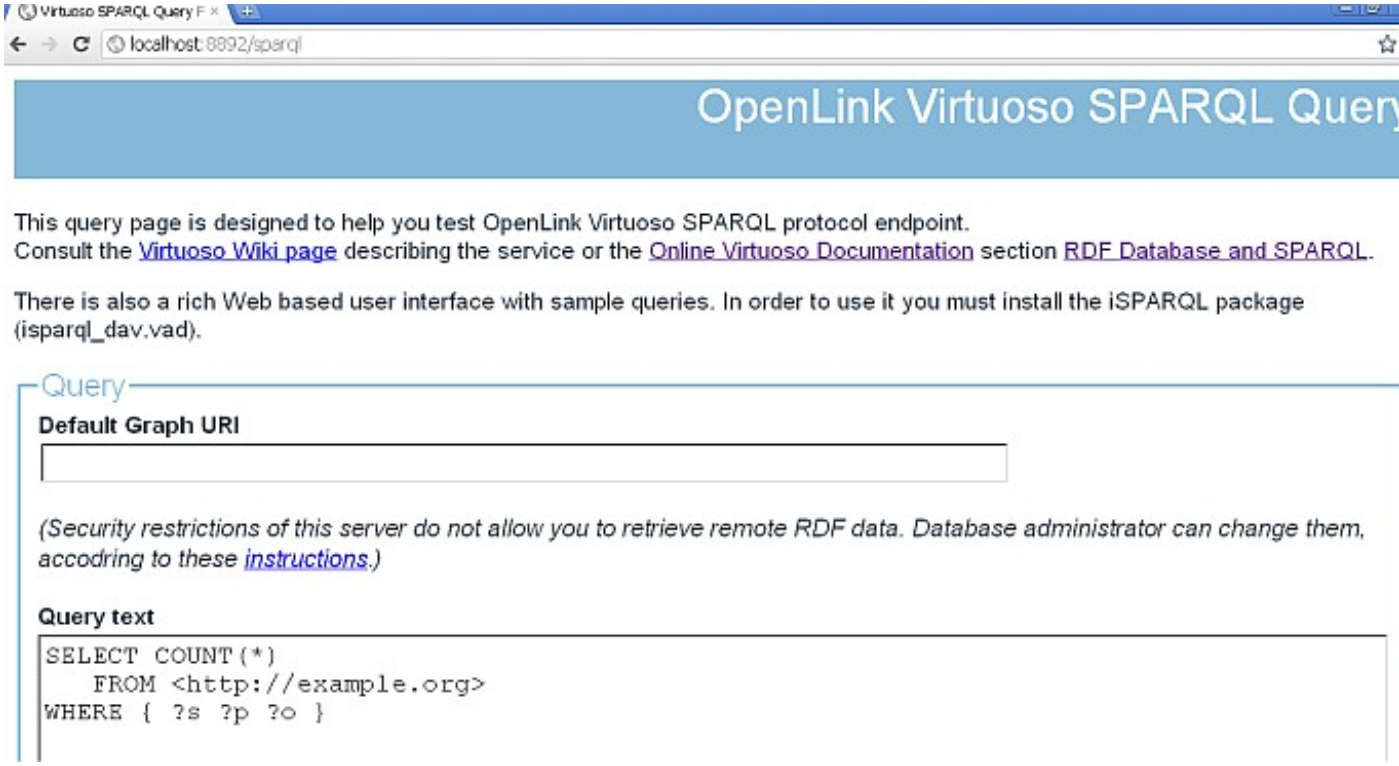
Figure 16.298. Bi-directional Replication Topology



8. Check the count of the inserted triples by executing a query like the following against the SPARQL endpoint, http://example.com:8892/sparql:

```
SELECT COUNT(*)
   FROM <http://example.org>
WHERE { ?s ?p ?o }
```

Figure 16.299. Bi-directional Replication Topology



9. Should return

   57

   as total.

Figure 16.300. Bi-directional Replication Topology

callret-0
57

Check data on the Destination instance db1

1. To check the starting count, execute from db1's SPARQL Endpoint:

```
SELECT COUNT(*)
   FROM <http://example.org>
WHERE { ?s ?p ?o }
```

2. Should return

   57

   as total.

Figure 16.301. Bi-directional Replication Topology

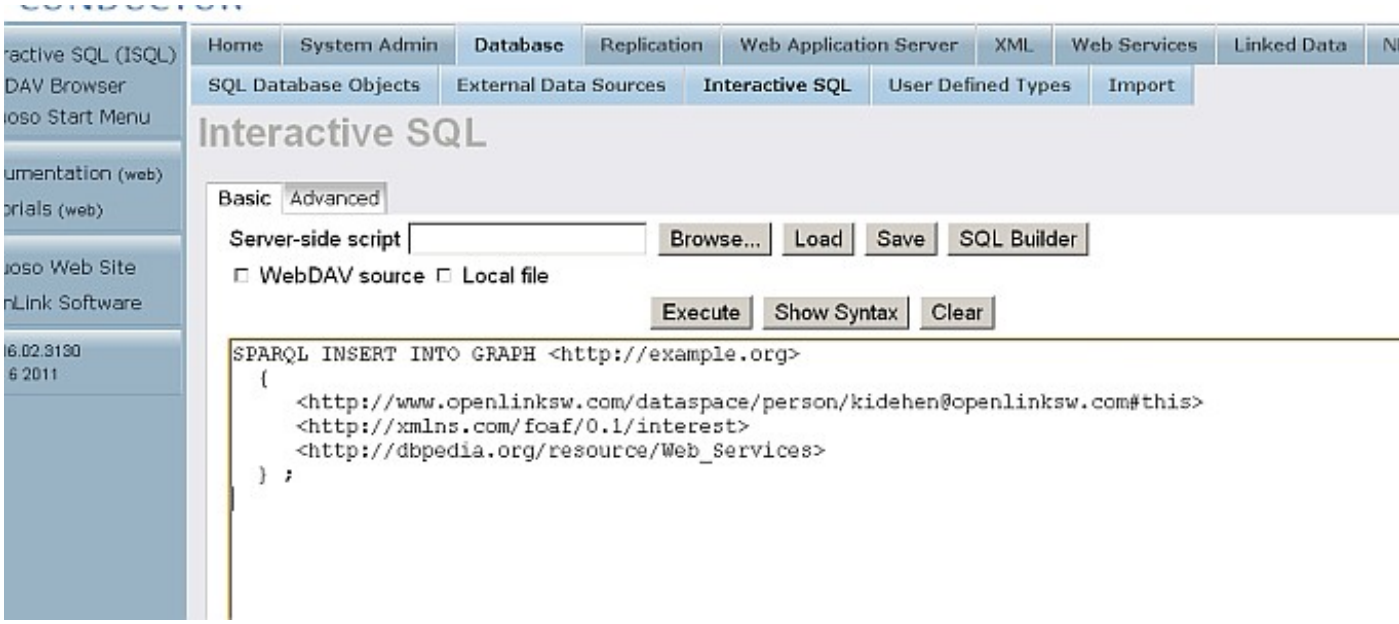
callret-0
57

Add new data on db2

1. Disconnect db1.
2. On the Host Virtuoso Instance db2 go to Conductor -> Database -> Interactive SQL enter the following statement:

```
SPARQL INSERT INTO GRAPH <http://example.org>
  {
     <http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#this>
     <http://xmlns.com/foaf/0.1/interest>
     <http://dbpedia.org/resource/Web_Services>
  } ;
```

Figure 16.302. Bi-directional Replication Topology



3. Click "Execute"
4. As result the triples will be inserted

Figure 16.303. Bi-directional Replication Topology

5. Check the count of the destination instance graph's triples by executing the following query like against the SPARQL endpoint, http://example.com:8892/sparql:

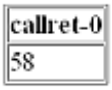
```
SELECT COUNT(*)
   FROM <http://example.org>
WHERE { ?s ?p ?o }
```

6. Should return

58

as total.

Figure 16.304. Bi-directional Replication Topology



Check data on the Destination instance db1

1. Start instance db1
2. To confirm that the triple count has increased by the number of inserted triples, execute the following statement on db1's SPARQL Endpoint:

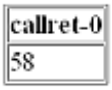
```
SELECT COUNT(*)
   FROM <http://example.org>
WHERE { ?s ?p ?o }
```

3. Should return

58

as total.

Figure 16.305. Bi-directional Replication Topology

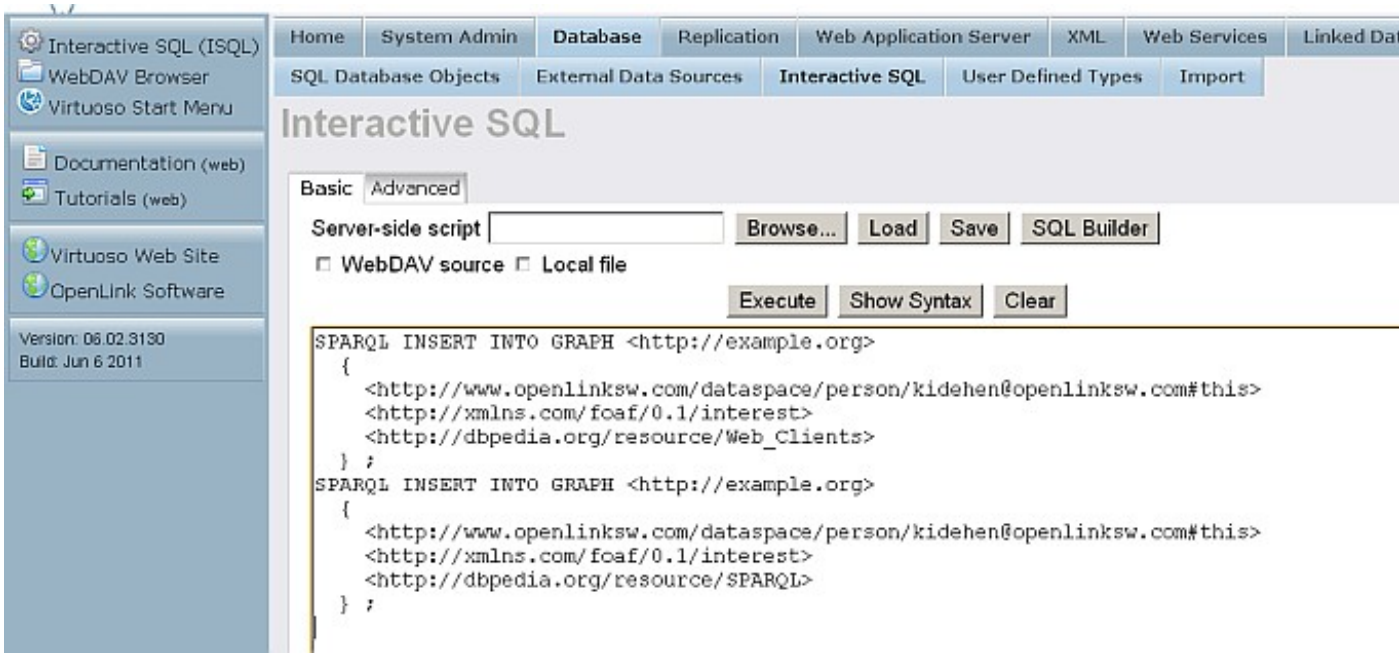


Add new data on db1

1. Disconnect db2.
2. On the Host Virtuoso Instance db1 go to Conductor -> Database -> Interactive SQL enter the following statement:

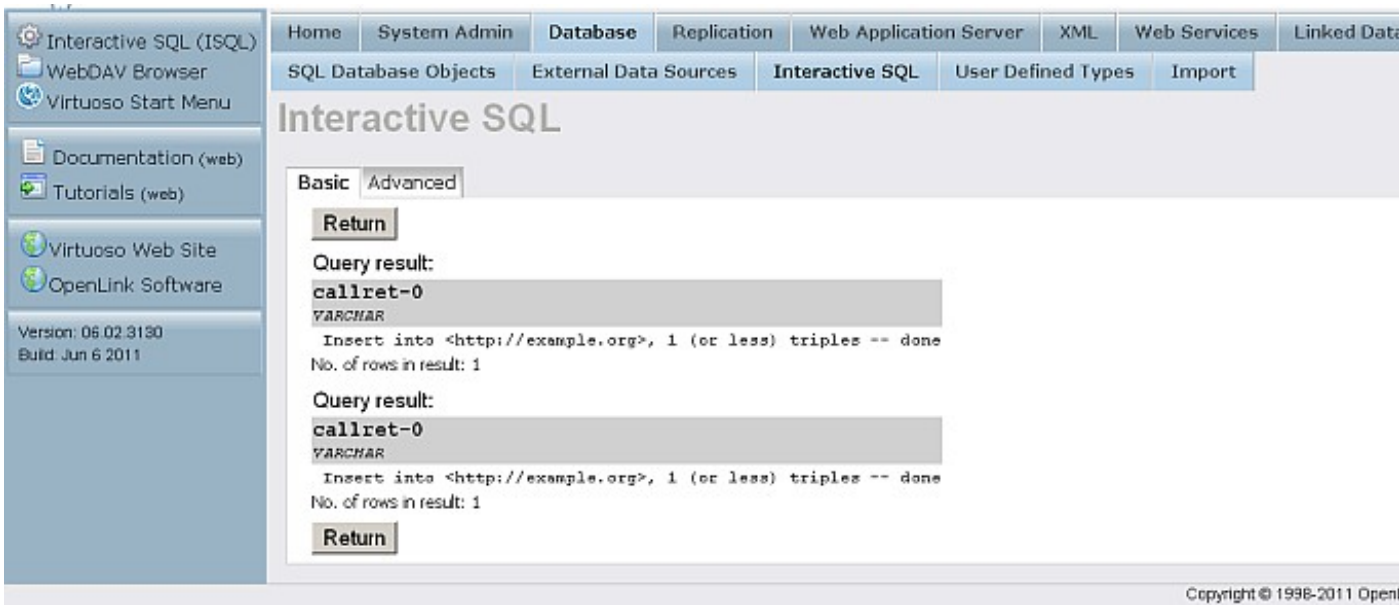
```
SPARQL INSERT INTO GRAPH <http://example.org>
  {
    <http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#this>
    <http://xmlns.com/foaf/0.1/interest>
    <http://dbpedia.org/resource/Web_Clients>
  } ;
SPARQL INSERT INTO GRAPH <http://example.org>
  {
    <http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#this>
    <http://xmlns.com/foaf/0.1/interest>
    <http://dbpedia.org/resource/SPARQL>
  } ;
```

Figure 16.306. Bi-directional Replication Topology



3. Click "Execute"
4. As result the triples will be inserted

Figure 16.307. Bi-directional Replication Topology



5. Check the count of the destination instance graph's triples by executing the following query like against the SPARQL endpoint, http://example.com:8891/sparql:

```
SELECT COUNT(*)
   FROM <http://example.org>
WHERE { ?s ?p ?o }
```

6. Should return

60

as total.

Figure 16.308. Bi-directional Replication Topology

callret-0
60

Check data on the Destination instance db2

1. Start instance db2
2. To confirm that the triple count has increased by the number of inserted triples, execute the following statement on db2's SPARQL Endpoint:

```
SELECT COUNT(*)
   FROM <http://example.org>
WHERE { ?s ?p ?o }
```

3. Should return

   60

   as total.

Figure 16.309. Bi-directional Replication Topology

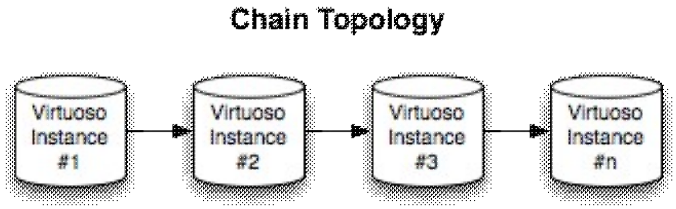
callret-0
60

16.19.3. Set up RDF Replication via procedure calls

Example

The following example shows how to use SQL procedures to set up Virtuoso RDF Graph Replication in a Chain Topology.

Figure 16.310. Chain Replication Topology



This can also be done through the HTTP-based Virtuoso Conductor .

Prerequisites

Database INI Parameters

Suppose there are 3 Virtuoso instances on the same machine.

The first instance holds the master copy of the data and publishes its changes to all other instances that subscribe to this master.

The second instance subscribes to the publication of the master copy, but also publishes all of these changes to any instance that subscribes to it.

The third instance only subscribes to the publication of the second instance.

Each of these 3 servers need unique ports and ServerName, DefaultHost for this replication scheme to work properly. Although not needed, this example also sets separate names for the database and related files. This results in the following ini parameters values (only changes are shown, the rest can remain default):

1. repl1/virtuoso.ini:

```
...
[Database]
DatabaseFile    = virtuoso1.db
TransactionFile = virtuoso1.trx
ErrorLogFile     = virtuoso1.log
...
[Parameters]
ServerPort               = 1111
SchedulerInterval        = 1
...
[HTTPServer]
ServerPort                  = 8891
...
[URIQA]
DefaultHost = example.com:8891
...
[Replication]
ServerName   = db1-r
...
```

2. repl2/virtuoso.ini:

```
...
[Database]
DatabaseFile    = virtuoso2.db
TransactionFile = virtuoso2.trx
ErrorLogFile     = virtuoso2.log
...
[Parameters]
ServerPort               = 1112
SchedulerInterval        = 1
...
[HTTPServer]
ServerPort                  = 8892
...
[URIQA]
DefaultHost = localhost:8892
...
[Replication]
ServerName   = db2-r
...
```

3. repl3/virtuoso.ini:

```
...
[Database]
DatabaseFile    = virtuoso3.db
TransactionFile = virtuoso3.trx
ErrorLogFile     = virtuoso3.log
...
[Parameters]
ServerPort               = 1113
SchedulerInterval        = 1
...
[HTTPServer]
ServerPort                  = 8893
...
[URIQA]
DefaultHost = example.com:8893
...
[Replication]
ServerName   = db3-r
...
```

Database DSNs

Use the ODBC Administrator on your Virtuoso host (e.g., on Windows, Start menu -> Control Panel -> Administrative Tools -> Data Sources (ODBC); on Mac OS X, /Applications/Utilities/OpenLink ODBC Administrator.app) to create a System DSN for each of db1, db2, db3, with names db1, db2 and db3, respectively.

Configure Publishers and Subscribers

1. Run the databases by starting start.sh, which has the following content:

```
cd repl1
virtuoso -f &
cd ../repl2
virtuoso -f &
cd ../repl3
virtuoso -f &
cd ..
```

2. Use the

   isql

   command to execute the following rep.sql file:

```
--
--  connect to the first database which is only a publisher
--
set DSN=localhost:1111;
reconnect;

--
-- start publishing the graph http://test.org
---
DB.DBA.RDF_REPL_START();
DB.DBA.RDF_REPL_GRAPH_INS ('http://test.org');

--
--  connect to the second database in the chain, which is both a publisher and a subscriber
--
set DSN=localhost:1112;
reconnect;

--
--  start publishing the graph http://test.org
--
DB.DBA.RDF_REPL_START();
DB.DBA.RDF_REPL_GRAPH_INS ('http://test.org');

--
--  contact the first database
--
repl_server ('db1-r', 'db1', 'localhost:1111');

--
--  subscribe to its RDF publication(s)
--
repl_subscribe ('db1-r', '__rdf_repl', 'dav', 'dav', 'dba', 'dba');

--
--  bring the replication service online
--
repl_sync_all();

--
--  and set scheduler to check every minute
--
DB.DBA.SUB_SCHEDULE ('db1-r', '__rdf_repl', 1);

--
--  connect to the third database in the chain, which is only a subscriber
--
set DSN=localhost:1113;
reconnect;

--
-- uncomment next 2 commands if this database should also be a publisher
--
--DB.DBA.RDF_REPL_START();
--DB.DBA.RDF_REPL_GRAPH_INS ('http://test.org');
```

```
--
--  contact second database
--
repl_server ('db2-r', 'db2', 'localhost:1112');

--
--  subscribe to its RDF publication(s)
--
repl_subscribe ('db2-r', '__rdf_repl', 'dav', 'dav', 'dba', 'dba');

--
--  bring the replication service online
--
repl_sync_all();

--
--  and set schedule to check every minute
--
DB.DBA.SUB_SCHEDULE ('db2-r', '__rdf_repl', 1);
```

Chapter 17. Web Services

Abstract

The term Web Services describes an application of XML for exposing application functionality to disparate clients via the Web. This paradigm provides access to entire applications, modules, objects, functions, and methods via HTTP and other transport protocols. Web Services are inherently platform- and programming-language independent. Services can be developed in one language and consumed in another; this holds true irrespective of client or server host operating system combinations.

A collection of core XML-based specifications and standards work in concert to fulfill the Web services value proposition. These standards include:

SOAP - XML notation that describes how messages are assembled and transmitted over HTTP between services and service consumers.

WSDL - XML notation for describing SOAP services.

UDDI - Application of the SOAP protocol for registering and publishing information about organizations, contacts within organizations, and Web Services that these organizations have chosen to expose to the public.

Virtuoso enables stored procedures to be exposed as Web services that are consumable by any Web services-aware development tool, application, or environment. The stored procedures exposed by Virtuoso may be native stored procedures, or may be hosted in any third-party database that supports stored procedures, and is accessible via an ODBC or JDBC driver. In Virtuoso, exposing stored procedures as Web services does not require any programming effort. You simply identify the stored procedures to be exposed using the Virtuoso Administrative Interface.

This feature is immensely valuable in situations where organizations are seeking to transform time-tested stored procedures into Web Services without having to upgrade or change databases or host operating systems. In modern enterprises, such stored procedures drive mission-critical solutions; Virtuoso's approach to Web services enables you to maximize current investment while minimizing the need for rework.

17.1. SOAP

The Simple Object Access Protocol (SOAP) is a lightweight, extensible, XML-based application layer protocol for information exchange in a decentralized, distributed environment. SOAP defines a framework for message structures and a message processing model. SOAP also defines a set of encoding rules for serializing data and a convention for making remote procedure calls. The SOAP extensibility model provides a foundation for a wide range of composable modules and protocols. Although the most common way to transport SOAP messages is HTTP, it may also be run on top of other protocols.

SOAP includes:

an envelope that defines a framework for describing what is in a message and how to process it

a set of encoding rules for expressing instances of application-defined datatypes

a convention for representing remote procedure calls and responses.

17.1.1. Virtuoso SOAP Support Overview

Virtuoso provides a framework for both consuming SOAP services (acting as a client) and producing them (acting as a server). The Virtuoso web server has a mechanism for handling SOAP messages and passing them to stored procedures for processing. Both SOAP 1.0 and SOAP 1.1 messages and data types are supported. You may use all base SQL data types, as well as heterogeneous arrays, as both arguments and return values of Virtuoso SOAP services. A full-featured set of functions for handling SOAP objects is provided. Services using a transport mechanism other than HTTP can also be constructed using the API. The SOAP framework may be used independently of any of the other web-related services.

Virtuoso/PL can also issue requests to SOAP servers. SOAP can be used to access any application servers, including those running within the Virtuoso server.

The Virtuoso SOAP server extends Virtuoso/PL parameter handling by adding complex data types declared with XML schema as parameter values for stored procedures. The Virtuoso SOAP server provides automatic validation of the parameters in requests, based on schema declarations.

17.1.2. Handling of SOAP HTTP Requests

The Virtuoso web server recognizes SOAP HTTP requests and their version in the POST method handler. When *SOAPMethodName* or *SOAPAction* HTTP header attributes are present with

```
Content-Type: text/xml
```

, the server initiates SOAP call handling. The XML namespace of the SOAP method name is stripped off and Virtuoso searches for a stored procedure with the same name, ignoring case.

The search is done within the default qualifier of the SQL user account assigned for SOAP call execution defined for the virtual host. For example, if the database user assigned in the virtual host's definition for SOAP execution is called SOAPDBUSER and this user has a default qualifier 'SOAPDB' and the request contains an invocation of method called

```
OurSoapMethod
```

, Virtuoso would attempt to find a stored procedure named `SOAPDB.SOAPDBUSER.OurSoapMethod`.

When a matching stored procedure is found, any of its parameters that have names matching parameter entity names in the SOAP call are bound to the call parameter. The parameter name match is also case-insensitive.

Virtuoso maps the procedure parameter datatypes internally by casting from XML data (a string) to the declared parameter datatype of the stored procedure. There is one exception: When an array is being passed, the server creates an array with values of types inferred from the XML Schema of its elements. It is possible to declare that a user defined SQL type be used to represent a

specific XML element in a SOAP request. Thus SQL objects can be constructed and serialized automatically. Note that this also means that the implementation of the user defined type instance may be in a hosted language, thus Java or CLR code may be transparently involved.

Two special parameters - *ws_soap_headers* and *ws_http_headers* - are available to a stored procedure handling a SOAP method invocation. If declared as input parameters for the procedure, *ws_soap_headers* must contain an XML parse tree of the *SOAP:Header* in same format as returned by `xml_tree()`. *ws_http_headers* should hold a one-dimensional array of attribute/value pairs representing the HTTP header fields in the request.

17.1.3. Extending Datatypes for SOAP Objects

Complex datatypes can be defined using XMLSchema and represented by WSDL. Any of the declared types may be used as arguments and return types of Virtuoso/PL procedures. Any procedures can thus be exposed as SOAP methods.

Complex data type definitions are used for values that cannot be contained by simple scalar datatypes. Typical examples are arrays of scalars, structures of scalars, arrays of structures or structures of arrays. A complex datatype may contain scalar and complex datatypes. When a complex type is used in the definition of another complex type, the definition of the contained complex type must exist.

In addition to 'usual' complex types as structures and arrays Virtuoso implements support for 'choice', 'enumeration', anyType and anyElement and extensions to the simple types. Inheritance of complex types is also possible and is discussed further in next chapter.

The 'nillable' and 'minOccurs' attributes in schema definitions have special meaning for PL values returned by PL procedure via SOAP. If this attribute is 'true' then output of NULL values will be serialized in their XML form with XMLSchema instance attribute 'nil' as 'true'. Otherwise if elements have 'minOccurs' equal to 0 (zero), the element will be omitted. If minOccurs is equal to 1 (one) an empty element will be sent to the client. The same algorithm applies to the serialization of PL values passed as parameters to soap_client() function. Therefore it's important to make proper use of these attributes when defining complex structures.

The '__VOID__' string constant has a special meaning in XMLSchema Datatypes. It is used to designate no output for return value. In other words returned value from PL procedure will not be serialized nor exposed in the WSDL file.

You define complex datatypes using `soap_dt_define()`. The function accepts a schema definition excerpt, based on the element *complexType*. The definition must be a valid XML document.

Example 17.1. Declaring and using complex datatypes in SOAP

In this example we define two complex datatypes. The first one, SOAPStruct , consists of scalars; the second one, ArrayOfSOAPStruct , is an array of these structures. These schema excerpts are stored in the filesystem as `struct.xsd` and `array.xsd`.

struct.xsd:

```
<!-- a SOAPStruct type declaration
     file name: struct.xsd -->

<complexType name="SOAPStruct"
   targetNamespace="http://tempuri.tmp/"
   xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:tns="services.wsdl">

   <sequence>
     <element name="varString" type="string" nillable="true"/>
     <element name="varInt" type="int" nillable="true"/>
     <element name="varFloat" type="float" nillable="true"/>
   </sequence>
</complexType>
```

array.xsd:

```
<!-- array of SOAPStruct
     file name: array.xsd -->

<complexType name="ArrayOfSOAPStruct"
   targetNamespace="http://tempuri.tmp/"
   xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:tns="services.wsdl">

   <complexContent>
   <restriction base="enc:Array">
   <sequence>
   <element name="item" type="tns:SOAPStruct" minOccurs="0" maxOccurs="unbounded"/>
   </sequence>
   <attribute ref="enc:arrayType" wsdl:arrayType="tns:SOAPStruct[]"/>
   <attributeGroup ref="enc:commonAttributes"/>
   <attribute ref="enc:offset"/>
   </restriction>
   </complexContent>
</complexType>
```

Next, we issue commands to define a new complex datatype. Correct order is important. (SQL> is the prompt of the Interactive SQL utility included with Virtuoso and should not be typed)

```
SQL> DB..soap_dt_define ('SOAPStruct', file_to_string ('struct.xsd'));
SQL> DB..soap_dt_define ('ArrayOfSOAPStruct', file_to_string ('array.xsd'));
```

Note:

The WSDL specification requires that array names be prefixed with `ArrayOf` .

17.1.4. Inheritance of Datatypes for SOAP Objects

The Virtuoso SOAP server implements handling of inherited XSD types. The simple example of such relation between types can be explained as

```
   Type  A, also know as the 'base' type,
   and type B an extension of A.
   +---+   +---+
   | a |-->

| c |
   | b |   | d |
   +---+   +---+

   which can be defined by two separate types without relation

   A type   B type
   +---+   +---+
   | a |   | a |
   | b |   | b |
   +---+   +- -+
           | c |
           | d |
           +---+
```

But when type A has changed, type B will not be changed in second representation. This is because B is not a relative to A per se.

To work in such situations Virtuoso SOAP server handles extensions to XSD types as follows:

1. each type and base type have defined a User Defined SQL type (UDT).
2. the XSD types defined for SOAP processing are defined with UDT relation (see soap_dt_define)
3. the inheritance is declared with 'extension' element in XSD type declaration

When we have these preliminaries the WSDL will declare in 'schema' part of WSDL all depending types. Furthermore the SOAP processor will handle inherited members of derived types.

Example 17.2. Declaration and usage of depending types

Consider the following XSD and User Defined Type declaration for a base type 'BaseStruct':

```
<!-- XSD type declaration, file base.xsd -->

<complexType name="BaseStruct">
        <sequence>
                <element name="floatMessage" type="xsd:float"/>
                <element name="shortMessage" type="xsd:short"/>
        </sequence>
</complexType>

-- corresponding user defined sql type
create type DB.DBA.BaseStruct as (floatMessage real, shortMessage int __soap_type 'short');
```

Furthermore we are extending the BaseStruct with adding three more elements (members) with declaration of ExtendedStruct:

```
<!-- XSD type declaration, file ext.xsd -->

<complexType name="ExtendedStruct">
        <complexContent>
                <extension base="tns:BaseStruct">
                        <sequence>
                                <element name="stringMessage" type="xsd:string"/>
                                <element name="intMessage" type="xsd:int"/>
                                <element name="anotherIntMessage" type="xsd:int"/>
                        </sequence>
                </extension>
        </complexContent>
</complexType>

 -- corresponding user defined SQL type
create type DB.DBA.ExtendedStruct under DB.DBA.BaseStruct as (
     stringMessage nvarchar __soap_type 'string',
     intMessage int __soap_type 'int',
     anotherIntMessage int __soap_type 'int');
```

Once we are done with declarations as XSD files and user defined SQL types, we must register them as SOAP types for processing:

```
SQL> soap_dt_define ('', file_to_string ('base.xsd'), 'DB.DBA.BaseStruct');
SQL> soap_dt_define ('', file_to_string ('ext.xsd'), 'DB.DBA.ExtendedStruct');
```

Now we are able to create a PL procedure to use as a SOAP method, which simply will accept an ExtendedStruct and echo it back to the client.

```
create procedure
echoExtendedStruct (in param DB.DBA.ExtendedStruct __soap_type 'http://soapinterop.org/types:ExtendedStru
returns DB.DBA.ExtendedStruct __soap_type 'http://soapinterop.org/types:ExtendedStruct'
{
  --  All members  of DB.DBA.ExtendedStruct and DB.DBA.BaseStruct are available in param.
  return param;
}
;

grant execute on echoExtendedStruct to SOAP;
```

The SOAP request to that method will be as follows:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
```

```
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:m0="http://soapinterop.org/types">
  <SOAP-ENV:Body>
    <m:echoExtendedStruct xmlns:m="http://soapinterop.org/wsdl">
      <param xsi:type="m0:ExtendedStruct">
        <floatMessage xsi:type="xsd:float">3.14159</floatMessage>
        <shortMessage xsi:type="xsd:short">4096</shortMessage>
        <stringMessage xsi:type="xsd:string">String</stringMessage>
        <intMessage xsi:type="xsd:int">0</intMessage>
        <anotherIntMessage xsi:type="xsd:int">0</anotherIntMessage>
      </param>
    </m:echoExtendedStruct>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The SOAP response to the above request will be as follows:

```
<SOAP:Envelope
SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:dt="urn:schemas-microsoft-com:datatypes"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/07/secext"
xmlns:ref="http://schemas.xmlsoap.org/ws/2002/04/reference/"
xmlns:ns0="http://soapinterop.org/types" xmlns:wsdl="services.wsdl">

  <SOAP:Body>
    <cli:echoExtendedStructResponse xmlns:cli="http://soapinterop.org/wsdl">
      <CallReturn xsi:type="ns0:ExtendedStruct">
        <stringMessage xsi:type="xsd:string">String</stringMessage>
        <intMessage xsi:type="xsd:int">0</intMessage>
        <anotherIntMessage xsi:type="xsd:int">0</anotherIntMessage>
        <floatMessage xsi:type="xsd:float">3.14159</floatMessage>
        <shortMessage xsi:type="xsd:short">4096</shortMessage>
      </CallReturn>
    </cli:echoExtendedStructResponse>
  </SOAP:Body>
</SOAP:Envelope>
```

Note:
Although the namespace declarations of XSD types are skipped for better readability, these must be present when declaring (see the Extending Datatypes for SOAP Objects section, discussed earlier)

17.1.5. Complex Types in PL Procedure and UDT Method Definition

Virtuoso/PL allows parameters to be declared as complex objects (structures and arrays) without special XMLSchema datatype defined. To declare a structure as a type of a parameter an UDT must be created and parameter to have it as datatype reference. Also all permitted datatypes (including UDTs) could be declared as elements of an ARRAY of unlimited or limited length.

Important: when a UDT is used in a SOAP context, it MUST be granted to the SQL user for SOAP invocation. In other words the user on whose behalf the SOAP call is processed.

Example 17.3. Procedure definition with a input and output as a structure

The following example defines a UDT 'SOAP_Struct' (containing varchar, integer and float members) and declares the input parameter and return value of a PL procedure to be of the SOAP_Struct type. The input will be verified, UDT will be instantiated with given values for members and it will be echoed back to the client.

```
create type SOAP_Struct as (varString varchar, varInt integer, varFloat real);
```

```
create procedure echoStruct (in s DB.DBA.SOAP_Struct) returns DB.DBA.SOAP_Struct
{
  return s;
};
```

Example 17.4. Procedure definition with a input and output as an integer array

This example declares that input must be an array of integer values with maximum length of 5. If input or output contains more than five integers then a SOAP Fault will be sent back to the client containing an appropriate error message ; otherwise the input array will be echoed back.

```
create procedure echoIntArray (in ia integer array[5]) returns integer array[5]
{
  return ia;
};
```

Example 17.5. Procedure definition with a input and output as a two-dimensional varchar array

This example declares that the input must be an array of integer array values with unlimited length. If the input SOAP message contains a valid array following the current XML encoding rules then an array of integer arrays (vector containing vectors of integers) will be created and passed to the procedure. On success the input array will be echoed back to the client.

```
create procedure echoIntMulArray (in iaa integer array array) returns integer array array
{
  return iaa;
};
```

Example 17.6. Procedure definition with a input and output as an struct array

This example shows how to use an array of structures (UDTs) and also shows usage of the array type as an member of the structure. The UDT 'SOAP_StructA' is similar to the those in first example except 4the member which is an array of integers. This is to demonstrate that arrays are not limited to the Stored Procedure's parameters declaration, they also can be used as a type of UDT member. Upon success the procedure will echo of the input back to the client.

```
create type SOAP_StructA as (varString varchar, varInt integer, varFloat real, varArray integ

create procedure echoStructArray (in sa DB.DBA.SOAP_StructA array) returns DB.DBA.SOAP_Struct
{
  return sa;
};
```

The SOAP request to an endpoint which exposes the echoStructArray as a document/literal encoded SOAP method would be as follows:

```
<?xml version="1.0" ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <ns0:echoStructArray xmlns:ns0="http://temp.uri">
      <sa>
        <item>
          <varString>abcd</varString>
          <varInt>1234</varInt>
          <varFloat>3.14</varFloat>
          <varArray>
            <item>3</item>
            <item>4</item>
```

```
            </varArray>
          </item>
        </sa>
      </ns0:echoStructArray>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The SOAP server will receive and array of one element containing a structure with string, integer, float and integer array of two elements. Then the response from the SOAP server to the requestor will be:

```
<?xml version="1.0" ?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP:Body>
    <cli:echoStructArrayResponse xmlns:cli="http://temp.uri">
      <CallReturn>
        <item>
          <varString>abcd</varString>
          <varInt>1234</varInt>
          <varFloat>3.14</varFloat>
          <varArray>
            <item>3</item>
            <item>4</item>
          </varArray>
        </item>
      </CallReturn>
    </cli:echoStructArrayResponse>
  </SOAP:Body>
</SOAP:Envelope>
```

See also the WSDL file generation section for details how such PL procedures with parameters of complex datatypes are exposed via SOAP enabled virtual HTTP directories.

17.1.6. Complex Types in Procedure Definition using a pre-defined XML Schema datatypes

Declaration of a complex datatype as a parameter is done by adding a special keyword `__soap_type` followed by the name of the defined complex type after normal parameter declaration in the parameter list. The type name is given as a string literal. The same syntax extension also applies to declaration of the return type. This is shown in the following example.

Example 17.7. Procedure Definition with Complex Datatype Parameters

We create a procedure that will accept an array of structures (as defined in the previous example) and return it to the client. It instructs the WSDL generator to assign ArrayOfSOAPStruct as the input parameter and return value types when `WS.SOAP.echoSOAPArray()` is exposed as a SOAP method. The type information is available to SOAP clients that read the WSDL description. Upon receiving an incoming SOAP request, Virtuoso converts the XML representation of the data, after validation, to the form

```
vector(vector([varchar],[integer],[real]), ...)
```

and passed to the `WS.SOAP.echoSOAPArray` . Failed parameter validation is reported to the client.

```
SQL> CREATE PROCEDURE WS.SOAP.echoSOAPArray (in inArray any __soap_type 'ArrayOfSOAPStruct')
             RETURNS any __soap_type 'ArrayOfSOAPStruct'
    {
      return inArray;
    };
```

17.1.7. Default SOAP-SQL Datatype Mappings

When no alternative datatype is assigned, the WSDL generator and SOAP server will use the default mapping described below:

Table 17.1. Default datatype mappings in SOAP

Datatype	Maps to
integer	xsd:int
real	xsd:float
double precision	xsd:double
numeric	xsd:decimal
datetime	xsd:timeInstant
any other type	xsd:string

The REAL SQL type is mapped to the xsd:float SOAP datatype by default and so loss of precision can occur. To improve the precision, the SOAP server will map the xsd:float to the PL double precision datatype instead, but only if the SOAP type is specified. The explicit declaration of __soap_type 'xsd:float' is required to instruct Virtuoso to use the mapping to double precision.

All strings from a SOAP request declared with the SOAP datatype xsd:string will be treated as NVARCHARs on input. All string data such a CHAR, VARCHAR, or NVARCHAR will be encoded as UTF-8 in a SOAP response. This makes processing of wide character sets in SOAP operations possible.

If a User Defined Type (UDT) is used as a type of parameter and no explicit XML Schema datatype given (see special syntax for PL procedures) then in WSDL will be included as a struct definition. Further upon SOAP processing the input struct will be encoded as a UDT instance and passed to the given PL procedure.

The parameters which are declared as an array (see PL procedure syntax) and having no explicit XML Schema datatype given will be exposed as array by means of SOAP encoding rules (see also 'Use' SOAP option to the virtual directory).

Some SOAP applications need a void return as opposed to an empty return, from SOAP operations. To distinguish the empty return from the void return a special SOAP datatype '__VOID__' has been introduced. This will cause the SOAP server to omit the procedure return value when responding to a SOAP request. Also, the return message will be discarded from the WSDL description file.

17.1.8. Exposing Stored Procedures as SOAP Objects

The special physical path `/SOAP/` in the Virtuoso Web server is reserved for SOAP objects. Virtuoso makes available any stored procedure created in the default qualifier of the SOAP user, with execution privileges granted to the SOAP user. You can also use Virtuoso's virtual host mechanism to create new logical paths for accessing SOAP objects. A logical path property `soap_user` determines the db user for SOAP. If a logical path points to the `/SOAP/` special physical path, it will expose any procedures created in the default qualifier of, and with execution privileges to, `soap_user` to the world as SOAP objects.

If the physical path of `/SOAP` exists under the VSP root directory then any non-SOAP specific HTTP requests will be directed there for content. This can be useful for helping to establish the presence and location of a SOAP endpoint - some applications attempt a standard HTTP connection first. You might configure a virtual directory, intended for SOAP, with a default page referencing a description of the SOAP endpoint, a page in the <VSPROOT>/SOAP directory, preventing an `HTTP 404` style error misleading an application into believing the SOAP endpoint is down regardless of whether it tried to talk SOAP to it or not.

Note:
Procedures exposed as SOAP procedures run as any other stored procedure in Virtuoso and can call and get return values from other procedures and functions not exposed through SOAP. The ability to execute procedures attached from remote data sources facilitates SOAP-enabling existing database applications in a heterogeneous environment.

Example 17.8. Creating a new virtual host for SOAP execution

Create new user in the database for SOAP:

```
SQL>CREATE USER SOAPDEMO;
```

Set the default catalogue/qualifier for the new user to WS. This is where procedures to be used as SOAP objects will be created:

```
SQL>USER_SET_QUALIFIER ('SOAPDEMO', 'WS');
```

Create a new virtual host definition, using `vhost_define()` .

```
SQL>VHOST_DEFINE (vhost=>'*ini*',lhost=>'*ini*',lpath=>'/mysoapdomain',ppath=>'/SOAP/',soap_user=>'SOAPDE
```

An existing mapping could be removed using the command:

```
SQL>VHOST_REMOVE (vhost=>'*ini*',lhost=>'*ini*',lpath=>'/mysoapdomain')
```

Note:
`'*ini*'` is a special value that instructs Virtuoso to use the default values from the Virtuoso initialization file.

All procedures that are created with the WS.SOAPDEMO qualifier and then granted execution to SOAPDEMO will be visible to SOAP. Make a simple SOAPTEST procedure and grant the appropriate privileges to the SOAPDEMO user:

```
SQL> create procedure
  WS.SOAPDEMO.SOAPTEST (in par varchar)
{
  return (upper(par));
};

SQL> grant execute on WS.SOAPDEMO.SOAPTEST to SOAPDEMO;
```

The SOAP object may now be tested by using the `soap_client()` function, which returns a vector representation of the SOAP object returned by the call. The example below simply extracts the returned string with `aref()` , as the exact format of the object returned is known:

```
SQL>select aref(aref(
        soap_client (url=>sprintf ('http://example.com:%s/mysoapdomain', server_http_port ()),
        operation=>'SOAPTEST',
        parameters=>vector('par', 'demotext')),
        1), 1);
callret
VARCHAR
_______

DEMOTEXT
```

Printing the output on the console or server log with `dbg_obj_print()` would output something like:

```
(("SOAPTESTResponse" ) (("CallReturn" ) "DEMOTEXT" ) )
```

The automatic service description generation can be verified by retrieving *http://<server:port>/mysoapdomain/services.wsdl* , and preferably tested by pointing a web browser at *http://<server:port>/mysoapdomain/services.vsmx*

```
SQL> select http_get (sprintf ('http://example.com:%s/mysoapdomain/services.wsdl', server_http_port()));
callret
VARCHAR
_______________________________________________________________________________

<?xml version="1.0"?>
<definitions
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
 xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:s="services.wsdl"
 xmlns:tns="services.wsdl"
 targetNamespace="services.wsdl"
 name="VirtuosoSOAP" xmlns="http://schemas.xmlsoap.org/wsdl/">

  <types>
  <schema targetNamespace="services.wsdl"
   xmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    <complexType name="echoStringArrayResponse">
      <sequence>
        <element name="return" type="ArrayOfstring_literal"/>
      </sequence>
    </complexType>
```

```
<complexType name="echoVoid"/>
<complexType name="ArrayOffloat">
  <complexContent>
    <restriction base="soapenc:Array">
      <sequence>
        <element name="item" type="float" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
      <attributeGroup ref="soapenc:commonAttributes"/>
      <attribute ref="soapenc:offset" />
      <attribute ref="soapenc:arrayType" wsdl:arrayType="float[]"/>
    </restriction>
  </complexContent>
</complexType>
<complexType name="SOAPStruct">
  <sequence>
    <element name="varString" type="string"/>
    <element name="varInt" type="int"/>
    <element name="varFloat" type="float"/>
  </sequence>
</complexType>
<complexType name="echoStructResponse">
  <sequence>
    <element name="return" type="SOAPStruct"/>
  </sequence>
</complexType>
<complexType name="echoVoidResponse"/>
<complexType name="ArrayOfString2D">
...
```

See Also:

Testing Web Services using VSMX

17.1.9. Creation of SOAP proxy based on User Defined Types

It is possible to automatically generate PL procedures or UDT classes for invoking a remote SOAP service.

See Also

The `WSDL_IMPORT_UDT()` function for details and examples.

The proxy-creation function `WSDL_IMPORT_UDT()` performs the following purposes:

retrieve and expand the WSDL file published by the end point to be called

compile the result and make SQL script with UDT definition

generate and register XML Schema definition for special types used in the source service

optionally execute the SQL script generated

Once such UDT SOAP proxy is defined it can be used within application code or be re-exposed as a SOAP service on local server instance (see next chapter how to expose UDT as service).

See Also

The Virtuoso Administration Interface provides a web based interface for importing WSDL definitions and creating UDTs and procedures. This can be found in the Virtuoso Server Administration Interface Chapter.

17.1.10. Exposing User Defined Type Methods as SOAP Objects

SQL User Defined Types may define methods. In context of Virtuoso SOAP server they can be exposed as SOAP methods. To do that the UDT must be published at an endpoint. So publishing could be done in two ways: using SQL INSERT statement or using Admin UI: Publishing UI via Virtual directories section.

The published UDTs will then expose all methods to the given virtual directory assigned for SOAP execution. In this case the default constructor will be called for method invocation if the UDT method is non-static.

Note: The method definitions may also contains special SOAP syntax for XML Schema datatypes, using the same options as for PL procedures. (see "PL Procedures and UDT Methods Syntax Affecting WSDL & SOAP Processing" section for details)

The following table specifies which UDTs are published at which end points.

```
create table SYS_SOAP_UDT_PUB
            (SUP_CLASS varchar, -- name of the published UDT, referencing SYS_USER_TYPES.UT_NAME
             SUP_LHOST varchar, -- listen host, referencing HTTP_PATH.HP_LISTEN_HOST
             SUP_HOST varchar,  -- virtual host, referencing HTTP_PATH.HP_HOST
             SUP_END_POINT varchar, -- logical path, referencing HTTP_PATH.HP_LPATH
             primary key (SUP_LHOST, SUP_HOST, SUP_END_POINT, SUP_CLASS))
;
```

Example 17.9. Exposing a UDT Method using SQL statement

The below code creates a UDT containing two methods: static and non-static and exposes them on a virtual directory '/soap-udt'

```
create user SOAP_U2;

VHOST_DEFINE (lpath=>'/soap-udt', ppath=>'/SOAP/', soap_user=>'SOAP_U2',
    soap_opts=>
    vector ('ServiceName', 'UDT',
            'Namespace', 'http://temp.uri',
            'SchemaNS', 'http://temp.uri',
            'MethodInSoapAction', 'yes',
            'elementFormDefault', 'unqualified',
            'Use', 'encoded')
);

create type MyWebSvc
static method echoStatInt (in a int) returns int,
method echoInt (in a int) returns int;

create static method echoStatInt (in a int)
returns int for MyWebSvc
{
  return a;
}
;

create method echoInt (in a int)
returns int for MyWebSvc
{
  return a;
}
;

-- Important: without grant publishing is not final as
-- user for SOAP invocation will not have permissions to instantiate the UDT nor
-- to call its methods
grant execute on MyWebSvc to SOAP_U2;

-- exposing the UDT methods to the /soap-udt endpoint
insert soft SYS_SOAP_UDT_PUB values ('MyWebSvc', '*ini*', '*ini*', '/soap-udt');
```

Exposing the methods of a UDT could be done using Admin UI/Virtual Directories: Create a new or edit an existing SOAP enabled virtual directory and navigate to the SOAP options section, click on the 'Publish' button and from presented list of Database qualifiers select the qualifier containing target UDT, then select it from the User Defined Types list and follow the wizard.

17.1.11. Exposing Remote Third Party SQL Stored Procedures as SOAP Services

Virtuoso can expose any of its available PL resources to the SOAP world. This includes data from remote attached tables and procedures. To do this, one needs to write a wrapper procedure in Virtuoso/PL.

Example 17.10. Exposing a MS SQL Server procedure to SOAP using Virtuoso

Here we have a sample MS SQL Server procedure and an accompanying Virtuoso wrapper function. The MS SQL Server function returns a result set based on a simple join query with a filter input. The Virtuoso procedure calls the remote procedure, iterates through the result set returned and produces XML output. First the MS SQL Server procedure:

```
create procedure ms_remote
        @mask varchar(15)
as
  select c.CustomerID, c.CompanyName, o.OrderDate,
      o.ShippedDate,ol.ProductID, ol.Quantity, ol.Discount
    from Northwind..Customers c
      inner join Northwind..Orders o on c.CustomerID = o.CustomerID
      inner join Northwind.."Order Details" ol on o.OrderID = ol.OrderID
    where c.CustomerID like @mask
;
```

Then the Virtuoso wrapper function:

```
create procedure WS.SOAP.ms_remote_call (
  in dsn varchar, in uid varchar, in pwd varchar, in mask varchar)
{
  declare m, r, ses any;
  vd_remote_data_source (dsn, '', uid, pwd);
  rexecute (dsn, 'ms_remote ?', null, null, vector (mask), 1000, m, r);
  ses := string_output ();
  http ('<?xml version="1.0" ?>\n<remote>\n', ses);
  if (isarray(m) and isarray (r))
    {
      declare i, l, j, k integer;
      declare md, rs any;
      md := m[0];
      i := 0; l := length (md); k := length (r); j := 0;
      while (j < k)
       {
         http ('<record ', ses);
         i:=0;
         while (i < l)
           {
             dbg_obj_print (md[i][0],r[j][i]);
             http (sprintf (' %s="%s"', trim(md[i][0]), trim(cast (r[j][i] as varchar))), ses);
             i := i + 1;
           }
         http (' />\n', ses);
         j := j + 1;
       }
    }
  http ('</remote>', ses);
  return string_output_string (ses);
};
```

Now, as before, we grant execute rights to the SOAP user:

```
grant execute on WS.SOAP.ms_remote_call to SOAP;
```

The remote procedure `ms_remote()` can now be accessed via SOAP.

See Also:
The Virtual Database chapter for information regarding use of remote datasources and their tables.

17.1.12. Virtuoso/PL SOAP Client

Virtuoso has generic SOAP client functionality. This was demonstrated in an example above, where we showed that we had correctly exposed a stored procedure as a SOAP object. The entry point to the SOAP client is `soap_client ()`.

See Also:
Importing A WSDL File

17.1.13. Execution Privileges

Virtual directory mappings allow you to define a specific database user on behalf of which to execute code invoked via SOAP. By default Virtuoso disables SOAP calls unless the database account 'SOAP' exists or a virtual directory mapping is defined for SOAP call execution. If we map a logical HTTP path to `/SOAP` and specify the user 'demo' as the SOAP user then stored procedures or UDT methods will be executed with demo's privileges.

17.1.14. Custom Soap Server Support

Virtuoso allows any VSP page to act as a SOAP endpoint. This permits preprocessing of the SOAP requests to extract additional information - such as one placed for ebXML - and conversion of the SOAP replies to put any additional information in them. SOAP messages with attachments can also be processed this way.

SOAP extensions, such as the ones required for ebXML, can be programmed as VSP services that can handle the additional information contained in the SOAP requests. The `xpath_eval()` function is useful here. The SOAP server could be called after removing extension information; this removal could be done with an XSL transformation. After the SOAP request is processed, additional information can be placed in the result by another XSL transformation.

Having a SOAP server outside the `/SOAP` physical path allows a greater degree of control over what procedures are executed by providing a list of mappings. Having this suite of functions allows SOAP requests to be processed outside an HTTP context (for example after doing `mime_tree()` over an e-mail) and sending the SOAP replies as SMTP messages.

The following built-in functions are relevant in this context:

`soap_server()`

`soap_make_error()`

`soap_box_xml_entity()`

`soap_print_box()`

`http_body_read()`

Example 17.11. Sample SOAP 1.1 server

```
<?vsp
        dbg_obj_print ('vspsoap called');
        declare content_type, soap_method, soap_xml varchar;
        declare payloads any;

        -- get the encoding to find out where the SOAP request should be searched for

        content_type := http_request_header (lines, 'Content-Type');
        if (isstring (content_type))
           content_type := lower (content_type);

        -- get the SOAP method name to execute

        soap_method := http_request_header (lines, 'SOAPAction');
        soap_xml := NULL;
        payloads := NULL;

        -- get the SOAP request
        if (content_type = 'multipart/related')
          {
            -- as in SOAP messages with attachments
            declare attrs any;
            declare inx integer;
            declare start_req varchar;

            -- the SOAP body is in the root part
            -- so get the root part's name
            start_req := http_request_header (lines, 'Content-Type', 'start');

            -- loop over the parts and get the root one.
```

```
    -- Others are placed in the payload array

    inx := 1;
    soap_xml := null;
    attrs := vector (1);
    while (isarray (attrs))
     {
       declare content_id varchar;

       -- get the part's MIME header
       attrs := get_keyword (sprintf ('attr-mime_part%d', inx), params);

       if (isarray (attrs))
         {
           -- extract the Content-ID from it
           content_id := get_keyword ('Content-ID', attrs);
           dbg_obj_print ('cont-id', content_id);

           if (isstring (content_id))
             {
               -- if it is the root part (SOAP request) parse it.
               if (content_id = start_req)
                 soap_xml := xml_tree_doc (xml_tree (
              get_keyword (sprintf ('mime_part%d', inx), params)));
               else
                 {
                   -- otherwise consider it a payload and store a info about the payload
                   -- for later retrieval by get_keyword () VSE based on Content-ID
                   if (payloads is null)
                     payloads := vector (vector (content_id, inx));
                   else
                     payloads := vector_concat (payloads, vector (content_id, inx));
                 }
             }
         }
       inx := inx + 1;
     }
  }
else if (content_type = 'text/xml')
  {
    -- it's a SOAP request without attachments
    -- so get the POST body and parse it.
    soap_xml := xml_tree_doc (xml_tree (http_body_read ()));
  }
else
  signal ('42000', 'unsupported encoding');

-- the things retrieved so far
dbg_obj_print ('vspsoap message', soap_xml);
dbg_obj_print ('vspsoap payloads', payloads);

-- execute the message

-- catch any subsequent SQL error and generate and return SOAP reply XML for it.

declare exit handler for SQLSTATE '*' {
  dbg_obj_print ('vspsoap in error handler for ', __SQL_MESSAGE);
  declare err_msg varchar;
  err_msg := soap_make_error ('300', __SQL_STATE, __SQL_MESSAGE);
  dbg_obj_print ('vspsoap error', err_msg);
  http (err_msg);

  -- note the SOAP SQL state - this is required since based on this value the
  -- HTTP server will not generate any additional reply if the SQL state starts with SOAP
  -- and this way the client will get a properly formatted reply
  resignal 'SOAP';
};

-- now check what is required and act accordingly
if (soap_method = 'ebXML')
  {
    signal ('42000', 'ebXML not implemented yet');
  }
```

```
        else if (soap_method in ('fake#test'))
          {
            declare res any;

            -- note the mapping here : the SOAP call to fake:test will result in a
            -- call to DB.DBA.SOAPTEST PL procedure and it's results returned.

            res := soap_server (soap_xml, soap_method, lines, 11,
                    vector ('fake:test', 'DB.DBA.SOAPTEST'));

            dbg_obj_print ('vspsoap result', res);
            http (res);
          }
        else
          {
            -- simple signal will do as this will be cached by the handler
          -- and formatted as an SOAP error XML
            signal ('42000', concat ('Procedure ', soap_method, ' not defined'));
          }
?>
```

17.1.15. PL Procedures and UDT Methods Syntax Affecting WSDL & SOAP Processing

Special PL syntax can be applied to any of the parameters (including the return value) in a declaration. All of these begins with __SOAP_ prefix and have special meaning. To manipulate more than the XMLSchema type representation and SOAP encoding style, extended syntax is available. With this syntax we can further override the default request/response namespace, name of the output elements, "soapAction" corresponding to the PL procedure and such.

The syntax is as follows:

```
   ...
   CREATE (PROCEDURE|METHOD) ([param_decl [rout_alt_type]] ...) { [BODY] } [RETURNS ....] [rout_alt_type]
   ...

rout_alt_type
        :  /* no SOAP options */
        | soap_kwd STRING opt_soap_enc_mode     /* the basic syntax */
        | __SOAP_OPTIONS '(' soap_kwd EQUALS STRING opt_soap_enc_mode ',' soap_proc_opt_list ')'/* extend
        ;

soap_proc_opt_list
        : soap_proc_opt
        | soap_proc_opt_list ',' soap_proc_opt
        ;

soap_proc_opt /* extension options as PartName:='part2' */
        : NAME EQUALS signed_literal
        ;

soap_kwd
        : __SOAP_TYPE   /* denotes XML datatype, RPC encoding style if applied to the procedure */
        | __SOAP_HEADER /* the parameter is a message in the SOAP Header */
        | __SOAP_FAULT  /* the parameter is a message in SOAP Fault */
        | __SOAP_DOC    /* applies to the procedure, free-form of encoding (literal) */
        | __SOAP_XML_TYPE /*applies to the parameters, the input will be XML tree */
        | __SOAP_DOCW           /* applies to the procedure, literal encoding in style like RPC */
        | __SOAP_HTTP           /* HTTP GET/POST binding will be used */
        ;

opt_soap_enc_mode               /* which part of traffic will be encapsulated and in what way : DIME or M
        : /* no encapsulation */
        | __SOAP_DIME_ENC IN
        | __SOAP_DIME_ENC OUT
        | __SOAP_DIME_ENC INOUT
        | __SOAP_MIME_ENC IN
        | __SOAP_MIME_ENC OUT
        | __SOAP_MIME_ENC INOUT
        ;
```

```
param_decl
        : (IN|OUT|INOUT) param_name data_type_ref [(DEFAULT|:=) literal]
        ;

data_type_ref
        : (data_type_name|udt_name) [ARRAY [intnum] ...]
        ;
```

The above syntax can be applied to the parameter and to the whole procedure, so both places designate different purposes and limitations. When it is applied to the parameter the following keywords can be used: __SOAP_TYPE, __SOAP_HEADER, __SOAP_FAULT and __SOAP_XML_TYPE. The __SOAP_TYPE means that only XSD type will be used to interpret the data, in contrast __SOAP_XML_TYPE designates no deserialization from XML, only parses the parameter XML representation to XML tree and passes it to the procedure. The __SOAP_HEADER and __SOAP_FAULT designate that parameter will be exposed in the SOAP Header or in the SOAP Fault elements. In the second case, that parameter needs to be an 'OUT' parameter (not IN or INOUT). The string after these keywords always denotes the XSD type for SOAP serialization. When it is applied to the PL procedure (after procedure's body), the __SOAP_TYPE, __SOAP_DOC, __SOAP_DOCW, __SOAP_HTTP, __SOAP_DIME_ENC and __SOAP_MIME_ENC can be used. The string after these keywords always denotes the XSD type for SOAP serialization, except __SOAP_DIME_ENC and __SOAP_MIME_ENC which are used for other purposes and can be combined with other keywords. The __SOAP_TYPE denotes RPC style encoding, __SOAP_DOC for document literal (bare parameters) encoding, __SOAP_DOCW for the free-form literal (wrapped) encoding. __SOAP_HTTP is used to denote HTTP style binding instead of SOAP one, in that way procedure can be called via HTTP GET/POST methods without SOAP XML encoding.

The following keywords are supported as extended options:

PartName - changes the name of a OUT parameter to the string as specified, affects WSDL generation and SOAP serialization.

RequestNamespace - designate namespace for the message in the request, affects header, fault and body WSDL declaration, and serialization of SOAP in RPC encoding style.

ResponseNamespace - the same as RequestNamespace, but for SOAP response and output in WSDL declaration.

soapAction - sets the 'soapAction' attribute in WSDL generation, can be applied to the procedure only.

The RequestNamespace and ResponseNamespace can be used only for the procedure and together with the __SOAP_FAULT and __SOAP_HEADER keywords.

The 'ARRAY' modifier to the SQL datatype is allowed when no XML Schema datatype is assigned to the given parameter of the PL procedure or UDT method. In this case the input and output value will be verified to confirm to the rules applicable for an array. Furthermore in this case an XSD definition will be added in the WSDL file at run time.

Example 17.12. SOAP Extension

This example shows both approaches to define parameters and SOAP encoding style. In practice this definition is part of the Interop tests round 4 (group H). The meaning of this is: the SOAP operation is uses RPC encoding style, 'whichFault' is integer, 'param1' and 'param2' are strings. The out parameters 'part2_1' and 'part2_2' will be printed in SOAP:Fault element (see Exposing & Processing SOAP Fault Messages for more details). The interesting fact is that the last two parameters will be serialized as "part2" in different namespaces. And finally no return of the SOAP operation is defined (it's empty).

```
create procedure
"echoMultipleFaults3" (
    in whichFault int __soap_type 'http://www.w3.org/2001/XMLSchema:int',
    in param1 varchar __soap_type 'http://www.w3.org/2001/XMLSchema:string',
    in param2 varchar __soap_type 'http://www.w3.org/2001/XMLSchema:string',
    out part2_1 varchar __soap_options (
        __soap_fault:='http://www.w3.org/2001/XMLSchema:string',
        PartName:='part2',
        ResponseNamespace:='http://soapinterop.org/wsdl/fault1'),
    out part2_2 varchar __soap_options (
        __soap_fault:='http://www.w3.org/2001/XMLSchema:string',
        PartName:='part2',
        ResponseNamespace:='http://soapinterop.org/wsdl/fault2')
    )
   __soap_type '__VOID__'
{

  if (whichFault > 2)
    whichFault := mod (whichFault, 3) + 1;
  declare exit handler for sqlstate 'SF000'
```

```
    {
      http_request_status ('HTTP/1.1 500 Internal Server Error');
      if (whichFault = 1)
        {
          part2_1 := param1;
        }
      else if (whichFault = 2)
        {
          part2_2 := param2;
        }
      connection_set ('SOAPFault', vector ('400', 'echoMultipleFaults3'));
      return;
    };
  signal ('SF000', 'echoEmptyFault');
}
;
```

17.1.16. Exposing & Processing SOAP Header Messages

The Virtuoso SOAP server can be used to process the SOAP Header messages as described in the W3C recommendation (http://www.w3c.org/TR/SOAP , SOAP Header section). They can also be exposed in the WSDL file (services.wsdl) as per W3C WSDL recommendation, using the RPC style encoding.

To bind a message to a SOAP header the special keyword __soap_header is reserved for input and output parameters. The __soap_header followed by the SOAP datatype can be specified for any input or output parameter after normal datatype declarations. This will expose parameters as input or output messages separately. Header binding will also be added to an appropriate section of the WSDL description file for the SOAP message.

Example 17.13. Processing of the SOAP Header element

Consider the following simple SOAP request message with Header element:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
       xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
       xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Header>
      <h:echoMeStringRequest
         xmlns:h="http://soapinterop.org/echoheader/"
         SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next"
         mustUnderstand="1">hello world</h:echoMeStringRequest>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
      <m:echoVoid xmlns:m="http://soapinterop.org/"></m:echoVoid>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

This request will be processed by the Virtuoso SOAP server in the following way:

1. Check whether the echoVoid operation is defined for the given web directory mapping (see: exposing a PL procedure as a SOAP operation)
2. Test whether there is an in-parameter echoMeStringRequest defined for header processing (see below exposing a header parameters)
3. Test the mustUnderstand attribute:
    - If mustUnderstand is 0 or is undefined the request will continue without an error.
    - If mustUnderstand is 1 and the actor attribute is not empty or defined with the http://schemas.xmlsoap.org/soap/actor/next special URI, the request will continue without an error.
    - If the two conditions about fail then the request will be rejected with a SOAP MustUnderstand error.
4. The value of the echoMeStringRequest will be passed as a parameter to the echoVoid procedure.
5. If the call to the echoVoid succeeds, and the corresponding out parameter is supplied for the SOAP response header then it will be sent to the SOAP client.

The following procedure, which represents a part from echoHeaderBindings iterop test (round C), for the demonstration purposes is designed to process the above SOAP message.

```
create procedure
Interop.INTEROP.echoVoid
   (in echoMeStringRequest nvarchar := NULL __soap_header 'http://www.w3.org/2001/XMLSchema:string',
    out echoMeStringResponse nvarchar := NULL __soap_header 'http://www.w3.org/2001/XMLSchema:string')
   __soap_type '__VOID__'
{
  if (echoMeStringRequest is not null)
    echoMeStringResponse := echoMeStringRequest;
};
```

Note:

The __soap_header keyword that instructs the SOAP server to process this parameter via a SOAP Header with datatype string. Also, the condition in the procedure is needed to return the value in SOAP header only if it is supplied. In some other cases it can be returned always, but in this particular example it will be echoed only if the appropriate header is sent.

17.1.17. Exposing & Processing SOAP Fault Messages

The SOAP:Fault message is used to indicate which part of SOAP request fails, so in its general form it may not have a detailed error. But in some cases it is useful to report in detail which element's input(s) are not correct.

Custom soap:fault messages can be generated by application logic as illustrated below:

Have a procedure to generate custom SOAP:Fault messages with at least one OUT parameter denoted by __SOAP_FAULT instead of __SOAP_TYPE keyword following by type to be returned as literal.

Once we have such parameter(s) declared we can set these to some value (of atomic, simple or complex type) as may be appropriate.

And finally we need to set a special connection variable 'SOAPFault', in order to signal custom SOAP:Fault on output. The value of the connection variable needs to be an array of two elements : An integer of 100, 200, 300, 400 which represents the SOAP:VersionMismatch, SOAP:MustUnderstand, SOAP:Client and SOAP:Server errors. And a string which will be printed in textual explanation, human readable format. In real life we will not need to generate 100 or 200 fault messages, but anyway it is possible to do that.

Example 17.14. Signalling a custom SOAP Fault element

Consider we need to indicate to the client that some string is not a valid input, we can use the custom fault message mechanism as.

```
create procedure
echoStringFault (in param nvarchar,
                 out part2 nvarchar __soap_fault 'string')
returns nvarchar
{
  declare exit handler for sqlstate 'SF000'
    {
      http_request_status ('HTTP/1.1 500 Internal Server Error');
      -- we are setting the fault message
      part2 := param;
      -- and instructing the SOAP server to make error 400 with text explanation StringFault
      connection_set ('SOAPFault', vector ('400', 'StringFault'));
      ----------------^^^^^^^^^^
      return;
    };
  -- in real life signalling of the error is under some condition
  -- for example if string is longer that 10 chars
  signal ('SF000', 'echoEmptyFault');
}
;
```

And an wire dump of SOAP request

```
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" ...>
  <SOAP-ENV:Body>
    <m:echoStringFault xmlns:m="http://soapinterop.org/wsdl">
      <param xsi:type="xsd:string">String</param>
    </m:echoStringFault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

And SOAP Fault response

```
<?xml version="1.0"?>
<SOAP:Envelope SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" ...>
  <SOAP:Body>
    <SOAP:Fault>
      <faultcode>SOAP:Server</faultcode>
      <faultstring>[Virtuoso SOAP server] StringFault</faultstring>
      <detail>
        <h:part2 xmlns:h="http://soapinterop.org/wsdl" xsi:type="xsd:string">String</h:part2>
      </detail>
    </SOAP:Fault>
  </SOAP:Body>
</SOAP:Envelope>
```

Please note that in wire dumps there is no namespace declarations for brevity (places are denoted with '...').

17.1.18. Document Literal Encoding

The Virtuoso SOAP server and client support Document Literal encoding for processing as an alternative to SOAP/RPC. The document/literal encoding allows the transmission of any arbitrary valid XML document instead of a SOAP call following rules from section 5 from SOAP/1.1 specification. This allows us to send and receive SOAP packets that are more free-form ("document" style). If you create a service that can accept more free-form type packets, you can employ constraints within the methods so that they can be independent (bare) or serialized as embedded elements within the method's SOAP structure (wrapped parameters style).

Example 17.15. Comparing SOAP Types

Here are examples of SOAP requests that represent the RPC, Doc/Literal and Doc/Literal with parameters types of SOAP message

-- RPC encoded --

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
      xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
     <m:echoString
         SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
         xmlns:m="http://soapinterop.org/">
           <param0 xsi:type="xsd:string">Enter a message here</param0>
     </m:echoString>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

-- Document Literal --

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
      xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
     <ns1:echoStringParam xmlns:ns1="http://soapinterop.org/xsd">Enter a message here</ns1:echoStringPara
```

```
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

-- Document Literal with parameters --

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
      xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
     <ns1:echoString xmlns:ns1="http://soapinterop.org/xsd">
       <param0>Enter a message here</param0>
     </ns1:echoString>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP operations can be designated as document/literal or RPC by using the appropriate values in the WSDL description file associated to that SOAP endpoint. As Virtuoso SOAP operations are PL procedures special keywords are used within the procedure to indicate that the document/literal encoding should be used. These special keywords are:

__soap_doc

__soap_docw

These should be placed after the 'returns' keyword in a Virtuoso procedure definition. If 'returns ... __soap_type' is omitted the procedure return type will be equivalent to 'returns varchar __soap_type 'http://www.w3.org/2001/XMLSchema:string'.

Another way to expose a PL procedure or UDT method as a document/literal SOAP methods is to use non-explicit XMLSchema datatypes and to force encoding rules via virtual directory option 'Use' (see also SOAP options section in this chapter and in WSDL chapter section: "Exposing SQL Stored Procedures containing complex datatype definitions" for details and examples).

See Also:

WSDL 1.1 Specification

Example 17.16. SOAP Returns RPC

The following example shows a procedure that will be exposed as an RPC encoded SOAP operation:

```
create procedure
Import1.echoString (in x nvarchar __soap_type 'http://www.w3.org/2001/XMLSchema:string')
returns nvarchar __soap_type 'http://www.w3.org/2001/XMLSchema:string'
{
  return x;
};
```

Example 17.17. SOAP Returns Document Literal

The following example shows a procedure that will be exposed as a document literal encoded operation. Note the __soap_doc keyword after 'returns', also in this case __soap_type for each parameter must be specified since the incoming request must be validated by the given schema element declaration (see below for XMLSchema elements declaration).

```
create procedure
DocLit.echoString (in echoStringParam varchar __soap_type 'http://soapinterop.org/xsd:echoStringParam')
      returns any __soap_doc 'http://soapinterop.org/xsd:echoStringReturn'
{
      return echoStringParam;
};
```

Example 17.18. SOAP Returns Document Literal with Parameters

The following example shows a procedure that will be exposed as document literal encoding operation with parameters style (wrapped). note the __soap_docw keyword after 'returns'.

```
create procedure
DocPars.echoString (in echoString varchar __soap_type 'http://soapinterop.org/xsd:echoString')
      returns any __soap_docw 'http://soapinterop.org/xsd:echoStringResponse'
{
      return echoString;
};
```

In both cases of Document Literal encoding we need to specify the schema element for validation of the incoming SOAP request. Furthermore, this applies to the output elements and return value, as they need to be encoded/validated properly.

Defining WSDL Schema Data Type and Elements

When defining a schema data type (for use within SOAP) the 'targetNamespace' attribute on top level element must be specified in order to describe in which namespace this type is valid. In other words, this type will be used to validate request only within this namespace. Therefore it will be exposed only at this WSDL point where it is used to describe a parameter of an operation associated to it.

Important

All datatypes and elements defined for use in SOAP must have namespace (QName), which means that 'targetNamespace' must be specified in the definition. All non-qualified types will be rejected in SOAP validation and will not be described in the WSDL file.

Example 17.19. Making an array of string data type

Here is an example demonstrating making an array-of-string datatype:

```
select soap_dt_define('','<complexType name="ArrayOfstring"
   targetNamespace="http://soapinterop.org/xsd"
   xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:tns="http://soapinterop.org/xsd">
  <complexContent>
     <restriction base="enc:Array">
        <sequence>
           <element name="item" type="string" minOccurs="0" maxOccurs="unbounded" nillable="true"/>
        </sequence>
        <attributeGroup ref="enc:commonAttributes"/>
        <attribute ref="enc:arrayType" wsdl:arrayType="string[]"/>
     </restriction>
  </complexContent>
</complexType>');
```

As document literal encodings work with elements, the elements must be declared as a part of the WSDL file (in the types/schema section). The declared elements can be used to define a doc/literal encoded SOAP operation. This allows for the definition of an element of request and response to enable the server to understand the requests (validate and process) and respond to them (validate the PL data and serialize properly).

Example 17.20. Example of defining elements

Here is an example for the DocLit.echoString SOAP operation using parameters (input parameter and return type):

```
select soap_dt_define('','<element xmlns="http://www.w3.org/2001/XMLSchema"
                                   name="echoStringParam"
                                   targetNamespace="http://soapinterop.org/xsd" type="string" />');

select soap_dt_define('','<element xmlns="http://www.w3.org/2001/XMLSchema"
                                   name="echoStringReturn"
                                   targetNamespace="http://soapinterop.org/xsd" type="string" />');
```

Extensions to Simple Types

The attribute extensions to the simple types (string, float, etc...) can be defined and used in SOAP messages. In that case a PL value is represented as a special structure of 3 elements as follows:

```
vector (<composite>, vector (<attr-name>, <attr-value>, ...), <simple type value>)
```

Example 17.21. An example to define a simple type 'Document'

```
select soap_dt_define('','<complexType name="Document"
             xmlns="http://www.w3.org/2001/XMLSchema"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             targetNamespace="http://soapinterop.org/xsd">
  <simpleContent>
    <extension base="string">
      <xsd:attribute name ="ID" type="string"/>
    </extension>
  </simpleContent>
</complexType>');
```

Note that soap_dt_define() does not need the name to be specified when adding a new type, the name/namespace will be extracted from XSD fragment.

WSDL Generation

As the WSDL file generation is based on granted PL procedures exposed to a given SOAP endpoint, only SOAP datatypes and schema elements used for them will be printed in <types> section. If an undeclared datatype is used for an exposed procedure, the error will be printed in an XML comment where the type definition was expected and not found. If an element or datatype refers to other (dependent) types they will also be automatically included. For example, if we have exposed for a SOAP endpoint only the following procedure:

```
create procedure
INTEROP.echoStructArray (
    in inputStructArray any __soap_type 'http://soapinterop.org/xsd:ArrayOfSOAPStruct')
    __soap_type 'http://soapinterop.org/xsd:ArrayOfSOAPStruct'
{
  return inputStructArray;
};
```

The schema fragment will consist of both SOAPStructure and ArrayOfSOAPStruct data types declaration:

```
<schema targetNamespace="http://soapinterop.org/xsd"
   xmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" >
     <complexType name="ArrayOfSOAPStruct" >
       <complexContent>
         <restriction base="soapenc:Array">
           <sequence>
             <element name="item" type="ns0:SOAPStruct" minOccurs="0" maxOccurs="unbounded"/>
           </sequence>
           <attribute ref="soapenc:arrayType" wsdl:arrayType="ns0:SOAPStruct[]"/>
           <attributeGroup ref="soapenc:commonAttributes"/>
           <attribute ref="soapenc:offset"/>
         </restriction>
      </complexContent>
    </complexType>
    <!-- Note this fragment, it's included because ArrayOfSOAPStruct depends from it -->

    <complexType name="SOAPStruct" >
       <all>
          <element name="varString" type="string" nillable="true"/>
          <element name="varInt" type="int" nillable="true"/>
          <element name="varFloat" type="float" nillable="true"/>
       </all>
    </complexType>
</schema>
```

Multiple Namespaces in WSDL and SOAP

When you define a SOAP operation that has parameters from different namespaces or a type referring to a type in another namespace, both will be defined and printed as a separate schema definition in the WSDL file. Hence, we can define a data type in different namespace so they will live together in a single WSDL file. This allows us to make more complex and flexible document-centric style SOAP operations.

Example 17.22. Example from the SOAP Interop 3 Tests

This example is of the echoEmployee operation from interop 3 tests:

```
create procedure
Compound2.echoEmployee (in x any __soap_type 'http://soapinterop.org/employee:x_Employee')
      returns any __soap_doc 'http://soapinterop.org/employee:result_Employee'
{
  return x;
};
```

This will generate the following schema in the WSDL file (only affected parts are shown):

```
<definitions
...
xmlns:ns1="http://soapinterop.org/person"
xmlns:ns0="http://soapinterop.org/employee"
... >

<types>
        <schema targetNamespace="http://soapinterop.org/person"
                xmlns="http://www.w3.org/2001/XMLSchema"
                xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" elementFormDefault="qualified" >
           <complexType name="Person" >
              <sequence>
                <element minOccurs="1" maxOccurs="1" name="Name" type="string"/>
                <element minOccurs="1" maxOccurs="1" name="Male" type="boolean"/>
              </sequence>
           </complexType>
        </schema>

        <schema targetNamespace="http://soapinterop.org/employee"
                xmlns="http://www.w3.org/2001/XMLSchema"
                xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" elementFormDefault="qualified" >
                <import namespace='http://soapinterop.org/person' />
            <complexType name="Employee" >
                <sequence>
                   <element minOccurs="1" maxOccurs="1" name="person" type="ns1:Person"/>
                   <element minOccurs="1" maxOccurs="1" name="salary" type="double"/>
                   <element minOccurs="1" maxOccurs="1" name="ID" type="int"/>
                </sequence>
           </complexType>

           <element name="result_Employee" type="ns0:Employee" />
           <element name="x_Employee" type="ns0:Employee" />

        </schema>
</types>
...
```

The PL procedure is defined to use element declaration x_Employee and result_Employee, so this will automatically include the Employee and Person type, upon which they depend. Also, as these types are defined in different namespace, two schema parts will be specified in the WSDL file.

In practice the SOAP developer needs to define elements and types (using soap_dt_define() function), after this, specifying a parameter of PL procedure (or return type) will cause automatic generation of the associated WSDL description in the manner described. Hence, no user intervention is required besides the initial element/type definition.

SOAP Interop round III Endpoints

The following endpoints are pre-defined in the Demo database for SOAP interop III testing (the WSDL files are in the usual services.wsdl for each group of tests):

- *D tests*
  - /r3/EmptySA/ - echoString operation with empty ("") soapAction (PRC encoded)
  - /r3/Import1/ - echoString operation, rpc encoded
  - /r3/Import2/ - echoStruct operation, rpc encoded
  - /r3/Import3/ - echoStruct and adds method echoStructArray, rpc encoded (echoStruct is in different namespace)
  - /r3/Compound1/ - Use of attributes in SOAP payload, including attribute on element of simpleType , doc/literal
  - /r3/Compound2/ - Two schema sections, types in 1st schema references types in the 2nd schema, doc/literal
  - /r3/DocPars/ - Reduced version of SOAPBuilders Interop test wsdl with "parameters" way of describing rpc requests in Document/Literal (Document/Literal - Wrapped). Version has operations echoString, echoArrayOfString and echoStruct
  - /r3/DocLit/ - Reduced version of SOAPBuilders InteropTest test, document/literal mode. Version has operations echoString, echoArrayOfString and echoStruct
  - /r3/RpcEnc/ - Reduced version of SOAPBuilders InteropTest test, rpc/encoded mode. Version has operations echoString, echoArrayOfString and echoStruct
- *E tests*
  - /r3/List/ - echo of list structure (as shown) , RPC encoded

```
struct list {
  int varInt;
  string varString;
  list child; //nullable
}
```

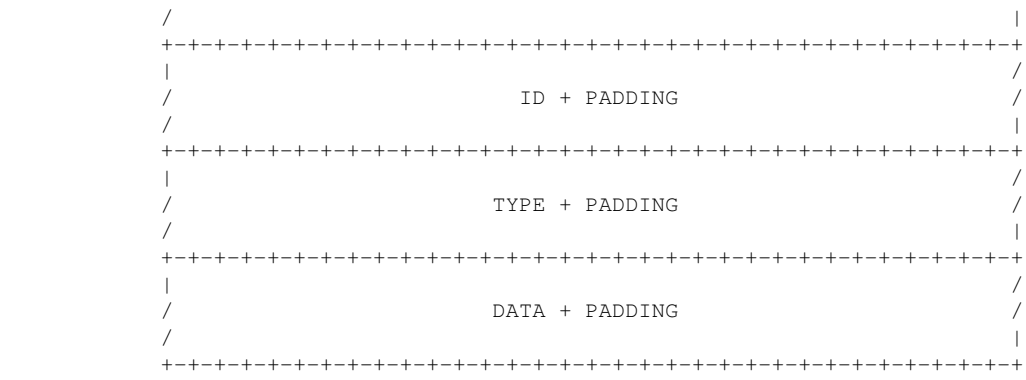
- *F tests*
  - /r3/Hdr/ - Modified version of SOAPBuilders InteropTest test, document/literal mode Version has one operation echoString with 2 headers defined.

17.1.19. DIME encapsulation of SOAP messages

The Direct Message Encapsulation (DiME) format is a message format that can be used to encapsulate one or more payloads of arbitrary type and size. This format can be used in place of MIME, but benefits of DIME are ease of parsing and low memory consumption, as DIME does not require loading the whole message body in order to parse it. This is due to the fact that MIME does not have mechanism for specifying the length of payloads etc. DIME prefixes all data with length and type information.

The structure of a DIME message as per draft-nielsen-dime-02 is:

```
/*
      Legend:

      VERSION = 0x01
      RESRVD  = 0x00
      MB - begin mark
      ME - end mark
      CF - chunked flag
      TYPE_T - type of content type field

      0                   1                   2                   3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |          |M|M|C|       |       |                               |
     | VERSION  |B|E|F| TYPE_T| RESRVD|         OPTIONS_LENGTH        |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |             ID_LENGTH          |           TYPE_LENGTH         |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                            DATA_LENGTH                        |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                                                               /
     /                     OPTIONS + PADDING                         /
```



The MB,ME,CF flags are used to indicate which part of the DIME message is the current block of data. Also, we notice that there are four length fields of fixed length before any data, id or type payload. This is to make the payload easier to read.

The Virtuoso server implements a DIME parser and composer as functions and filter for DIME in SOAP server. Furthermore the Virtuoso WSDL generator can be instructed to specify a DIME extension to the PL procedure exposed as SOAP method. The implementation is based on draft-nielsen-dime-02 RFC proposal. Please note that in the rest of document we will use 'DIME attachment' term , which is about SOAP message with attachment encapsulated with DIME as per draft-nielsen-dime-soap-01. The special case in these messages is type of first payload, so it's supposed to be a SOAP:Envelope message.

Note: Option fields are not supported.

To setup a SOAP endpoint to recognize DIME encapsulation the "DIME-ENC" option to SOAP in virtual directory must be set to 'yes'. Furthermore the WSDL description of endpoint defined as DIME enabled will contain WSDL extensions to DIME.

As not in all cases input and output of the SOAP server needs to be DIME encoded, the particular PL procedure exposed as SOAP method needs to be defined in special way to indicate which traffic is encoded as DIME. This is done by using special keywords on procedure declaration:

```
CREATE PROCEDURE ([PARAMETERS DECLARATION])
 [RETURNS TYPE] [(__SOAP_TYPE|__SOAP_DOC|_SOAP_DOCW) 'LITERAL'] [__SOAP_DIME_ENC (IN/OUT/INOUT)]
```

The '__SOAP_DIME_ENC IN' indicate that the procedure expects a DIME attachments on input. This can also be used with OUT and INOUT. This will also be indicated in WSDL file (services.wsdl) as DIME extension in appropriate place of 'soap:operation' element.

The format of SOAP attachments passed to PL procedure defined in this way is an array which consists of three string elements: ID, content-type, and attachment data itself. The same format must be used when parameter is an output which needs to be sent as DIME attachment. There is also a special parameter of PL procedure exposed as SOAP method named 'ws_soap_attachments', so when we have such, all attachments received will be passed thru it. In practice we will not need to use 'ws_soap_attachments' , but anyway it's practical use is to handle unreferenced parameters or to debug the request.

Finally we must say that type of parameter needs to have datatype declared as per 'WSDL Extension for SOAP in DIME' proposal, this is needed for indicating in the WSDL what to expect and how to send the attachment. See also the example below.

Example 17.23. Using DIME encapsulation

Suppose we need to accept a binary attachment and echo it back as string encoded in the popular 'base64'.

We first need to enable DIME encapsulation to an endpoint, with virtual directory definition:

```
SQL> VHOST_DEFINE (lpath=>'/r4/groupG/dime/rpc', ppath=>'/SOAP/', soap_user=>'interop4',
    soap_opts => vector ('DIME-ENC', 'yes')) ;
```

The sample PL procedure that takes a binary attachment and transforms it to a base64 encoded string must be declared as:

```
create procedure
```

```
EchoAttachmentAsBase64 (in "In" nvarchar __soap_type 'http://soapinterop.org/attachments/xsd:ReferencedBi
returns nvarchar __soap_type 'base64Binary'
__soap_dime_enc in
{
  -- we are getting the attachment as the 3rd element of input,
  -- do the base64 encoding for it and return it to the requestor
  return encode_base64 (cast ("In"[2] as varchar));
}
;
```

As we have noticed an 'ReferencedBinary' is used to declare 'In' parameter. This has a special purpose for WSDL definition, not for SOAP processing itself. In that case clients are instructed to look at annotation/appinfo of a simple type declared as:

```
<complexType name="ReferencedBinary">
        <simpleContent>
                <restriction base="soap-enc:base64Binary">
                        <annotation>
                                <appinfo>
                                        <content:mediaType value="application/octetstream"/>
                                </appinfo>
                        </annotation>
                        <attributeGroup ref="soap-enc:commonAttributes"/>
                </restriction>
        </simpleContent>
</complexType>
```

This is a little-bit tricky, but this is how to indicate the type of the content and how to resolve the references to the attachments as per the WSDL Extension for SOAP in DIME' proposal.

17.1.20. SOAP Endpoint Options

The virtual directory mechanism provides a special SOAP options for SOAP processing. The SOAP options are name-value pairs contained in a vector: i.e. vector ('name1', 'value1', ...). The SOAP server accepts the following optional parameters settable in the SOAP Options field of the HTTP Virtual Directories Setup interface, or using the `vhost_define()` function:

ServiceName : name of the SOAP service, will be prefixed with 'Virtuoso'. That name is shown in WSDL description.

Namespace : namespace URI of the SOAP:Body request and response.

HeaderNS : namespace URI for SOAP:Header messages.

FaultNS : namespace URI for SOAP:Fault messages.

MethodInSoapAction : enable or disable appending of the method name in the soapAction attribute (WSDL) after namespace URI.

CR-escape : enable or disable escaping of the CRs on wire as &#0xd

elementFormDefault=(unqualified|qualified); Sets the elementFormDefault for schema specification. if qualified is used the elementFormDefault attribute will be set to qualified, in which case elements required to be unqualified can be declared with value of "form" attribute "unqualified".

Use=(encoded|literal) Sets the default SOAP message encoding rules for those PL procedures which have no explicit encoding rule assigned (see SOAP special syntax for PL procedures). The default is 'encoded' which means to follow SOAP RPC encoding as described in SOAP v1.1 specification section 5.1. The 'literal' mode forces the SOAP server to expose PL procedures with the document/literal parameter encoding style.

MethodInSoapAction=(no|yes|empty|only); Controls soapAction attribute manipulation. *no* - only URL for soap requests will be printed. *yes* (default) - the URL and soap method will be printed in form: <url>#<method name>. *empty* - no value will be specified for soapAction. *only* - only the method will be specified in form #<method name>.

DIME-ENC : Controls DIME encapsulation on particular SOAP endpoint, valid values are *no* - (default) not enabled. *yes* - DIME encapsulation is enabled on endpoint

WS-SEC : WS-Security processing is enabled on the endpoint, if it's *yes* , otherwise disabled (default)

WSS-KEY : name of PL procedure, which is supposed to return a key instance, used together with "WS-SEC" option.

WSS-Template : path to the file for making the XML Signature in response message. The "[key reference for signing]" denotes using a default template for signing, see WS Security signing SOAP messages.

WSS-Validate-Signature : This option controls the input behavior, i.e. how to verify the incoming message. Possible values are "0", "1" or "2", where 0 does not verify signatures, 1 expects a signature to exist, 2 will verify signature if one exists.

WS-RP : to enable WS-Routing protocol on particular endpoint, if it's *yes* , otherwise disabled (default).

wsrp-from : Constant for identification of endpoint, an example is 'some@user.network'. This will be included in 'form' element in WS Routing header.

17.2. WSDL

The Web Services Description Language (WSDL) is a standard, structured way of describing SOAP messages and Web services. It is an XML format for describing the network services offered by a service provider. The provider will publish a WSDL file that contains details about the services provided, and the set of operations within each service that the provider supports. For each of the operations, the WSDL file also describes the format that the client must follow in requesting an operation.

Since the WSDL file sets up requirements for both the provider and service requester, this file is like a contract between the two. The provider agrees to provide certain services if the client sends a properly formatted SOAP request. Suppose that we have a WSDL file defining a service called StockQuoteService. This service describes operations such as getTradePrice, getLowestTradePrice, and getHighestTradePrice. You place this WSDL file on the service provider server. A client who wishes to send a SOAP request to this server must first obtain a copy of the WSDL file from the provider, and then use it to format a suitable SOAP request. The client sends this request to the provider. The provider executes the requested operation and sends the results back to the client requester as a SOAP response.

See Also:

The specification of WSDL and its file structures can be found on the W3C site .

17.2.1. Exposing Stored Procedures as WSDL Services

Virtuoso can be both a provider and a client of WSDL. In this section we will explain how to use Virtuoso to expose procedures as SOAP messages, and then publish them as WSDL consumables.

In the descriptions below, lines preceded by

```
SQL>
```

denote that the command is intended to be issued using the ISQL command line interface to Virtuoso.

Virtuoso procedures can easily be published as WSDL consumables. We follow the same steps as we would take to create SOAP objects and then for every SOAP object Virtuoso automatically generates a WSDL file entry. The default Virtuoso has the user SOAP and reserved HTTP path of `/SOAP/` . All procedures that are created in the default qualifier namespace of the SOAP user (WS.SOAP) and have had a 'grant execute to SOAP' permissions established in the database are available to SOAP and thus are automatically available to WSDL. In Virtuoso this is done by requesting the file `services.wsdl` from the server via HTTP from the `/SOAP/` path:

```
http://[host:port]/SOAP/services.wsdl
```

. WSDL files such as this are often referred to as "endpoints" for services.

Virtual directories increase our flexibility by allowing us to map logical HTTP paths to the location `/SOAP/` . This means that we can separate WSDL/SOAP functionality, making groups of services available under different locations. We will now demonstrate this:

1. First, create a new user in the database for creating the stored procedures as SOAP messages:

   ```
   SQL>CREATE USER SOAPDEMO;
   ```
2. Now, set the default catalogue/qualifier for the new user to the WS catalogue where we will create procedures to be used as SOAP objects:

   ```
   SQL>USER_SET_QUALIFIER ('SOAPDEMO', 'WS');
   ```
3. Now create a new virtual host definition, using the `vhost_define()` , so that we can find our SOAP objects later at a desired location.

   ```
   SQL>VHOST_DEFINE (vhost=>'*ini*',lhost=>'*ini*',lpath=>'/services',ppath=>'/SOAP/',soap_user=>'SOA
   ```

   If the mapping already exists, producing an error in the call above, and is not being used, then you can remove it using the command:

```
SQL>VHOST_REMOVE (vhost=>'*ini*',lhost=>'*ini*',lpath=>'/services')
```

Note:
ini is a special indicator telling Virtuoso to take the default values from its initialization file.

4. Now create a simple SOAPTEST procedure and grant the appropriate privileges to the SOAPDEMO user:

```
SQL> create procedure
  WS.SOAPDEMO.SOAPTEST (in par varchar)
{
  return (upper(par));
};

SQL> grant execute on WS.SOAPDEMO.SOAPTEST to SOAPDEMO;
```

5. Now test this new SOAP object's availability by using `soap_client()` . This function would normally return a vector representation of the SOAP object but since we know the dimensions of the object ahead of time we can pin-point the entry using the `aref()` function as follows:

```
SQL> select aref(aref(
        soap_client (url=>sprintf ('http://localhost:%s/services', server_http_port ()),
        operation=>'SOAPTEST', parameters=>vector('par', 'demotext')), 1), 1);
callret
VARCHAR
_______

DEMOTEXT
```

   The actual SOAP object looks more like:

```
(("SOAPTESTResponse" ) (("CallReturn" ) "DEMOTEXT" ) )
```

   which was generated in a Virtuoso server log for debugging purposes using the `dbg_obj_print()` function.

6. Procedures that exist under the WS.SOAPDEMO namespace and have been granted execution to the new SOAPDEMO user are now available as SOAP services and described by WSDL in this example, Virtuoso would publish them from the URL:

```
http://example.com/services/services.wsdl
```

   which will yield the following WSDL description:

```
<?xml version='1.0'?>
<definitions
 targetNamespace='services.wsdl'
 xmlns:xsd='http://www.w3.org/1999/XMLSchema'
 xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
 xmlns:tns ='services.wsdl'
 xmlns ='http://schemas.xmlsoap.org/wsdl/'
 name='VirtuosoSOAP'>
        <message name='SOAPTEST'>
                <part name='par' type='xsd:string'/>
        </message>
        <message name='SOAPTESTResponse'>
                <part name='CallReturn' type='xsd:string'/>
        </message>
        <portType name='SOAPPortType'>
                <operation name='SOAPTEST'>
                        <input message='tns:SOAPTEST' name='SOAPTEST'/>
                        <output message='tns:SOAPTESTResponse' name='SOAPTESTResponse'/>
                </operation>
        </portType>
        <binding name='SOAPBinding' type='tns:SOAPPortType'>
                <soap:binding style='rpc' transport='http://schemas.xmlsoap.org/soap/http'/>
                <operation name='SOAPTEST'>
                        <soap:operation soapAction='urn:openlinksw.com:virtuoso_soap_schema#SOAPTE
                        <input>
                                <soap:body use='encoded' namespace='urn:openlinksw.com:virtuoso_so
                        </input>
                        <output>
                                <soap:body use='encoded' namespace='urn:openlinksw.com:virtuoso_so
```

```
                        </output>
                    </operation>
                </binding>
                <service name='SOAPService'>
                    <port name='SOAPPort' binding='tns:SOAPBinding'>
                        <soap:address location='http://example.com/services'/>
                    </port>
                </service>
            </definitions>
```

See Also:

The Testing Web Services (VSMX) section describes Virtuoso's ability to also automatically generate a test page for your SOAP services, simply by replacing services.wsdl with services.vsmx in the URL.

17.2.2. Exposing SQL Stored Procedures containing complex datatype definitions

When parameters of a PL procedure or UDT (User Defined Type) methods contain parameters declared as UDT or/and as ARRAY then WSDL generation will include XML Schema for them. The schema types in this case will be generated every time WSDL URL is accessed. Also the XMLSchema datatypes will be generated following the default encoding rules forced via 'Use' SOAP option to the given virtual directory.

Example 17.24. Exposing a PL Stored procedures containing complex datatypes

The following example will create a virtual directory '/soap-lit' on default HTTP listener and will expose a single method accepting a array of structures which contains an array of integers, integer, varchar and float members. The SOAP message will use the document/literal encoding rules (option Use=literal).

```
create user SOAP_U1;

VHOST_DEFINE (  lpath=>'/soap-lit',
                ppath=>'/SOAP/', soap_user=>'SOAP_U1',
                soap_opts=>
                vector ('ServiceName', 'Literal',
                    'Namespace', 'http://temp.uri',
                    'SchemaNS', 'http://temp.uri',
                    'MethodInSoapAction', 'yes',
                    'elementFormDefault', 'unqualified',
                    'Use', 'literal'));

create type SOAP_StructA as (varString varchar, varInt integer, varFloat real, varArray integer array);

create procedure echoStructArray (in sa DB.DBA.SOAP_StructA array) returns DB.DBA.SOAP_StructA array
{
  return sa;
};

grant execute on SOAP_StructA to SOAP_U1;
grant execute on echoStructArray to SOAP_U1;
```

This would produce the following WSDL file when accessing the http://[host:port]/soap-lit/services.wsdl URL.

```
<?xml version="1.0"?>
<definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:http="http://schemas.xmlsoap.org/wsdl/htt
  <types>
    <schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" target
      <complexType name="echoStructArray_Response_t">
        <sequence>
          <element name="item" type="ns0:SOAP_StructA" minOccurs="0" maxOccurs="unbounded" nillable="true
        </sequence>
      </complexType>
      <complexType name="echoStructArray_sa_t">
        <sequence>
          <element name="item" type="ns0:SOAP_StructA" minOccurs="0" maxOccurs="unbounded" nillable="true
        </sequence>
      </complexType>
      <complexType name="SOAP_StructA">
        <all>
```

```
            <element name="varString" type="string" nillable="true"/>
            <element name="varInt" type="int" nillable="true"/>
            <element name="varFloat" type="float" nillable="true"/>
            <element name="varArray" type="ns0:SOAP_StructA_varArray_t" nillable="true"/>
          </all>
        </complexType>
        <complexType name="SOAP_StructA_varArray_t">
          <sequence>
            <element name="item" type="int" minOccurs="0" maxOccurs="unbounded" nillable="true"/>
          </sequence>
        </complexType>
        <element name="echoStructArray">
          <complexType>
            <sequence>
              <element minOccurs="1" maxOccurs="1" name="sa" type="ns0:echoStructArray_sa_t"/>
            </sequence>
          </complexType>
        </element>
        <element name="echoStructArrayResponse">
          <complexType>
            <all>
              <element minOccurs="1" maxOccurs="1" name="CallReturn" type="ns0:echoStructArray_Response_t"/
            </all>
          </complexType>
        </element>
      </schema>
    </types>
    <message name="echoStructArrayRequest">
      <part element="dl:echoStructArray" name="parameters"/>
    </message>
    <message name="echoStructArrayResponse">
      <part element="dl:echoStructArrayResponse" name="parameters"/>
    </message>
    <portType name="LiteralDocLiteralPortType">
      <operation name="echoStructArray">
        <input message="tns:echoStructArrayRequest" name="echoStructArrayRequest"/>
        <output message="tns:echoStructArrayResponse" name="echoStructArrayResponse"/>
      </operation>
    </portType>
    <binding name="LiteralDocLiteralBinding" type="tns:LiteralDocLiteralPortType">
      <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="echoStructArray">
        <soap:operation soapAction="http://temp.uri#echoStructArray" style="document"/>
        <input name="echoStructArrayRequest">
          <soap:body use="literal"/>
        </input>
        <output name="echoStructArrayResponse">
          <soap:body use="literal"/>
        </output>
      </operation>
    </binding>
    <service name="VirtuosoLiteral">
      <documentation>Virtuoso SOAP services</documentation>
      <port name="LiteralDocLiteralPort" binding="tns:LiteralDocLiteralBinding">
        <soap:address location="http://example.com/soap-lit"/>
      </port>
    </service>
  </definitions>
```

17.2.3. Exposing Third Party SQL Stored Procedures as WSDL-Compliant Web Services

Virtuoso can expose any of its available PL resource to the SOAP world, and subsequently to the WDSL file. This includes data from remote attached tables and procedures. All you have to do is make sure that the entry point exists as a stored procedure in the correct namespace with the appropriate grants, as before.

See Also

The Virtual Database chapter for information regarding use of remote data sources and their tables.

Because remote procedures may not be directly compatible you are required to write a Virtuoso wrapper function first to handle the remote procedure. Below is a sample MS SQLServer procedure and an accompanying Virtuoso wrapper function. The MS SQLServer function returns a result set based on a simple join query with a filter input. The Virtuoso procedure calls the remote procedure, iterates through the result set returned and produces XML output.

MS SQLServer procedure:

```
create procedure ms_remote
        @mask varchar(15)
as
  select c.CustomerID, c.CompanyName, o.OrderDate,
      o.ShippedDate,ol.ProductID, ol.Quantity, ol.Discount
    from Northwind..Customers c
      inner join Northwind..Orders o on c.CustomerID = o.CustomerID
      inner join Northwind.."Order Details" ol on o.OrderID = ol.OrderID
    where c.CustomerID like @mask
;
```

Virtuoso wrapper function:

```
create procedure WS.SOAP.ms_remote_call (
  in dsn varchar, in uid varchar, in pwd varchar, in mask varchar)
{
  declare m, r, ses any;
  vd_remote_data_source (dsn, '', uid, pwd);
  rexecute (dsn, 'ms_remote ?', null, null, vector (mask), 1000, m, r);
  ses := string_output ();
  http ('<?xml version="1.0" ?>\n<remote>\n', ses);
  if (isarray(m) and isarray (r))
    {
      declare i, l, j, k integer;
      declare md, rs any;
      md := m[0];
      i := 0; l := length (md); k := length (r); j := 0;
      while (j < k)
       {
         http ('<record ', ses);
         i:=0;
         while (i < l)
           {
             dbg_obj_print (md[i][0],r[j][i]);
             http (sprintf (' %s="%s"', trim(md[i][0]), trim(cast (r[j][i] as varchar))), ses);
             i := i + 1;
           }
         http (' />\n', ses);
         j := j + 1;
       }
    }
  http ('</remote>', ses);
  return string_output_string (ses);
};
```

Now, as before, we grant execute rights to the SOAP user:

```
grant execute on WS.SOAP.ms_remote_call to SOAP;
```

The third-party procedures can now be accessed via SOAP and are listed in the WSDL file.

17.2.4. WSDL Descriptions of SOAP Header Messages

The Virtuoso web server automatically generates WSDL descriptions for procedures exposed as SOAP messages, and those exposed to have parameters bound to SOAP Header messages.

Consider the sample of the Interop.INTEROP.echoVoid procedure defined as an example in the SOAP section as:

```
create procedure
Interop.INTEROP.echoVoid
   (in echoMeStringRequest nvarchar := NULL __soap_header 'http://www.w3.org/2001/XMLSchema:string',
    out echoMeStringResponse nvarchar := NULL __soap_header 'http://www.w3.org/2001/XMLSchema:string')
```

```
    __soap_type '__VOID__'
{
  if (echoMeStringRequest is not null)
    echoMeStringResponse := echoMeStringRequest;
};
```

The WSDL description will now contain the header messages also. No extra user intervention is required. The WSDL file that will result will look like:

```
..... WSDL excerpt (consider http://[host:port]/Interop/services.wsdl of an demo DB) ...........
       <message name="echoVoidRequest"></message>
       <message name="echoVoidResponse"></message>
       <message name="echoVoidechoMeStringRequest">
               <part name="echoMeStringRequest" type="xsd:string" />
       </message>
       <message name="echoVoidechoMeStringResponse">
               <part name="echoMeStringResponse" type="xsd:string" />
       </message>
..... in bindings section note the soap:header sections.............
       <operation name="echoVoid">
       <soap:operation soapAction="http://soapinterop.org/" />
       <input>
       <soap:body use="encoded" namespace="http://soapinterop.org/" encodingStyle="http://schemas.xmlsoa
       <soap:header use="encoded" message="tns:echoVoidechoMeStringRequest" part="echoMeStringRequest" n
       </input>
       <output>
       <soap:body use="encoded" namespace="http://soapinterop.org/" encodingStyle="http://schemas.xmlsoa
       <soap:header use="encoded" message="tns:echoVoidechoMeStringResponse" part="echoMeStringResponse"
       </output>
       </operation>
.....more.........
```

17.2.5. Importing A WSDL File & SOAP/WSDL Proxying

Virtuoso can import WSDL files from other locations using the function:

`soap_wsdl_import()`

This function reads the descriptions of SOAP messages available in the WSDL file and automatically creates Virtuoso stored procedure wrappers for executing the SOAP messages directly from Virtuoso in procedures. These generated procedures can then be exposed as SOAP messages in the normal way from the Virtuoso SOAP server, and of course fully described by an automatically generated WSDL file for them, thus creating a proxy service for original messages.

17.2.6. SOAP/WSDL Interoperability

A key feature of the Web services promise is that Web services published with one server can be called from any other client. The ability of each implementation to make use of each others' output is called interoperability. Thus, Web services created with Virtuoso should interoperate smoothly with services created with Microsoft's .NET, Sun's Java, and so on. Interoperable Web services mean that developers and users do not have to think about which programming language or operating system the services are hosted on.

The evolving nature of the SOAP specification, as well as its complexity, leads to differences in SOAP implementations. Unfortunately, these implementation differences decrease interoperability. To counteract this problem, a volunteer group of SOAP application builders has developed a series of interoperability tests.

OpenLink, as a participant in this group, ensures that its SOAP implementation interoperates fully. This means that Virtuoso's SOAP server properly exposes your Web services so they can be used by any client. It also means that Virtuoso can call services published by any compliant provider.

You can view the Round 1 "Interoperability Lab" at www.xmethods.net and the Round 2 at www.whitemesa.com . Please note that OpenLink has no connection with these companies; they are simply places that volunteered to host the testing reference documents on their servers.

The Round 2 tests include base functionality (which duplicates Round 1) as well as more advanced tests labeled "Group B" and "Group C". The WSDL published by OpenLink containing the descriptions of these tests can be found at http://demo.openlinksw.com:8890/Interop/ . This URL will be updated as new interoperability tests are devised.

The SOAP implementation passes all known interoperability tests.

17.3. WebID Protocol Support

WebID Protocol is an authentication and authorization protocol that links a "Web ID" or "Personal URI " to a public key to create a global, decentralized, distributed, and secure authentication system that functions with existing browsers.

WebID Protocol uses PKI standards - usually thought of as hierarchical trust management tools - in a decentralized web-of-trust way . The web of trust is built using semantic web vocabularies (particularly FOAF) published in RESTful manner to form Linked Data.

Based on well known existing standards, WebID Protocol is currently in development, and is being discussed on the FOAF protocols mailing list .

For the most recent description of the protocol, read the one-page "WebID Protocol: Adding Security to Open Distributed Social Networks ". For a more detailed explanation of how the authentication works, see "WebID Protocol: Creating a Web of Trust without Key Signing Parties ".

Automatic discovery of interpersonal trust relationships enables automatic application of appropriate permissions.

In other words, data owners can set fuzzy permissions like "only let my friends see this" or "only let my family edit this." Applications can discover the relationships between the data owner and the data requester/user, and permit (or disallow) any attempted actions, without needing the data owner to explicitly set permissions for each potential user.

One example might be a parent setting permissions on a photo gallery, to permit viewing only by "immediate family". The parent need not list each and every such relative specifically for this application - and need not add new permissions for a new family member (whether by marriage, birth, or otherwise), though they do need to be added to the owner's FOAF. When a new user comes and asks to see the pictures, the gallery application would check the relationships declared by each person (the owner and the visitor), and if they matched up (in other words, the visitor could not get in simply by claiming a family relationship; the relationship must be confirmed by the owner's FOAF data), the pictures would be shown.

17.3.1. x.509 certificate

The WebID Protocol consumer needs an x509 certificate with v3 extension "Subject Alternate Name". This attribute is used for the owner's Web ID. For testing purposes we used OpenSSL demo CA to generate such certificates. If you are not using the OpenSSL demo CA, you must first setup a self-signed CA; read OpenSSL documents on how to do this.

1. Add the following line to the [usr_cert] section of the openssl.cnf file:

   ```
   subjectAltName=$ENV::ALTNAME
   ```
2. Set the environment variable ALTNAME to the owner's Web ID, e.g.,

   ```
   export ALTNAME=URI:http://example.com/dataspace/person/myname#this
   ```
3. Make a self-signed certificate, e.g.,

   ```
   $ CA.pl -newreq (follow the dialog)
   $ CA.pl -sign
   ```
4. When asked to commit the certificate, make sure you see several lines above, like:

   ```
    X509v3 Subject Alternative Name:
                   URI:http://example.com/dataspace/person/myname#this
   ```
5. If your browser wants a PKCS#12 bundle, you must make one:

   ```
   $ openssl pkcs12 -export -in newcert.pem -inkey newkey.pem -out mycert.p12
   ```
6. Rename newcert.pem and newkey.pem, to mycert.pem and mykey.pem for example. The PEM format of the certificate will be needed below.

17.3.2. Setting up Virtuoso HTTPS

To enable the HTTPS listener, you will need another certificate. Existing certificates may not have Subject Alternate Name, so you may want to generate one as in the previous section.

1. The next step is to move newcert.pem, newkey.pem, and cacert.pem into the server's working directory. In our test case, we put the keys in a 'keys' sub-directory, and added the following lines to the [HTTPServer] section of the Virtuoso INI file, virtuoso.ini:

   ```
   SSLPort                     = 4443
   SSLCertificate              = ./keys/localhost.cert.pem
   SSLPrivateKey               = ./keys/localhost.key.pem
   X509ClientVerifyCAFile      = ./keys/localhost.ca.pem
   X509ClientVerify            = 1
   X509ClientVerifyDepth       = 15
   ```
2. Also in the Virtuoso INI file, in the [URIQA] section, DefaultHost (localhost:8890 below) must be edited to correspond to the DNS-resolvable host name ("CNAME") of the Virtuoso host, combined with the ServerPort as set in the [HTTPServer] section of the same INI file.

   ```
   [URIQA]
   DynamicLocal = 1
   DefaultHost  = localhost:8890
   ```

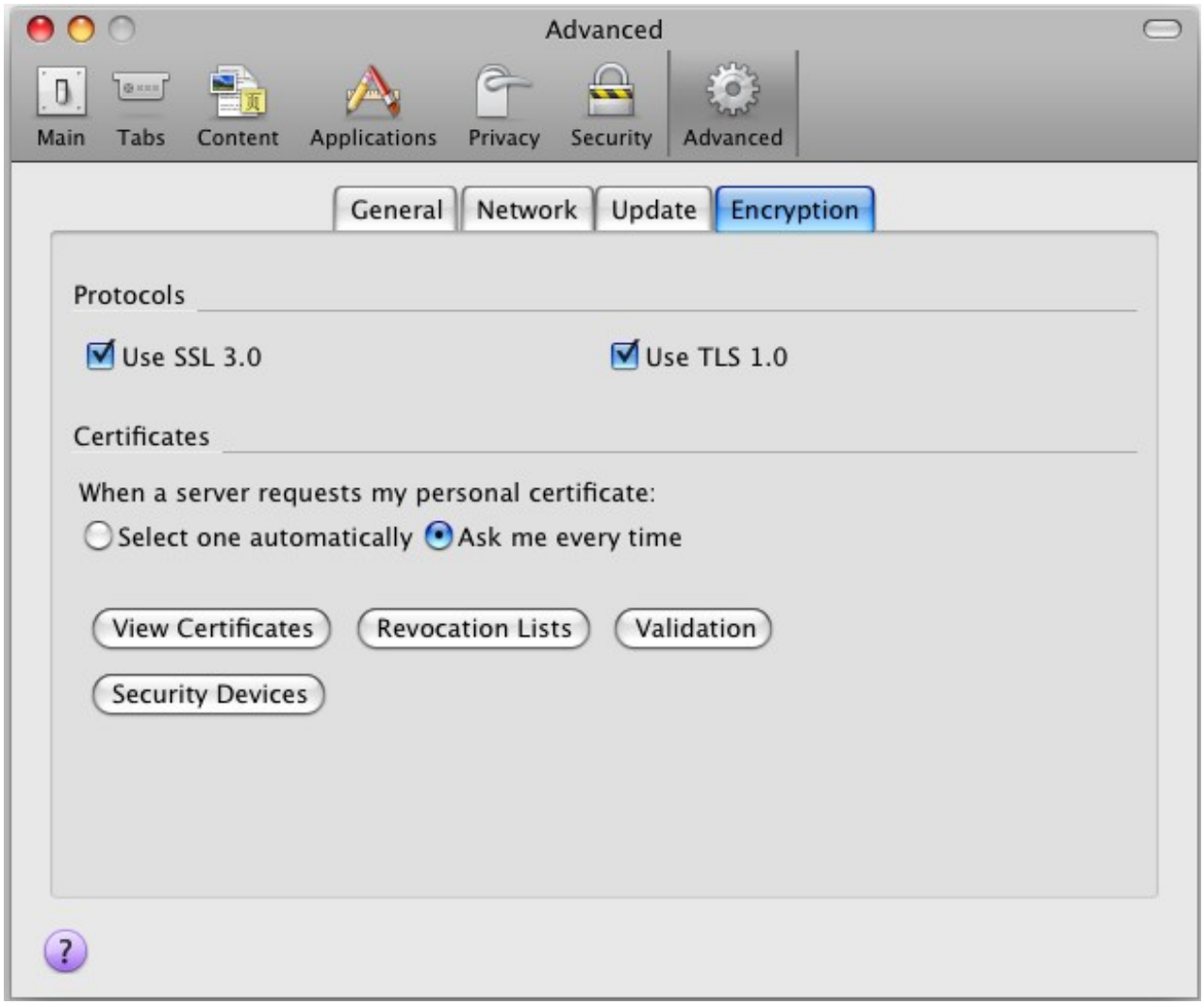
   For instance, if the CNAME of the host is virtuoso.example.com, and the ServerPort is 4321, the DefaultHost should be set to virtuoso.example.com:4321

   ```
   [URIQA]
   DynamicLocal = 1
   DefaultHost  = virtuoso.example.com:4321
   ```
3. Start the Virtuoso server, and look at the log file. Once HTTPS is up, you should see something like:

   ```
   HTTPS Using X509 Client CA ....
   HTTPS/X509 server online at 4443
   ```

17.3.3. Setting Up Firefox

1. **Figure 17.1. Setting Up Firefox**



2. Click the "Add exception" button and enter the address of the HTTPS server you've just configured, i.e., https://virtuoso.example.com:4443/
3. Click OK, and confirm the exception.

Figure 17.2. Setting Up Firefox

4. Click to the "Your Certificates" tab, and import mycert.p12.

17.3.4. Configuring ODS Account to use WebID Protocol

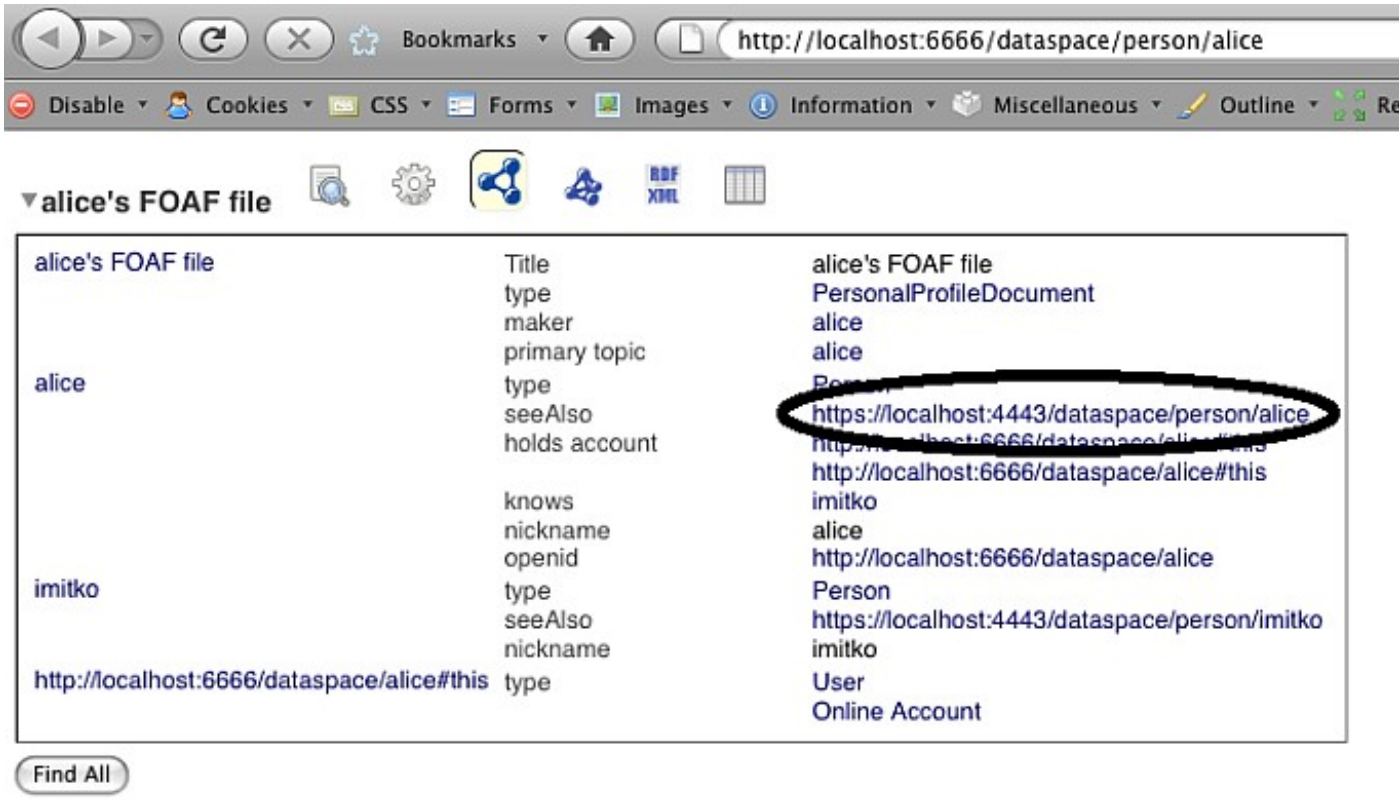
1. Log in to your ODS account, and edit your profile.
2. Click to the Security Tab, and scroll to the bottom, where you will find the X.509 certificate entry area.
3. Copy and paste the PEM format of the certificate (i.e., the content of mykey.pem, from earlier).
4. Press "Save Certificate" button, and you are set.

17.3.5. Testing the setup

To test, we recommend Firefox v3 with the Tabulator extension . Firefox must be set to ask for RDF, as instructed in the Tabulator documentation .

1. Enter an ODS user's URI in the address bar.
2. You should see a protected document's URI. Note that there is no specific "address" data seen:

Figure 17.3. Testing the setup

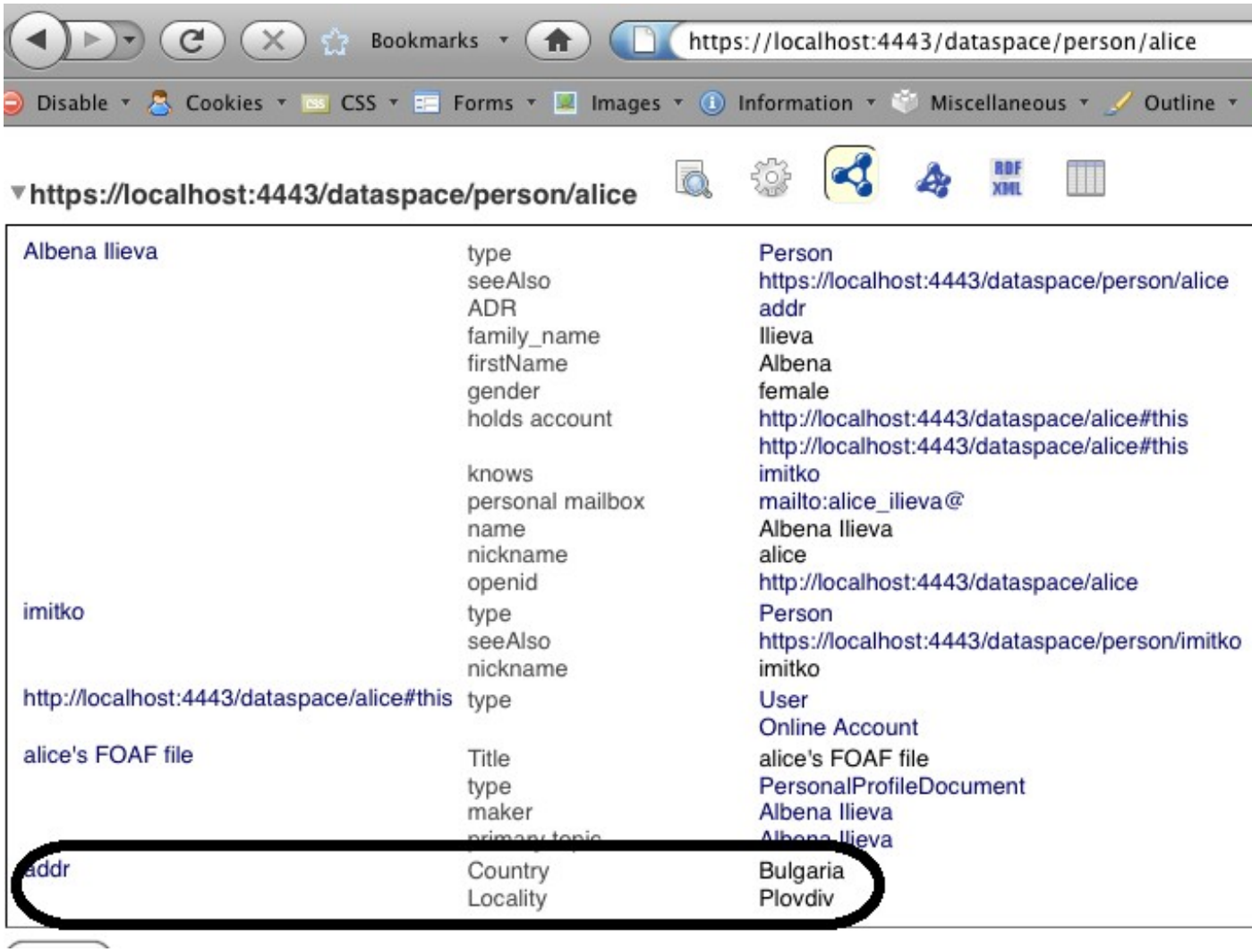


3. When clicked, the browser will ask the user to select a certificate (note: certificate details are erased in the picture below).

Figure 17.4. Testing the setup

4. Now the protected document includes the private address data alongside the previously visible public data!

Figure 17.5. Testing the setup



17.3.6. WebID Protocol ACLs

You can set WebID Protocol ACLs from the Virtuoso Authentication Server UI.

17.3.7. SPARQL-WebID based Endpoint

See details how to create and use a SPARQL-WebID based Endpoint .

17.3.8. CA Keys Import using Conductor

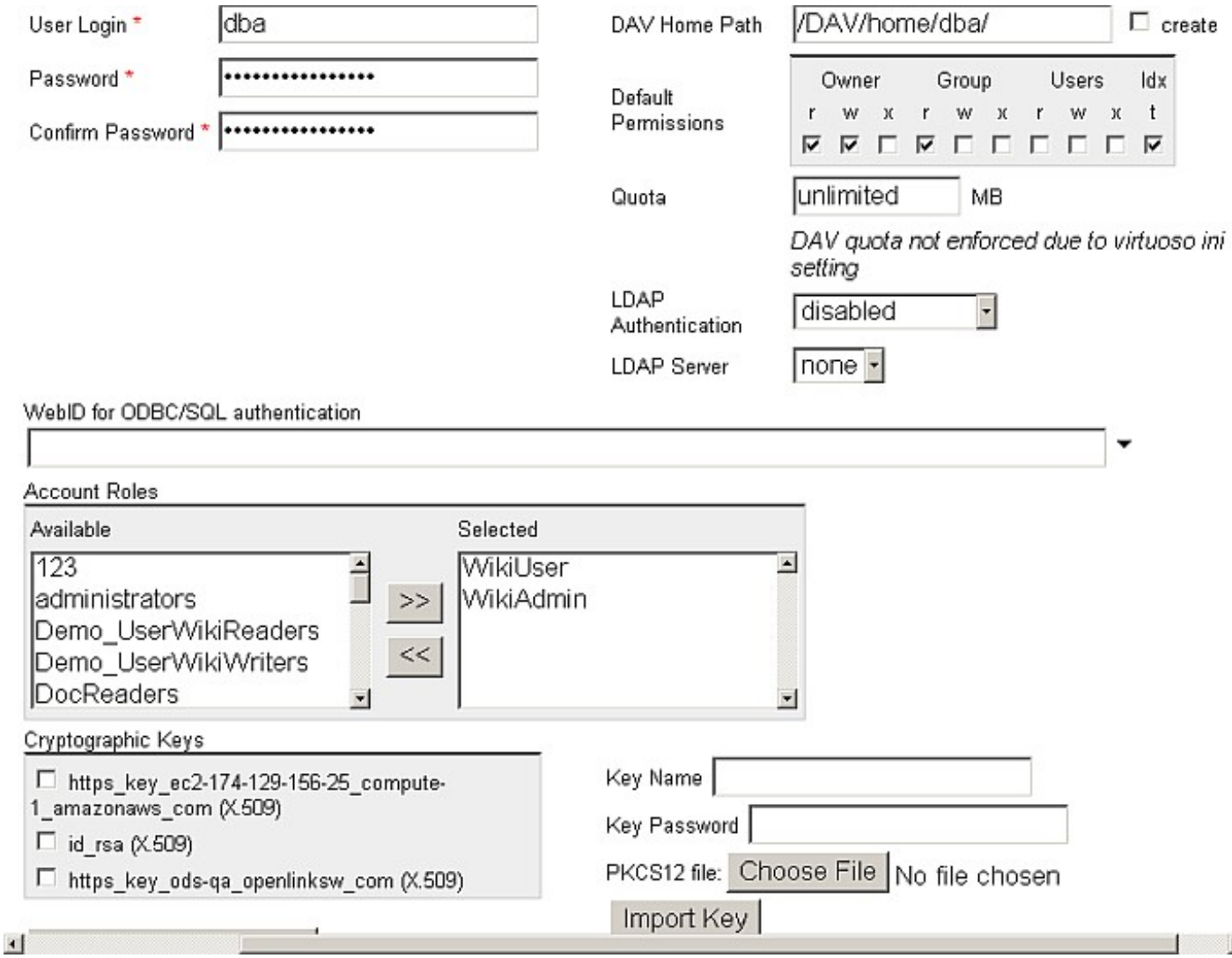
The Virtuoso Conductor allows easy import of user-level CA (Certificate Authority) Keys through System Admin -> User Accounts -> Edit.

The dba user (typically) or other users may need CA keys to execute different services.

For ODS, the dba user must import a certificate with primary key and with name id_rsa. The process takes the following steps:

1. A signing authority (e.g., RSA Labs) generates a site certificate.
2. The site certificate is used to generate certificates for ODS users.
3. The site certificate is imported with name id_rsa for user dba using the Conductor.

Figure 17.6. Conductor CA Keys Import



4. As a result, the ODS user certificates can be authenticated against the site certificate.

If there is a service that requires a different key, the Conductor can be used to import another certificate for the relevant user.

You can use a Key from a global signing authority or produce a self-signed key using the Virtuoso Conductor .

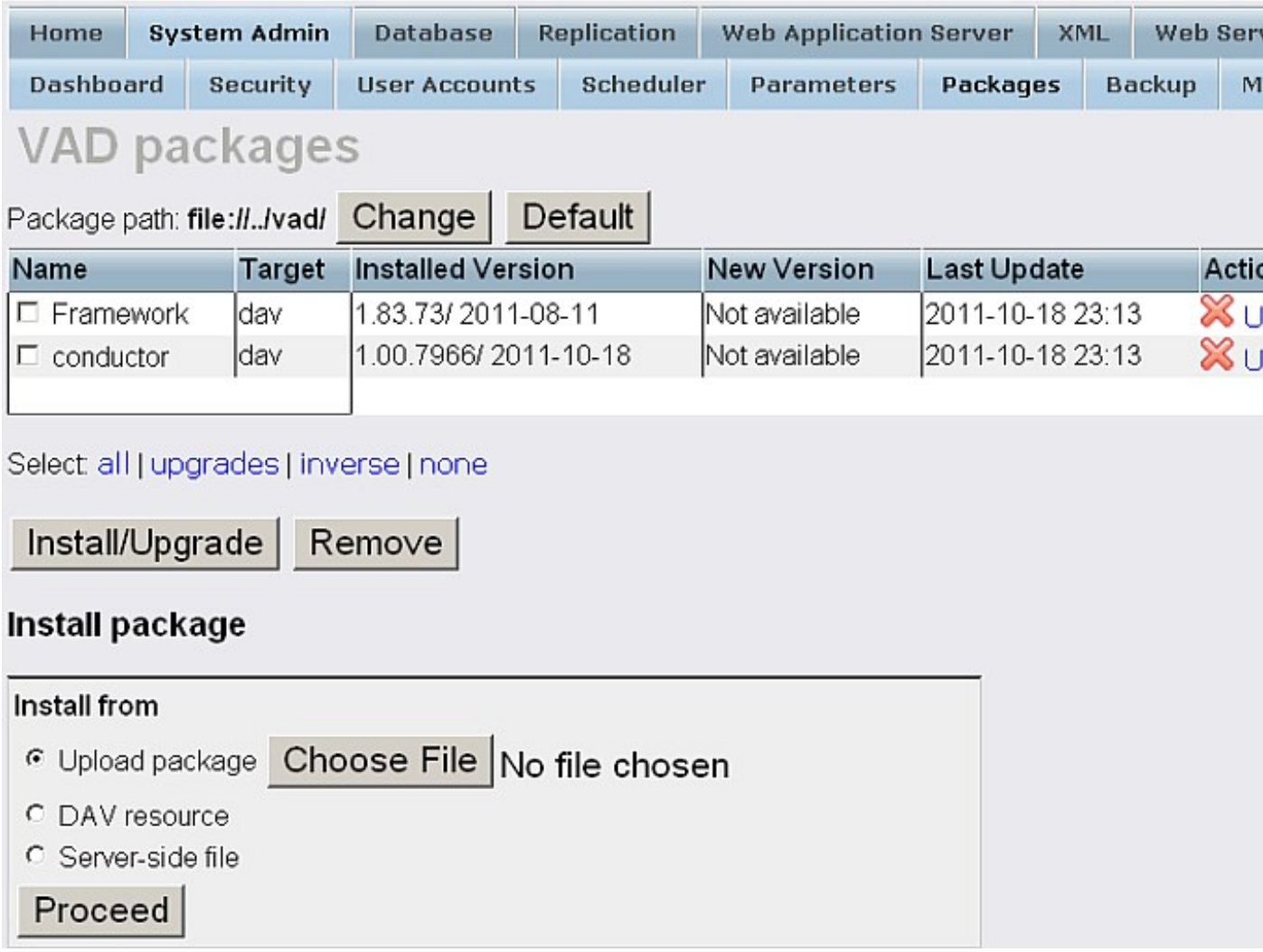
Details and more information how to generate the key see in the next section.

17.3.9. Set Up X.509 certificate issuer, HTTPS listener and generate ODS user's certificates

The following Step-by-Step guide walks you through set up of an X.509 certificate issuer and HTTPS listener, and generation of ODS user certificates.

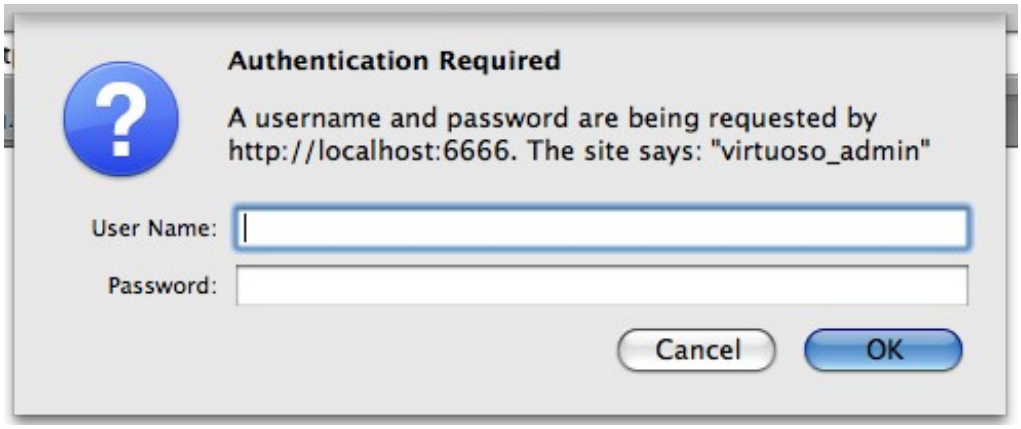
1. Install ODS and Virtuoso Conductor VAD packages.

Figure 17.7. Setting-Up issuer CA



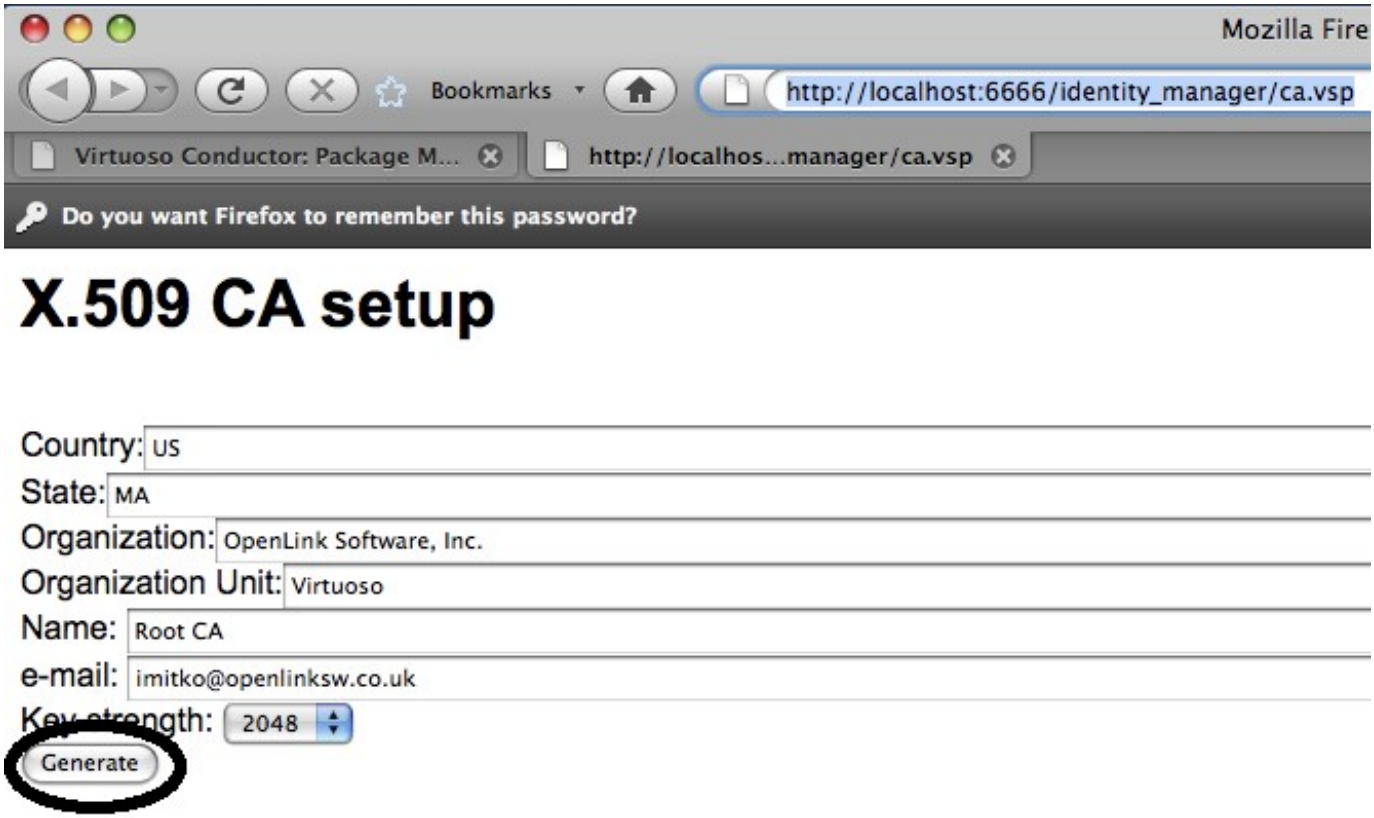
2. Go to the http://cname:port/identity_manager URL, enter the DBA user credentials in the dialog presented.

Figure 17.8. Setting-Up issuer CA



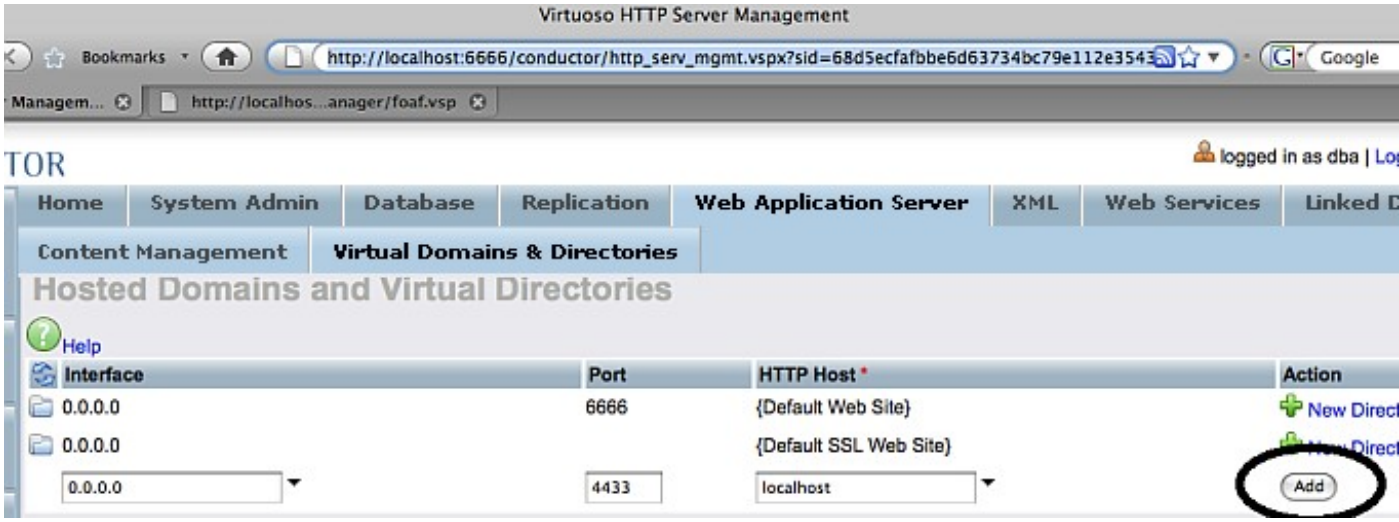
3. Enter the Issuer details and click generate.

Figure 17.9. Setting-Up issuer CA



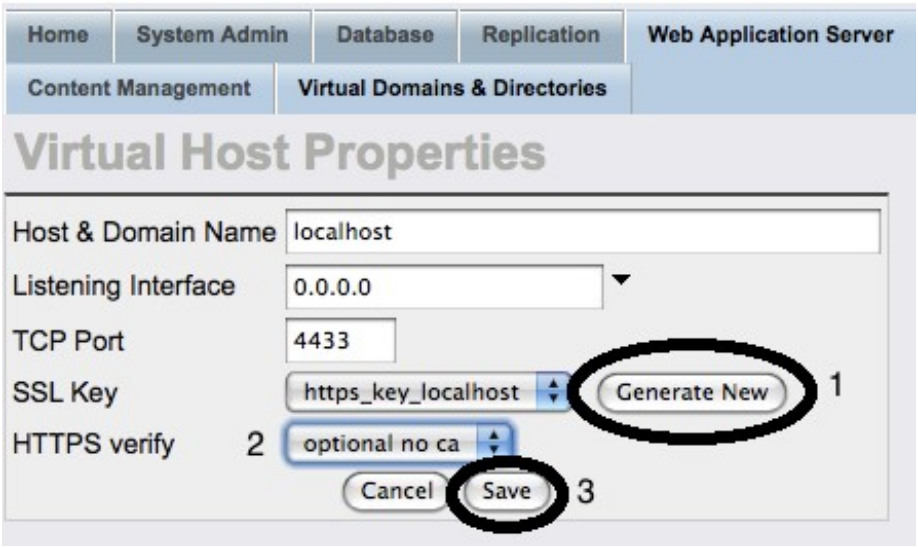
4. Go to Conductor -> Web Application Server -> Virtual Domains & Directories, and add a new listener.

Figure 17.10. Setting-Up issuer CA



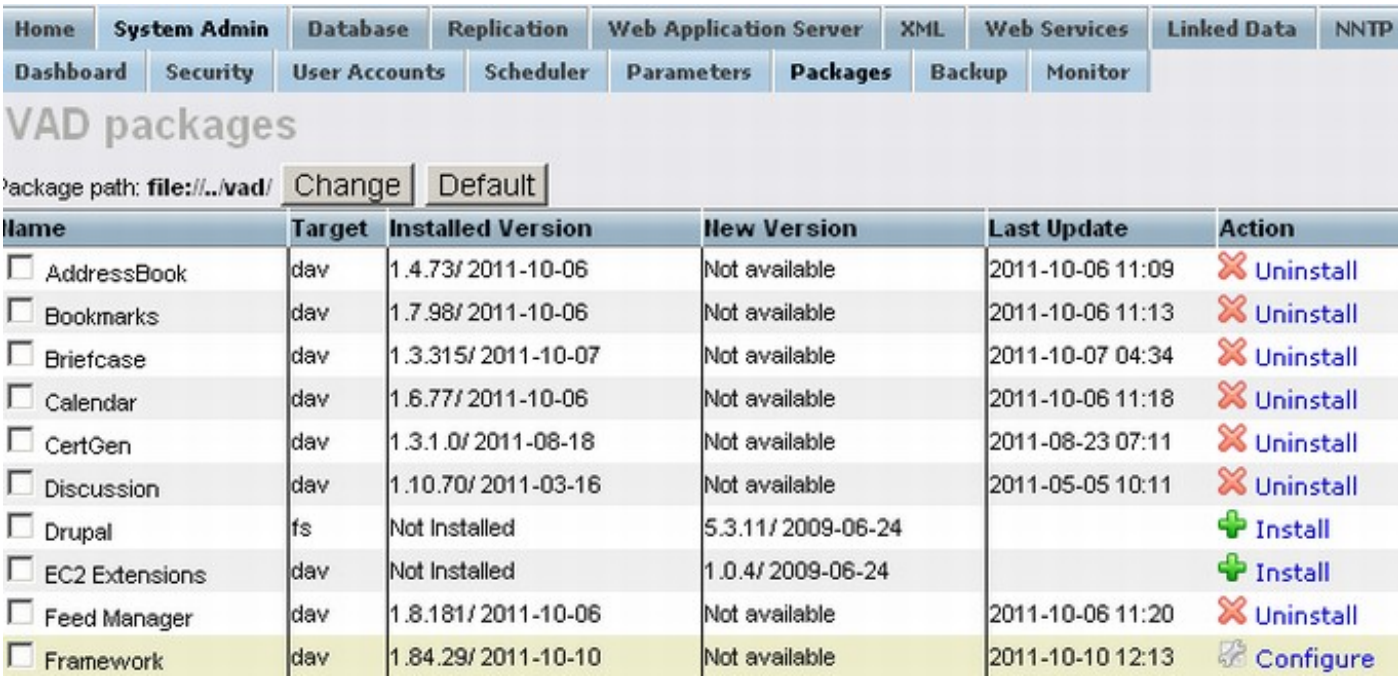
5. Edit the new listener, and generate new key.

Figure 17.11. Setting-Up issuer CA



6. Go to Packages list and select Configure for ODS Framework.

Figure 17.12. Setting-Up issuer CA

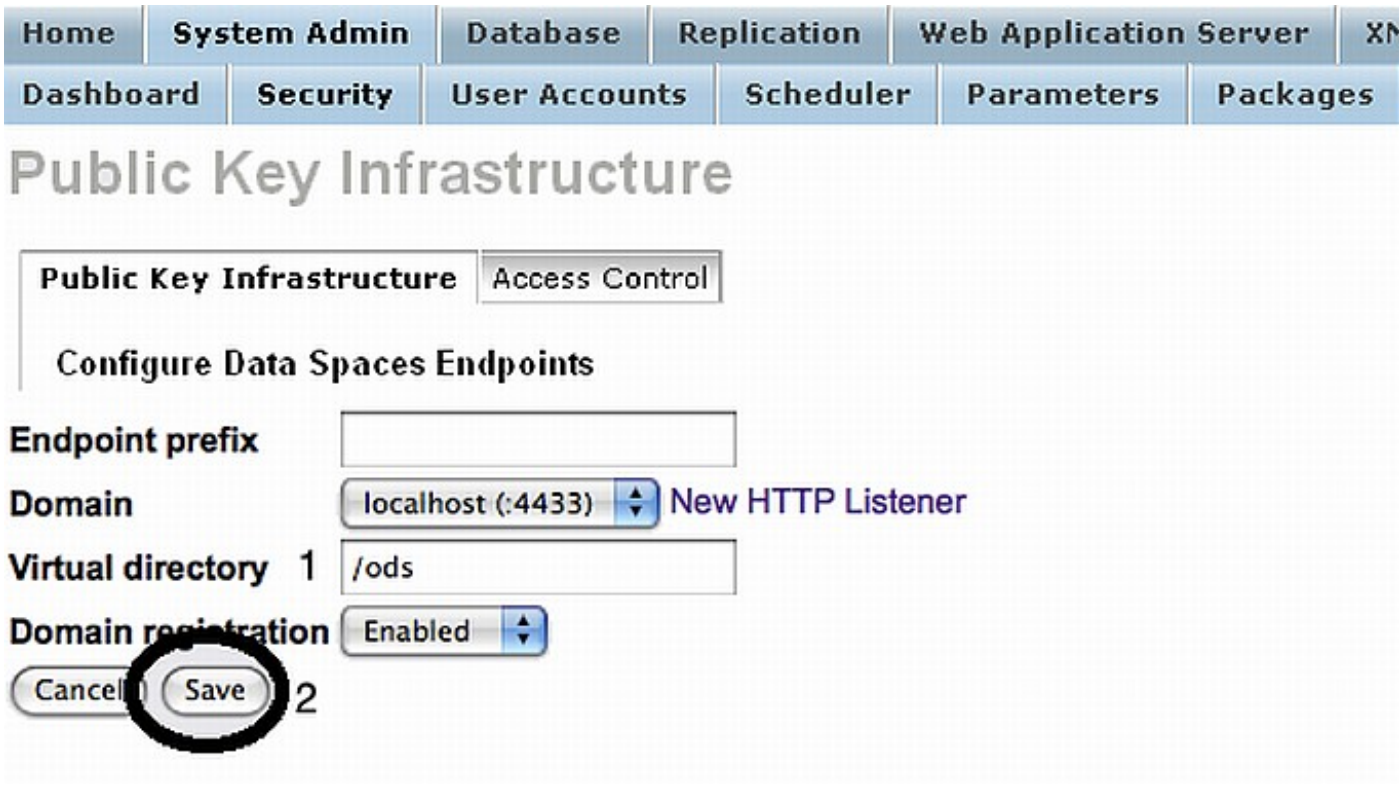


7. Select Create New Endpoint.

Figure 17.13. Setting-Up issuer CA

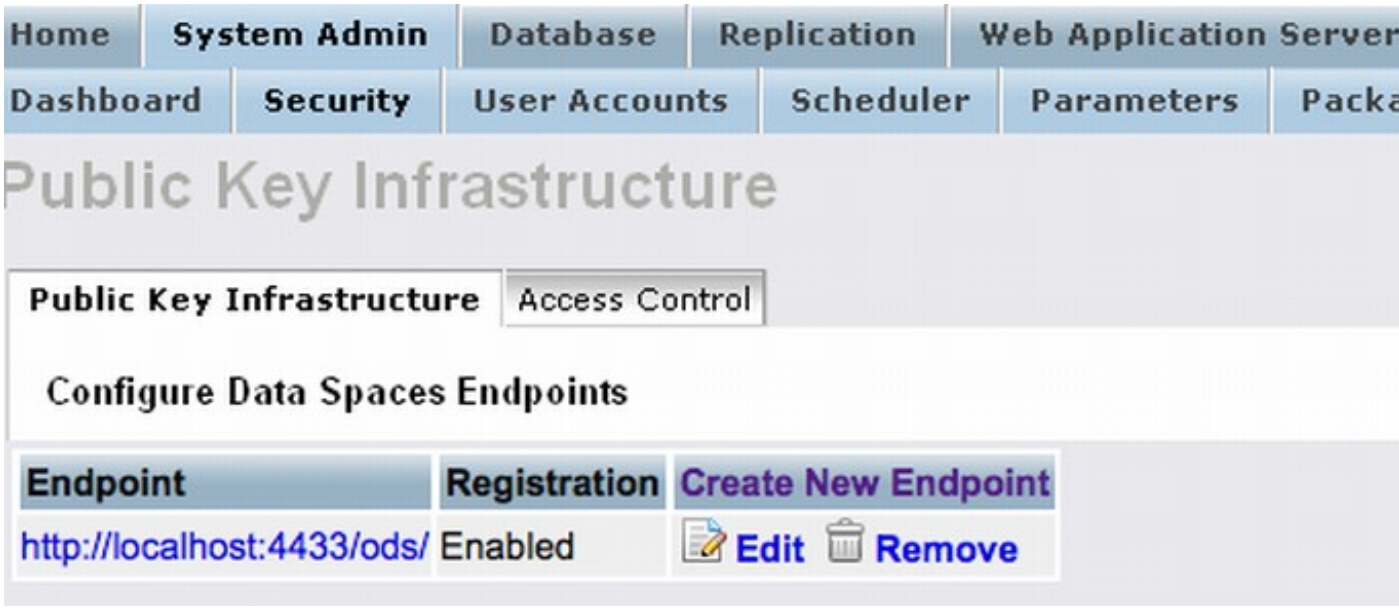
8. Enter the home path for ODS, and save.

Figure 17.14. Setting-Up issuer CA



9. The new endpoint should now appear in the endpoint list.

Figure 17.15. Setting-Up issuer CA

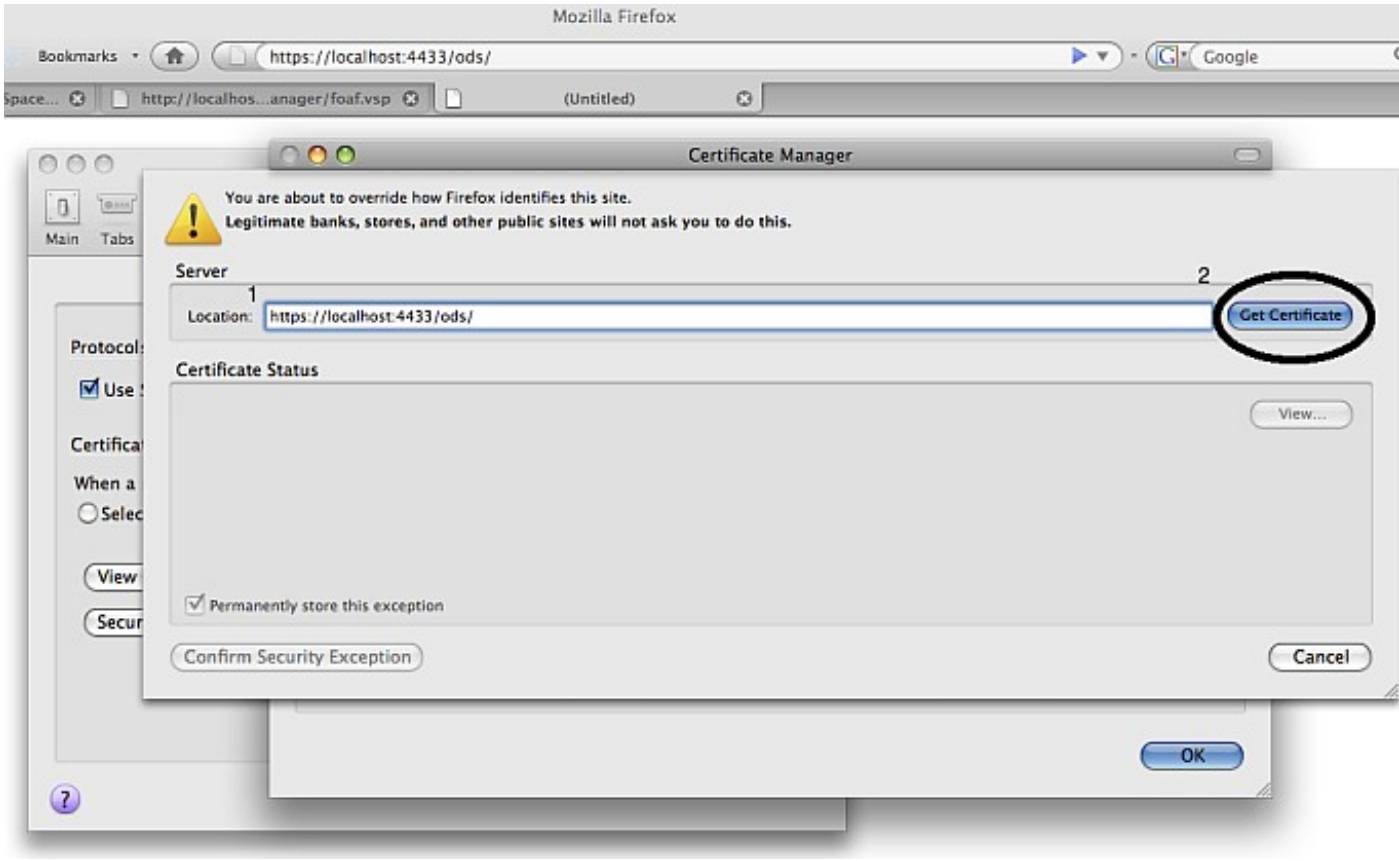


10. Go to the HTTPS site, e.g., https://cname:port/ods; in our example, https://localhost:4433/ods/. If Firefox is used, it will complain that the certificate is not valid, so we must register the site's certificate.

Figure 17.16. Setting-Up issuer CA

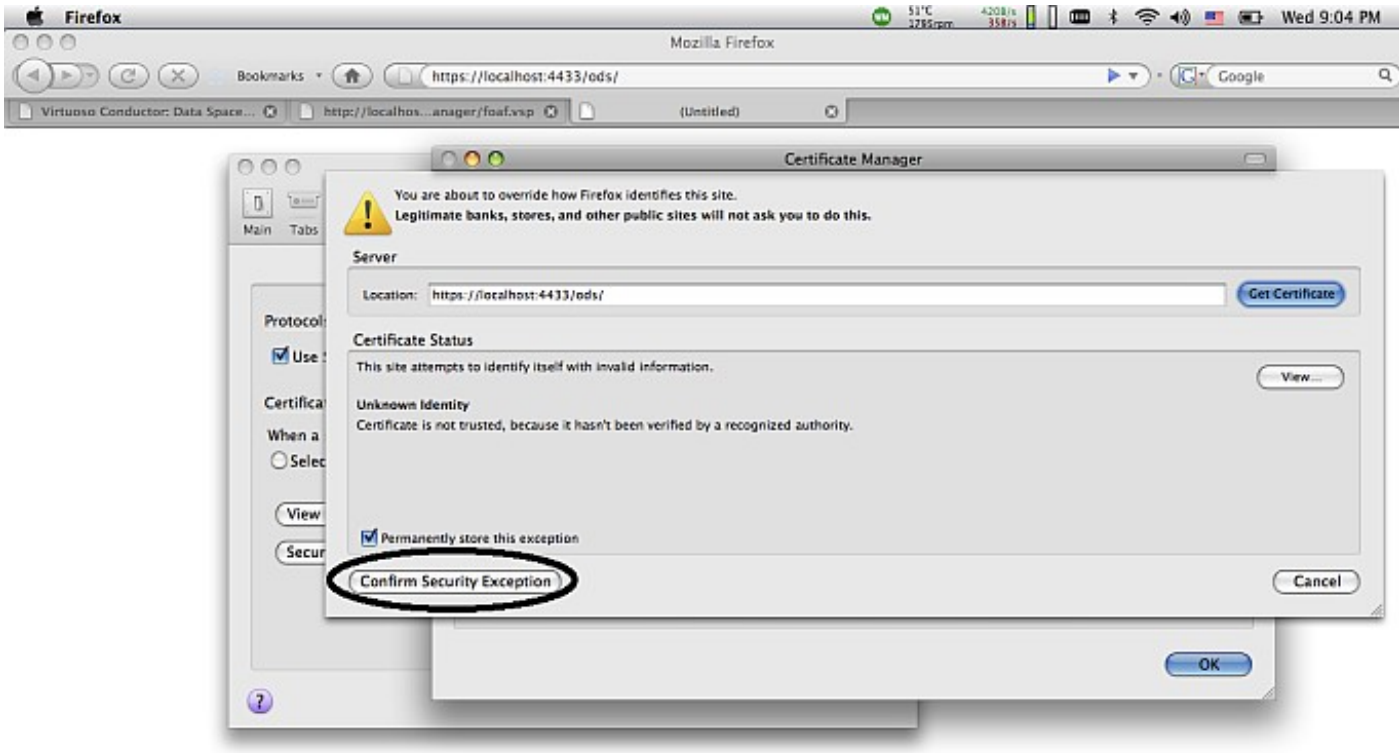
11. In Firefox certificate manager, in Site's certificates, add an exception.

Figure 17.17. Setting-Up issuer CA



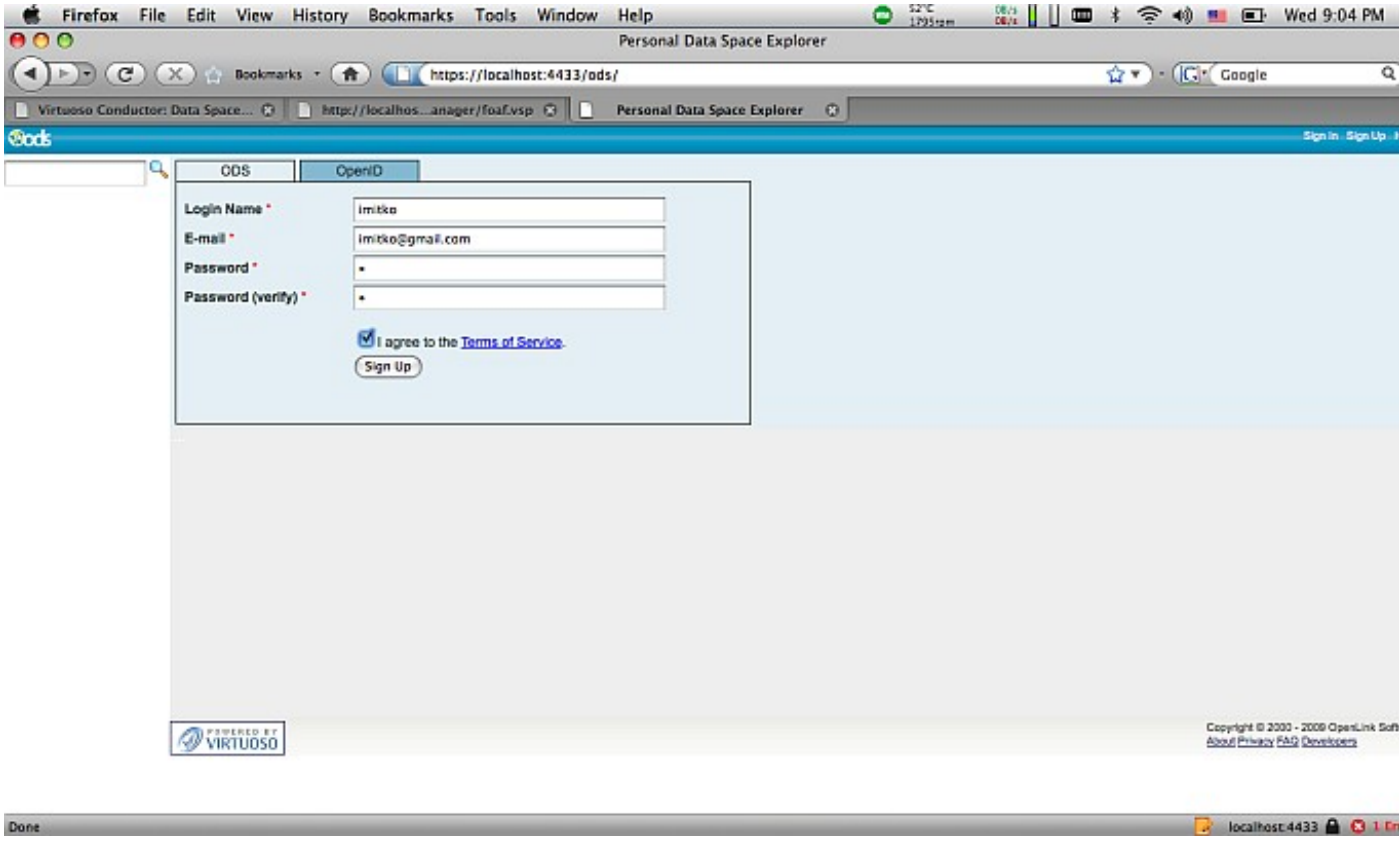
12. Confirm exception.

Figure 17.18. Setting-Up issuer CA



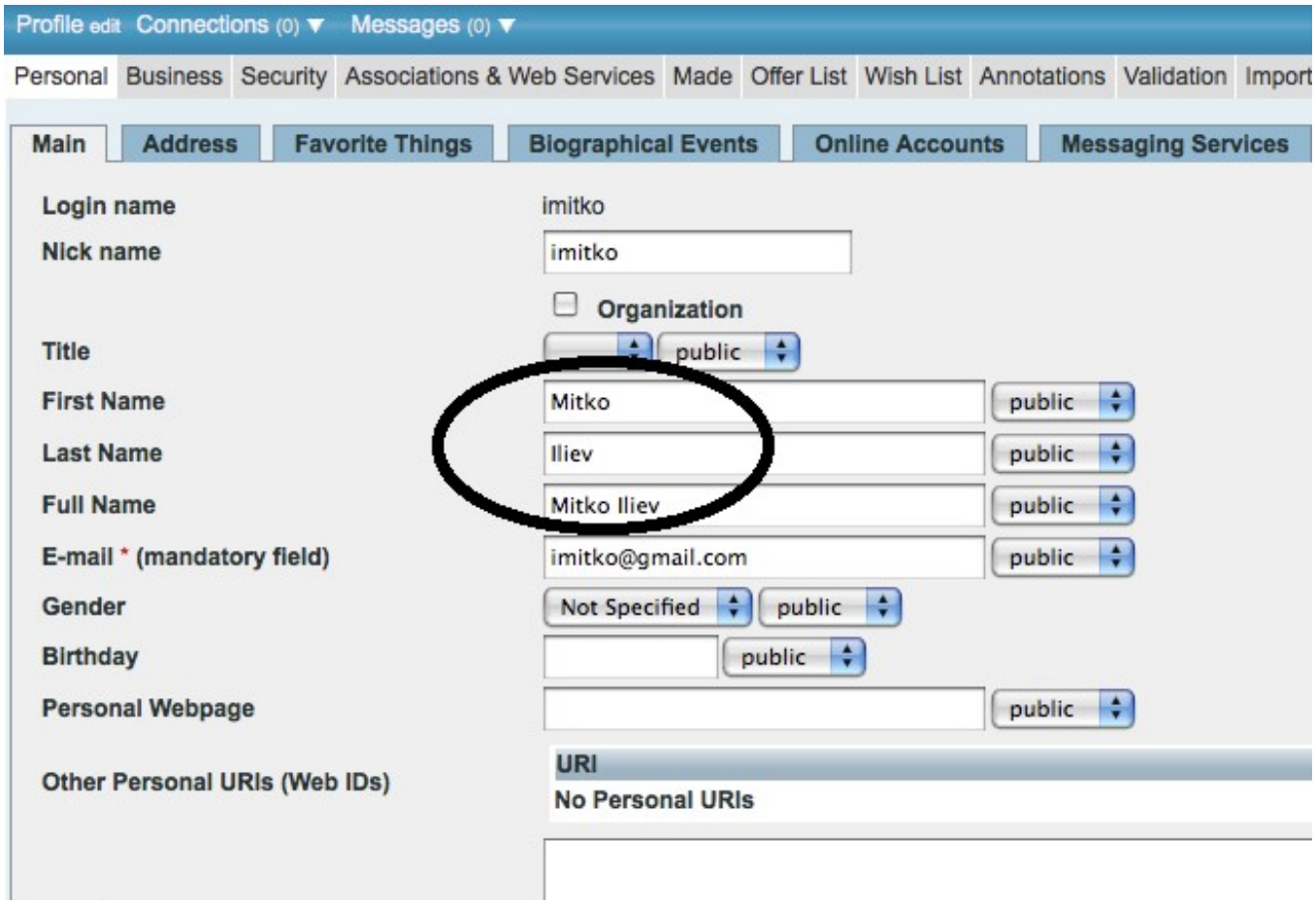
13. Return to ODS site, and register new user.

Figure 17.19. Setting-Up issuer CA



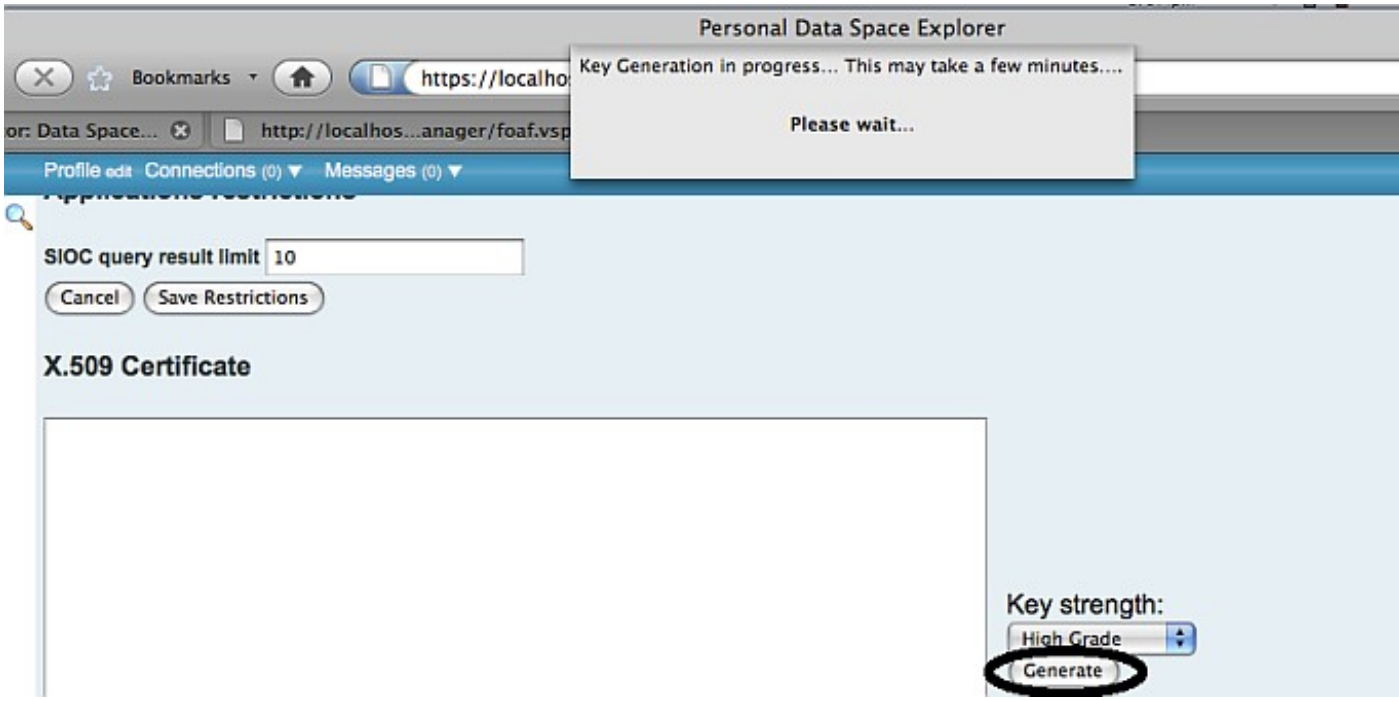
14. Edit the user's profile and enter his/her name(s). If this step is skipped, the certificate will not have a human readable name.

Figure 17.20. Setting-Up issuer CA



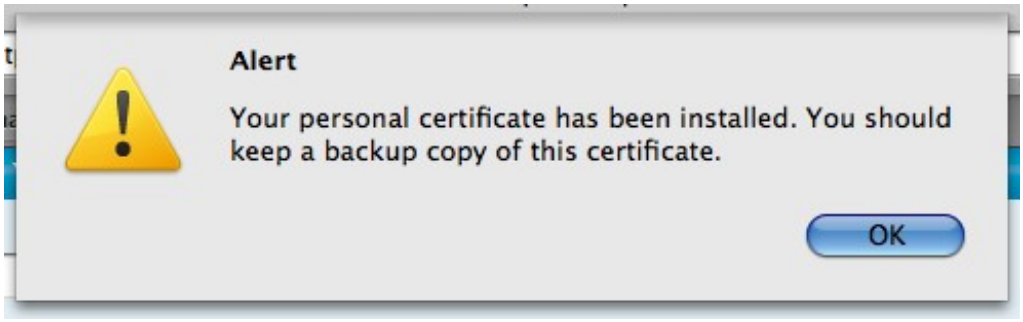
15. Open the Security tab in the Profile Editor, and generate the client key.

Figure 17.21. Setting-Up issuer CA



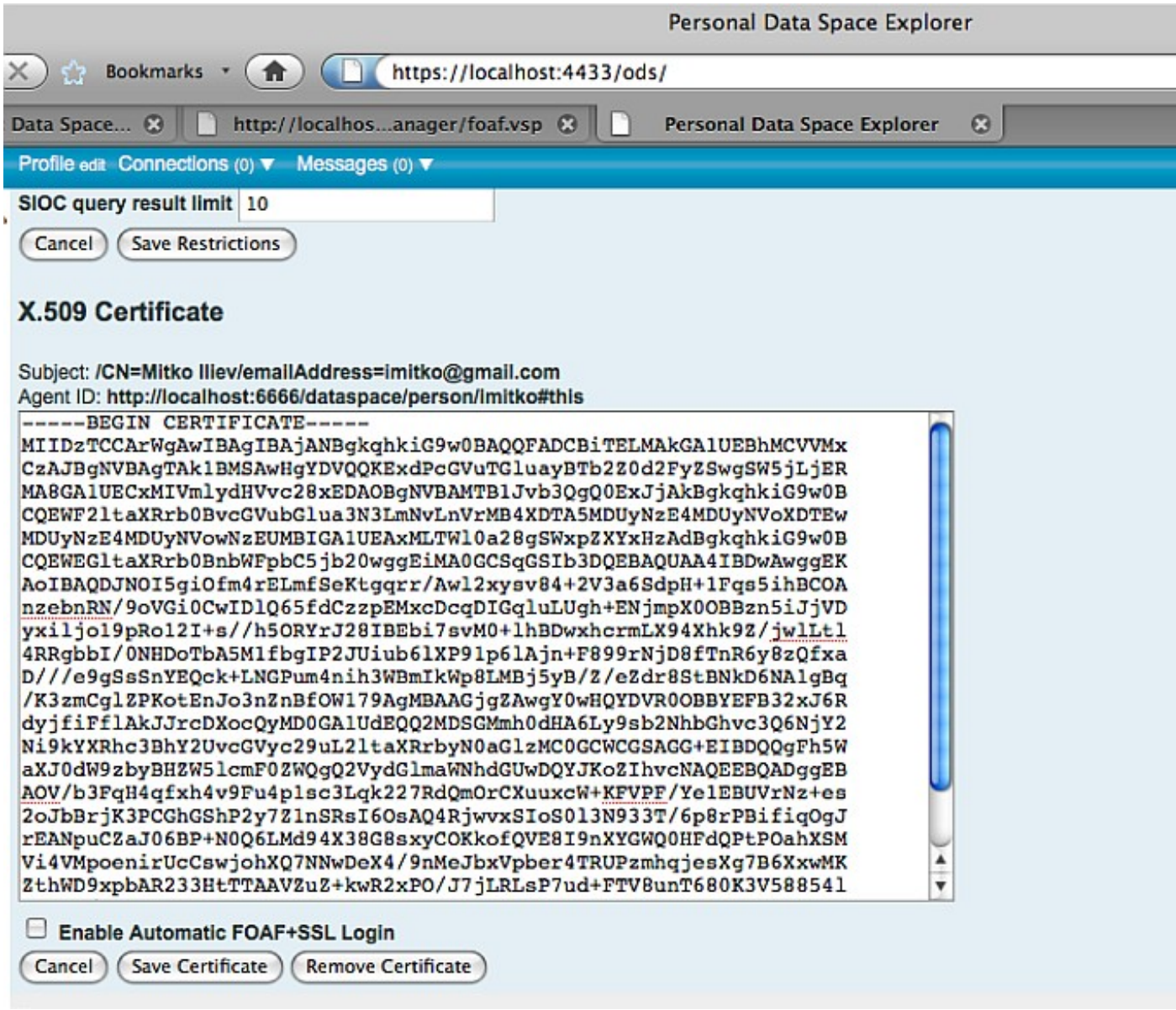
16. If all is set up correctly, you should see this message. It means Firefox has the private key, and has obtained a new certificate from the server.

Figure 17.22. Setting-Up issuer CA



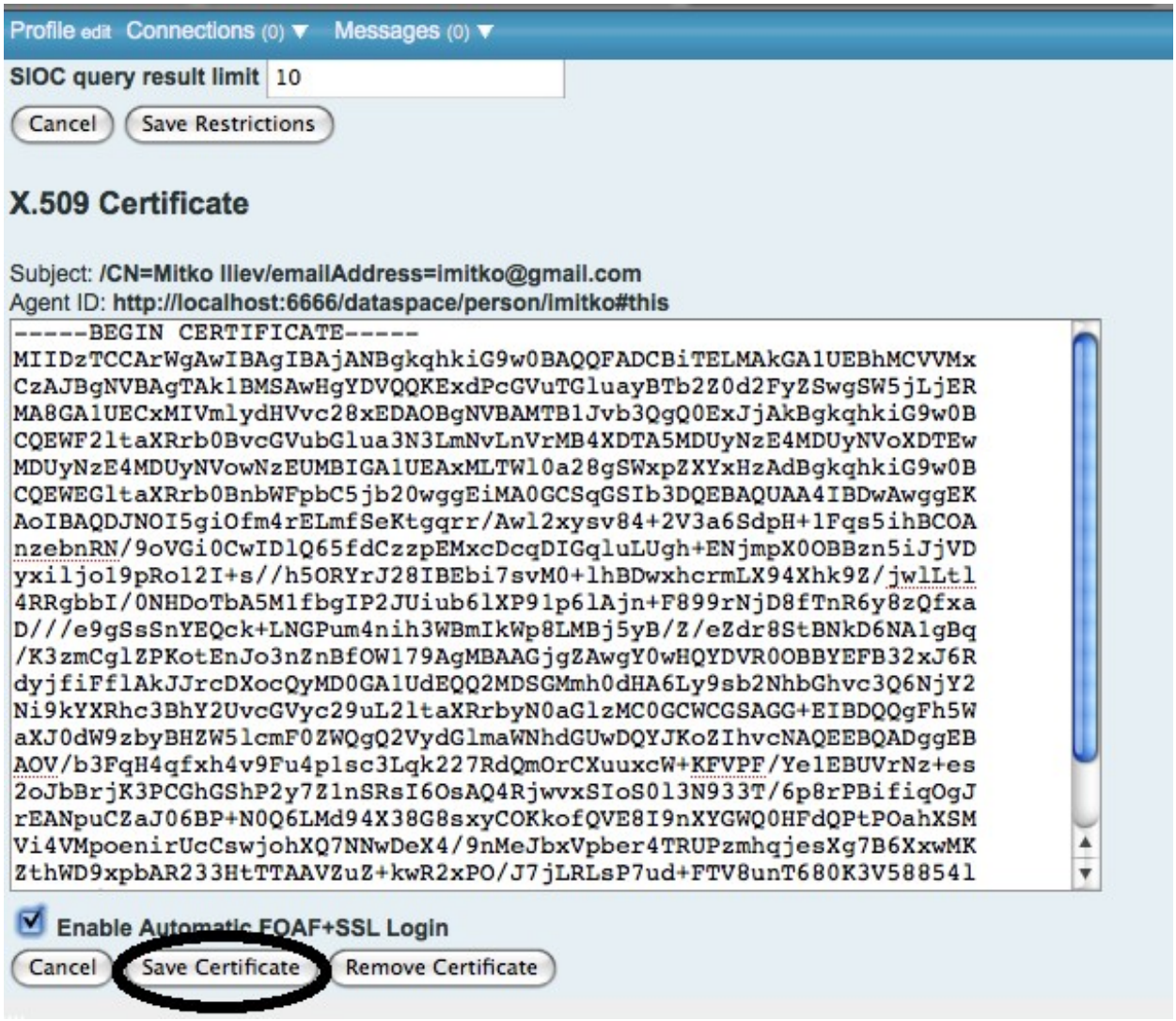
17. Refresh the Security tab by clicking on same tab.

Figure 17.23. Setting-Up issuer CA



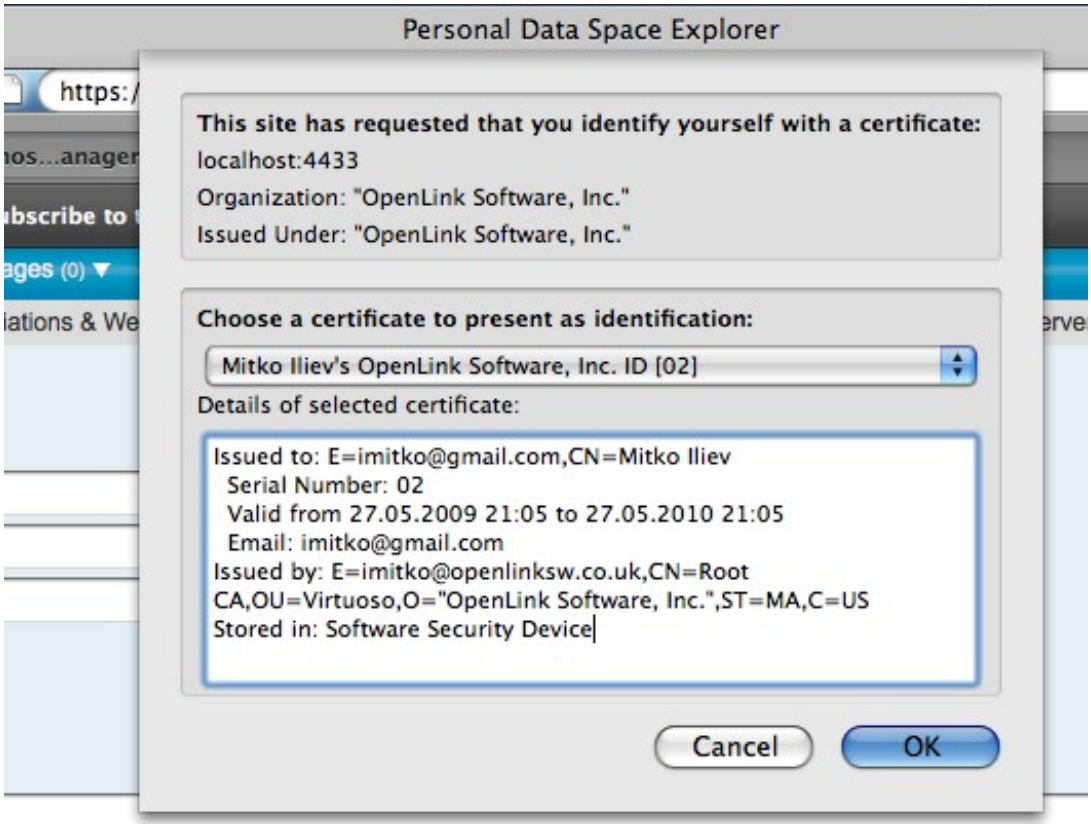
18. Select automatic login option, and save.

Figure 17.24. Setting-Up issuer CA



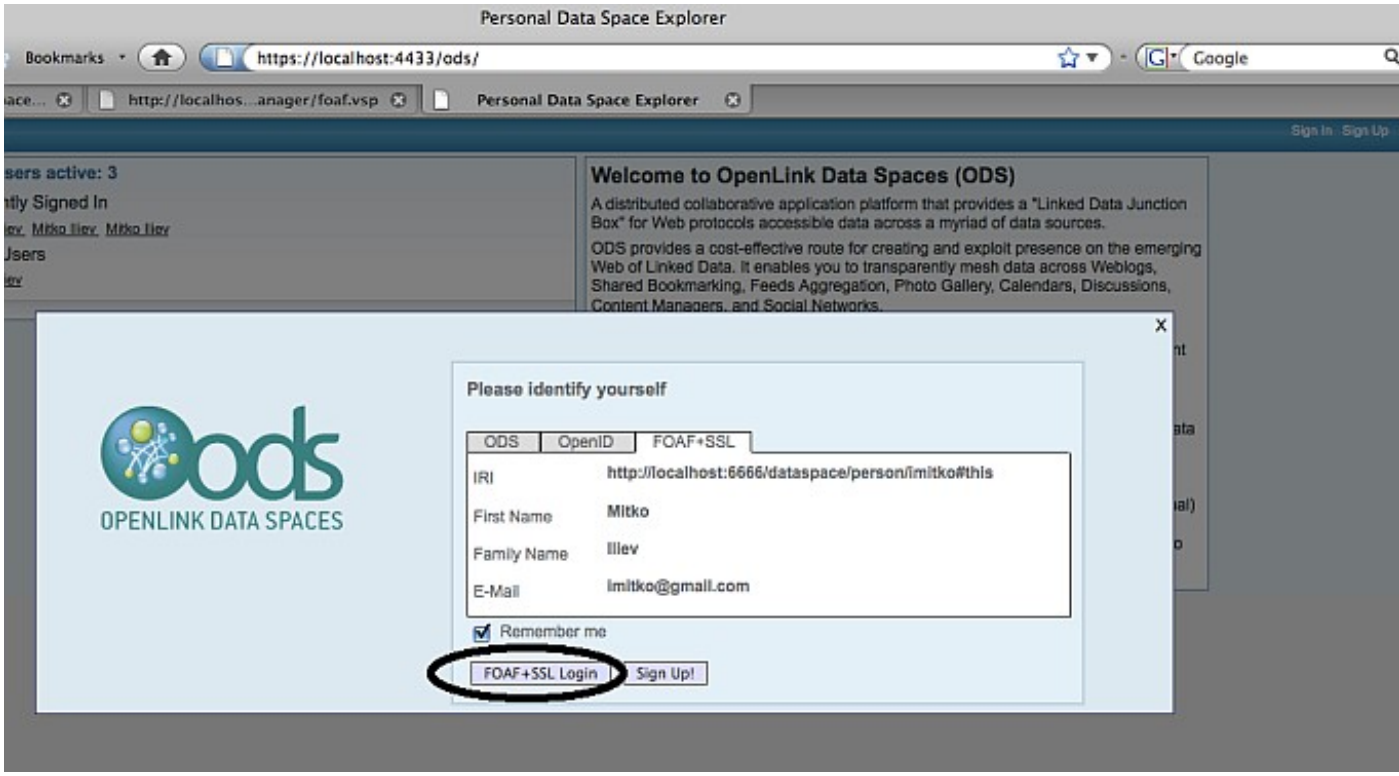
19. Log out from ODS and refresh browser to simulate opening the ODS site. The browser will ask for a certificate; select the one generated in the steps above.

Figure 17.25. Setting-Up issuer CA



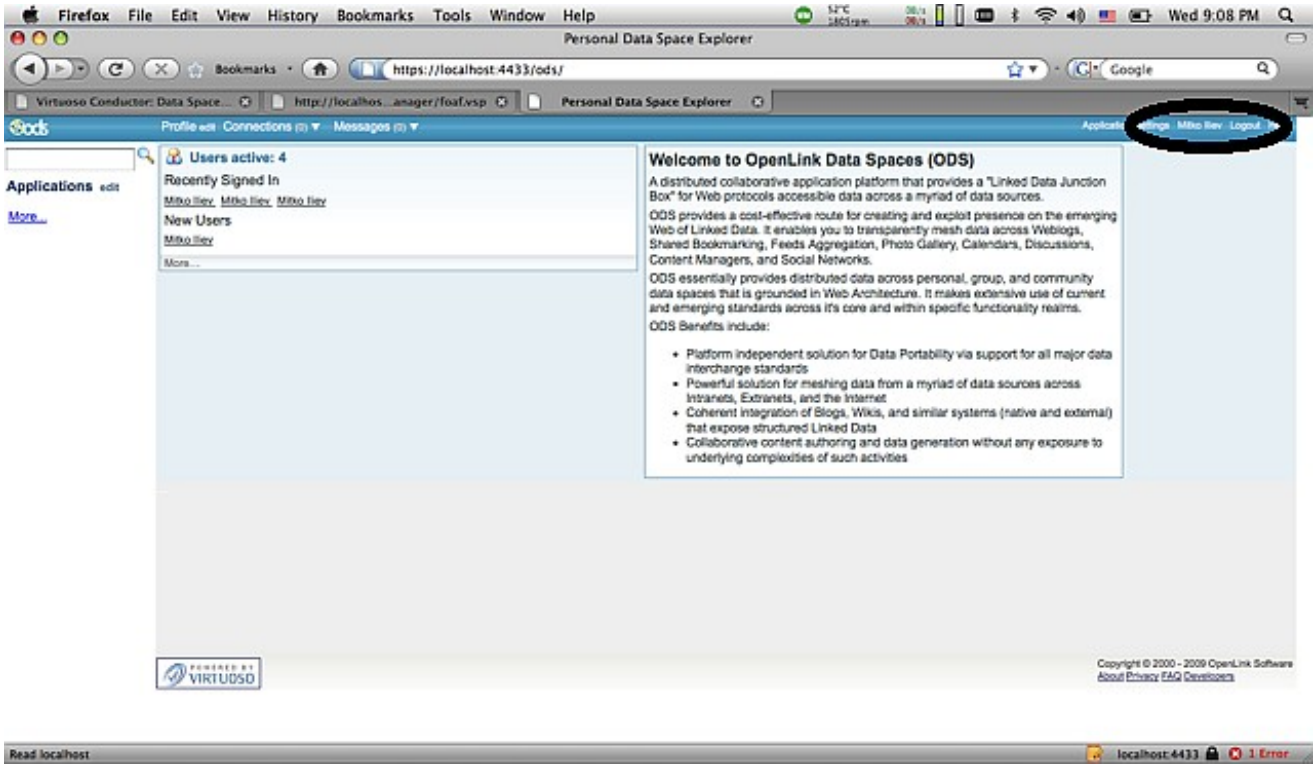
20. ODS presents your card, and asks to login with certificate. Confirm it.

Figure 17.26. Setting-Up issuer CA



21. You should now be logged in to ODS via WebID Protocol.

Figure 17.27. Setting-Up issuer CA



17.3.10. WebID Protocol ODBC Login

See details and examples here .

17.4. OAuth Support

The OAuth protocol enables websites or applications (Consumers) to access Protected Resources from Web services (Service Providers) via an API, without requiring Users to disclose their Service Provider credentials to those Consumers. More generally, OAuth creates a freely-implementable and generic methodology for API-oriented authentication.

For Consumer developers, OAuth is a method to publish and interact with protected data.

For Service Provider developers, OAuth gives users access to their data while protecting their account credentials.

One use case would be allowing a printing service, printer.example.com (the Consumer), to access private photos stored on photos.example.net (the Service Provider), without requiring Users to reveal their photos.example.net credentials to printer.example.com.

OAuth allows the user to selectively grant access to their private resources housed on one site (called the Service Provider), to another site (called the Consumer). In other words, OAuth enables you to grant access to some of your information without sharing all of your identity.

OAuth does not require a specific user interface or interaction pattern, nor does it specify how Service Providers authenticate Users, making the protocol ideally suited for cases where authentication credentials are unavailable to the Consumer, such as with OpenID.

OAuth aims to unify the experience and implementation of delegated web service authentication with a single, community-driven protocol. OAuth builds on existing protocols and best practices that have been independently implemented by various websites. An open standard, supported by large and small providers alike, promotes a consistent and trusted experience for both application developers and the users of those applications.

What is publicly known as "OAuth" is really the "OAuth Core 1.0" specification. The "Core" designation is used to stress that this is the skeleton upon which other extensions and protocols may be built. OAuth Core 1.0 does NOT by itself provide many desired features such as automated discovery of endpoints, language support, support for XML-RPC and SOAP, standard definition of resource access, OpenID integration, a full range of signing algorithms, or any number of other great ideas posted to the OAuth

group.

This was intentional and is viewed by the authors as a benefit. As the name implies, Core deals only with the most fundamental aspects of the protocol:

- Establish a mechanism for exchanging a username and password for a token with defined rights
- Provide tools to protect these tokens

More details can be found here .

17.4.1. OAuth Access Tokens

Credentials bearing tokens enable a user to provide their credentials in tokenized form in cases where HTTP redirection to a browser plus human interaction is unavailable or unsuitable. For example, intermediary intelligent agents, mobile phones, or set-top devices.

Request Authentication

To request an Access Token in this model, the Consumer makes an HTTP request to the Service Provider's Access Token URL. The authentication request contains [nine] parameters contained in the HTTP Authorization header or as URL parameters. Parameter names and values must be "percent-encoded" to handle characters in different character sets. The request should be performed using TLS, and should use HTTP POST.

Receive Authentication

Before granting an access token, the Service Provider must ensure that the request signature has been successfully verified as per OAuth, that a request with the supplied timestamp and nonce has never been received before, and that the supplied username and password match a User's credentials. If successful, the Service Provider generates an Access Token and Token Secret using a 200 Ok response and returns them in the HTTP response body.

Access Protected Resources

After successfully receiving the Access Token and Token Secret, the Consumer is able to access the Protected Resources on behalf of the User as per section 7 of the OAuth core specification. In other words, the Access Token obtained here is no different in capability to the Access Token specified by OAuth. Once authenticated using the above process, the Consumer will sign all subsequent requests for the User's Protected Resources using the returned Token Secret.

17.4.2. Virtuoso OAuth server

Virtuoso implements the OAuth Core 1.0 specification, and exposes the following API endpoints:

Request token

Endpoint:

```
http://server-cname/OAuth/request_token
```

Parameters:

- *oauth_consumer_key* : The Consumer Key.
- *oauth_signature_method* : The signature method the Consumer used to sign the request.
- *oauth_signature* . The signature as defined in Signing Requests (Signing Requests).
- *oauth_timestamp* : As defined in Nonce and Timestamp (Nonce and Timestamp).
- *oauth_nonce* : As defined in Nonce and Timestamp (Nonce and Timestamp).
- *oauth_version* : OPTIONAL. If present, value MUST be 1.0 .

Example

Request:

```
http://example.com/OAuth/request_token?oauth_version=1.0&oauth_nonce
```

```
=dad4cb071e2169cbcaa051d404ac61a3&oauth_timestamp=1201873644&oauth_cons
umer_key=f756023be5ff1f20881cf8fe398069f3976b2304&oauth_signature_metho
d=HMAC-SHA1&oauth_signature=z76k5fQ0msFsQzCmhO%2FJZ329ZUE%3D
```

Note: all long lines in example texts are split, i.e., the GET request is single line.

Response:

```
oauth_token=b4e22daa117b0bebf60ab6ba6e401edc7addd78c&oauth_token_secret
=4de6e3ab17553a0a385ebf6a3b4dd30f
```

Authorization

Endpoint:

```
http://server-cname/OAuth/authorize
```

Parameters:

- *oauth_token* : The Request Token obtained in the previous step. The current implementation of the Service Provider declare this parameter as REQUIRED.
- *oauth_callback* : OPTIONAL. The Consumer MAY specify a URL the Service Provider will use to redirect the User back to the Consumer when Obtaining User Authorization (Obtaining User Authorization) is complete.

Example

Request:

```
http://example.com/OAuth/authorize?oauth_token=b4e22daa117b0bebf60ab
6ba6e401edc7addd78c&oauth_callback=http%3A%2F%2Flocalhost%3A8890%2Foaut
h%2Fexample%2Fclient.php%3Fkey%3Df756023be5ff1f20881cf8fe398069f3976b23
04%26secret%3Dcc249bfb732039d8ecba9e4f94fdead7%26token%3Db4e22daa117b0b
ebf60ab6ba6e401edc7addd78c%26token_secret%3D4de6e3ab17553a0a385ebf6a3b4
dd30f%26endpoint%3Dhttp%253A%252F%252Flocalhost%253A8890%252FOAuth%252F
authorize
```

The User will be asked via web page to accept or decline the token.

Access token

Endpoint:

```
http://server-cname/OAuth/access_token
```

Parameters:

- *oauth_consumer_key* : The Consumer Key.
- *oauth_token* : The Request Token obtained previously.
- *oauth_signature_method* : The signature method the Consumer used to sign the request.
- *oauth_signature* : The signature as defined in Signing Requests (Signing Requests).
- *oauth_timestamp* : As defined in Nonce and Timestamp (Nonce and Timestamp).
- *oauth_nonce* : As defined in Nonce and Timestamp (Nonce and Timestamp).
- *oauth_version* : OPTIONAL. If present, value MUST be 1.0 .

Example

Request:

```
http://example.com/OAuth/access_token?oauth_version=1.0&oauth_nonce=
8ad75091a66bdd741472be42149c828e&oauth_timestamp=1201873800&oauth_consu
mer_key=f756023be5ff1f20881cf8fe398069f3976b2304&oauth_token=b4e22daa11
7b0bebf60ab6ba6e401edc7addd78c&oauth_signature_method=HMAC-SHA1&oauth_s
ignature=tCxy0Lod4%2Bp%2FCBPV7Ph7RrsHXe4%3D
```

Response:

```
oauth_token=8c03b3da93480ca4728cc1194d6d03962f3bb5bb&oauth_token_secret
=854fd29c00adcedff4fbeaeb96584911
```

In addtion to the endpoints it define a API for PL applications to check authentication:

Authentication verification

```
OAUTH.DBA.check_authentication (in params any, in lines any)
```

Parameters:

- *params* : an array of strings representing the HTTP parameters
- *lines* : an array of HTTP request headers

Result:

- on success it returns integer 1,
- on failure it signal an SQL error.

Sample code:

A sample service (oauth.vsp):

```
<html>
<body>
<?vsp
  OAUTH..check_authentication (params, lines);
?>
An OAuth testing page
</body>
</html>
```

Sample request:

```
http://example.com/admin/oauth.vsp?oauth_version=1.0&oauth_nonce=d57
640869b994b2d51bf9800229c4997&oauth_timestamp=1201873935&oauth_consumer
_key=f756023be5ff1f20881cf8fe398069f3976b2304&oauth_token=8c03b3da93480
ca4728cc1194d6d03962f3bb5bb&oauth_signature_method=HMAC-SHA1&oauth_sign
ature=X3K4lr9bJVz5YLnnyJDkykQZivY%3D
```

Sample request:

```
http://example.com/admin/oauth.vsp?oauth_version=1.0&oauth_nonce=d57
640869b994b2d51bf9800229c4997&oauth_timestamp=1201873935&oauth_consumer
_key=f756023be5ff1f20881cf8fe398069f3976b2304&oauth_token=8c03b3da93480
ca4728cc1194d6d03962f3bb5bb&oauth_signature_method=HMAC-SHA1&oauth_sign
ature=X3K4lr9bJVz5YLnnyJDkykQZivY%3D
```

Sample response:

```
<html>
<body>
An OAuth testing page
</body>
</html>
```

17.4.3. OAuth Implementation in OpenLink Data Spaces

OAuth tokens must be generated for each user account, for each ODS application, via ODS -> Settings -> OAuth Keys. This UI provides a list of all applications to which the logged-in user has access, i.e., of which the user is a member or owner. To enable access with OAuth, the user simply selects the desired application instance from the list, and clicks the 'generate keys' button. The generated Consumer Key and Secret will be associated with the given ODS user account & application instance.

Once a consumer token is available, the sequence below must be executed in order to establish an authorized session:

1. client uses request_token to get a token/secret pair for establishing session using consumer token

2. client asks OAuth server to authorize the token from step 1.
3. using authorized token from step 1 client asks OAuth server for authentication token.
4. with authentication token from step 3, clients can access instance data associated with the consumer token from step 1.

Sample OAuth Client

To demonstrate the Virtuoso implementation of OAuth, we have written this sample client, in Virtuoso/PL.

```
<html>
<?vsp
    declare consumer_key, consumer_secret, oauth_token, oauth_secret, signature, timest, nonce varchar;
    declare srv, res, signature_base, ret_url, url, tmp, sid varchar;
    declare meth, pars any;
    consumer_key := {?'key'};
    srv := sprintf ('http://example.com:%s/OAuth/', server_http_port ());
    sid := null;
    res := '';

    sid :=  {?'sid'};
    if (sid = '')
      sid := null;
    else
      {
        -- if selected token is not same as one from the session we need to get new authentication token
        if (consumer_key <> OAUTH..get_consumer_key (sid))
      {
        OAUTH..session_terminate (sid);
            sid := null;
      }
      }

    meth := get_keyword ('meth', params, 'weblog.post.new');;
    pars := get_keyword ('pars', params, 'inst_id=15&title=testing&description=Some test post');;

    if ({?'rt'} is not null and sid is null) -- request new token for the session
      {
    url := srv||'request_token';
    url := OAUTH..sign_request ('GET', url, '', consumer_key, sid);
        res := http_get (url);
    sid := OAUTH..parse_response (sid, consumer_key, res);
    OAUTH..set_session_data (sid, params);
    ret_url := sprintf ('http://example.com:%s/oauth/oauth_client.vsp?ready=%U', server_http_port (), sid
    -- call authorize
        url := sprintf ('%sauthorize?oauth_token=%U&oauth_callback=%U', srv, OAUTH..get_auth_token (sid),
    http_status_set (301);
        http_header (sprintf ('Location: %s\r\n', url));
        return;
      }
    else if ({?'ready'} is not null) -- get access token
      {
    -- we go here when token above is authorized
        sid := {?'ready'};
    url := srv||'access_token';
    consumer_key := OAUTH..get_consumer_key (sid);
    url := OAUTH..sign_request ('GET', url, '', consumer_key, sid);
        res := http_get (url);
    sid := OAUTH..parse_response (sid, consumer_key, res);

      }
    if (sid is not null) -- access token is ready, and we can call API
      {
        -- here we are ready to call service
    if ({?'rt'} is null)
          {
            tmp := OAUTH..get_session_data (sid);
            pars := get_keyword ('pars', tmp, pars);
        meth := get_keyword ('meth', tmp, meth);
      }
    url := sprintf ('http://example.com:%s/ods/api/%s', server_http_port (), meth);
    tmp := split_and_decode (pars);
    params := '';
    for (declare i,l int, l:=length (tmp); i < l; i := i + 2)
```

```
        {
          params := params || sprintf ('%U=%U&', tmp[i], tmp[i+1]);
        }
      --params := sprintf ('inst_id=%d&description=%U&title=%U', 15, 'Some test post', 'testing');
      consumer_key := OAUTH..get_consumer_key (sid);
      url := OAUTH..sign_request ('GET', url, params, consumer_key, sid);
          res := http_get (url);
          --dbg_obj_print (res);
      }
?>
      <head><title>OAuth Client</title></head>
      <body>
      <h1>OAuth client for ODS Controllers</h1>
      <form method="POST" action="oauth_client.vsp">
      <input type="hidden" name="sid" value="<?V sid ?>"/>
      APPLICATION : <br /> <select name="key">
      <?vsp for select a_name, a_key from OAUTH..APP_REG do { ?>
          <option value="<?V a_key ?>" <?vsp if (consumer_key = a_key) http ('selected'); ?>><?V a_name ?><
      <?vsp } ?>
          </select>
      <?vsp
      if (sid is not null) http (sprintf (' TOKEN: %s', OAUTH..get_auth_token (sid)));
      ?>
      <br />
      ODS API: <br /><input type="text" name="meth" value="<?V meth ?>" size=50/> <br />
      PARAMETERS: <br /> <textarea name="pars" rows="5" cols="50"><?V pars ?></textarea> <br />
          <input type="submit" value="Execute" name="rt"/><br />
      </form>
      RESULT:
      <hr/>
      <pre><?V res ?></pre>
      </body>
</html>
```

Sample OAuth Session

The following shows a sample session, using the above Virtuoso/PL OAuth client.

```
GET /OAuth/request_token?oauth_consumer_key=50082d0fb861b0e6e67d5d986b8
333607edc5f36&oauth_nonce=b8f1089077cbce6e&oauth_signature_method=HMAC-
SHA1&oauth_timestamp=1211212829&oauth_version=1.0&oauth_signature=V1zmk
757LBHcmqVJ6obMhNX5hKA%3D HTTP/1.1
Host: example.com

HTTP/1.1 200 Ok
Content-Length: NNN
<CR/LF>
oauth_token=86da75079d3aee0fab57a36fcffbf900768e4de3&oauth_token_secret
=M...

GET /OAuth/authorize?oauth_token=86da75079d3aee0fab57a36fcffbf900768e4d
e3&oauth_callback=http%3A//localhost%3A6666/oauth/oauth_client.vsp%3Fre
ady%3D00c874b2fab2f6424008b5064fe83e88 HTTP/1.1
Host: example.com

HTTP/1.1 301 Moved
Location: /ods/oauth_authorize.vspx?....

GET /OAuth/access_token?oauth_consumer_key=50082d0fb861b0e6e67d5d986b83
33607edc5f36&oauth_nonce=242cc4875a0059f6&oauth_signature_method=HMAC-S
HA1&oauth_timestamp=1211212831&oauth_token=86da75079d3aee0fab57a36fcffb
f900768e4de3&oauth_version=1.0&oauth_signature=sqs/8nmNNnNJiZ/eBa688uNe
g9o%3D HTTP/1.1
Host: example.com

HTTP/1.1 200 Ok
Content-Length: NNN
<CR/LF>
oauth_token=N..&oauth_token_secret=M...

GET /ods/api/weblog.post.new?description=Some%20test%20post&inst_id=15&
oauth_consumer_key=50082d0fb861b0e6e67d5d986b8333607edc5f36&oauth_nonce
```

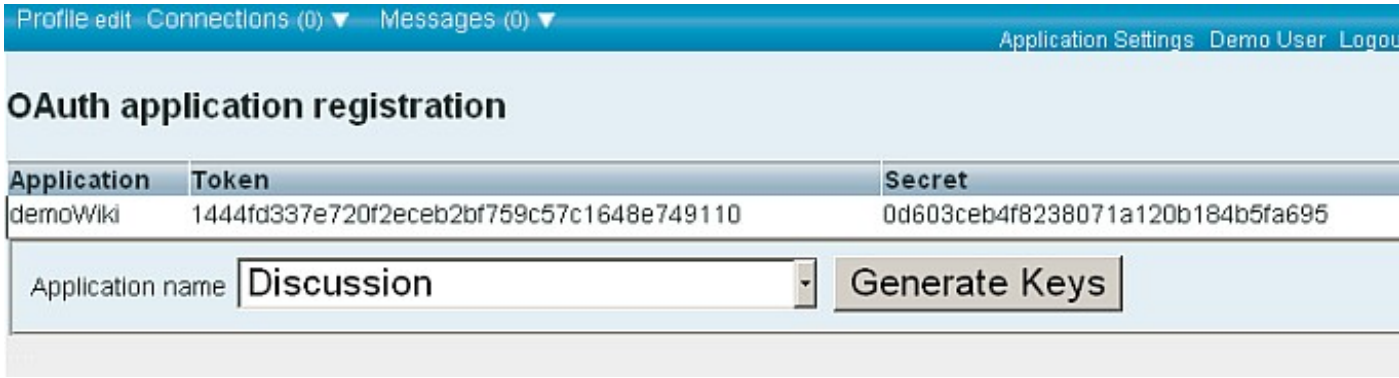
```
=2f4765d20664e696&oauth_signature_method=HMAC-SHA1&oauth_timestamp=1211
213321&oauth_token=db83a37d74faf53edc8ed56d418ded3a70fcc0bc&oauth_versi
on=1.0&title=testing&oauth_signature=oocIyr6kgoEQkC3WVwzvl6w62W4%3D HTT
P/1.1
Host: example.com

HTTP/1.1 200 Ok
Content-Length: NNN
<CR/LF>
<result><code>1&lt/code></result>
```

17.4.4. OAuth Generate Keys for ODS Controllers (Web Services)

The following steps describe how to Setup Application OAuth keys in ODS:

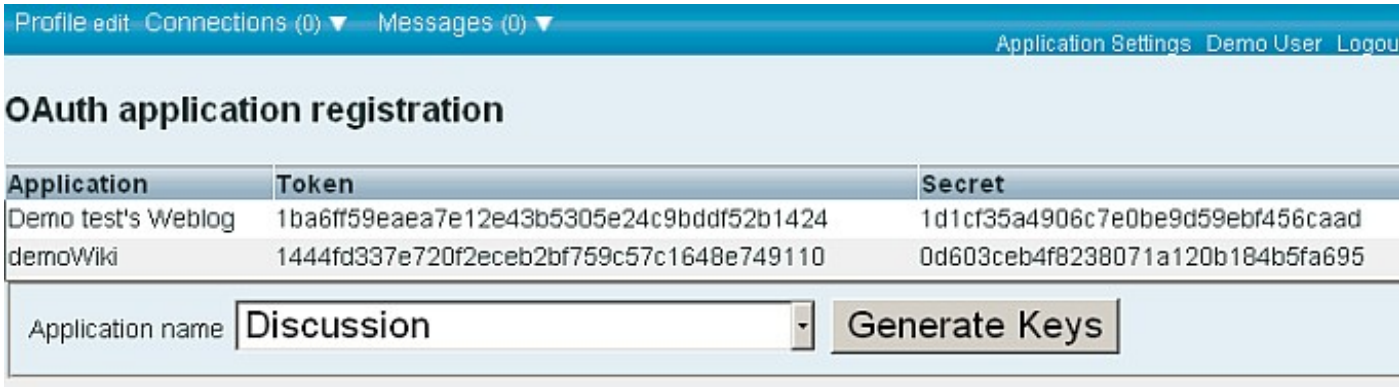
1. Login as user at ODS Data Space.
2. Go to Settings -> OAuth keys

Figure 17.28. OAuth Generate Keys



3. Select Application name from the drop-down list and click the "Generate Keys" button.
4. The generated Token and Secret values for the selected application will be shown as a result.

Figure 17.29. OAuth Generate Keys



17.4.5. ODS Ubiquity Commands

ODS provides Ubiquity commands to manipulate user accounts as well as instances and instance specific data using the ODS Controllers API. This command set is available only for the Firefox browser with the Ubiquity extension installed.

Installation

First, the Ubiquity Firefox extension must be installed. Then, ODS Ubiquity commands can be installed by accessing http://host:port/ods/ods_ubiquity.html page. Ubiquity commands are accessed by the ctrl-space key combination.

Basic Session Setup Commands

ODS OAuth Endpoint Identification

Assuming the OAuth endpoint: http://myopenlink.net/OAuth, in order to change it use the command:

```
Syntax: ods-oauth-host <host_url>
Example: ods-oauth-host http://demo.openlinksw.com/OAuth
```

ODS Command Mode

There are two command modes available to ubiquity commands when working against an ODS instance: oauth or sid.

In OAuth mode, for every ODS application instance, an OAuth key must be obtained and then initialized via the following steps:

- obtain an ODS based OAuth session-key via the ODS example endpoint: http://host:port/ods/oauth_sid.vsp

  ```
  http://myopenlink.net/ods/oauth_sid.vsp
  ```
- bind session key to command session via the command: *ods-set-oauth <oauth-key>* or use *ods-set-[class name]-oauth <oauth-key>* when oauth-key was generated for particular application instance.

If you choose the SID mode of interaction with ODS you have to perform the following steps:

- set mode via command: ods-set-mode sid or ods-set-mode oauth
- set sid via command: ods-set-sid sid

Note: The sid is a session ID applicable to all ODS commands for a given bound instance and user combination. It's obtained from an ODS session.

Example

The following example demonstrates oauth authentication getting the OAuth SID value for ex. for existing Bookmark instance "mybookmarks" with owner user demo at the OpenLink Demo server.

In order to execute correctly the example, you need to have OAuth Generate Key for the Bookmark "mybookmarks" instance at OpenLink Demo server. More information and simple scenario how to be done you can find here.

1. Execute the command:

   ```
   ods-oauth-host <host_url>
   -- for ex.:
   ods-oauth-host http://demo.openlinksw.com/OAuth
   ```

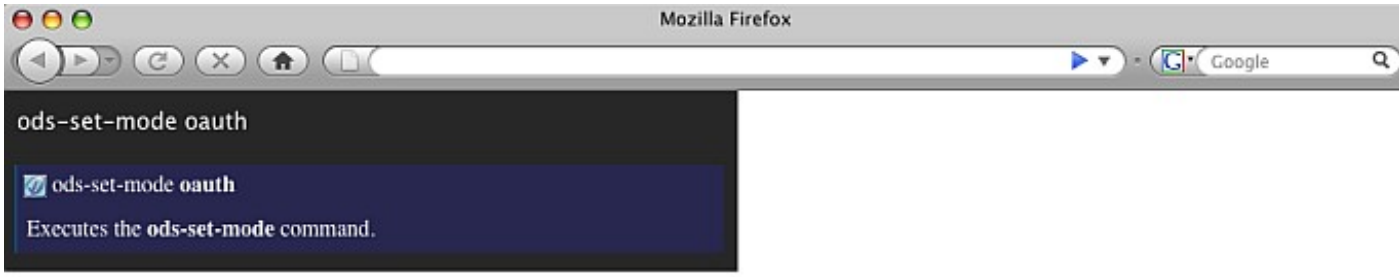
   Figure 17.30. OAuth Authentication

2. Execute the command:

   ```
   ods-set-mode oauth
   ```

   Figure 17.31. OAuth Authentication



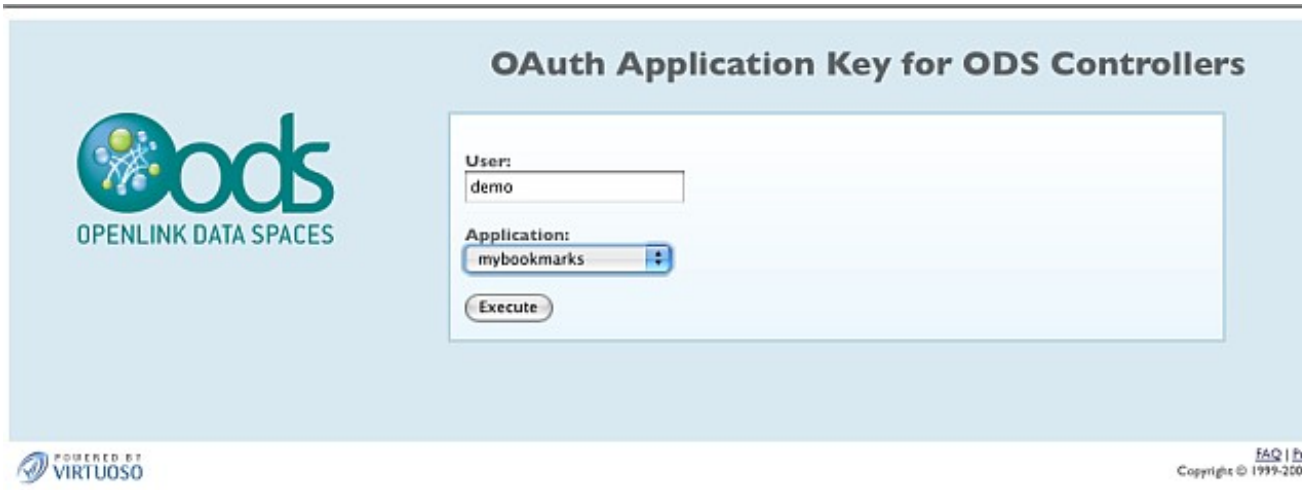
3. Obtain the OAuth Generated Key:

   a. Go to http://cname:port/ods/oauth_sid.vsp . For ex. http://demo.openlinksw.com/ods/oauth_sid.vsp

   Figure 17.32. OAuth Authentication

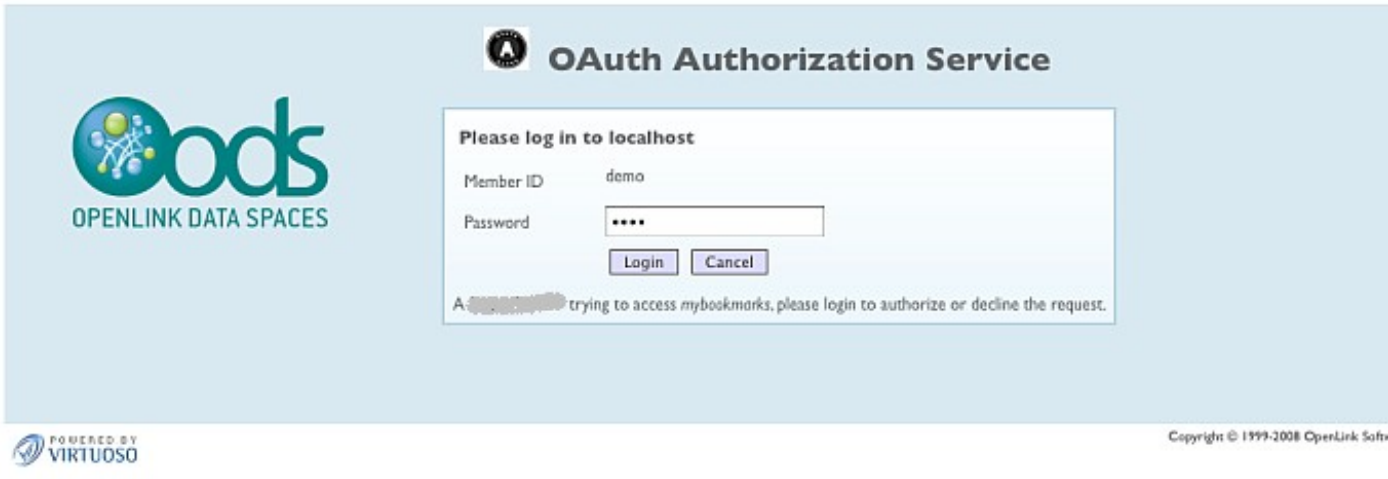
   b. In the shown form enter user demo and choose the application "mybookmarks". Click the "Execute" button.

   Figure 17.33. OAuth Authentication



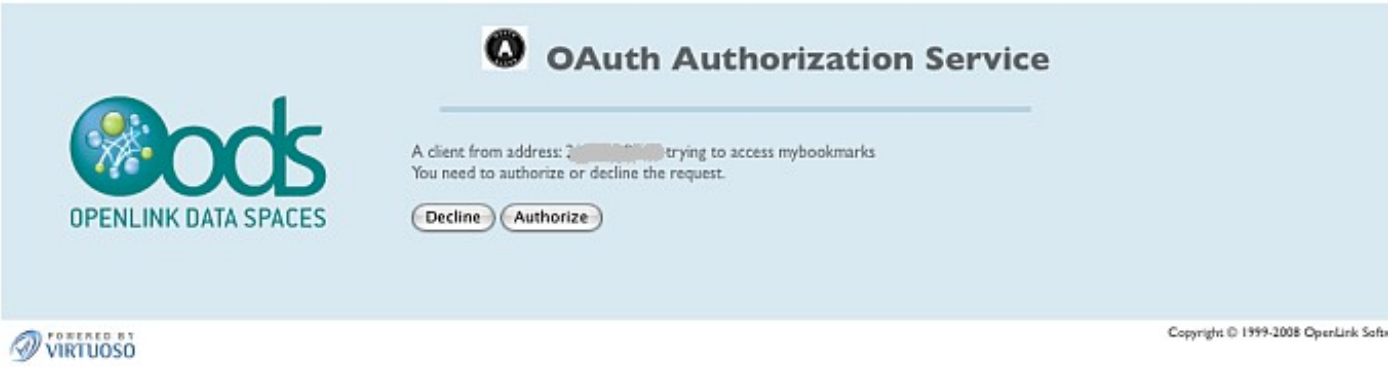
4. As result will be shown a form where you need to enter the user password.
5. Enter for user demo its password and click the "Login" button.

   Figure 17.34. OAuth Authentication



6. Click the "Authorize" button.

Figure 17.35. OAuth Authentication



7. Copy the obtains OAuth SID value for ex.: 7ef4dcf23869488823b771b09b952cc9

Figure 17.36. OAuth Authentication

8. Execute the following command:

```
ods-set-sid 7ef4dcf23869488823b771b09b952cc9
```

```
or

ods-set-addressbook-oauth 7ef4dcf23869488823b771b09b952cc9
```

Figure 17.37. OAuth Authentication

9. You can also execute the command(s) depending on the application type:

```
ods-set-briefcase-oauth <the obtained oauth session-key>
-- or
ods-set-bookmark-oauth <the obtained oauth session-key>
-- or
ods-set-feeds-oauth <the obtained oauth session-key>
-- or
ods-set-calendar-oauth <the obtained oauth session-key>
-- or
ods-set-addressbook-oauth <the obtained oauth session-key>
-- or
ods-set-poll-oauth <the obtained oauth session-key>
-- or
ods-set-weblog-oauth <the obtained oauth session-key>
-- or
ods-set-discussion-oauth <the obtained oauth session-key>
```

See Also:
the Virtuoso SPARQL OAuth Tutorial and the full list of ODS Application Authentication Examples using OAuth

See Also:
the full list of ODS Ubiquity Commands and the full list of OpenLink Data Spaces (ODS) Ubiquity Commands Tutorials

17.4.6. OAuth Test Tool for ODS Controllers

The ODS OAuth Test Tool creates examples to show users the correct format for constructing HTTP requests signed according to OAuth specifications. The users use this format in their applications to make successful requests to the ODS REST APIs.

The ODS users generate a Consumer Key and a Consumer Secret for their application instances by ODS application UI (Setings -> OAuth Keys). You can find more information and sample scenario here.

To reach a specific ODS resource via the ODS REST API, a user must also specify a API method and associated API parameters.

ODS OAuth standards validate the credentials of an external user by means of the digital signature. If the user signs the request, and the ODS server validates the digital signature, the developer is granted access to the requested resource.

To tool is accessible via http://host:port/ods/oauth_test.vsp

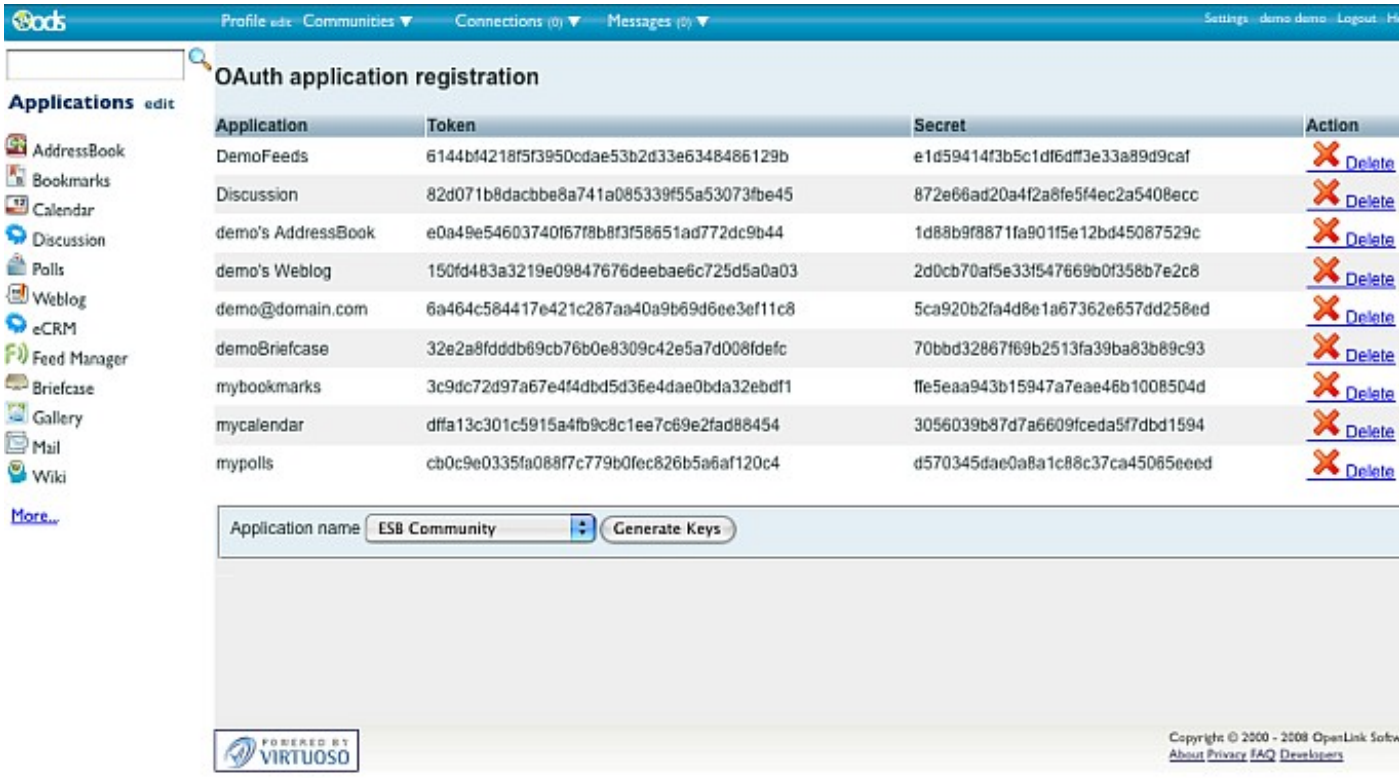
Example

The following example demonstrates creating and getting contact info by the ODS REST APIs weblog.post.new and weblog.post.get using the OAuth Test Tool.

1. Go to http://demo.openlinksw.com/ods
2. Log in as user demo with password demo

3. Go to Settings->OAuth Keys

Figure 17.38. Weblog OAuth



4. Copy the Token value for an Weblog instance or create such in not existing. For ex.:

```
150fd483a3219e09847676deebae6c725d5a0a03
```

5. Go to http://demo.openlinksw.com/ods/oauth_test.vsp
6. In the shown form enter for:

   a. Server: http://demo.openlinksw.com/ods/api
   b. Consumer Key: 150fd483a3219e09847676deebae6c725d5a0a03
   c. API method: weblog.post.new
   d. API parameters:

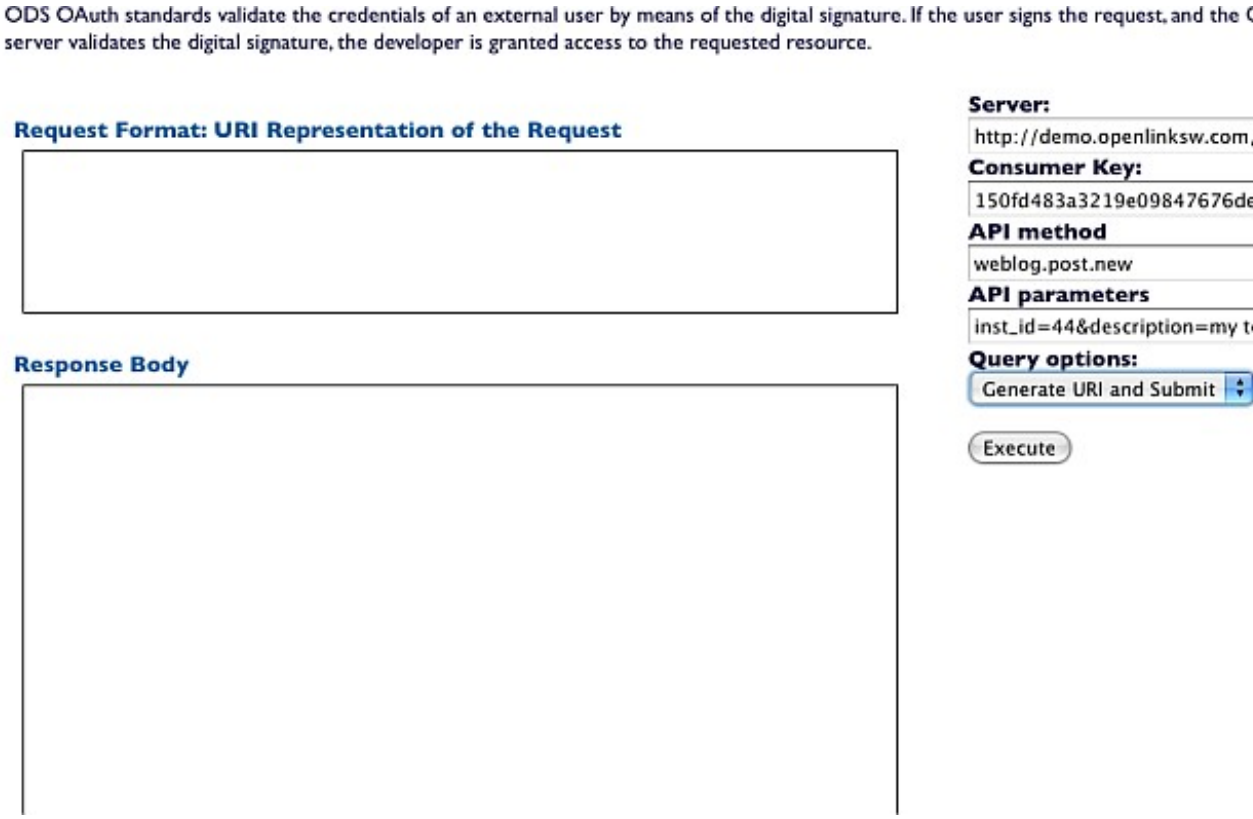
```
inst_id=44&description=my test&title=my first post
```

   - An easy way to get the instance id is using the ODS ubiquity commands sequence:

```
ods-host http://demo.openlinksw.com/ods
ods-set-mode sid
ods-authenticate-user demo password demo
ods-get-instance-id demo's Weblog
```

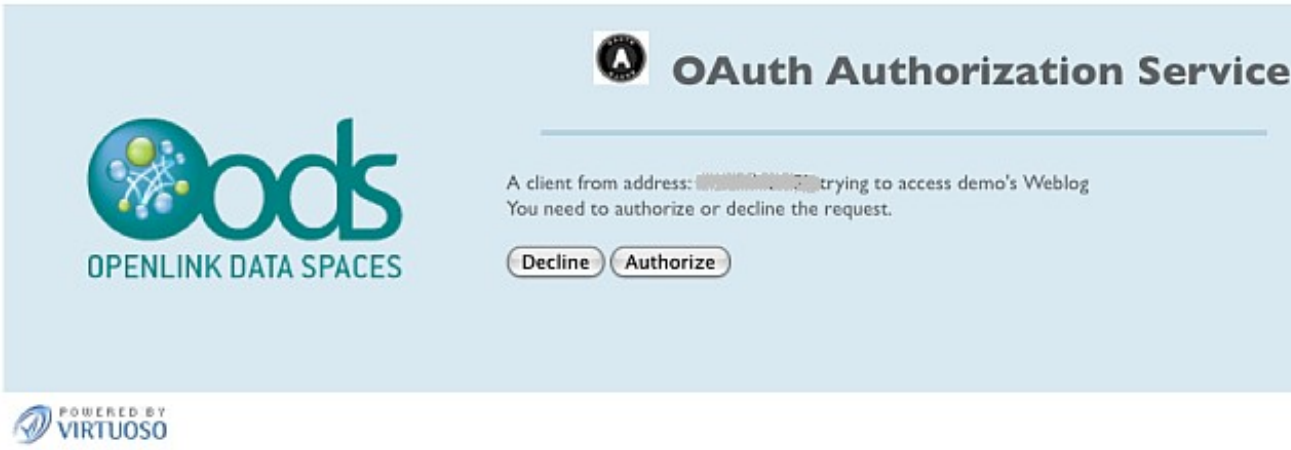
   e. Select for Query options: "Generate URI and Submit".

Figure 17.39. Weblog OAuth



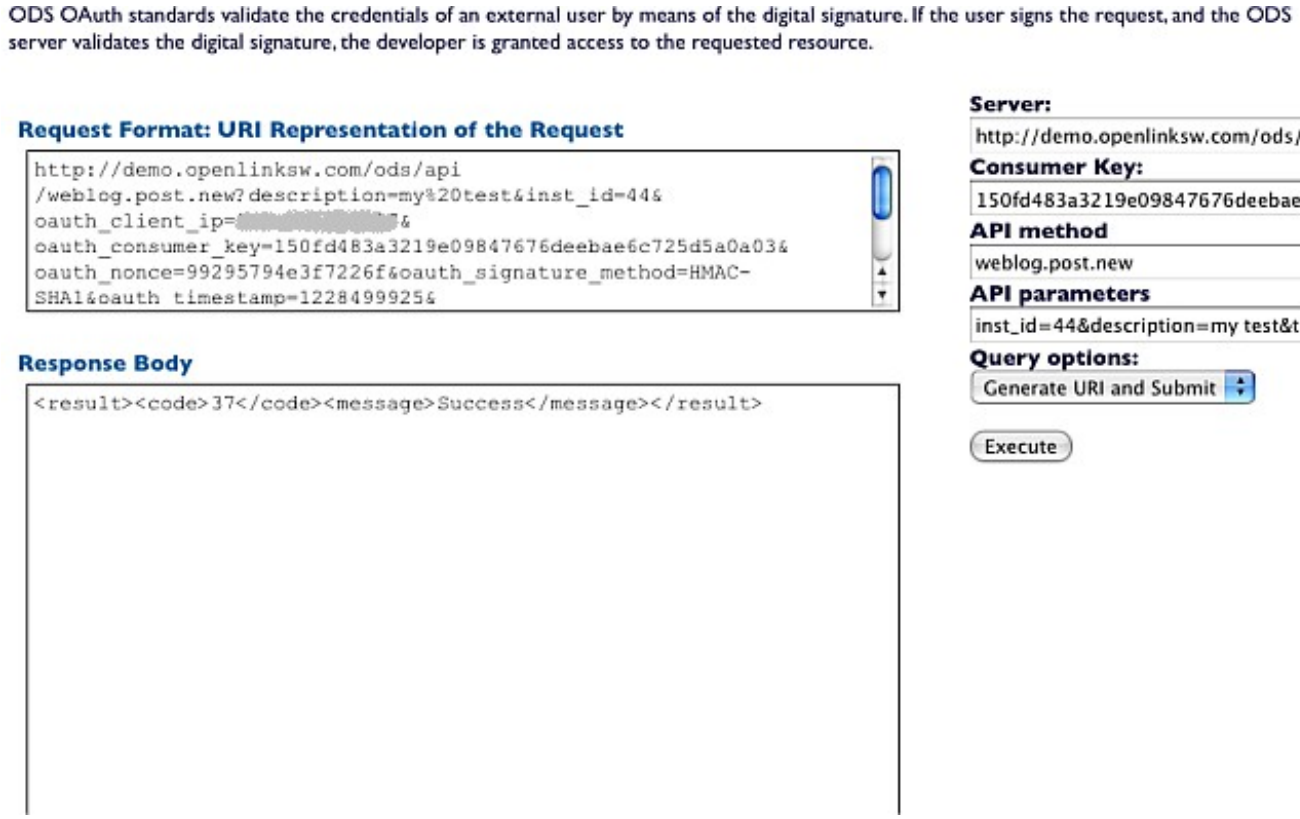
f. Click the "Execute" button.
g. In the shown authorization form click the "Authorize" button.

Figure 17.40. Weblog OAuth



h. As result will be shown the generated URI and the output message of executing the ODS REST API. In our case, the id of the new created post will be: 37

Figure 17.41. Weblog OAuth



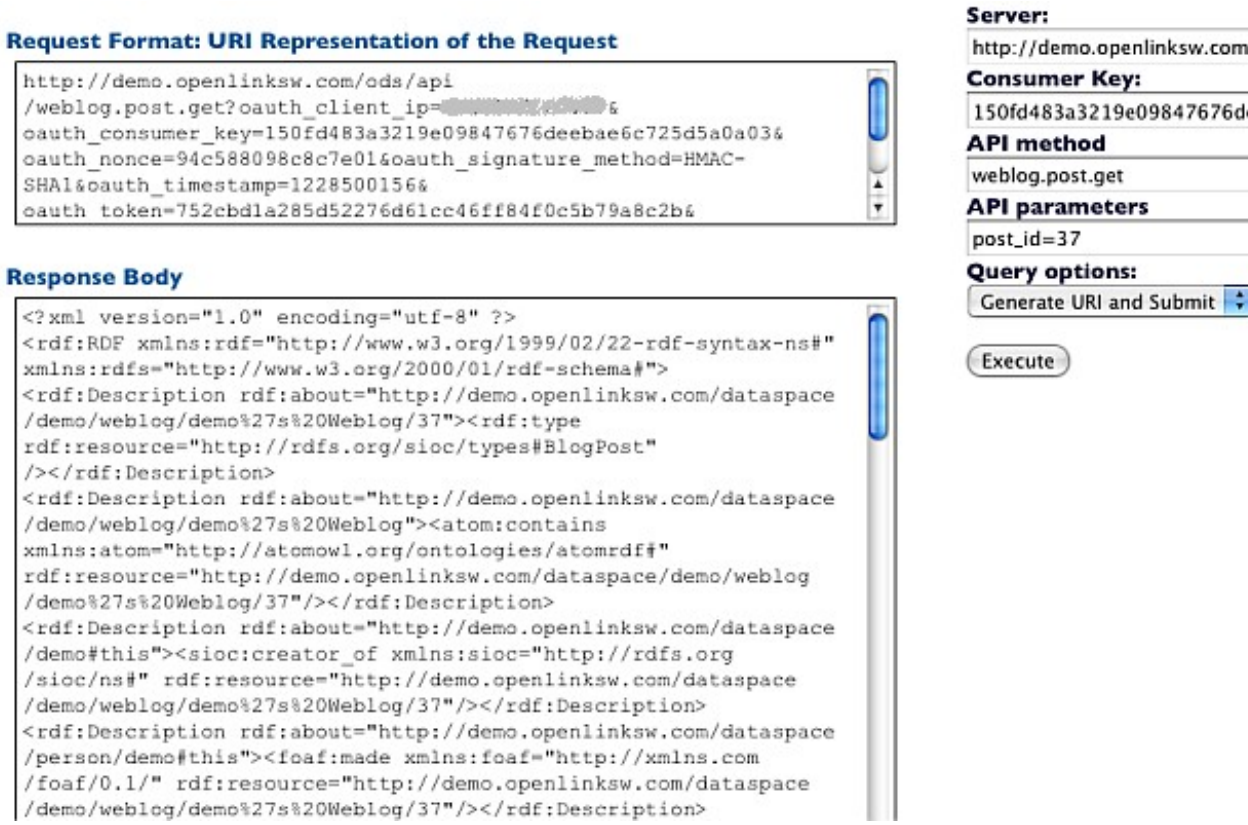
i. Now let's get the data for the created post. In the form enter:

   i. API method: weblog.get
   ii. API parameters: post_id=37
   iii. Select for Query options: "Generate URI and Submit".

j. Click the "Execute" button.

k. As result in the "Response Body" text-area will be shown the retrieved details for the posts with the given above id.

Figure 17.42. Weblog OAuth



17.4.7. OAuth QA

MySpace Tools

In order to use the MySpace OAuth Testing Tool :

1. Need to have registered myspace account.
2. Need to apply to build apps on the MySpace Developer Platform.

Google Tools

Google's OAuth playground tool can be tried here .

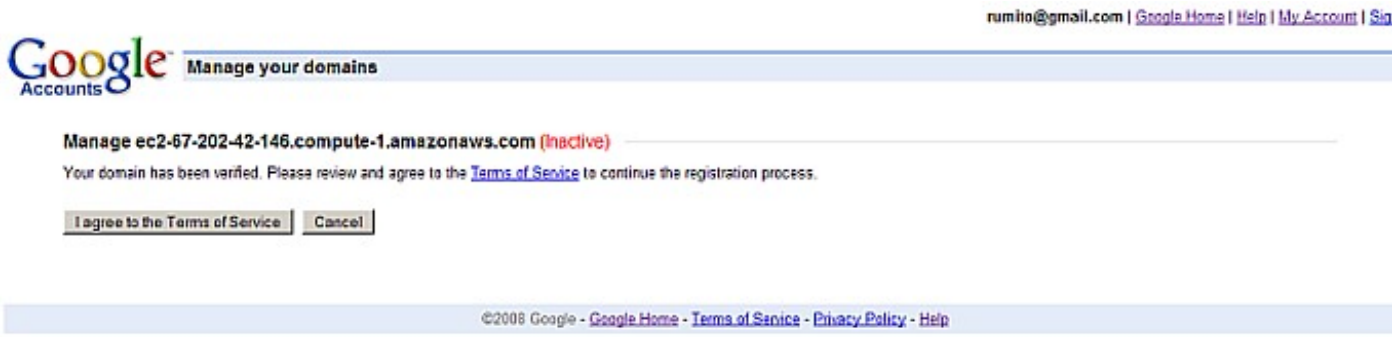
Register your domain in Google

In order to use the tool, you need to register the web application as domain:

1. Login at your gmail account
2. Go to https://www.google.com/accounts/ManageDomains
3. Enter in the field for ex. the QA server ec2-67-202-42-146.compute-1.amazonaws.com and click the "Add Domain" button.

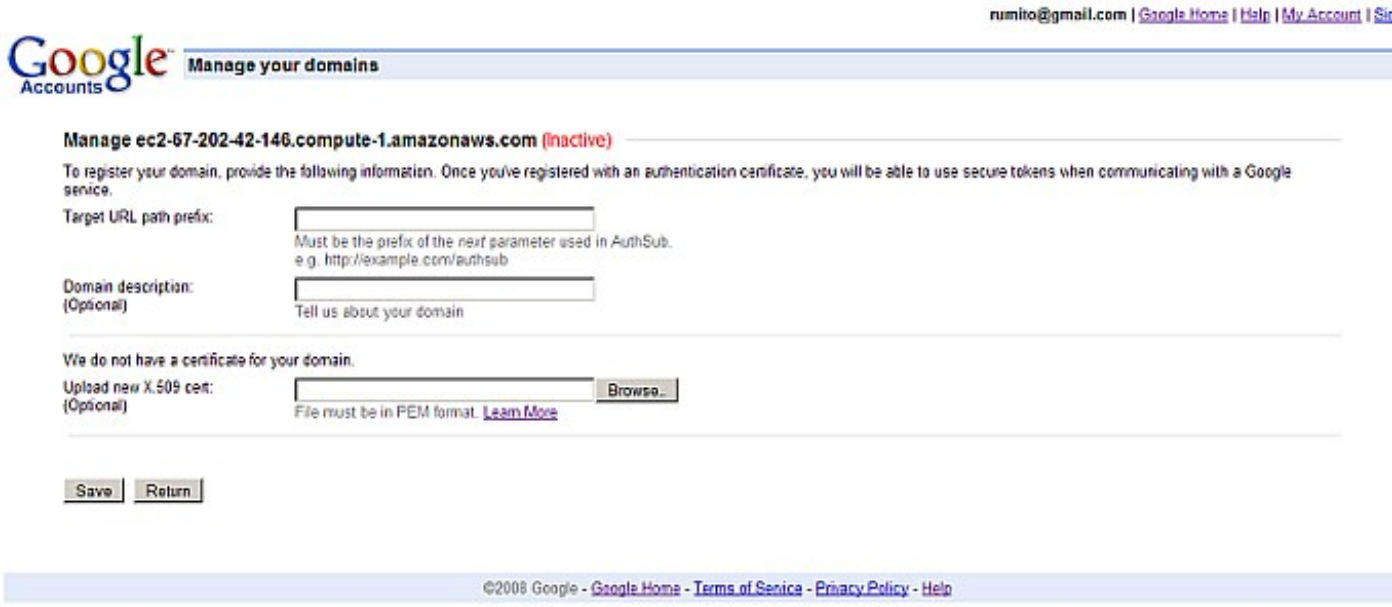
   Note: for now registered domains cannot be deleted (not supported from the Google UI)
4. As result the domain will be created and the manage domain page will be opened. Here you need to verify you have admin privileges by choosing verification method
5. Choose "Upload an HTML file" and follow further the instructions.
6. Once the file is put in the correct server root, you should get the confirmation page:

Figure 17.43. Google OAuth



7. Click the "Agree .." buttons.
8. As result will be additional setting page you need to change. Type in for "Target URL path prefix:": http://ec2-67-202-42-146.compute-1.amazonaws.com/ods

Figure 17.44. Google OAuth

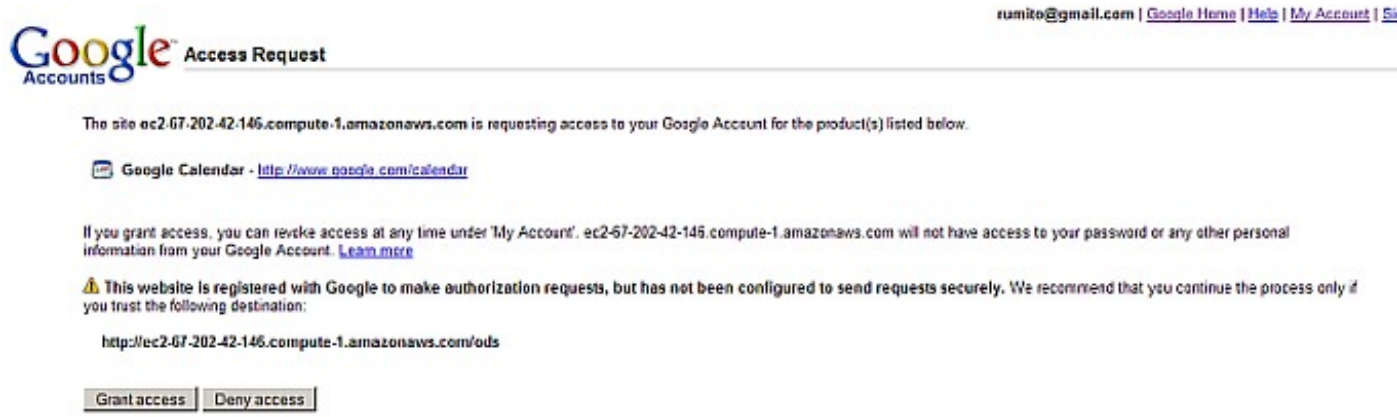


9. Click the "Save" button.
10. As result will be shown the generated OAuth Consumer Secret for the OAuth Consumer Key: ec2-67-202-42-146.compute-1.amazonaws.com:

```
OAuth Consumer Key:      ec2-67-202-42-146.compute-1.amazonaws.com
OAuth Consumer Secret:  uEkfBvpMhTTT/VyFItEnEYt4
```

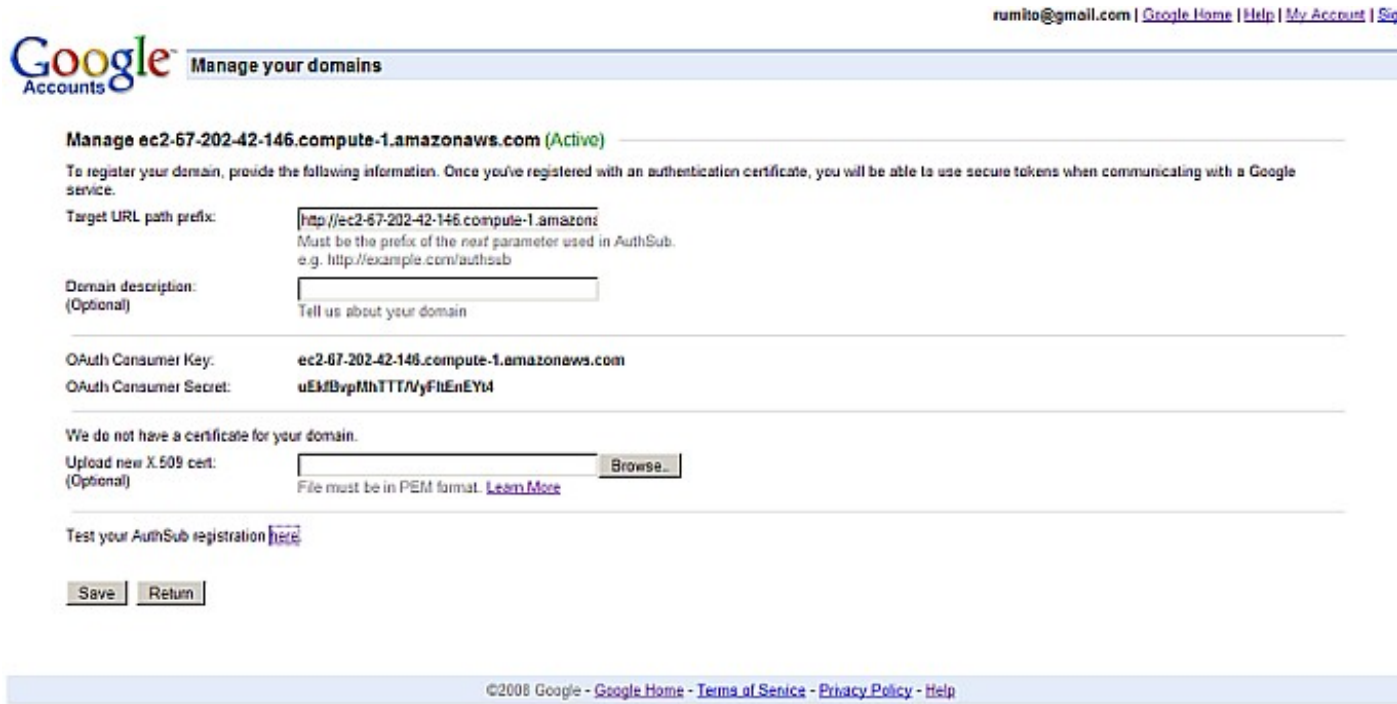
11. Also will be shown "test" link "Test your AuthSub registration here". Click it.
12. As result will be opened new window with OAuth request
13. Click the "Grant Access" button.

Figure 17.45. Google OAuth



14. As result will be redirected to http://ec2-67-202-42-146.compute-1.amazonaws.com/ods/
15. Now lets return to our initial window and click there the button "Save".
16. Note that already is shown for the domain that is "Active":

Figure 17.46. Google OAuth

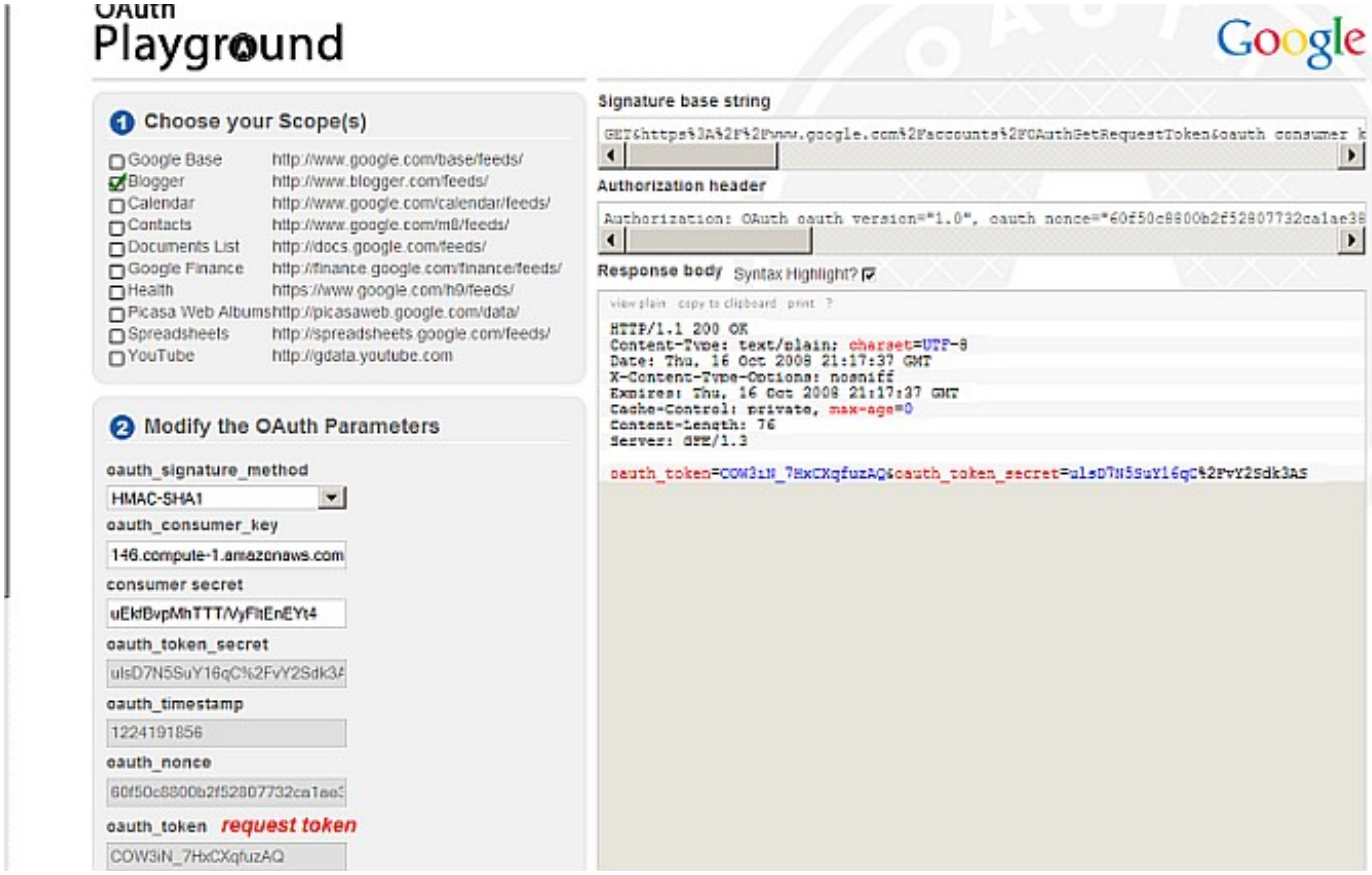


The Playground Tool qa steps

Let's try the playground tool:

1. Go to http://googlecodesamples.com/oauth_playground/
2. Select Scope, for ex. check the check-box for "Blogger".
3. Change oauth signature_method to HMAC-SHA1
4. Enter for "oauth consumer_key": ec2-67-202-42-146.compute-1.amazonaws.com
5. Enter for "consumer secret" the value generated from above.
6. Click the "Request token" button.
7. As result will get Response with Status Ok

Figure 17.47. Google OAuth



- the Signature base string:

```
GET&https%3A%2F%2Fwww.google.com%2Faccounts%2FOAuthGetRequestToken&oauth_consumer_key
%3Dec2-67-202-42-146.compute-.amazonaws.com%26oauth_nonce%3D60f50c8800b2f52807732ca1ae3855e
%26oauth_signature_method%3DHMACSHA1%26oauth_timestamp%3D1224191856%26oauth_version%3D1.0
%26scope%3Dhttp%253A%252F%252Fwww.blogger.com%252Ffeeds%252F
```

- the Authorization header:

```
Authorization: OAuth oauth_version="1.0",
oauth_nonce="60f50c8800b2f52807732ca1ae3855ef",
oauth_timestamp="1224191856",
oauth_consumer_key="ec2-67-202-42-146.compute-1.amazonaws.com",
oauth_signature_method="HMAC-SHA1",
oauth_signature="nohPMCw%2BMrO8%2FwslS4oEm2wfuhg%3D"
```

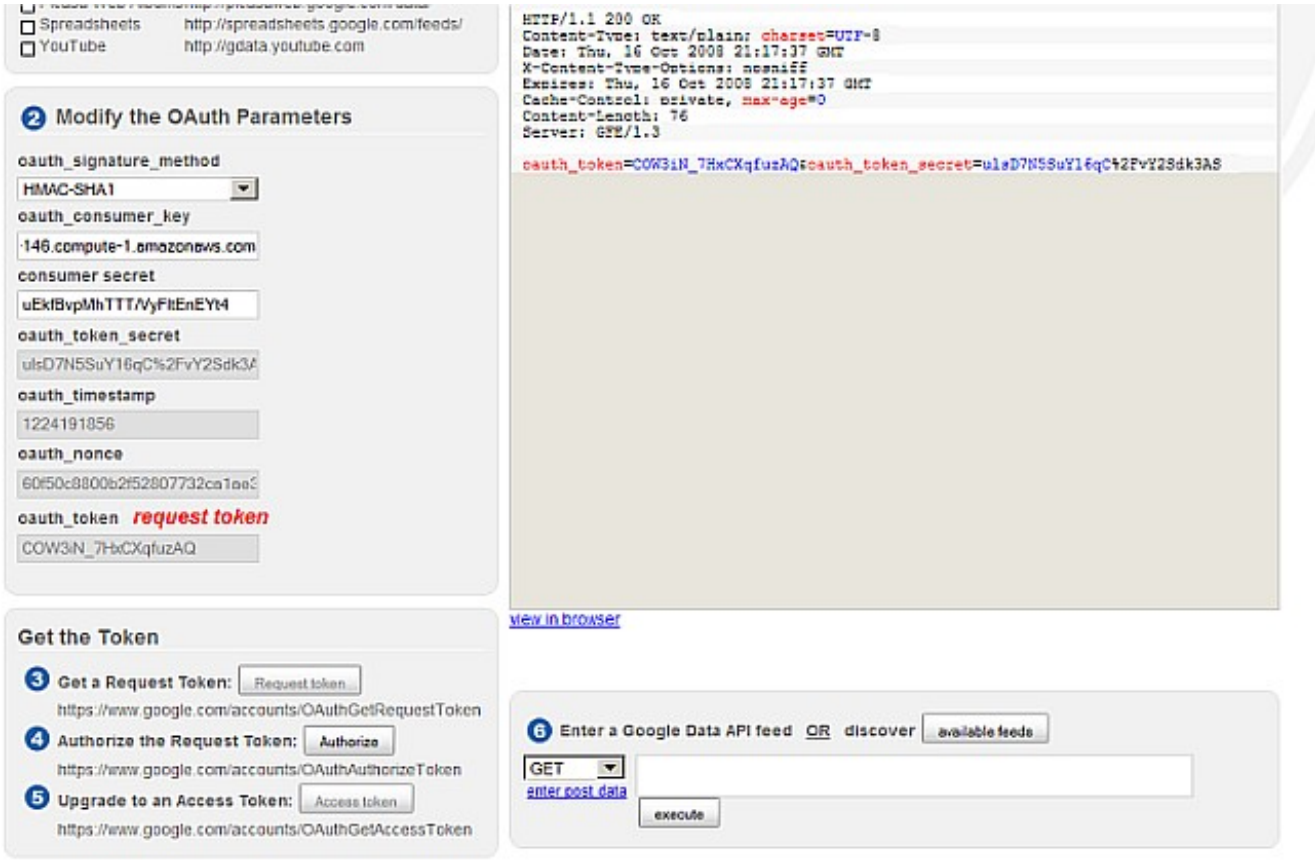
- the Response body:

```
HTTP/1.1 200 OK
Content-Type: text/plain; charset=UTF-8
Date: Thu, 16 Oct 2008 21:17:37 GMT
X-Content-Type-Options: nosniff
Expires: Thu, 16 Oct 2008 21:17:37 GMT
Cache-Control: private, max-age=0
```

```
Content-Length: 76
Server: GFE/1.3
 oauth_token=COW3iN_7HxCXqfuzAQ&oauth_token_secret=ulsD7N5SuY16qC%2FvY2Sdk3AS
```

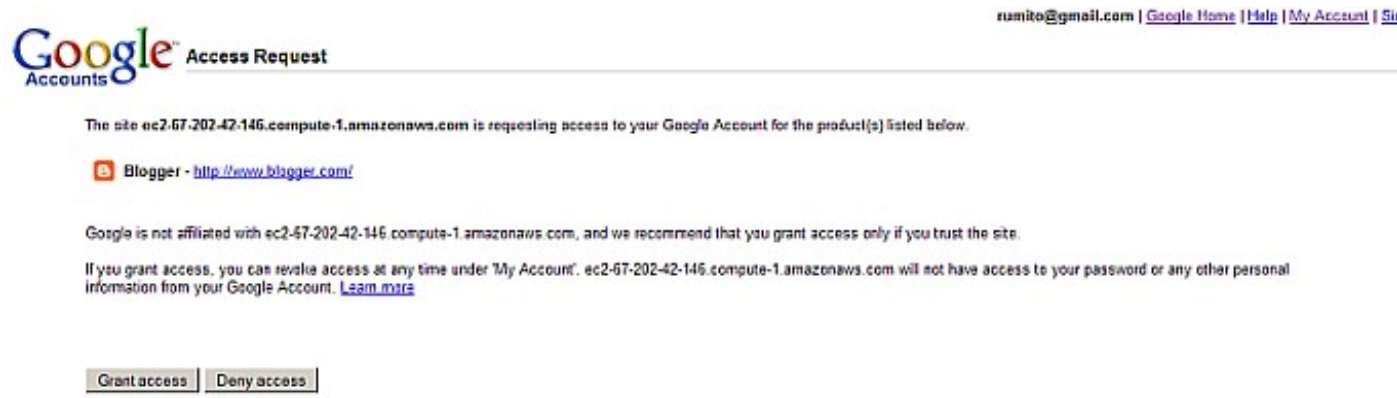
8. And also in the "Get the Token" section now will be active the "Authorize" button. Click it.

Figure 17.48. Google OAuth



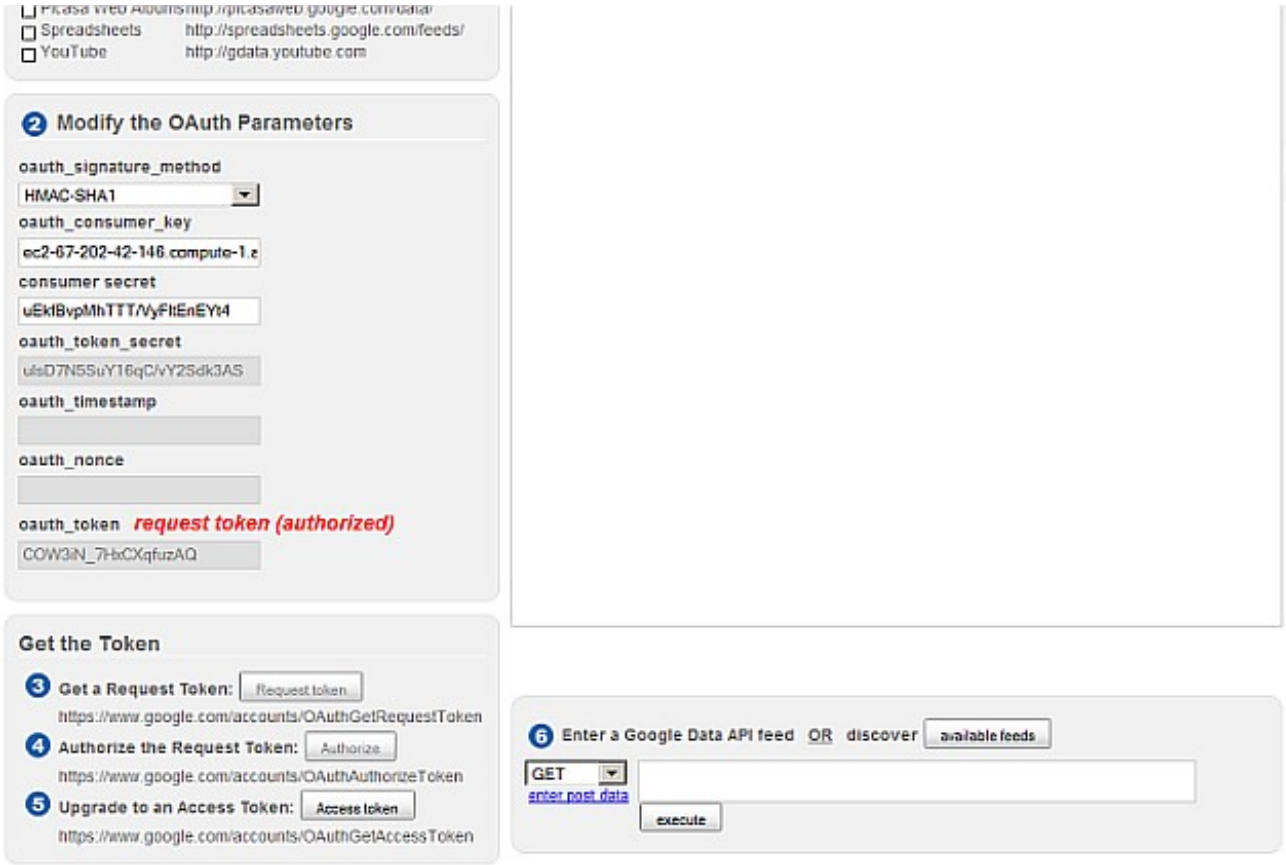
9. As result will be shown the authentication page where you need to click the "Grant Access" button:

Figure 17.49. Google OAuth



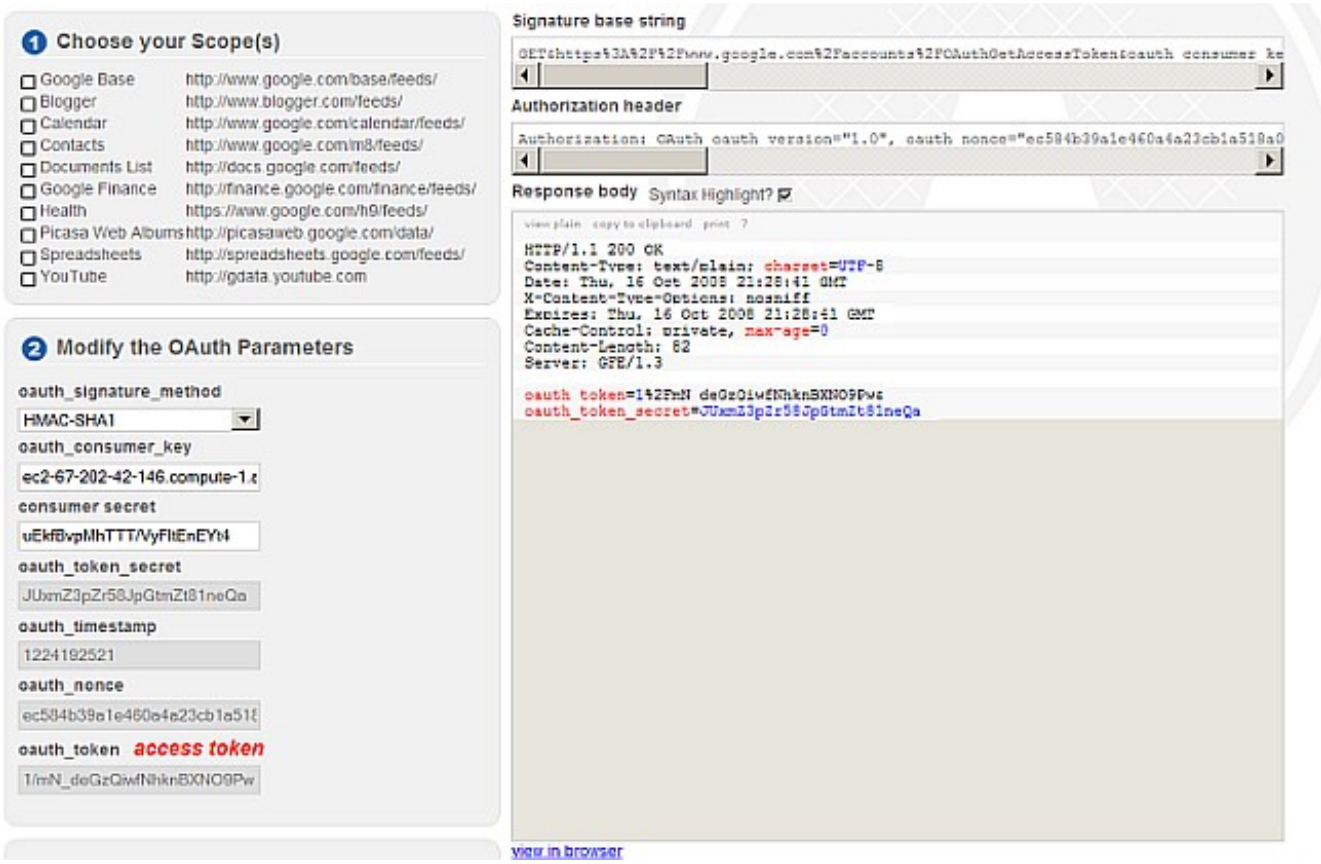
10. As result now in the "Get the Token" section will be active the "Access token" button. Click it.

Figure 17.50. Google OAuth



11. As result the access_token will be obtained:

Figure 17.51. Google OAuth

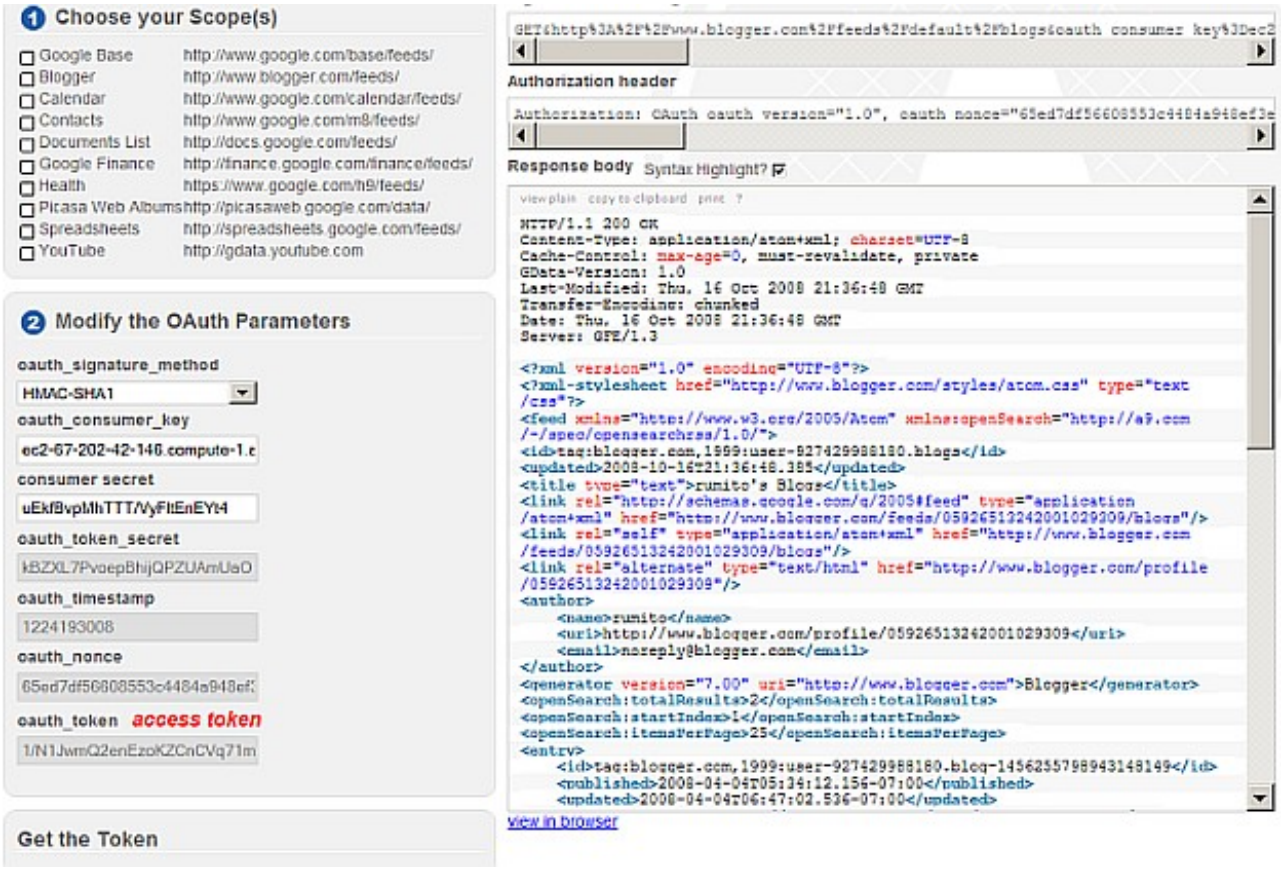


12. Click the "available feeds" button marked as 6.
13. As result in the "Response" body section will be shown:

```
Blogger
http://www.blogger.com/feeds/default/blogs
http://www.blogger.com/feeds/<blogID>/posts/default
http://www.blogger.com/feeds/<blogID>/[<postID>]/comments/default
```

14. Copy the 1st URL and paste in the field after the method GET.
15. Click "execute".
16. As result will find your blogs, post, info at Blogger:

Figure 17.52. Google OAuth



17.5. WS-Security (WSS) Support in Virtuoso SOAP Server

The following terms are used in this section in the following meanings:

- **encryption .** The process of making data unreadable using some secret (see 'key')
- **decryption .** The opposite of encryption
- **signature .** A sequence of binary codes that is calculated based on the original data and a key to secure the content from undetected change.
- **key .** A secret, depending on the type can be symmetric or asymmetric.

  Typical symmetric keys are passwords. Symmetric keys are by their nature more at risk of being compromised if the secret key is stolen. Symmetric keys are usually used to encrypt large amounts of data as they are very fast.

  Asymmetric keys are more secure and their structure is more complicated. Asymmetric keys generally consist of a private and public key pair. The owner of the key can sign data to be verified by recipient with public key. A correspondent can encrypt the data using public key to be decrypted by private key owner. Asymmetric keys are used to sign data, or encrypt small amounts of data as they are slower to process than symmetric ones.
- **certificate .** This is a structure of secure data, which identifies a certificate owner. This is similar to having a user-name and password but more secure. A 'certificate' can be associated with a document as an alias of the X.509 certificate. Certificates are issued by a third-party, not the owner or recipient in the verification path, namely the Certificate Authority (CA). The CA's function is to guarantee that a receiver can trust data accompanied by a particular certificate. Certificates will contain a public key, but never the private key. These must be distinguished from certificates that are exported together with their private key in PKCS#12 format, these are different things.
- **user account space .** This is a data area where a user can store private data. Only ODBC sessions and web page processing code which runs on behalf of a certain SQL user account has access to this area. This is useful for caching private security related information such as keys or certificates.
- **key reference .** A short hand for a key, sometime called a 'key name'. This is a string to uniquely identify a key in a user account space.
- **key instance .** An entity representing a key in PL, this is a not the key per se. The key instance is used in encryption or decryption routines.

The following algorithms are supported:

RSA (http://www.w3.org/2001/04/xmlenc#rsa-1_5)
DSA (http://www.w3.org/2001/04/xmlenc#dsa)
triple DES (http://www.w3.org/2001/04/xmlenc#tripledes-cbc)
AES128 (http://www.w3.org/2001/04/xmlenc#aes128-cbc)
AES192 (http://www.w3.org/2001/04/xmlenc#aes192-cbc)
AES256 (http://www.w3.org/2001/04/xmlenc#aes256-cbc)
Digest algorithms:

SHA1 (http://www.w3.org/2000/09/xmldsig#sha1)
Signing algorithms:

RSA-SHA1 (http://www.w3.org/2000/09/xmldsig#rsa-sha1)
DSA-SHA1 (http://www.w3.org/2000/09/xmldsig#dsa-sha1)
Canonicalization algorithms with comments:

(http://www.w3.org/TR/2001/REC-xml-c14n-20010315)
(http://www.w3.org/TR/2001/REC-xml-c14n)
The keys can be temporary or persistent keys. Temporary keys are loaded only in memory an will be lost when the server is restarted. These are usually used for making symmetric session keys. Persistent keys are kept on the file-system or in the database and will be loaded upon server startup. These are asymmetric keys, certificates that belong to the user.

The location where persistent keys are stored depends on the key reference. If the key reference is a 'file:' URI, then the API's assume file-system storage, otherwise they will be kept in the U_OPT column of the SYS_USERS table as a serialized string. The API functions are described below. Whether keys are stored on the file-system or within the database they will have an in-memory representation which is used in crypto functions. The memory cache of keys is contained in the user account space, hence no user can access or reference another user's keys. Furthermore if a user account is removed all associated keys will also be removed if they were stored in the database. If the keys were on the file-system only the in-memory cache will be deleted.

17.5.1. Client and Server side Certificates & Keys

Since XML encoding routines are executed on server-side, we cannot strictly say that we have client and server certificates (as browser and HTTPS server). To perform the encoding on behalf of a client or user account, the XML encoding functions need to have the keys and certificates loaded in the memory cache of the user. The same applies to server keys and certificates. Therefore we will refer to these as client or server security tokens that are kept on server-side.


Note:
Uploading of keys and certificate must be done under a secure SSL/TLS connection to minimize the risk of vulnerability.

Key Definition & Persistence

```
xenc_key_3DES_rand_create()

USER_KEY_LOAD()
```

Key Encryption

To minimize the risk of non-authorized private key usage, keys are usually kept encrypted with password. If a key or certificate containing a key is loaded using a password, the API will assume it is an encrypted private key and will be kept in that form. In other words encrypted keys are kept in their encrypted form in memory, there will be no raw certificate or key data if the encrypted form is used to import. Virtuoso will ask for passwords to unlock persisted encrypted keys upon server restart. This is only possible when the server is running in foreground mode. If the server is started as background process and the key needs a password to decrypt and load, an error will be logged in the virtuoso log file and that particular key will not be loaded.

Example 17.25. An example of a password prompt and log on success

```
Enter a password for key "wss.pfx":
13:14:04 PL LOG: XENC:   Loaded : wss.pfx
```

Referencing Keys

To make a run-time key instance, used for XML encryption functions, and to perform server or client side the `xenc_key_inst_create (in key_name varchar[, super key inst])` function can be used.

Example 17.26. Example

```
create procedure
DB.DBA.WSDK_GET_KEY ()
{
  -- this returns a instance of shared secret suitable for content encryption
  return xenc_key_inst_create ('WSDK Sample Symmetric Key');
}
;

create procedure
DB.DBA.WS_SOAP_GET_KEY ()
{
  declare superkey, keyinst any;
  superkey := xenc_key_inst_create ('file:dsa.der'); -- already loaded asymmetric key (see above example)
  -- this returns a session key , to be encrypted with a asymmetric one
  keyinst := xenc_key_inst_create (xenc_key_3DES_rand_create (NULL), superkey);
  return keyinst;
}
;
```

Key Removal

To delete a key, persistent or otherwise, the following function is used:

`xenc_key_remove()`

17.5.2. SOAP Server WS-Security Endpoint

The WS-Security processing is performed by filtering incoming and outgoing messages of the Virtuoso SOAP server. These filters are activated when a special SOAP option is set on the current virtual directory that is the SOAP endpoint. The WS-Security filters are invoked on input to verify / decrypt the message. Upon success the message will be processed by the SOAP server, otherwise a `SOAP:Fault` will be returned to the SOAP requester. On outgoing messages, depending of SOAP options, messages produced by the SOAP server can be encoded and signed as well. The administrator configures the WS Security subsystem at the access point level. Different security schemes involving encryption and/or signature can be selected.

The keys and certificates need to be defined in order to get a working secure service. The key/certificate(s) for the SOAP endpoint that is WS-Security enabled are referenced in a special PL hook and/or signature template. If none of these (signature or key instance hook) are defined the response will not be encrypted or signed.

Here are the steps involved in processing a message to a secure end point.

1. SOAP server receives a message on a secure endpoint
2. The message is determined to be for this endpoint, otherwise will be sent to the next SOAP router if routing is enabled.
3. The message (as is) is passed to the decoding routine. At this point keys that are referenced in SOAP message need to be in the user space of the SQL account on whose behalf SOAP accessible procedures of this end point run. If any key does not exist in the user space, the requested processing will fail.

   
   Note:
   Signatures can be verified in the following manners:

   never try signatures
   expect signatures, explicit
   try signature if exists
   This behavior depends on the "WSS-Validate-Signature" option set for the virtual directory.
4. If step 3 completes without problem, security related headers are stripped from the decoded message.
5. The result of point 4 is passed to the usual SOAP server for processing.

6. Once a response is generated by the SOAP method (i.e. corresponding PL procedure, exposed as SOAP method) the result will be encoded and/or signed. This is the last step before the result is sent back to the requestor client. At this point the server behavior is controlled by a few options defined in the virtual directory. See below: "WSS-KEY", "WSS-Template", and "WSS-Type" options.

17.5.3. Virtual Directory SOAP WSS Options

The following SOAP options are available for configuring a virtual directory:

- **WS-SEC .** WS-Security processing is enabled on the endpoint, in practice this enables the input and output message filters for Security Header processing. This MUST be set to enable WS-Security endpoints. All other WS related options will be ignored if this option is disabled. The possible values for this option are "yes" or "no"
- **WSS-KEY .** The name of a PL procedure that is expected to return a key instance. Examples of this can be found in the Referencing Keys section. If this NULL or undefined the SOAP server will not encrypt outgoing messages.
- **WSS-Template .** path and file name to a file containing a template - an instruction for how to make the XML signature for the response message. For more information see the Security Templates section. If this option is NULL or undefined the SOAP server will not sign any outgoing messages from this endpoint. As an example, if the value "sig.xml" was supplied to this option this would imply that the content of template is stored in the file called sig.xml located in the servers working directory. A default template can be used by setting this option to the value `[key reference for signing]` - (note '[]' around name is required). In this case the server will generate a default template based on a key, and produce a default rule for making a signature.
- **WSS-Validate-Signature.** This option controls how the SOAP server responds to incoming messages i.e. how to verify the incoming message.

  possible values are:

  0 do not verify signatures

  1 expects signature to exist

  2 will verify signature if exists

Example 17.27. Configuring WS Secure Endpoints

```
-- definition of /SecureWebServices endpoint
VHOST_DEFINE (
  lpath=>'/SecureWebServices', ppath=>'/SOAP/', soap_user=>'WSS',
  soap_opts=>vector(
    'Namespace','http://temp.uri/',
    'MethodInSoapAction','yes',
    'ServiceName', 'WSSecure',
    'CR-escape', 'no',
    'WS-SEC','yes',
    'WSS-Validate-Signature', 2)
  )
;
```

this will define a WS-Security enabled endpoint that is compatible with .NET WSDK examples. This endpoint will accept encrypted/or signed SOAP messages and will generate an error if security checking fails. Upon success it will return non-encrypted message. Therefore it's a one-way encryption example.

```
-- complex example
VHOST_DEFINE (
  lpath=>'/wss', ppath=>'/SOAP/', soap_user=>'WSS',
  soap_opts=>vector(
    'Namespace','http://soapinterop.org/',
                'MethodInSoapAction','no',
                'ServiceName', 'WSSecure',
                'HeaderNS', 'http://soapinterop.org/echoheader/',
                'CR-escape', 'no',
    'WS-SEC','yes',
                'WSS-KEY','DB.DBA.WS_SOAP_GET_KEY',
                'WSS-Template','wss_tmpl.xml',
    'WSS-Validate-Signature', 1)
  )
;
```

This endpoint will expect signed and encrypted incoming messages and will sign and encrypt outgoing messages too. This is a two-way encryption example, for client to server and reverse.

17.5.4. Accounting & Accounting Hook

If an X.509 certificate is used to sign an incoming message, the following connection variables will be available to the SOAP processing function and accounting hook:

wss-token-owner owner of certificate, an example is "C=US/ST=MA/CN=User Name".

wss-token-issuer issued by , it's like above , but it's name of who is issued the certificate.

wss-token-serial serial number, an integer specific to certificate issuer.

wss-token-start valid from, a string containing a data of validity.

wss-token-end valid to, a string containing data of end of validity.

These can be accessed by invoking connection_get([name-of-info]) when processing the SOAP request, i.e. in the procedure being invoked or from a user defined accounting hook.

A user defined procedure hook named DB.DBA.WS_SOAP_ACCOUNTING can be used for accounting purposes. If it exists it will be invoked after the connection information is set. This PL hook should return 0 when an error occurs or 1 upon success. Hence, a SOAP request may be rejected based on some custom condition.

The procedure prototype is :

```
create procedure DB.DBA.WS_SOAP_ACCOUNTING ()
{
  -- custom logic
  return 1; -- on success
  return 0; -- on failure
}
;
```

Note that the usage of this functionality is global to the Virtuoso server, To get similar functionality for each SOAP method, the developer will need to include account checking within the PL procedures that are exposed as SOAP methods.

17.5.5. Signature Templates

Signature templates are used to define how signatures are generated for SOAP messages. They are XML files containing a structure of Signature elements as per XML Signature standard. The DigestValue and SignatureValue elements are empty, so they will be filled upon output. The most important element is the KeyName element. This must contain a reference to an existing key from the user account space. This key reference does not need to be the same as key for data encryption, it can be a different key. Please note that XML encoding routines will resolve the key automatically, so there is no need to specify how the key instance will be obtained, hence no relation to "WSS-Key" from SOAP options section (above).

Default Template Generation

The default template will create an XML Signature using the following definitions:

a) have a simple template corresponding to a key reference (and it's algorithm) as:

```
TEMP := <?xml version="1.0" encoding="UTF-8"?>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#" >
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
    <SignatureMethod Algorithm="[key algorithm]" />
  </SignedInfo>
  <SignatureValue></SignatureValue>
  <KeyInfo>
    <KeyName>[key reference]</KeyName>
  </KeyInfo>
</Signature>'
```

b) generate references to template a), using the `dsig_template_ext()` function and BODY of the SOAP request.

```
TEMPLATE :=  dsig_template_ext (xtree_doc (SOAP_BODY), TEMP,
      'http://schemas.xmlsoap.org/soap/envelope/', 'Body',
      'http://schemas.xmlsoap.org/rp', 'action',
      'http://schemas.xmlsoap.org/rp', 'to',
      'http://schemas.xmlsoap.org/rp', 'id',
      'http://schemas.xmlsoap.org/ws/2002/07/utility', 'Created',
      'http://schemas.xmlsoap.org/ws/2002/07/utility', 'Expires');
```

In other words, the 'Body' of the SOAP message, part of the routing header elements ('action,'to','id') and, if it exists, Timestamp headers ('Created', 'Expires').

Example 17.28. Using default templates

This example demonstrates the use of the default template mechanism and X.509 certificate with RSA key:

```
--------------- SOAP message
<SOAP:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
 xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
 xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/07/secext"
 xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <SOAP:Header />
  <SOAP:Body
xml:id="id-2e7c29c7-8645-4ad8-af59-06bed179d2fb">
    <AddInt xmlns="http://microsoft.com/wsdk/samples/SumService">
      <a>3</a>
      <b>4</b>
    </AddInt>
  </SOAP:Body>
</SOAP:Envelope>

--------------- Generated signature template
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
              xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/07/secext"
              xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
   <ds:SignedInfo>
      <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
      </ds:CanonicalizationMethod>
      <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"></ds:SignatureMethod>
      <ds:Reference URI="#Id-2e7c29c7-8645-4ad8-af59-06bed179d2fb">
        <ds:Transforms>
          <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
            </ds:Transform>
          </ds:Transforms>
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></ds:DigestMethod>
        <ds:DigestValue></ds:DigestValue>
      </ds:Reference>
   </ds:SignedInfo>
   <ds:SignatureValue></ds:SignatureValue>
    <ds:KeyInfo>
      <wsse:SecurityTokenReference>
        <wsse:Reference URI="#SecurityToken-24c152e3-26e0-d611-bb59-90b4c67d3be5"></wsse:Reference>
      </wsse:SecurityTokenReference>
    </ds:KeyInfo>
</ds:Signature>
```

Now, an example of a user defined template:

```
<?xml version="1.0" encoding="UTF-8"?>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1" />
      <Reference URI="">
        <Transforms>
          <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
        </Transforms>
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
```

```
            <DigestValue></DigestValue>
          </Reference>
        </SignedInfo>
        <SignatureValue></SignatureValue>
        <KeyInfo>
            <KeyName>file:dsa.der</KeyName>
        </KeyInfo>
    </Signature>
```

17.5.6. SOAP Client

The `soap_client()` function is used to make soap requests. Upon success it will return an array containing the XML tree of the result. However, it can return more complex results when the debug parameter is enabled, in which case it can return the request and the response messages as well. Some of the parameters are optional, not all are required and they can be named.

See Also:
`soap_client()`

17.6. Web Services Routing Protocol (WS-Routing)

The SOAP Routing Protocol (WS-Routing) is a SOAP-based, stateless protocol for exchanging SOAP messages from an initial sender to an ultimate receiver, potentially via a set of intermediaries. The WS-Routing protocol is implemented as a SOAP extension, embedded in a SOAP Header entry.

The WS-Routing implementation consists of SOAP Header processing. The header processing handler for the WS-Routing header determines if the Virtuoso SOAP server can act as an intermediary or a endpoint depending on the rules in message path.

This implementation supports HTTP only. Message Id's generated are UUIDs.

17.6.1. Configuration

Setting-up the WS-Routing for a SOAP service requires you to:

1. make a virtual directory bound to the SOAP endpoint processor (/SOAP)
2. add the following SOAP options:

   WS-RP=yes;
   wsrp-from=[identification for endpoint];

17.6.2. Traversing Message Paths

The initial WS-Routing sender generates a WS-Routing "path" header that indicates the route. The path can indicate a route via one or more intermediaries using the "via" elements as sub-elements of the "fwd" element. The initial sender indicates the ultimate destination by using a "to" element. In the absence of a "to" element the ultimate destination is deduced by the last "via" in the "fwd" element. The second option occurs most commonly when an ultimate destination reverses roles, becomes an initial sender, and uses the reverse path in a received message as a forward path to send a response to the original sender.

In addition, the initial WS-Routing sender inserts a reverse path for indicating where the initial sender can receive reverse path messages. The initial sender sets the ultimate destination in the reverse path using a "via" element as a sub-element of the "rev" element.

A WS-Routing receiver receiving a WS-Routing message inspects the SOAP header and performs the following operations:

- If no "fwd" element is present or if the "fwd" element does not contain any "via" elements then inspect the "to" element and verify that the value identifies itself. If this is the case then THIS receiver is the ultimate destination. If there is no "to" element or if the value of the "to" element does not identify THIS WS-Routing receiver then generate a fault.
- If the "fwd" element is present and contains one or more "via" element(s) then remove the top "via" element listed in the "fwd" element and verify that the value of that "via" element is either empty or identifies THIS WS-Routing receiver. Failing that generate an appropriate WS-Routing fault

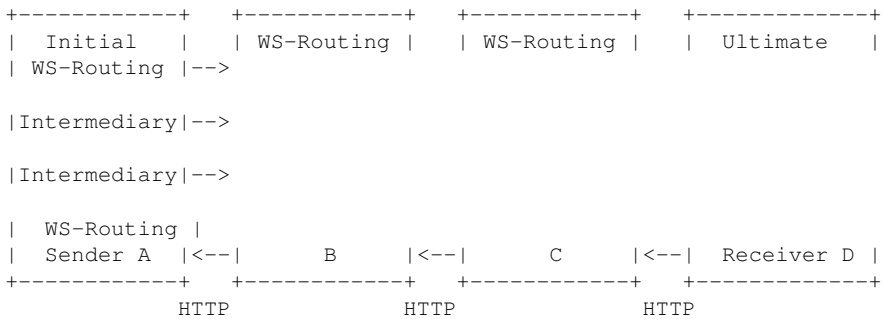
If, after removing the top "via" element there are no remaining "via" element(s) listed in the "fwd" element, and there is no "to" element, then THIS WS-Routing receiver is the ultimate destination.

A WS-Routing intermediary MUST follow these additional rules:

- If a "rev" element is present then add a "via" element as the first "via" element listed in the "rev" element with a value indicating the reverse path endpoint. If a reverse path endpoint cannot be provided then generate a 751 "Reverse Path Unavailable" WS-Routing fault.
- If one or more "via" element(s) remain in the "fwd" element then forward the WS-Routing message to the endpoint identified by the new top "via" element listed in the "fwd" element.
- If there are no remaining "via" element(s) listed in the "fwd" element but there is a "to" element then forward the WS-Routing message to the endpoint identified by the "to" element.
- In the last two cases if the forwarding does not succeed then generate the appropriate WS-Routing fault.

Example 17.29. WS Routing Example

Here is an example SOAP call from a client to an endpoint D.com via intermediaries B.com and C.com:



Request from client to intermediary B

```
<S:Envelope>
<S:Header>
  <m:path
     xmlns:m="http://schemas.xmlsoap.org/rp/"
     S:actor="http://schemas.xmlsoap.org/soap/actor/next"
     S:mustUnderstand="1">
   <m:action>http://soapinterop.org/</m:action>
   <m:to>http://D.com/router</m:to>
   <m:id>uuid:09233523-345b-4351-b623-5dsf35sgs5d6</m:id>
   <m:fwd>
    <m:via>http://B.com/router</m:via>
    <m:via>http://C.com/router</m:via>
   </m:fwd>
   <m:rev />
  </m:path>
</S:Header>
<S:Body>
...
</S:Body>
</S:Envelope>
```

Request from intermediary B to intermediary C

```
<n0:Envelope>
<n0:Header>
  <n2:path
     xmlns:n2="http://schemas.xmlsoap.org/rp/"
     n0:actor="http://schemas.xmlsoap.org/soap/actor/next"
     n0:mustUnderstand="1">
   <n2:action>http://soapinterop.org/</n2:action>
   <n2:to>http://D.com/router</n2:to>
   <n2:id>uuid:09233523-345b-4351-b623-5dsf35sgs5d6</n2:id>
   <n2:fwd>
     <n2:via>http://C.com/router</n2:via>
   </n2:fwd>
```

```
    <n2:rev>
      <n2:via />
    </n2:rev>
  </n2:path>
</n0:Header>
<n0:Body>
...
</n0:Body>
</n0:Envelope>
```

Request from intermediary C to endpoint D

```
<n0:Envelope>
  <n0:Header>
    <n2:path
      xmlns:n2="http://schemas.xmlsoap.org/rp/"
      n0:actor="http://schemas.xmlsoap.org/soap/actor/next"
      n0:mustUnderstand="1">
     <n2:action>http://soapinterop.org/</n2:action>
     <n2:to>http://D.com/router</n2:to>
     <n2:id>uuid:09233523-345b-4351-b623-5dsf35sgs5d6</n2:id>
     <n2:fwd />
     <n2:rev>
      <n2:via />
      <n2:via />
     </n2:rev>
   </n2:path>
</n0:Header>
<n0:Body>
...
</n0:Body>
</n0:Envelope>
```

Response from endpoint to C

```
<n0:Envelope>
<n0:Header>
  <n2:path xmlns:n2="http://schemas.xmlsoap.org/rp/"
          n0:actor="http://schemas.xmlsoap.org/soap/actor/next"
          n0:mustUnderstand="1">
        <n2:action>http://soapinterop.org/</n2:action>
        <n2:id>uuid:2b2d09ec-a93a-11d6-be21-c9f55c969fe7</n2:id>
        <n2:relatesTo>uuid:09233523-345b-4351-b623-5dsf35sgs5d6</n2:relatesTo>
        <n2:fwd>
          <n2:via />
          <n2:via />
        </n2:fwd>
        <n2:rev>
           <n2:via>http://D.com/router</n2:via>
        </n2:rev>
        <n2:from />
  </n2:path>
</n0:Header>
<n0:Body>
...
</n0:Body>
</n0:Envelope>
```

Response from intermediary C to B

```
<n0:Envelope>
 <n0:Header>
    <n2:path xmlns:n2="http://schemas.xmlsoap.org/rp/"
             n0:actor="http://schemas.xmlsoap.org/soap/actor/next"
             n0:mustUnderstand="1">
        <n2:action>http://soapinterop.org/</n2:action>
        <n2:id>uuid:2b2d09ec-a93a-11d6-be21-c9f55c969fe7</n2:id>
        <n2:relatesTo>uuid:09233523-345b-4351-b623-5dsf35sgs5d6</n2:relatesTo>
        <n2:fwd>
          <n2:via />
        </n2:fwd>
        <n2:rev>
```

```
            <n2:via>http://C.com/router</n2:via>
            <n2:via>http://D.com/router</n2:via>
          </n2:rev>
          <n2:from />
       </n2:path>
</n0:Header>
<n0:Body>
...
</n0:Body>
</n0:Envelope>
```

Response from intermediary B to client

```
<n0:Envelope>
<n0:Header>
    <n2:path xmlns:n2="http://schemas.xmlsoap.org/rp/"
             n0:actor="http://schemas.xmlsoap.org/soap/actor/next"
             n0:mustUnderstand="1">
           <n2:action>http://soapinterop.org/</n2:action>
           <n2:id>uuid:2b2d09ec-a93a-11d6-be21-c9f55c969fe7</n2:id>
           <n2:relatesTo>uuid:09233523-345b-4351-b623-5dsf35sgs5d6</n2:relatesTo>
              <n2:fwd />
           <n2:rev>
              <n2:via>http://B.com/router</n2:via>
              <n2:via>http://C.com/router</n2:via>
              <n2:via>http://D.com/router</n2:via>
           </n2:rev>
           <n2:from />
      </n2:path>
</n0:Header>
<n0:Body>
...
</n0:Body>
</n0:Envelope>
```

17.7. Web Services Reliable Messaging Protocol (WS-ReliableMessaging)

The WS-ReliableMessaging protocol is a SOAP-based RPC protocol for guaranteed delivery of messages; possibly in specific order from one sender to one receiver. Such messages are usual SOAP messages - XML documents conforming to the SOAP specification. The Sender is an alias of the transmission initiator, i.e. the originator of the message transfer. The Receiver is a recipient, that which accepts the messages. How accepted messages should be processed is not covered in this document.; What to do with the data and whether to send replies is at the discretion of the application.

See Also:
WS Reliable Messaging

WS Reliable Messaging Exec Summary

Further in this section for brevity WS-RM will be used in place of "Web Services Reliable Messaging Protocol".

Delivery Assurances Types:

AtMostOnce - Delivery at most once without duplication or an error will be raised on at least one endpoint. It is possible that some messages in a sequence may not be delivered.

AtLeastOnce - Every message sent will be delivered or an error will be raised on at least one endpoint. Some messages may be delivered more than once.

ExactlyOnce - Every message sent will be delivered without duplication or an error will be raised on at least one endpoint. This delivery assurance is the logical "and" of the two prior delivery assurances.

InOrder - Messages will be delivered in the order that they were sent. This delivery assurance may be combined with any of the above delivery assurances. It requires that the sequence observed by the ultimate receiver be non-decreasing. It says nothing about duplications or omissions.

See Also:
WS-RM System Table Definitions in the Appendix section

17.7.1. SOAP CLIENT API Extensions

The SOAP Client API is used for handling, building and accessing complex types required to perform a SOAP request. It allows you to build a structures to access their elements and to build values suitable for passing to the `SOAP_CLIENT()` function. It also allows to de-serialize a SOAP response to a `soap_parameter` and access its members and attributes safely.

Vectors are used to pass or retrieve a complex type using SOAP These vectors contain special content or UDTs. In SOAP, structures are allowed to have multiple members of the same name, or conditional members (choices). Thus it is not possible to cover all variants of XSD types with UDTs (which would be the most elegant way to represent structures). So to cover all variants of structure handling or to express structures containing attributes, we have to use vectors. Since we used a special structure with vectors for expressing such complex types we introduce the following API to deal with them.

The base of API is a UDT 'soap_parameter':

```
create type soap_parameter as
                (
                  s any default null,
                  param_type int default 1,
                  param_xsd varchar default null,
                  ver int default 11
                )
                temporary self as ref
;
```

Its members are:

's' - a vector representing a structure, array or simple type.
'param_type' - designates what is kept in 's' : 1 is struct ; 0 - simple type; 2 - array
'param_xsd' - URL; QName of the type stored in 's'
'ver' - soap version to be used for serialization/deserialization if RPC style is used.
Its Constructors are:

```
constructor method soap_complex_parameter ()
Instantiate an empty structure placeholder;

constructor method soap_simple_parameter (val any),
Instantiate an empty simple type placeholder

constructor method soap_array_parameter (n int),
Instantiate an array placeholder

constructor method soap_single_parameter (elm soap_parameter),
Instantiate an element of containing a type of 'elm'
```

Its Methods are:

```
method get_length () returns any,
Returns the  length of the parameter

method add_member (name varchar, val any) returns any,
Add a new member element to the structure placeholder

method set_member (name varchar, val any) returns any
Sets the value of an existing member by name or will add a new member  if  the member does not exist

method set_member (pos int, val any) returns any,
the same as above but using ordinal position of the member;

method get_member (name varchar) returns any,
Returns a member's value by name

method get_member (pos int) returns any,
Returns member value by ordinal position

method get_value () returns any,
```

```
Returns value for simple types. Only for simple type is applicable

method set_value (val any) returns any,
Sets the value of a simple type

method set_attribute (name varchar, val any) returns any,
Set an attribute value of a structure or simple type

method get_attribute (name varchar) returns any,
Return an attribute value of a structure or simple type

method get_call_param (name varchar) returns any,
```

Serializes the parameter in a form suitable for passing to the SOAP_CLIENT() function. When several parameters needs to be passed to it, then result of those calls must be concatenated (see vector_concat()) Important: set_xsd have to be called with valid ExQName, before doing get_call_param call.

```
method set_xsd (xsd varchar) returns any,
```

Establish a SOAP data-type Expanded QName, referencing the data kept in the soap_parameter. This is a reference to a SOAP complex data-type already defined with soap_dt_define () function (see function reference). This method only sets the relation between data kept in soap_parameter and doesn't check its validity unless serialize is performed. The data-type Expanded QName also will be included into output of the get_call_param method.

Example 17.30. Complex WS Type Example

Consider the following complex type:

```
<xsd:complexType
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    name="SOAPComplexType"
    targetNamespace="http://soapinterop.org/xsd">
 <xsd:sequence>
        <xsd:element minOccurs="1" maxOccurs="1" name="varInt" type="xsd:int" />
        <xsd:element minOccurs="0" maxOccurs="1" name="varString" type="xsd:string" />
        <xsd:element minOccurs="1" maxOccurs="1" name="varFloat" type="xsd:float" />
 </xsd:sequence>
</xsd:complexType>

declare s soap_parameter ;
s := new soap_parameter ();
s.set_xsd ('http://soapinterop.org/xsd:SOAPComplexType');
s.add_member ('varString', 'string');
s.serialize ('mystruct'); -- will generate an error as, varInt and varFloat are
                                    -- missing but they are declared as minOccurs="1"
```

So if we add two more members, and remove varString, to the statements above.

```
s.add_member ('varInt', 123);
s.add_member ('varFloat', 3.14);
s.serialize ('mystruct');
```

Will produce valid output:

```
<mystruct><varInt>123</varInt><varFloat>3.14</varFloat></mystruct>
```

and finally we can make a parameter for the SOAP_CLIENT() function:

```
par := s.get_call_param ('par1');
```

here are the contents of 'par':

```
(
  ("par1", "http://soapinterop.org/xsd:SOAPComplexType" ), -- designates parameter par1, with type SOAPCo
  (<COMPOSITE>, "",     -- this is a marker that it's a struct
        "varInt", 123,  -- the members name/value pairs follows
        "varFloat", 3.14
```

```
    )
)
```

Further methods are:

```
method deserialize (xs any, elem varchar) returns any,
```

Deserializes a element 'elem' from 'xs' (XML tree object) the the soap_parameter.

```
method serialize (tag varchar) returns any,
```

Returns an XML document representing the data kept into the soap_parameter.

```
method set_struct (s any) returns any
```

Explicitly sets the structure/array/simple type kept in the soap_parameter UDT. This can be used to disassemble a nested complex type into its components.

Example 17.31. Example Using WS-RM

echoStruct invocation

```
declare ret any;
declare pa soap_parameter;

pa := new soap_parameter ();
pa.add_member ('varString', 'My String');
pa.add_member ('varInt', 12345);
pa.add_member ('varFloat', 3.1415926);
pa.set_xsd ('http://soapinterop.org/xsd:SOAPStruct');

ret := SOAP_CLIENT (url=>'http://.../interop',
  operation=>'echoStruct', parameters=>pa.get_call_param ('inputStruct'));

pa := new soap_parameter ();
pa.set_xsd ('http://soapinterop.org/xsd:SOAPStruct');
pa.deserialize (ret, 'CallReturn');
return pa.get_member ('varString');
```

echoDocument invocation

```
declare ret any;
declare doc, pa soap_parameter;

doc := new soap_parameter ('The document content');
doc.set_attribute ('ID', uuid());
doc.set_xsd ('http://soapinterop.org/xsd:x_Document');

ret := SOAP_CLIENT (url=>'http://.../r3/Compound1', operation=>'echoDocument',
  parameters=>doc.get_call_param ('x'), style=>1);

pa := new soap_parameter ();
pa.set_xsd ('http://soapinterop.org/xsd:Document');
pa.deserialize (ret, 'result_Document');
dbg_obj_print (pa.s);

return pa.get_attribute ('ID');
```

See Also:

`SOAP_CLIENT()`

17.7.2. WS-RM Sender API

The WS-RM API allows for:

register/start a conversation
set parameters on both sides (receiver or sender)
check the acknowledgment
retry if needed
to finish or cancel the conversation (can be with cancel)
check for faults
The User Defined Types defined in support of WS-RM are as follows:

- *soap_client () parameters wrapper*

```
create type soap_client_req as
        (
        url varchar,
        operation varchar,
        target_namespace varchar default null,
        parameters any default null,
        headers any default null,
        soap_action varchar default '',
        attachments any default null,
        ticket any default null,
        passwd varchar default null,
        user_name varchar default null,
        user_password varchar default null,
        auth_type varchar default 'none',
        security_type varchar default 'sign',
        debug integer default 0,
        template varchar default null,
        style integer default 0,
        version integer default 11,
        direction integer default 0
        )
;
```

  This UDT is used for passing various parameters to the WS-RM client to send a SOAP messages along with WS-RM specific headers. The members are with same names as for SOAP_CLIENT () function. In other words applications must fill a soap_client_req specific data and pass this to the wsrm_cli methods (see below).
- *client addressing UDT:* To facilitate two-way transport the receiver must know the address of the sender's responder (endpoint for asynchronous Acknowledgment or response). To pass such addresses to the WS-RM client (wsrm_cli) the following UDT is used.

```
create type wsa_cli as
        (
        mid varchar default null,
        "to" varchar default null,
        "from" varchar default null,
        action varchar default null,
        fault_to varchar default null,
        reply_to varchar default null
        )
```

  Whose members are as follows:

  mid - message identifier; unique across space and time
  to - where message traveling to
  from - from where it's sent
  action - WS-Addressing Action headed element
  fault_to - where to return the fault
  reply_to - where to reply; if no such URL given the "to" will be used to reply.

  Important Note:

Important Note:

A special "to" equal to 'http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous ' is used to designate an 'anonymous' sender. Further Acknowledgment can be made as a synchronous reply only. There is no way to return a response in asynchronous manner.

Constructors:

```
constructor method wsa_cli ()
```

Instantiate a new addressing placeholder ; with new message id.

- *WS-RM Client UDT* The following is a core UDT for the sender's activity. It is used to establish a message sequence context, to set various parameters and to send/check/cancel WS-RM encapsulated SOAP messages to the sender.

```
create type wsrm_cli as
        (
        url varchar,
        seq varchar,
        msgno int default -1,
        assurance varchar,
        expiry datetime,
        address wsa_cli,
        i_timeout int,
        resend_intl int,
        ack_intl int,
        dirty int default 0,
        is_last int default 0,
        is_finish int default 0
        )
        temporary self as ref
```

Members:

url - the recipient's URL

seq - identifier; unique Id for batch of messages to be sent together

msgno - unique number/unsigned integer greater than 1, unique identifying a message within a sequence

assurance - policy to be applied on receiver side : AtLeastOne; InOrder; AtMostOne

expiry - message expiration

address - a UDT of type `wsa_cli` ; identifying sender and receiver URLs

i_timeout - inactivity timeout

ack_intl - Acknowledgment interval

dirty - (internal use)

is_last - the current message is a last in sequence

is_finish - transfer is finished; no more messages are pending

Constructors:

```
constructor method wsrm_cli (addr wsa_cli, url varchar),
```

Instantiate a new WS-RM sender object; new message sequence This will generate a new sequence identifier, and will store it into a outgoing sequences table. The default policy will be established.

```
constructor method wsrm_cli (addr wsa_cli, url varchar, seq varchar),
```

Instantiate a WS-RM sender object from a persisted state; The policy and parameters will be read from database and instantiated in WS-RM sender's object.

Methods:

```
method create_sequence () returns any,
```

Start a new sequence with current WS-RM sender's object;

```
method send_message (req soap_client_req, last int) returns any,
```

Send a message to the recipient; depending on 'last' flag this can be designate open or closed message sequence. The 'req' must be fulfilled with appropriate parameters (see above).

```
method send_message (req soap_client_req) returns any,
```

same as send_message (req soap_client_req, 0)

```
method finish (req soap_client_req) returns any,
```

same as send_message (req soap_client_req, 1)

```
method check_state () returns any,
```

retrieves a state for a sequence. If sequence is closed and all is delivered no remote call will be performed. If an message is in pending state a remote call will be issued to check the state of message sequence. The return value is an array containing message numbers (id's within sequence) and their state: 0 or 1 - send or pending

```
method cancel () returns any,
```

cancel the sequence; kill it on WS-RM sender site ; say that to recipient to do that also.

```
method set_parameter (name varchar, val any) returns any,
```

Set a parameter to the current message sequence : like policy; timeout etc. The parameters will be sent together with next WS-RM conversation. Note: Some of parameters may invalidate previous state of message transfer; so in this case the sender will receive a fault and application must handle the situation properly.

Valid 'name's are :

Assurance - 'AtMostOnce','AtLeastOnce','ExactlyOnce','InOrder'
Expiry - date of expiration
Timeout - inactivity timeout in milliseconds
RetryInterval - retry on every n milliseconds if message is not sent
AckInterval - Acknowledgment interval in milliseconds

Sender's responder (endpoint) . To receive an asynchronous Acknowledgment the sender must have an endpoint to handle them. The `WSRMSequenceAcknowledgment()` procedure must be exposed at that endpoint. This accepts and processes asynchronous Acknowledgment. This is used to accept a SequenceAcknowledgment response from a remote party so it will process the response and will set the state of messages that are acknowledged.

17.7.3. WSRM Receiver API

On the receiver side we have PL wrappers that take as arguments all *known* WS-RM headers. This procedure, when granted to a SOAP endpoint will process the incoming requests. It will also answer send the appropriate answers to the sender.

The following is the list of WS-RM receiver wrappers:

1. **WSRMSequence .** accept and store a Sequence requests. This is used to accept Sequence requests from sender and will perform the following actions:

   store the message in incoming message log table

   process and apply the Policy attachment (if supplied); otherwise apply default rule: AtMostOnce

   depending of a addressing headers will reply in synchronous or asynchronous manner. This depends on the value of "From" and "ReplyTo" headers. If these contains non-anonymous URL (see above); an asynchronous Acknowledgment will be sent in a separate TCP connection. Thus in this mode the sender must have defined a listener with WSRMSequenceAcknowledgment exposed to accept such responses. Otherwise synchronous Acknowledgment will be sent in the same TCP connection.

If an error occurred a Fault will be generated

2. **WSRMSequenceTerminate .** accept and process SequenceTerminate requests. This is used to accept cancel request; to kill an existing message sequence. Further actions on such sequence will be rejected.
3. **WSRMAckRequested .** accept and process AckRequested requests. This is used to process AckRequested WS-RM messages and will produce a SequenceAcknowledgment containing info about messages been accepted. Used from sender's side to check the message sequence state.

These PL procedures are built-in to the server, and have to be granted to the user that is assigned as the SOAP execution account for a given virtual directory designated as a WS-RM receiver endpoint.

17.7.4. WS-RM Protocol Endpoint Configuration

The setup is a virtual directory definition and grant of rights to the procedures for sender and receiver endpoints.

This will be demonstrated as an example of setting up both endpoints:

- *Sender*

```
create user WSRMS;
grant execute on WSRMSequenceAcknowledgment to WSRMS;
vhost_define (lpath=>'/replyto', ppath=>'/SOAP/', soap_user=>'WSRMS');
```

- *Reply*

```
create user WSRMR;
grant execute on WSRMSequence to WSRMR;
grant execute on WSRMSequenceTerminate to WSRMR;
grant execute on WSRMAckRequested to WSRMR;
vhost_define (lpath=>'/wsrm', ppath=>'/SOAP/', soap_user=>'WSRMR');
```

Example 17.32. WS-RM Ping Test Example

This is an example client used to perform the interoperability test "Ping" as proposed in "Interop Workshop Scenarios"

```
soap_dt_define ('',
                '<element
                        xmlns="http://www.w3.org/2001/XMLSchema"
                        name="Ping" type="test:Ping_t"
                        targetNamespace = "http://tempuri.org/" xmlns:test="http://tempuri.org/" />');

soap_dt_define ('',
                '<complexType xmlns="http://www.w3.org/2001/XMLSchema"
                              name="Ping_t" targetNamespace = "http://tempuri.org/">
                                <sequence>
                                        <element minOccurs="1" maxOccurs="1" name="Text" type="string"/>
                                </sequence>
                </complexType>');

create procedure WSRMTestPing (in _to varchar, in _from varchar)
  {
    declare addr wsa_cli;
    declare test wsrm_cli;
    declare req soap_client_req;
    declare finish any;
    declare ping soap_parameter;

    ping := new soap_parameter (1);

    ping.set_xsd ('http://tempuri.org/:Ping');
    ping.s := vector ('Hello World');

    addr := new wsa_cli ();
    addr."to" := _to;
    addr."from" := _from;
    addr.action := 'urn:wsrm:Ping';

    req := new soap_client_req ();
    req.url := _to;
    req.operation := 'Ping';
```

```
    req.parameters := ping.get_call_param ('');

    test := new wsrm_cli (addr, _to);

    test.send_message (req);
    test.send_message (req);
    test.finish (req);
    return test.seq;
  }
;
```

17.7.5. Message Examples

Example 17.33. Example: Initial request

```
<SOAP:Envelope>
  <SOAP:Header>
    <wsa:MessageID>uuid:aa8bd508-110b-11d8-8344-8cfad4a25a87</wsa:MessageID>
    <wsa:To>http://host:9999/wsrm</wsa:To>
    <wsa:From>
      <wsa:Address>http://host:9999/replyto</wsa:Address>
    </wsa:From>
    <wsa:Action><!-- depending of application -->

</wsa:Action>
    <wsrm:Sequence>
      <wsu:Identifier>uuid:aa8bfa74-110b-11d8-8344-8cfad4a25a87</wsu:Identifier>
      <wsrm:MessageNumber>1</wsrm:MessageNumber>
    </wsrm:Sequence>
  </SOAP:Header>
  <SOAP:Body>
    <!-- some application data -->

  </SOAP:Body>
</SOAP:Envelope>
```

receiver accepts and return just

```
HTTP/1.1 202 Accepted
Content-Length: 0
```

Last message

```
<SOAP:Envelope>
  <SOAP:Header>
    <wsa:MessageID>uuid:aa8bd508-110b-11d8-8344-8cfad4a25a87</wsa:MessageID>
    <wsa:To>http://host:9999/wsrm</wsa:To>
    <wsa:From>
      <wsa:Address>http://host:9999/replyto</wsa:Address>
    </wsa:From>
    <wsa:Action><!-- depending of application -->

</wsa:Action>
    <wsrm:Sequence>
      <wsu:Identifier>uuid:aa8bfa74-110b-11d8-8344-8cfad4a25a87</wsu:Identifier>
      <wsrm:MessageNumber>2</wsrm:MessageNumber>
      <wsrm:LastMessage/>
    </wsrm:Sequence>
  </SOAP:Header>
  <SOAP:Body>
    <!-- some application data -->

  </SOAP:Body>
</SOAP:Envelope>
```

receiver accepts and returns

```
HTTP/1.1 202 Accepted
Content-Length: 0
```

Sequence Acknowledgment response; sent via another channel

```
<SOAP:Envelope>
  <SOAP:Header>
    <wsa:MessageID>uuid:aadedf64-110b-11d8-8344-8cfad4a25a87</wsa:MessageID>
    <wsa:To>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</wsa:To>
    <wsa:From>
      <wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</wsa:Address>
    </wsa:From>
    <wsa:Action>http://schemas.xmlsoap.org/ws/2004/08/rm#SequenceAcknowledgment</wsa:Action>
    <wsrm:SequenceAcknowledgment>
      <wsu:Identifier>uuid:aa8bfa74-110b-11d8-8344-8cfad4a25a87</wsu:Identifier>
      <wsrm:AcknowledgmentRange Upper="2" Lower="1"/>
    </wsrm:SequenceAcknowledgment>
  </SOAP:Header>
  <SOAP:Body/>
</SOAP:Envelope>
```

initial sender accepts and returns

```
HTTP/1.1 202 Accepted
Content-Length: 0
```

17.7.6. WS-RM Schema

```
<xsd:schema
    targetNamespace="http://schemas.xmlsoap.org/ws/2003/03/rm"
        xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
        xmlns:wsrm="http://schemas.xmlsoap.org/ws/2003/03/rm"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified" attributeFormDefault="unqualified">

        <!-- *** BASE *** -->

    <xsd:complexType name="SequenceType">
        <xsd:sequence>
            <xsd:element ref="wsu:Identifier"/>
            <xsd:element name="MessageNumber" type="xsd:unsignedLong"/>
            <xsd:element name="LastMessage" type="xsd:ENTITY" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="SequenceTerminate_t">
        <xsd:sequence>
            <xsd:element ref="wsu:Identifier"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="AcknowledgmentRange_t">
        <xsd:sequence/>
        <xsd:attribute name="Upper" type="xsd:unsignedLong" use="required"/>
        <xsd:attribute name="Lower" type="xsd:unsignedLong" use="required"/>
    </xsd:complexType>

    <xsd:complexType name="SequenceAcknowledgment_t">
        <xsd:sequence>
            <xsd:element ref="wsu:Identifier"/>
            <xsd:element name="AcknowledgmentRange" type="wsrm:AcknowledgmentRange_t" maxOccurs="unbounde
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="AckRequestedType">
        <xsd:sequence>
            <xsd:element ref="wsu:Identifier"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:element name="Sequence" type="wsrm:SequenceType"/>
    <xsd:element name="SequenceTerminate" type="wsrm:SequenceTerminate_t"/>
```

```
<xsd:element name="SequenceAcknowledgment" type="wsrm:SequenceAcknowledgment_t" />
<xsd:element name="AckRequested" type="wsrm:AckRequestedType"/>

<!-- *** FAULTS *** -->

<xsd:simpleType name="FaultCodes">
    <xsd:restriction base="xsd:QName">
        <xsd:enumeration value="wsrm:UnknownSequence"/>
        <xsd:enumeration value="wsrm:SequenceTerminated"/>
        <xsd:enumeration value="wsrm:InvalidAcknowledgment"/>
        <xsd:enumeration value="wsrm:MessageNumberRollover"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="SequenceFaultType">
    <xsd:sequence>
        <xsd:element ref="wsu:Identifier"/>
        <xsd:element name="FaultCode" type="wsrm:FaultCodes"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:element name="SequenceFault" type="wsrm:SequenceFaultType"/>

<!-- *** ASSERTIONS *** -->

<xsd:complexType name="InactivityTimeout_t">
    <xsd:sequence/>
    <xsd:attribute name="Milliseconds" type="xsd:unsignedLong" use="required"/>
</xsd:complexType>

<xsd:complexType name="BaseRetransmissionInterval_t">
    <xsd:sequence/>
    <xsd:attribute name="Milliseconds" type="xsd:unsignedLong" use="required"/>
</xsd:complexType>

<xsd:complexType name="AcknowledgmentInterval_t">
    <xsd:sequence/>
    <xsd:attribute name="Milliseconds" type="xsd:unsignedLong" use="required"/>
</xsd:complexType>

<xsd:complexType name="PolicyAssertionType">
    <xsd:sequence>
        <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other"/>
</xsd:complexType>

<xsd:simpleType name="DeliveryAssuranceEnum">
    <xsd:restriction base="xsd:QName">
        <xsd:enumeration value="wsrm:AtMostOnce"/>
        <xsd:enumeration value="wsrm:AtLeastOnce"/>
        <xsd:enumeration value="wsrm:ExactlyOnce"/>
        <xsd:enumeration value="wsrm:InOrder"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="DeliveryAssurance_t">
    <xsd:sequence/>
    <xsd:attribute name="Value" type="xsd:QName" use="required"/>
</xsd:complexType>

<xsd:element name="InactivityTimeout" type="wsrm:InactivityTimeout_t" />
<xsd:element name="BaseRetransmissionInterval" type="wsrm:BaseRetransmissionInterval_t" />
<xsd:element name="ExponentialBackoff" type="wsrm:PolicyAssertionType"/>
<xsd:element name="AcknowledgmentInterval" type="wsrm:AcknowledgmentInterval_t"/>
<xsd:element name="DeliveryAssurance" type="wsrm:DeliveryAssurance_t"/>

<!-- *** Sequence Reference *** -->

<xsd:complexType name="SequenceRefType">
    <xsd:sequence />
    <xsd:attribute name="Identifier" type="xsd:anyURI" use="required"/>
    <xsd:attribute name="Match" type="wsrm:MatchChoiceType" use="optional"/>
```

```
        </xsd:complexType>

        <xsd:simpleType name="MatchChoiceType">
            <xsd:restriction base="xsd:QName">
                <xsd:enumeration value="wsrm:Exact"/>
                <xsd:enumeration value="wsrm:Prefix"/>
            </xsd:restriction>
        </xsd:simpleType>
        <xsd:element name="SequenceRef" type="wsrm:SequenceRefType"/>

    </xsd:schema>
```

17.8. Web Services Trust Protocol (WS-Trust)

In order to secure communication between two parties, the two parties must exchange security credentials (either directly or indirectly). However, each party needs to determine if they can "trust" the asserted credentials of the other party. WS-Security defines the basic mechanisms for providing secure SOAP messaging. WS-Trust is an extension of WS-Security for security token exchange to enable the issuance and dissemination of credentials within different trust domains, and thus manage trust relationships. The goal of WS-Trust is to enable applications to construct trusted SOAP message exchanges.

Using these extensions, applications can engage in secure communication designed to work with the general Web Services framework, including WSDL service descriptions, UDDI businessServices and bindingTemplates, and SOAP messages.

WSS (WS-Security) enabled endpoint can make use of (WST) WS-Trust by exposing the "`RequestSecurityToken` " method. It then will check the WSS headers, decode if appropriate and pass the request parameters to the RequestSecurityToken method.

Virtuoso supports and can generate many session key types. Supported key types are: DSA, 3des and AES. RSA keys can be imported but not generated, likewise x509 certificate generation, however they will be added in the near future.

WSS uses the `UsernameToken` method to bind an issued security token to a particular user.

Each WS enabled SOAP endpoint should have a list of supported encryption methods, keys that may be issued, and authorized users. This can be achieved using a PL (Stored Procedure) hook.

See Also:
Web Services Trust Language (WS-Trust)

The message flow involving WST endpoints will be as follows:

The client (1st instance) must ask the WST endpoint for token (security token, may be a 3des key x509 certificate or whatever security tokens are supported)

WST endpoint may or may not issue a token to the client.

Client sends a message to 2nd instance (the target) to perform the main request.

The 2nd instance (recipient) may accept or reject the request.

The 2nd instance may also ask WST for the token, to encrypt the data for client, which depends on the policy to be applied.

The client has to have a way to know what policy to apply. This can be a UDT that is initialized appropriately and passed to the client routines.

SOAP clients have to have an API to perform:

Request a security token from WST

Invoke the method from recipient with token obtained from WST endpoint.

```
wst_cli(req, policy)
```

The call to `wst_cli` performs the following actions:

For the URL of request determine whether a policy is needed; scan over policy array by URL.

If token has an issuer, then ask issuer with policy conforming to it (obtain via URL from policy array).

Apply the obtained token to the request and pass to the ultimate receiver.

Return response from ultimate receiver.

Server tokens are stored in the `WST_SERVER_ISSUER_TOKENS` system table.

See Also:

WST_SERVER_ISSUER_TOKENS

Tokens can be selected using the system procedure:

```
DB.DBA.WS_TRUST_TOKEN_GEN (
  in "From" any,
  in "MessageID" any,
  in "RequestSecurityToken" any,
  in "Timestamp" any,
  in "To" any).
```

This procedure can be over-ridden for specific cases. The definition of the default procedure is shown below.

```
create procedure  DB.DBA.WS_TRUST_TOKEN_GEN (
  in "From" any,
  in "MessageID" any,
  in "RequestSecurityToken" any,
  in "Timestamp" any, in "To" any )
{
   declare ret any;
   declare t_type, r_type, l_from varchar;

   t_type := cast ("RequestSecurityToken"[3] as varchar);
   r_type := cast ("RequestSecurityToken"[5] as varchar);
   l_from := cast ("From"[3] as varchar);

   select WSK_TOKEN into ret from WST_SERVER_ISSUER_TOKENS
               where WSK_TOKEN_TYPE = t_type and
                 WSK_REQUEST_TYPE = r_type and WSK_FROM = l_from;

   return ret;
}
;
```

Example 17.34. Example

1) client ask for context token token service:

```
<soap:Envelope
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
  xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header>
      <wsa:To>http://tokenservice</wsa:To>
      <wsse:Security soap:mustUnderstand="1">
        ....
      </wsse:Security>
    </soap:Header>
    <soap:Body wsu:Id="Id-d7fceab4-62ed-45fb-bc09-69310ff1712e">
      <wsse:RequestSecurityToken>
        <wsse:TokenType>wsse:SecurityContextToken</wsse:TokenType>
        <wsse:RequestType>wsse:ReqIssue</wsse:RequestType>
        <wsp:AppliesTo xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy">
          <wsa:EndpointReference>
            <wsa:Address>http://example.com/SecureConvPolicyService/SecureConvService.asmx</wsa:Address>
          </wsa:EndpointReference>
        </wsp:AppliesTo>
      </wsse:RequestSecurityToken>
    </soap:Body>
  </soap:Envelope>
```

2) response from token service

```
<soap:Envelope xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
```

```
  xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header>
      <wsse:Security soap:mustUnderstand="1">
        ....
      </wsse:Security>
    </soap:Header>
    <soap:Body wsu:Id="Id-ee536e2b-3911-46c8-9a51-850b11ecf866">
      <xenc:EncryptedData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
        <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
        <xenc:CipherData>
          <xenc:CipherValue>...</xenc:CipherValue>
        </xenc:CipherData>
      </xenc:EncryptedData>
    </soap:Body>
  </soap:Envelope>
```

3) using the token from response above ; perform request to the ultimate service

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
  xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext">
    <soap:Header>
      <wsa:To>http://quoteservice</wsa:To>
      <wsse:Security soap:mustUnderstand="1">
         ....
      </wsse:Security>
    </soap:Header>
    <soap:Body wsu:Id="Id-a8a78a3b-6775-470d-96d8-ca3f96fd2715">
      <xenc:EncryptedData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
        ...
      </xenc:EncryptedData>
    </soap:Body>
  </soap:Envelope>
```

4) response from ultimate service

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
    <soap:Header>
    </soap:Header>
    <soap:Body>
      <StockQuotes xmlns="http://temp.uri">
        <StockQuote>
        ...
        </StockQuote>
        ...
      </StockQuotes>
    </soap:Body>
  </soap:Envelope>
```

Example 17.35. Full WS-Trust Programmatic Sample

Table for sample results

```
create table WS_S_5 (
        ID      varchar primary key,
        LINK    varchar,
        TITLE   varchar,
        AUTHOR  varchar,
        ISSUED  datetime,
        CONTENT varchar
)
```

```
;
```

User used for UsernameToken

```
create user WS_TRUST;
USER_SET_PASSWORD ('WS_TRUST', 'TRUST_PASSWORD');
```

Endpoint user

```
create user WSE;
```

client test procedure

```
create procedure trust_client ()
{
   declare token POLICY_STRUCT;
   declare req SOAP_CLIENT_REQ;
   declare ret any;

   token := new POLICY_STRUCT ();
   req := new SOAP_CLIENT_REQ ();

   -- Issuer parameters

   token.usage := 'ReqIssue';
   token.token_type := 'X509v3';
   token.token_issuer := 'http://localhost:' || server_http_port () || '/ws_s_5ts';
   token.user_name := 'WS_TRUST';
   token.user_pass := 'TRUST_PASSWORD';
   token.debug := 0;

   -- End point parameters

   req.url := 'http://localhost:' || server_http_port () || '/ws_s_5';
   req.parameters := vector (vector ('AddEntry', 'http://weblogs.contoso.com/wse/samples/2003/07:AddEntry
                     vector (soap_box_structure ('title', 'Test title', 'author', 'Test author', 'issued'
                                                 'content', 'Test content')));

   req.soap_action := 'http://weblogs.contoso.com/wse/samples/2003/07:AddEntry';
   req.operation := 'AddEntry';

   ret := WST_CLI (req, token);  -- call the WS-Trust client.

   if (token.debug <> 0)
     return ret;

   -- Fill result to table

   insert into WS_S_5 (ID, LINK, TITLE, AUTHOR, ISSUED, CONTENT) values
                (ret[2][2][1], ret[2][4][1], ret[2][6][1], ret[2][8][1], ts (ret[2][10][1]), ret[2][12][1
}
;
```

End point virtual directory

```
VHOST_REMOVE (lpath=>'/ws_s_5');

VHOST_DEFINE (lpath=>'/ws_s_5', ppath=>'/SOAP/', soap_user=>'WSE',
              soap_opts=>vector('Namespace','http://temp.uri/',
                                'MethodInSoapAction','yes',
                                'ServiceName', 'WSSecure',
                                'CR-escape', 'no',
                                'WS-SEC','yes',
                                'WSS-Type', 0,
                                'WSS-Validate-Signature', 2,
                                'WSS-Func-Template', 'DB.DBA.SOAP_WS_TRUST_OUT_XENC_TEMPLATE'))
;
```

Issuer virtual directory

```
VHOST_REMOVE (lpath=>'/ws_s_5ts');

VHOST_DEFINE (lpath=>'/ws_s_5ts', ppath=>'/SOAP/', soap_user=>'WSE',
              soap_opts=>vector('Namespace','http://temp.uri/',
                                'MethodInSoapAction','yes',
                                'ServiceName', 'WSSecure',
                                'CR-escape', 'no',
                                'WS-SEC','yes',
                                'WSS-KEY', 'ws_s_5',
                                'WSS-Template', 'ws_s_5',
                                'WSS-Type', 0,
                                'WSS-Validate-Signature', 2,
                                'WSS-Func-Template', 'DB.DBA.SOAP_WS_TRUST_OUT_XENC_TEMPLATE'))
;

grant execute on WS.SOAP.RequestSecurityToken to WSE
;

CREATE PROCEDURE WS_S_5_XSD ()
{
  declare ses any;
  ses := string_output ();
  http ('<xsd:schema\n', ses);
  http ('    xmlns:xsd="http://www.w3.org/2001/XMLSchema"\n', ses);
  http ('    xmlns:tns="http://weblogs.contoso.com/wse/samples/2003/07"\n', ses);
  http ('    targetNamespace="http://weblogs.contoso.com/wse/samples/2003/07">\n', ses);
  http ('    <xsd:element name="AddEntry">\n', ses);
  http ('       <xsd:complexType>\n', ses);
  http ('           <xsd:sequence>\n', ses);
  http ('               <xsd:element name="entry" minOccurs="1" maxOccurs="1" type="tns:entry_t" />\n', s
  http ('           </xsd:sequence>\n', ses);
  http ('       </xsd:complexType>\n', ses);
  http ('    </xsd:element>\n', ses);
  http ('    <xsd:element name="WeblogEntry">\n', ses);
  http ('       <xsd:complexType>\n', ses);
  http ('           <xsd:sequence>\n', ses);
  http ('               <xsd:element name="WeblogEntry" minOccurs="1" maxOccurs="1" type="tns:entry_t" />
  http ('           </xsd:sequence>\n', ses);
  http ('       </xsd:complexType>\n', ses);
  http ('    </xsd:element>\n', ses);
  http ('    <xsd:element name="AddEntryResponse">\n', ses);
  http ('       <xsd:complexType>\n', ses);
  http ('           <xsd:sequence>\n', ses);
  http ('               <xsd:element name="WeblogEntry" minOccurs="1" maxOccurs="1" type="tns:entry_t" />
  http ('           </xsd:sequence>\n', ses);
  http ('       </xsd:complexType>\n', ses);
  http ('    </xsd:element>\n', ses);
  http ('    <xsd:complexType name="entry_t">\n', ses);
  http ('           <xsd:sequence>\n', ses);
  http ('               <xsd:element name="id" minOccurs="0" maxOccurs="1" type="xsd:string" />\n', ses);
  http ('               <xsd:element name="link" minOccurs="0" maxOccurs="1" type="xsd:string" />\n', ses
  http ('               <xsd:element name="title" minOccurs="0" maxOccurs="1" type="xsd:string" />\n', se
  http ('               <xsd:element name="author" minOccurs="0" maxOccurs="1" type="xsd:string" />\n', s
  http ('               <xsd:element name="issued" minOccurs="0" maxOccurs="1" type="xsd:dateTime" />\n',
  http ('               <xsd:element name="content" minOccurs="0" maxOccurs="1" type="xsd:string" />\n',
  http ('           </xsd:sequence>\n', ses);
  http ('    </xsd:complexType>\n', ses);
  http ('</xsd:schema>\n', ses);
  return string_output_string (ses);
}
;
```

XSD used from end point

```
SOAP_LOAD_SCH (WS_S_5_XSD ())
;
```

End point procedure

```
create procedure WS.SOAP.AddEntry
(
in  AddEntry any := null __soap_type 'http://weblogs.contoso.com/wse/samples/2003/07:AddEntry',
```

```
out AddEntryResponse any __soap_type 'http://weblogs.contoso.com/wse/samples/2003/07:AddEntryResponse',
inout "From" any __soap_header 'http://schemas.xmlsoap.org/ws/2004/08/addressing:From',
inout "MessageID" any __soap_header 'http://schemas.xmlsoap.org/ws/2004/08/addressing:MessageID',
  out "Timestamp" any __soap_header 'http://schemas.xmlsoap.org/ws/2002/07/utility:Timestamp',
inout "To" any __soap_header 'http://schemas.xmlsoap.org/ws/2004/08/addressing:To'
) __soap_doc '__VOID__'

{
   declare ret any;
   declare param any;
   declare wsa_from, wsu_time, created, expr, m_id, a_to, headers soap_parameter;
   declare in_title, in_author, in_content, out_id, out_link any;

   in_title := get_keyword ('title', AddEntry[0], '');
   in_author := get_keyword ('author', AddEntry[0], '');
   in_content := get_keyword ('content', AddEntry[0], '');

   out_id := lower (uuid ());
   out_link := sys_connected_server_address () || '/ws-trust/sample?' || out_id;

   wsa_from := new soap_parameter ();
   wsa_from.set_xsd ('http://schemas.xmlsoap.org/ws/2004/08/addressing:From');
   wsa_from.add_member ('Address', 'http://' || sys_connected_server_address () || '/WSE');
   wsa_from.set_attribute ('Id', 'Id-' || uuid());

   created := new soap_parameter (dt_set_tz (now (), 0));
   created.set_xsd ('http://schemas.xmlsoap.org/ws/2002/07/utility:Created');
   created.set_attribute ('Id', 'Id-' || uuid());

   expr := new soap_parameter (dt_set_tz (dateadd ('minute', 500, now ()), 0));
   expr.set_xsd ('http://schemas.xmlsoap.org/ws/2002/07/utility:Expires');
   expr.set_attribute ('Id', 'Id-' || uuid());

   wsu_time := new soap_parameter ();
   wsu_time.set_xsd ('http://schemas.xmlsoap.org/ws/2002/07/utility:Timestamp');
   wsu_time.add_member ('Created', created);
   wsu_time.add_member ('Expires', expr);

   m_id := new soap_parameter (lower ('UUID:'||uuid ()));
   m_id.set_xsd ('http://schemas.xmlsoap.org/ws/2004/08/addressing:MessageID');
   m_id.set_attribute ('Id', 'Id-' || uuid());

   a_to := new soap_parameter ('http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous');
   a_to.set_xsd ('http://schemas.xmlsoap.org/ws/2004/08/addressing:To');
   a_to.set_attribute ('Id', 'Id-' || uuid());

   param :=  (vector ('WeblogEntry', 'http://weblogs.contoso.com/wse/samples/2003/07:AddEntry'),
              vector (soap_box_structure ('id', out_id,
                                          'link', out_link,
                                          'title', in_title,
                                          'author', in_author,
                                          'issued', now (),
                                          'content', in_content)));

    AddEntryResponse := param;
    "From" := wsa_from.s;
    "MessageID" := m_id.s;
    "Timestamp" := wsu_time.s;
    "To" := a_to.s;
}
;

grant execute on WS.SOAP.AddEntry to WSE;
```

Server enc. template. Can be over-ridden

```
create procedure
DB.DBA.SOAP_WS_TRUST_OUT_XENC_TEMPLATE (in body varchar)
{
  declare tmpl varchar;

  tmpl := sprintf ('<?xml version="1.0" encoding="UTF-8"?>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#" >
```

```
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
  </SignedInfo>
  <SignatureValue></SignatureValue>
  <KeyInfo>
    <KeyName>ws_s_5</KeyName>
  </KeyInfo>
</Signature>');

  return dsig_template_ext (body, tmpl,
      'http://schemas.xmlsoap.org/soap/envelope/', 'Body',
      'http://schemas.xmlsoap.org/ws/2004/08/addressing', 'MessageID',
      'http://schemas.xmlsoap.org/ws/2004/08/addressing', 'From',
      'http://schemas.xmlsoap.org/ws/2004/08/addressing', 'RelatesTo',
      'http://schemas.xmlsoap.org/ws/2004/08/addressing', 'To',
      'http://schemas.xmlsoap.org/ws/2002/07/utility', 'Expires',
      'http://schemas.xmlsoap.org/ws/2002/07/utility', 'Created'
      );

}
;
```

Certificate from server

```
create procedure cert ()
{
    return uudecode (
   'MIIKIQIBAzCCCecGCSqGSIb3DQEHAaCCCdgEggnUMIIJ0DCCBs8GCSqGSIb3' ||
   'DQEHBqCCBsAwgga8AgEAMIIGtQYJKoZIhvcNAQcBMBwGCiqGSIb3DQEMAQYw' ||
   'DgQIYR9Q5x78Es4CAggAgIIGiHRAz7QEEk6jrI3un28yD7YOO3G+Sm33abCa' ||
   'jCwA3x5lT4ShZxaRrIB5Xaykr4gfTWwa3+/eFFwqaHdae9XNAjsOCvWYftFU' ||
   'mRpxwJcuY0C1yOlMxG2SyLSJNDEGY8p/uY9Okw5e5iQuzMEvDxaU+j2PSum+' ||
   'QWg94obEAJkwmCqelMwKH7aVGlFNtkphGbrl8egJzfJUCIqC6vsMYA6KSurN' ||
   'Nv05Vk2/w9Av7q2DrkSfqNMOgYluZ+OKzbTnSq2kg42F/Qd9qJye3iUusi1j' ||
   'bcIqZBCFddIFNUR+Yxa/GWD720DngBquiagqaO5Tm0vvORk/hhLx3x4cJRra' ||
   '4CFHswtVSq8JHDgyF6goMifHPsv9HTnK5r3MzQFQVITS/26NCcoj3vf9G/ka' ||
   'fRZZCAtD14lRYvENoDBFZfjUfrbHTT7VrcXbDfhYuXopUMa/Zr6fJM8ELNgE' ||
   'QmAttT4+fEnL9tNaY3VRQVkxCAl+2dvZsOqNDOh8RqaeeEumPgNUKtGr6ppW' ||
   'DXIOAg3L8r/0CwDEQArNh1HZ+SQ5leUyswsnkDG9PY3LGdqYCJJDnhoxeDla' ||
   'hqlYmqjytyfkL96768CU5wL9eck+jKNySy3foDNKu0yVZVSvO4BP38OE+hzK' ||
   '4QrmFdSztousIgTw6fe73FmLgHMjrMTlp3OFXG0krH7AZvaxYvi0Xy6+g2zJ' ||
   'xOttT9O0kNYAt7tVk15n4/tkjlF/meS4Dhu8TnHTjTMX+kljYlNTsEewzn5r' ||
   'NfXQY0RMZa/zw8lS2G/vfT71UyCACPl/SYxkSYUht8kvZCc4L3Z0460IszpC' ||
   '+nQ9YFDLQqYX7VToVyKoGQWEfHN4z8FFoYHXY/e2NNacfZkBwhq7wfh4upWG' ||
   'kjHnDE2LC1EHSkPcdmeZoPZcXXve5/WZyPQEM3h5+rLca1F67lyD8a57nh2E' ||
   '7m916TO64V4mIfxjFwxZO+LF/MzRJDXyUlGWiHV2w363TIbgc6vD1/sed0yP' ||
   'xg6mTpFTkThj7mMcDFh5jO7p7JXeJU8v/uls7pb/HbfGcsSfXEHQcHSLqwM/' ||
   'kWk6KQRxvj+9wl7zglyrCU5ty3/0i5SOb4BL4DMtGeaLXgbhSczzA26kmhSN' ||
   'C9wuB535TE9X/msXxjKqJclRC/nQicsIJEpoilwKKh0lt39J5mQwpk/By7du' ||
   'qspLZzEfXhcQlrNVJa6cTM14GuMMh3RqPK2AvxxVbwvSmBRxDDX4Wq+E7AsY' ||
   'onr322L3YHAS+oRIp7onKJyHv4J8M26iRSRCl11Jtt3lKcSEHtQIO1hS+BOR' ||
   '1yAXJ+AOhvufpCqbOwV12Tw+wCUXVDrRdpaGL+laoNaqC7heo6HZkWFy6SSm' ||
   'CUbKhtk6P0IE8Db0GdIF3jzLGvKreFiiBKkwFI1g4+C9j2BaPL1F4JMmoEaa' ||
   'eFrLqtd66g6/n0zSxkA43H3qqfGTQJ/YkilRvuqZ3pNN9sklR2n7ti44TSb+' ||
   'LZofLerppJxgcJgT67wD7Mt58pekjnOKW2HwPt8hegrQh6juBHaFxn/BIZuh' ||
   'VivCCsfY2V/sZBl/uL9qvevnoQXKrvOks0XESRTpqc3PptgQdFTkUST3vc6o' ||
   'CtrLSyK6rLNVI5bP2QRuCQAPyhI9u6s6AC1uot9T/BooOLowzzpNLioWstsB' ||
   'Td9+64Ei1bvcmIZZ2Gq3p/gAXYnkw/VciQ/YET54nP95wUYSrbB8OLXJHPX6' ||
   'zaLryqbpPIcNSvGjneSf84a0NkMFkdq5H4m0lJQIJPIvi7qhGxpNGYEuaqgv' ||
   'NwGmhWKK4noHLuXIMOv5Cn10MHTaR7CVxOLX950RzitmIQ9xa7Qu2Ey+wzRM' ||
   'LvoxUf1+GMUCGyuVhQlCRmfCK7ts53WTCLywNsJcueImaLTjXOOoJNg1Baov' ||
   'C+RYwAvigUtp1aBY9XZRHMqHytLooGhPG/xgX1Mhe+1452YSutxIww+psC5E' ||
   '9LAkBMZ7mz9o6JJnk3IvJ+WhAZ+hV876T7yABTifxctfkOmNu3H/RcpDV4uk' ||
   'TZizoDttm3/Mj99V9U+elt/1YreXvB5kJ63o9nOeN3gBu8mEBhqGLGOWuibL' ||
   'RANKQ1es3jVGk5SMS0bi8BeG6nGw59xna1BZcpS3KnbgWdU4ek7mz+OO0fHe' ||
   'tQPGQ1pI0FA/UTBEoRUokZPjGlELL9su7bcAbgpTTS0vncGzUwO5yxRExFh7' ||
   'PJPVMmjrOphChDvBlgUESq9J9CmEUswp+IEwggL5BgkqhkiG9w0BBwGgggLq' ||
   'BIIC5jCCAuIwggLeBgsqhkiG9w0BDAoBAqCCAqYwggKiMBwGCiqGSIb3DQEM' ||
   'AQMwDgQIBnHBzK4ZZwwCAggABIICgO8D5hIqZZLOZmVMCWdTayS0joeE1W6H' ||
   '7J/IiiP3N5EQeALNvVaoI6EeNuap3W8lj89moUzCuScokct7jRaLOhjeOeRa' ||
   'osMRMOvdbSSIFS+QN/CT1mQ46+LeNuFocCW0M0RsFVgcSPdWuJUJzOq9qx7J' ||
   'XjkG8UHfwpjy1o9JZAqtjde+fNFHiuPLYI3oJBwNGfbe1QJlrVjf+MAziu6J' ||
```

```
    'iGt+QBNfWWLoFgDZegHWLcfwwXkmrzfM/4KIGEjX2DZhBrf5M5r+P6ZDJFFs' ||
    'NNNmUUjVvtz+PQIlVWrBJxh5r0Yyr/n37g2pEGKcq5PNxP+DZ1H/UCEObUzk' ||
    'H8afcU7uUq43t0Eyq4cs8VX7pytIoUgvMT5bcs0aU8gs9b3c33BjRv7uTB7q' ||
    'qTGaAQ+b4t5vAR/MVoHfVA1Sgq0D8mzJ8NtD6IMdbjsW0cSxwZM/pgPDmSI9' ||
    'AKi6t9E/UrzxwaJWBmEgy2Qup5n6VrxzWZ+TiAKAH4/Ma3kIUkYtgvrAH9Tf' ||
    'qY/7ZOHIVF93aEEcIshYYVyUAHsJVa1r7LXkfcm7ogxDi8vjmvtDZhxo7+i8' ||
    'TmrsO19FoDSGUNJlYFvPsGpOpnrw/VT7M9VEhF9nSznRRlDD+xidZdWf2GDe' ||
    'MxLg+7dLMkKqYgQbWKRO6y6ATJbSL+0wBRml1h5hvIhK+PsJeDHcVf3rl5my' ||
    'NZgBlFkHau9/2WohA428dwKDgFVFjgt8WfsweOW6QCYL5ezjtORDRZHg3YQL' ||
    'ZrB7jSJkx9WFq5O81YT5YqVvcDow7aoPpKJvZtFUkPPtgMTyIz6zOTCC9sTe' ||
    'lHu6m/Olizb3o/uOlxlcK3727SHSiBV8+4rhgIstIlYxJTAjBgkqhkiG9w0B' ||
    'CRUxFgQUjYbSw3MD4nRuny8vVKz5hZtCftwwMTAhMAkGBSsOAwIaBQAEFLRv' ||
    'tU3dr9bQEbcm2mcYE+KK33n3BAh/OvyukQvZpAICCAA=', 2);
}
;

create procedure server_pub_x509_key ()
{
  return
    'MIICxzCCAjCgAwIBAgIBADANBgkqhkiG9w0BAQQFADBSMQswCQYDVQQGEwJCRzEQMA4GA1UE' ||
    'CBMHUGxvdmRpdjEQMA4GA1UEBxMHUGxvdmRpdjEfMB0GA1UEChMWT3BlbkxpbmsgU29mdHdh' ||
    'cmUgTHRkLjAeFw0wNDAxMjExNDA4MzhaFw0wNTAxMjAxNDA4MzhaMFIxCzAJBgNVBAYTAkJH' ||
    'MRAwDgYDVQQIEwdQbG92ZGl2MRAwDgYDVQQHEwdQbG92ZGl2MR8wHQYDVQQKExZPcGVuTGlu' ||
    'ayBTb2Z0d2FyZSBMdGQuMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDp4LEkZOl/Nbve' ||
    'sKUYbJkYS615oB0nPbu3n0dCCC37xswbluBQcS+P/zHdvQZaWzWsluGpGctHzTYcD7+UkiLJ' ||
    'Xrd+PddqkgfogqaW7/9jB2CJSA1paoJTqX6b06/KOi4Jj1WYHwkGOfiD+WybUWcX65gtaM52' ||
    'OUoenVOy7v5zrwIDAQABo4GsMIGpMB0GA1UdDgQWBBTrS3v9pmTo/jCtrd9+7FBESXGVHDB6' ||
    'BgNVHSMEczBxgBTrS3v9pmTo/jCtrd9+7FBESXGVHKFWpFQwUjELMAkGA1UEBhMCQkcxEDAO' ||
    'BgNVBAgTB1Bsb3ZkaXYxEDAOBgNVBAcTB1Bsb3ZkaXYxHzAdBgNVBAoTFk9wZW5MaW5rIFNv' ||
    'ZnR3YXJlIEx0ZC6CAQAwDAYDVR0TBAUwAwEB/zANBgkqhkiG9w0BAQQFAAOBgQCCzqtd0ej6' ||
    'f5NSORqyLlJ90L1FPAiF1lg+dFSatMpxbv6zPTK9qnHp3VWK0cPwK1GxxC3B2QyuhCIkeRs7' ||
    'qymH8S6W9maUMIvLD1dDQFxKStgxJe0IDEIG9CygaDGsTpkPwq/qPqhRGamGeLO9GU8wPnUN' ||
    'OleyHzY8Y4ZkCznSFQ==';
}
;
```

Fill server public key to table.

```
insert soft WST_SERVER_ISSUER_TOKENS (WSK_TOKEN_TYPE, WSK_REQUEST_TYPE, WSK_APPLIES_TO, WSK_FROM,
                                      WSK_SERVICE_NAME, WSK_PORT_TYPE, WSK_TOKEN) values
                ('wsse:X509v3', 'wsse:ReqIssue', NULL,
                 'http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous', NULL, NULL,
                  server_pub_x509_key ())
;
```

Upload the server certificate

```
USER_KEY_LOAD ('ws_s_5', cert(), 'X.509', 'PKCS12', 'ws_s_5', null, 1)
;

reconnect WSE
;
```

Upload the client certificate

```
USER_KEY_LOAD ('ws_s_5', cert(), 'X.509', 'PKCS12', 'ws_s_5', null, 1)
;

checkpoint
;
```

17.9. XML for Analysis Provider

XML for Analysis (XMLA) is a SOAP based XML API for data access interaction between a client application and a data provider working over the Web. The Virtuoso SOAP server can act as an XMLA data provider, also the Virtuoso SOAP client can be used as a client to talk with an XMLA data provider.

The following terms: result-set and row-set will be used interchangeably to designate repeating content with identical structure. The mention of a result-set and row-set in this document is related to an SQL/ODBC result set which is represented as XML for use in XMLA.

The implementation is based on the "XML for Analysis Specification v 1.0" by Microsoft Corp, found on the MSDN website.

The XMLA protocol consists of two SOAP operations: "Discover" and "Execute" which are described below:

The *Discover* operation is used to retrieve settings and database object information. It accepts the following input parameters:

RequestType - enumerated string.
Restrictions - a structure of restrictions to be applied to the processing of request, the type values may vary, depending on the request.
Properties - properties to the request, such as data source name, initial catalog etc.
Upon success the "Discover" operation will return a specific rowset. The 'specific' means that its structure depends of type of request.

The XMLA data provider supports the following discovery rowsets:

`DISCOVER_DATASOURCES` - to discover available data sources
`DISCOVER_PROPERTIES` - the properties which can be used/set
`DISCOVER_SCHEMA_ROWSETS` - this list itself with allowed restrictions
`DISCOVER_ENUMERATORS` - enumeration values supported by provider
`DISCOVER_KEYWORDS` - reserved words
`DISCOVER_LITERALS` - restrictions on database object names etc.
`DBSCHEMA_CATALOGS` - list of catalogs
`DBSCHEMA_TABLES` - list of tables
`DBSCHEMA_TABLES_INFO` - list of info about tables
`DBSCHEMA_COLUMNS` - list of columns
`DBSCHEMA_PROVIDER_TYPES` - list of datatypes
Here is an example of a response message to the Discover invocation:

```
----
<SOAP:Envelope SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP:Body>
    <cli:DiscoverResponse xmlns:cli="urn:schemas-microsoft-com:xml-analysis">
      <Result xmlns="">
        <root xmlns="urn:schemas-microsoft-com:xml-analysis:rowset">
          <n0:schema xmlns:n0="http://www.w3.org/2001/XMLSchema"
                     targetNamespace="urn:schemas-microsoft-com:xml-analysis:rowset"
                     elementFormDefault="qualified">
            <n0:element name="root" type="n2:root" xmlns:n2="urn:schemas-microsoft-com:xml-analysis:rowse
            ... more declarations ...
          </n0:schema>
          <row xmlns="urn:schemas-microsoft-com:xml-analysis:rowset">
            ... rowset data ...
          </row>
        </root>
      </Result>
    </cli:DiscoverResponse>
  </SOAP:Body>
</SOAP:Envelope>
  ----
```

The client can use information retrieved with the "Discover" operation for automatically formulating queries etc.

The *Execute* operation is used to process a SQL statement on the server and to return resultset or/and schema description for the resultset. The current implementation supports only SQL statements. It accepts the following input parameters:

Command - a structure with single element - "Statement" , which contains the SQL statement.

Properties - properties to the execution process.
On success the "Execute" operation will return the rowset and/or XSD for the rowset.

```
----
<SOAP:Envelope SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP:Body>
    <cli:ExecuteResponse xmlns:cli="urn:schemas-microsoft-com:xml-analysis">
      <Result xmlns="">
        <root xmlns="urn:schemas-microsoft-com:xml-analysis:rowset">
          <n0:schema xmlns:n0="http://www.w3.org/2001/XMLSchema"
                     targetNamespace="urn:schemas-microsoft-com:xml-analysis:rowset"
                     elementFormDefault="qualified">
            <n0:element name="root" type="n2:root" xmlns:n2="urn:schemas-microsoft-com:xml-analysis:rowse
            ... more XSD declarations ...
          </n0:schema>
          <row xmlns="urn:schemas-microsoft-com:xml-analysis:rowset">
            ... rowset data ...
          </row>
          more <row> elements
        </root>
      </Result>
    </cli:ExecuteResponse>
  </SOAP:Body>
</SOAP:Envelope>
 ----
```

The XML representation of the result sets can have three forms: Tabular, Multidimensional and Custom. As the Virtuoso server is a tabular data provider itself, hence the current XMLA implementation supports only Tabular representation.

The Tabular representation of the result set consist of two main parts:

schema - XML Schema definition of result

data - sequence of repeating elements, each consisting of elements representing the cells in the rowset

Security . The Execute operation needs the properties Username and Password to be supplied in order to execute the statement on behalf of an SQL user account. If these are not supplied the request will be rejected with SOAP:Fault message. To prevent a network sniffer from catching the password in clear text, it is strongly recommended that sending of these properties be done via HTTPS (SSL/TLS) connection (for HTTPS setup see Web server section, and tutorials).

State Support . The Virtuoso XMLA provider implements statelessness for the settable properties. This means that properties such as UserName and Password can be set in the beginning and they will be restored on server side on the next request ID for which they are not supplied. This mechanism is very similar to URL poisoning state support, which is described in the Web server section of the documentation. In short, the XMLA client asks to begin a session, the XMLA provider returns a session ID and from that point the client sends this ID to the server. If the client wishes, it may cancel the session with an end session request.

The above requests are SOAP messages carried in the SOAP Header element. Please note that these headers can be sent together with Discover or Execute operations.

Here is a a simple session:

```
- client requests a session
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
                   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Header>
  <xmla:BeginSession xmlns:xmla="urn:schemas-microsoft-com:xml-analysis" mustUnderstand="1"/>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
   <!-- Discover or Execute call -->

</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

- server returns the SessionID
<SOAP-ENV:Header>
  <xmla:Session xmlns:xmla="urn:schemas-microsoft-com:xml-analysis" mustUnderstand="1" SessionID="NNNNNN"
</SOAP-ENV:Header>

- sequential requests of client contains:
```

```
<SOAP-ENV:Header>
  <xmla:Session xmlns:xmla="urn:schemas-microsoft-com:xml-analysis" mustUnderstand="1" SessionID="NNNNNN"
</SOAP-ENV:Header>

- and finally client cancel the session:
<SOAP-ENV:Header>
  <xmla:EndSession xmlns:xmla="urn:schemas-microsoft-com:xml-analysis" mustUnderstand="1" SessionID="NNNN
</SOAP-ENV:Header>
```

Namespaces

The request and response are in the "urn:schemas-microsoft-com:xml-analysis" namespace. Also SOAPAction header field must be "urn:schemas-microsoft-com:xml-analysis:Discover" and "urn:schemas-microsoft-com:xml-analysis:Execute" for Discover and Execute operations.

Virtuoso extensions

The following settable properties are available to the execute request:

direction - one of forward, backward

skip - integer

n-rows integer

bookmark-from - string

retrieve-row-count - string

The direction indicates in which direction the cursor moves from the bookmark-from. If there is no bookmark-from, the query is always forward from the start of the evaluation.

skip allows skipping a number of rows in the selected direction. n-rows gives the number of rows to return after the skip. Fewer rows will be returned if the evaluation does not extend this far.

The bookmark is an opaque string, acceptable values are values that have been in a bookmark element of a result row.

return-bookmark boolean

If true, a bookmark element is added to each row of the result set.

A simple request / response is added to samples.

The retrieve-row-count element of the query options must be true for rowcount returning to be enabled.

If retrieve-row-count is specified, the response has a row-count element with a text value containing the total result set size of the query, irrespective of the specified window of rows retrieved. Note that for large result sets getting the row count can be prohibitively expensive.

Virtual directory Setup

To make a virtual directory work as an XMLA data provider it is enough to grant execute permissions on Discover and Execute procedures to the SQL account specified for SOAP execution in this virtual directory.

For compliance with XMLA, the virtual directory attribute elementFormDefault must have the value "qualified" in the XMLA SOAP end point.

Here are the steps for setting up a virtual directory for use with XMLA:

```
create user "XMLA";

user_set_qualifier ('XMLA', 'XMLA');
```

```
VHOST_REMOVE (lpath=>'/XMLA');
VHOST_DEFINE (lpath=>'/XMLA', ppath=>'/SOAP/', soap_user => 'XMLA',
              soap_opts => vector ('ServiceName', 'XMLAnalysis', 'elementFormDefault', 'qualified'));

grant execute on DB.."Discover" to "XMLA";
grant execute on DB.."Execute" to "XMLA";
```

Example 17.36. Simple Discovery Request/Response examples

The following example shows a simple request for discovering data sources and response from server.

```
<?xml version="1.0"?>
<SOAP:Envelope SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP:Body>
    <cli:Discover xmlns:cli="urn:schemas-microsoft-com:xml-analysis">
      <RequestType xsi:type="xsd:string" dt:dt="string">DISCOVER_DATASOURCES</RequestType>
      <Restrictions xsi:nil="1"/>
      <Properties xsi:nil="1" />
      </Properties>
    </cli:Discover>
  </SOAP:Body>
</SOAP:Envelope>

<SOAP:Envelope SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP:Body>
    <cli:DiscoverResponse xmlns:cli="urn:schemas-microsoft-com:xml-analysis">
      <Result xmlns="">
        <root xmlns="urn:schemas-microsoft-com:xml-analysis:rowset">
          <n0:schema xmlns:n0="http://www.w3.org/2001/XMLSchema"
                     targetNamespace="urn:schemas-microsoft-com:xml-analysis:rowset"
                     elementFormDefault="qualified">
            <n0:element name="root" type="n2:root" xmlns:n2="urn:schemas-microsoft-com:xml-analysis:rowse
            <n0:complexType name="root">
              <n0:sequence minOccurs="0" maxOccurs="unbounded">
                <n0:element name="row" type="n2:row" xmlns:n2="urn:schemas-microsoft-com:xml-analysis:row
              </n0:sequence>
            </n0:complexType>
            <n0:complexType name="row">
              <n0:choice maxOccurs="unbounded" minOccurs="0">
                <n0:element name="DataSourceName" type="string" xmlns:n2="urn:schemas-microsoft-com:xml-s
                <n0:element name="DataSourceDescription" type="string" xmlns:n2="urn:schemas-microsoft-co
                <n0:element name="URL" type="string" xmlns:n2="urn:schemas-microsoft-com:xml-sql" n2:fiel
                <n0:element name="DataSourceInfo" type="string" xmlns:n2="urn:schemas-microsoft-com:xml-s
                <n0:element name="ProviderName" type="string" xmlns:n2="urn:schemas-microsoft-com:xml-sql
                <n0:element name="ProviderType" xmlns:n2="urn:schemas-microsoft-com:xml-sql" n2:field="Pr
                  <n0:complexType>
                    <n0:sequence minOccurs="0" maxOccurs="unbounded">
                      <n0:any processContents="lax" maxOccurs="unbounded"/>
                    </n0:sequence>
                  </n0:complexType>
                </n0:element>
                <n0:element name="AuthenticationMode" type="string" xmlns:n2="urn:schemas-microsoft-com:x
              </n0:choice>
            </n0:complexType>
          </n0:schema>
          <row xmlns="urn:schemas-microsoft-com:xml-analysis:rowset">
            <DataSourceName>Local Server</DataSourceName>
            <DataSourceDescription>Virtuoso Server</DataSourceDescription>
            <URL>http://example.com/XMLA</URL>
            <DataSourceInfo>DSN=__local</DataSourceInfo>
            <ProviderName>Virtuoso XML for Analysis</ProviderName>
            <ProviderType>
              <TDP xsi:type="xsd:string" dt:dt="string"/>
            </ProviderType>
            <AuthenticationMode>Unauthenticated</AuthenticationMode>
          </row>
        </root>
      </Result>
    </cli:DiscoverResponse>
  </SOAP:Body>
</SOAP:Envelope>
```

The following example shows a query against the Northwind's database's Customers table

Request/Response

```
<?xml version="1.0"?>
<SOAP:Envelope SOAP:encodingType="http://schemas.xmlsoap.org/soap/encoding/" SOAP:encodingStyle="http://s
  <SOAP:Body>
    <cli:Execute xmlns:cli="urn:schemas-microsoft-com:xml-analysis">
      <Command>
        <Statement xsi:type="xsd:string" dt:dt="string">select CustomerID , CompanyName from Demo..Custom
      </Command>
      <Restrictions xsi:nil="1"/>
      <Properties>
        <PropertyList>
          <DataSourceInfo xsi:type="xsd:string" dt:dt="string">DSN=__local</DataSourceInfo>
          <Password type="http://www.w3.org/2001/XMLSchema:string" dt="string" >demo</Password>
          <UserName type="http://www.w3.org/2001/XMLSchema:string" dt="string" >demo</UserName>
        </PropertyList>
      </Properties>
    </cli:Execute>
  </SOAP:Body>
</SOAP:Envelope>

<SOAP:Envelope SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP="http://schemas.
  <SOAP:Body>
    <cli:ExecuteResponse xmlns:cli="urn:schemas-microsoft-com:xml-analysis">
      <Result xmlns="">
        <root xmlns="urn:schemas-microsoft-com:xml-analysis:rowset">
          <n0:schema xmlns:n0="http://www.w3.org/2001/XMLSchema" targetNamespace="urn:schemas-microsoft-c
            <n0:element name="root" type="n2:root" xmlns:n2="urn:schemas-microsoft-com:xml-analysis:rowse
            <n0:complexType name="root">
              <n0:sequence minOccurs="0" maxOccurs="unbounded">
                <n0:element name="row" type="n2:row" xmlns:n2="urn:schemas-microsoft-com:xml-analysis:row
              </n0:sequence>
            </n0:complexType>
            <n0:complexType name="row">
              <n0:choice maxOccurs="unbounded" minOccurs="0">
                <n0:element name="CustomerID" type="string" xmlns:n2="urn:schemas-microsoft-com:xml-sql"
                <n0:element name="CompanyName" type="string" xmlns:n2="urn:schemas-microsoft-com:xml-sql"
              </n0:choice>
            </n0:complexType>
          </n0:schema>
          <row xmlns="urn:schemas-microsoft-com:xml-analysis:rowset">
            <CustomerID>ALFKI</CustomerID>
            <CompanyName>Alfreds Futterkiste</CompanyName>
          </row>
          <row xmlns="urn:schemas-microsoft-com:xml-analysis:rowset">
            <CustomerID>ANATR</CustomerID>
            <CompanyName>Ana Trujillo Emparedados y helados</CompanyName>
          </row>
          <row xmlns="urn:schemas-microsoft-com:xml-analysis:rowset">
            <CustomerID>ANTON</CustomerID>
            <CompanyName>Antonio Moreno Taquería</CompanyName>
          </row>
          <row xmlns="urn:schemas-microsoft-com:xml-analysis:rowset">
            <CustomerID>AROUT</CustomerID>
            <CompanyName>Around the Horn</CompanyName>
          </row>
        </root>
      </Result>
    </cli:ExecuteResponse>
  </SOAP:Body>
</SOAP:Envelope>
```

The following example shows a simple request for "Execute" operation whit Virtuoso extension. Sources and response from server.

```
  <Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/" xmlns:n1="http://schemas.xmlsoap.org/soap/e
    <Body xmlns="http://schemas.xmlsoap.org/soap/envelope/">
      <Execute xmlns="urn:schemas-microsoft-com:xml-analysis">
        <Command>
          <Statement xmlns:n1="http://www.w3.org/2001/XMLSchema-instance" xmlns:n2="urn:schemas-microsoft
```

```
      </Command>
      <Properties>
        <PropertyList>
          <DataSourceInfo xmlns:n1="http://www.w3.org/2001/XMLSchema-instance" xmlns:n2="urn:schemas-mi
          <UserName xmlns:n1="http://www.w3.org/2001/XMLSchema-instance" xmlns:n2="urn:schemas-microsof
          <Password xmlns:n1="http://www.w3.org/2001/XMLSchema-instance" xmlns:n2="urn:schemas-microsof
          <direction xmlns:n1="http://www.w3.org/2001/XMLSchema-instance" xmlns:n2="urn:schemas-microso
          <skip xmlns:n1="http://www.w3.org/2001/XMLSchema-instance" xmlns:n2="urn:schemas-microsoft-co
          <n-rows xmlns:n1="http://www.w3.org/2001/XMLSchema-instance" xmlns:n2="urn:schemas-microsoft-
          <return-bookmark xmlns:n1="http://www.w3.org/2001/XMLSchema-instance" xmlns:n2="urn:schemas-m
        </PropertyList>
      </Properties>
    </Execute>
  </Body>
</Envelope>

<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/" xmlns:n1="http://schemas.xmlsoap.org/soap/e
  <Body xmlns="http://schemas.xmlsoap.org/soap/envelope/">
    <ExecuteResponse xmlns="urn:schemas-microsoft-com:xml-analysis">
      <return xmlns="urn:schemas-microsoft-com:xml-analysis">
        <root xmlns="urn:schemas-microsoft-com:xml-analysis:rowset">
          <schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="urn:schemas-microsoft-com:x
            <element xmlns="http://www.w3.org/2001/XMLSchema" name="root" type="urn:schemas-microsoft-c
            <complexType xmlns="http://www.w3.org/2001/XMLSchema" name="root">
              <sequence xmlns="http://www.w3.org/2001/XMLSchema" minOccurs="0" maxOccurs="unbounded">
                <element xmlns="http://www.w3.org/2001/XMLSchema" name="row" type="urn:schemas-microsof
              </sequence>
            </complexType>
            <complexType xmlns="http://www.w3.org/2001/XMLSchema" name="row">
              <choice xmlns="http://www.w3.org/2001/XMLSchema" maxOccurs="unbounded" minOccurs="0">
                <element xmlns="http://www.w3.org/2001/XMLSchema" xmlns:n3="urn:schemas-microsoft-com:x
                <element xmlns="http://www.w3.org/2001/XMLSchema" xmlns:n3="urn:schemas-microsoft-com:x
                <element xmlns="http://www.w3.org/2001/XMLSchema" xmlns:n3="urn:schemas-microsoft-com:x
              </choice>
            </complexType>
          </schema>
          <row xmlns="urn:schemas-microsoft-com:xml-analysis:rowset">
            <BOOKMARK xmlns="urn:schemas-microsoft-com:xml-analysis:rowset">wbwCtQExvAI=</BOOKMARK>
            <CustomerID xmlns="urn:schemas-microsoft-com:xml-analysis:rowset">AROUT</CustomerID>
            <CompanyName xmlns="urn:schemas-microsoft-com:xml-analysis:rowset">Around the Horn</Company
          </row>
          <row xmlns="urn:schemas-microsoft-com:xml-analysis:rowset">
            <BOOKMARK xmlns="urn:schemas-microsoft-com:xml-analysis:rowset">wbwCtQExvAE=</BOOKMARK>
            <CustomerID xmlns="urn:schemas-microsoft-com:xml-analysis:rowset">ANTON</CustomerID>
            <CompanyName xmlns="urn:schemas-microsoft-com:xml-analysis:rowset">Antonio Moreno Taquera</
          </row>
          <row xmlns="urn:schemas-microsoft-com:xml-analysis:rowset">
            <BOOKMARK xmlns="urn:schemas-microsoft-com:xml-analysis:rowset">wbwCtQExvAA=</BOOKMARK>
            <CustomerID xmlns="urn:schemas-microsoft-com:xml-analysis:rowset">ANATR</CustomerID>
            <CompanyName xmlns="urn:schemas-microsoft-com:xml-analysis:rowset">Ana Trujillo Emparedados
          </row>
        </root>
      </return>
    </ExecuteResponse>
  </Body>
</Envelope>
```

17.10. XML-RPC support

The XML-RPC is a remote procedure calling system via HTTP using XML as the encoding. It is very much like the SOAP protocol, but the data encoding rules are different. XML-RPC supports fewer data types than SOAP. The data is self-describing and position bound.

The Virtuoso SOAP server can process XML-RPC requests using the XML-RPC to SOAP bridge. This is done with two filters : input and output filter. The input filter transforms XML-RPC into a SOAP PRC encoded message. Then the transformed message is passed to the SOAP server for processing. The response from the SOAP server will be re-coded into XML-RPC format in the output filter. The combination of these filters constitutes the bridge.

It is important to remember that XML-RPC defines two complex types : array and structure. These two types are represented by vector () and soap-structure respectively, when passing the data to the PL procedure in question.

Example 17.37. XML-RPC Procedure Definition

An example of a PL procedure representing an XML-RPC method "echoXRtypes" is:

```
create procedure echoXRtypes (
                                in  inInteger   integer,
                                in  inString    varchar,
                                in  inDate      datetime,
                                in  inDouble    double precision,
                                in  inBoolean   smallint,
                                in  inHex       varchar,
                                out outInteger  integer,
                                out outString   varchar,
                                out outDate     datetime,
                                out outDouble   double precision,
                                out outBoolean  smallint,
                                out outHex      varchar __soap_type 'http://www.w3.org/2001/XMLSchema:bas
                            )
{
    outInteger  := inInteger    ;
    outString   := inString     ;
    outDate     := inDate       ;
    outDouble   := inDouble     ;
    outBoolean  := soap_boolean (inBoolean);
    outHex      := inHex        ;
}
;
```

Note that the definition of such procedures does not differ from those of SOAP procedures.

An XML-RPC request may look like:

```
<?xml version="1.0"?>
<methodCall>
<methodName>echoXRtypes</methodName>
<params>
  <param> <value><i4>42</i4></value> </param>
  <param> <value>String</value> </param>
  <param> <value><dateTime.iso8601>1998-07-17T14:08:55</dateTime.iso8601></value> </param>
  <param> <value><double>1234.567</double></value> </param>
  <param> <value><boolean>1</boolean></value> </param>
  <param> <value><base64>eW91IGNhbid0IHJlYWQgdGhpcyE=</base64></value> </param>
</params>
</methodCall>
```

The response for the above message will be :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<methodResponse>
<params>
<param> <value> <i4>42</i4> </value> </param>
<param> <value> <string>String</string> </value> </param>
<param> <value> <dateTime.iso8601>1998-07-17T14:08:55.000+03:00</dateTime.iso8601> </value> </param>
<param> <value> <double>1234.567000</double> </value> </param>
<param> <value> <boolean>1</boolean> </value> </param>
<param> <value> <base64>eW91IGNhbid0IHJlYWQgdGhpcyE=</base64> </value> </param>
</params>
</methodResponse>
```

Enabling the XML-RPC -> SOAP bridge is very simple. You make a virtual directory with physical location pointing to /SOAP/ and specify the 'XML-RPC' SOAP option as 'yes'. The following methods are available:

1. **Virtual Directories Visual Administration Interface .** From the main administration menu go to `Web Servers -> Virtual Directories` and add or configure a virtual directory. For the directory definition add a new option in

SOAP options box: " `XML-RPC=yes;` ".

2. **The vhost_define() Function .** Using the ISQL utility one can use the command:

```
SQL> vhost_define (lpath=>'/RPC2', ppath=>'/SOAP/', soap_user=>'dba',
    soap_opts=>vector ('XML-RPC', 'yes'));
```

Note:

Virtual directories configured in this way can only be used for XML-RPC calls. If you need to make SOAP requests, then another virtual directory will be required.

17.11. SyncML

SyncML is a protocol for synchronization of data collections between two devices - a SyncML Client and SyncML Server using an XML data representation. The client is typically some mobile device or mobile PC. The Virtuoso server implements the SyncML server protocol over HTTP.

The SyncML server maintains the data collections within the WebDAV repository. The items (resources) in the collections (folders) represents items found on the client, e.g. VCARD, vcalendar records. As the WebDAV repository does not restrict the type of data that can be stored, likewise there are no restrictions on the type of data that can be synchronized. Also every WebDAV virtual directory can act as a SyncML server endpoint, the SyncML processing is triggered via the `Content-Type` , detected on the POST request (see below).

See Also:

SyncML Schema Tables

The SyncML server detects the following formats from the `Content-Type` header string:

`application/vnd.syncml+wbxml` - WBXML coded XML documents

`application/vnd.syncml+xml` - plain text XML document

These are detected during the POST request to a WebDAV virtual directory thus triggering SyncML processing. When a device makes creates '`application/vnd.syncml+wbxml` ' session, the SyncML server will respond with the WBXML format. Otherwise plain XML will be used to encode SyncML messages.

Basic and MD5 digest SyncML authorization schemes are supported. The WebDAV enabled user accounts are used to perform authentication.

It is possible to query the HTTP authentication type supported by the target device by assigning an authentication hook to a given virtual directory. By default SyncML authentication is digest-md5, so HTTP authentication is off (note that HTTP authentication is different from SyncML authentication).

The following synchronization methods are supported:

Two-way sync

Slow two-way sync

Every item sent from the client device is mapped to a WebDAV resource under a given WebDAV collection. Every device database, such as contacts, calendar, usually requires that a different WebDAV collection be specified because the devices usually can not maintain items of different kinds with a single database. For example, when synchronizing the Contacts of a Nokia 9210 with the server, we can specify `./contacts/` , and likewise for the Calendar database, `./calendar/` . This keeps VCARDS (contacts) and vCalendars in separate collections (folders) on a given virtual directory. It is not possible to keep calendar and contacts (for example) in the same WebDAV collection, unless the device supports such items in a single database.

Important Note:

The target folders (Virtuoso server-side collections) must exists and must be accessible by the WebDAV account used to authenticate against the SyncML server. If credentials or permissions are insufficient then an Error 403 (Forbidden) will be returned to the client.

Example 17.38. Example of a SyncML session

The client begins initial request

```
<SyncML>
  <SyncHdr>
```

```
    <VerDTD>1.0</VerDTD>
    <VerProto>SyncML/1.0</VerProto>
    <SessionID>88</SessionID>
    <MsgID>1</MsgID>
    <Target>
      <LocURI>http://192.168.1.1:6666/</LocURI>
    </Target>
    <Source>
      <LocURI>IMEI:57471724140229</LocURI>
    </Source>
    <Meta>
      <MaxMsgSize>10000</MaxMsgSize>
    </Meta>
  </SyncHdr>
  <SyncBody>
    <Alert>
      <CmdID>1</CmdID>
      <Data>201</Data>
      <Item>
        <Target>
          <LocURI>./calendar</LocURI>
        </Target>
        <Source>
          <LocURI>./C\System\Data\Calendar</LocURI>
        </Source>
        <Meta>
          <Anchor>
            <Next>20031202T165103Z</Next>
          </Anchor>
        </Meta>
      </Item>
    </Alert>
    <Put>
      <CmdID>2</CmdID>
      <Meta>
        <Type>application/vnd.syncml-devinf+wbxml</Type>
      </Meta>
      <Item>
        <Source>
          <LocURI>./devinf10</LocURI>
        </Source>
        <Data>
          <DevInf>
            <VerDTD>1.0</VerDTD>
            <Man>NOKIA</Man>
            <Mod>9210</Mod>
            <SwV>256</SwV>
            <HwV>1.0</HwV>
            <DevID>IMEI:57471724140229</DevID>
            <DevTyp>phone</DevTyp>
            ....
          </DevInf>
        </Data>
      </Item>
    </Put>
    <Get>
      <CmdID>3</CmdID>
      <Meta>
        <Type>application/vnd.syncml-devinf+wbxml</Type>
      </Meta>
      <Item>
        <Target>
          <LocURI>./devinf10</LocURI>
        </Target>
      </Item>
    </Get>
    <Final/>
  </SyncBody>
</SyncML>
```

Server rejects credential, because no credentials.

```
<SyncML xmlns:n0="syncml:SYNCML1.0">
```

```
  <SyncHdr>
    <VerDTD>1.0</VerDTD>
    <VerProto>SyncML/1.0</VerProto>
    <SessionID>88</SessionID>
    <MsgID>1</MsgID>
    <Target>
      <LocURI>IMEI:57471724140229</LocURI>
    </Target>
    <Source>
      <LocURI>http://192.168.1.1:6666/</LocURI>
    </Source>
  </SyncHdr>
  <SyncBody>
    <Status>
      <CmdID>1</CmdID>
      <MsgRef>1</MsgRef>
      <CmdRef>0</CmdRef>
      <TargetRef>http://192.168.1.1:6666/</TargetRef>
      <SourceRef>IMEI:57471724140229</SourceRef>
      <Cmd>SyncHdr</Cmd>
      <Chal>
        <Meta>
          <Type xmlns:n2="syncml:metinf">syncml:auth-md5</Type>
          <Format xmlns:n2="syncml:metinf">b64</Format>
          <NextNonce xmlns:n2="syncml:metinf">NzcyYTgyMDRjMGM2NzRlYTZjYWVmNGY4ZjNjMGYzMDk=</NextNonce>
        </Meta>
      </Chal>
      <Data>401</Data>
    </Status>
    ....
    <Final/>
  </SyncBody>
</SyncML>
```

Client sends back credentials.

```
<SyncML>
  <SyncHdr>
    <VerDTD>1.0</VerDTD>
    <VerProto>SyncML/1.0</VerProto>
    <SessionID>88</SessionID>
    <MsgID>2</MsgID>
    <Target>
      <LocURI>http://192.168.1.1:6666/</LocURI>
    </Target>
    <Source>
      <LocURI>IMEI:57471724140229</LocURI>
      <LocName>imitko</LocName>
    </Source>
    <Cred>
      <Meta>
        <Format>b64</Format>
        <Type>syncml:auth-md5</Type>
      </Meta>
      <Data>X8uGR8CX4ogw8Ux+ZWIzkg==</Data>
    </Cred>
    <Meta>
      <MaxMsgSize>10000</MaxMsgSize>
    </Meta>
  </SyncHdr>
  <SyncBody>
    <Status>
      <CmdID>1</CmdID>
      <MsgRef>1</MsgRef>
      <CmdRef>0</CmdRef>
      <Cmd>SyncHdr</Cmd>
      <TargetRef>IMEI:57471724140229</TargetRef>
      <SourceRef>http://192.168.1.1:6666/</SourceRef>
      <Data>200</Data>
    </Status>
    <Alert>
      <CmdID>2</CmdID>
      <Data>201</Data>
```

```
        <Item>
          <Target>
            <LocURI>./calendar</LocURI>
          </Target>
          <Source>
            <LocURI>./C\System\Data\Calendar</LocURI>
          </Source>
          <Meta>
            <Anchor>
              <Next>20031202T165103Z</Next>
            </Anchor>
          </Meta>
        </Item>
      </Alert>
      <Put>
        <CmdID>3</CmdID>
        <Meta>
          <Type>application/vnd.syncml-devinf+wbxml</Type>
        </Meta>
        <Item>
          <Source>
            <LocURI>./devinf10</LocURI>
          </Source>
          <Data>
            <DevInf>
              <VerDTD>1.0</VerDTD>
              <Man>NOKIA</Man>
              <Mod>9210</Mod>
              <SwV>256</SwV>
              <HwV>1.0</HwV>
              <DevID>IMEI:57471724140229</DevID>
              <DevTyp>phone</DevTyp>
              ...
            </DevInf>
          </Data>
        </Item>
      </Put>
      <Get>
        <CmdID>4</CmdID>
        <Meta>
          <Type>application/vnd.syncml-devinf+wbxml</Type>
        </Meta>
        <Item>
          <Target>
            <LocURI>./devinf10</LocURI>
          </Target>
        </Item>
      </Get>
      <Final/>
    </SyncBody>
  </SyncML>
```

Server accepts the request.

```
  <SyncML xmlns="syncml:SYNCML1.0">
    <SyncHdr>
      <VerDTD>1.0</VerDTD>
      <VerProto>SyncML/1.0</VerProto>
      <SessionID>88</SessionID>
      <MsgID>2</MsgID>
      <Target>
        <LocURI>IMEI:57471724140229</LocURI>
      </Target>
      <Source>
        <LocURI>http://192.168.1.1:6666/</LocURI>
      </Source>
    </SyncHdr>
    <SyncBody>
      <Status>
        <CmdID>5</CmdID>
        <MsgRef>2</MsgRef>
        <CmdRef>0</CmdRef>
        <TargetRef>http://192.168.1.1:6666/</TargetRef>
```

```
  <SourceRef>IMEI:57471724140229</SourceRef>
  <Cmd>SyncHdr</Cmd>
  <Data>212</Data> <!-- Authenticated for session -->

</Status>
<Status>
  <CmdID>7</CmdID>
  <MsgRef>2</MsgRef>
  <CmdRef>2</CmdRef>
  <Cmd>Alert</Cmd>
  <Data>200</Data> <!- two-way alert accepted, 'next' anchor echoed -->

  <Item>
    <Data>
      <Anchor xmlns:n2="syncml:metinf">
        <Next>20031202T165103Z</Next>
      </Anchor>
    </Data>
  </Item>
</Status>
<Status>
  <CmdID>8</CmdID>
  <MsgRef>2</MsgRef>
  <CmdRef>3</CmdRef>
  <Cmd>Put</Cmd>
  <Data>200</Data> <!-- the 'put' command succeeded;
                              device info is written ->
</Status>
<Results> <!-- the following are server's device info  -->

  <CmdID>9</CmdID>
  <MsgRef>2</MsgRef>
  <CmdRef>4</CmdRef>
  <Meta>
    <Type xmlns:n2="syncml:metinf">application/vnd.syncml-devinf+wbxml</Type>
  </Meta>
  <Item>
    <Source>
      <LocURI>./devinf10</LocURI>
    </Source>
    <Data>
      <DevInf xmlns:n2="syncml:devinf">
        <VerDTD>1.0</VerDTD>
        <Man>OpenLink Software Ltd</Man>
        <Mod>Virtuoso</Mod>
        <OEM>OpenLink</OEM>
        <FwV>3.5</FwV>
        <SwV>2602</SwV>
        <HwV>0</HwV>
        <DevID>http://example.com/</DevID>
        <DevTyp>server</DevTyp>
        <SyncCap>
          <SyncType>1</SyncType>
          <SyncType>2</SyncType>
        </SyncCap>
        ...
        <UTC/>
        <SupportLargeObjs/>
        <SupportNumberOfChanges/>
      </DevInf>
    </Data>
  </Item>
</Results>
<Alert>
  <CmdID>6</CmdID>
  <Data>201</Data> <!-- the ./calendar/ is new empty collection;
                              server asks client for slow two-way sync.  -->

  <Item>
    <Target>
      <LocURI>./C\System\Data\Calendar</LocURI>
    </Target>
    <Source>
```

```
            <LocURI>./calendar/</LocURI>
          </Source>
          <Meta>
            <Anchor xmlns:n2="syncml:metinf">
              <Last>1970-01-01T00:00:00.000+02:00</Last>
              <Next>2003-12-02T18:51:05.000+02:00</Next>
            </Anchor>
          </Meta>
        </Item>
      </Alert>
      <Final/>
    </SyncBody>
  </SyncML>
```

Client sends to server all calendar items. See 'Sync' element below.

```
  <SyncML>
    <SyncHdr>
      <VerDTD>1.0</VerDTD>
      <VerProto>SyncML/1.0</VerProto>
      <SessionID>88</SessionID>
      <MsgID>3</MsgID>
      <Target>
        <LocURI>http://192.168.1.1:6666/</LocURI>
      </Target>
      <Source>
        <LocURI>IMEI:57471724140229</LocURI>
      </Source>
      <Meta>
        <MaxMsgSize>10000</MaxMsgSize>
      </Meta>
    </SyncHdr>
    <SyncBody>
      <Status>
        <CmdID>1</CmdID>
        <MsgRef>2</MsgRef>
        <CmdRef>0</CmdRef>
        <Cmd>SyncHdr</Cmd>
        <TargetRef>IMEI:57471724140229</TargetRef>
        <SourceRef>http://192.168.1.1:6666/</SourceRef>
        <Data>200</Data>
      </Status>
      <Status>
        <CmdID>2</CmdID>
        <MsgRef>2</MsgRef>
        <CmdRef>9</CmdRef>
        <Cmd>Results</Cmd>
        <Data>200</Data>
      </Status>
      <Status>
        <CmdID>3</CmdID>
        <MsgRef>2</MsgRef>
        <CmdRef>6</CmdRef>
        <Cmd>Alert</Cmd>
        <TargetRef>./C\System\Data\Calendar</TargetRef>
        <SourceRef>./calendar/</SourceRef>
        <Data>200</Data>
        <Item>
          <Data>
            <Anchor>
              <Next>2003-12-02T18:51:05.000+02:00</Next>
            </Anchor>
          </Data>
        </Item>
      </Status>
      <Sync>
        <CmdID>4</CmdID>
        <Target>
          <LocURI>./calendar</LocURI>
        </Target>
        <Source>
          <LocURI>./C\System\Data\Calendar</LocURI>
        </Source>
```

```
<Meta>
  <Mem>
    <FreeMem>7627614408</FreeMem>
    <FreeID>59590737</FreeID>
  </Mem>
</Meta>
<Replace> <!-- the client uses 'Replace' command -->

  <CmdID>5</CmdID>
  <Meta>
    <Type>text/x-vcalendar</Type>
  </Meta>
  <Item>
    <Source>
      <LocURI>2</LocURI>
    </Source>
    <Data>
        BEGIN:VCALENDAR
        VERSION:1.0
        BEGIN:VEVENT
        UID:2
        DESCRIPTION:tests
        DTSTART:20031127T090000
        DTEND:20031127T090000
        X-EPOCAGENDAENTRYTYPE:APPOINTMENT
        CLASS:PUBLIC
        DCREATED:20031128T000000
        LAST-MODIFIED:20031201T123500
        END:VEVENT
        END:VCALENDAR
    </Data>
  </Item>
</Replace>
<Replace>
  <CmdID>6</CmdID>
  <Meta>
    <Type>text/x-vcalendar</Type>
  </Meta>
  <Item>
    <Source>
      <LocURI>3</LocURI>
    </Source>
    <Data>
        BEGIN:VCALENDAR
        VERSION:1.0
        BEGIN:VEVENT
        UID:3
        DESCRIPTION:tests more
        DTSTART:20031128T090000
        DTEND:20031128T190000
        X-EPOCAGENDAENTRYTYPE:APPOINTMENT
        CLASS:PUBLIC
        DCREATED:20031128T000000
        LAST-MODIFIED:20031201T123500
        END:VEVENT
        END:VCALENDAR
    </Data>
  </Item>
</Replace>
<Replace>
  <CmdID>7</CmdID>
  <Meta>
    <Type>text/x-vcalendar</Type>
  </Meta>
  <Item>
    <Source>
      <LocURI>5</LocURI>
    </Source>
    <Data>
        BEGIN:VCALENDAR
        VERSION:1.0
        BEGIN:VEVENT
        UID:5
```

```
                DESCRIPTION:today integration
                DTSTART:20031201T090000
                DTEND:20031201T090000
                X-EPOCAGENDAENTRYTYPE:APPOINTMENT
                CLASS:PUBLIC
                DCREATED:20031201T000000
                LAST-MODIFIED:20031201T125400
                END:VEVENT
                END:VCALENDAR
              </Data>
            </Item>
          </Replace>
        </Sync>
        <Final/>
      </SyncBody>
    </SyncML>
```

Server stores new items.

```
    <SyncML xmlns:n0="syncml:SYNCML1.0">
      <SyncHdr>
        <VerDTD>1.0</VerDTD>
        <VerProto>SyncML/1.0</VerProto>
        <SessionID>88</SessionID>
        <MsgID>3</MsgID>
        <Target>
          <LocURI>IMEI:57471724140229</LocURI>
        </Target>
        <Source>
          <LocURI>http://192.168.1.1:6666/</LocURI>
        </Source>
      </SyncHdr>
      <SyncBody>
        <Status>
          <CmdID>10</CmdID>
          <MsgRef>3</MsgRef>
          <CmdRef>0</CmdRef>
          <TargetRef>http://192.168.1.1:6666/</TargetRef>
          <SourceRef>IMEI:57471724140229</SourceRef>
          <Cmd>SyncHdr</Cmd>
          <Data>212</Data>
        </Status>
        <Status>
          <CmdID>11</CmdID>
          <MsgRef>3</MsgRef>
          <CmdRef>5</CmdRef>
          <Cmd>Replace</Cmd>
          <Data>201</Data> <!-- 201 (Added) is used to indicate
                                    that item is added as new. -->

        </Status>
        <Status>
          <CmdID>12</CmdID>
          <MsgRef>3</MsgRef>
          <CmdRef>6</CmdRef>
          <Cmd>Replace</Cmd>
          <Data>201</Data>
        </Status>
        <Status>
          <CmdID>13</CmdID>
          <MsgRef>3</MsgRef>
          <CmdRef>7</CmdRef>
          <Cmd>Replace</Cmd>
          <Data>201</Data>
        </Status>
        <Status>
          <CmdID>15</CmdID>
          <MsgRef>3</MsgRef>
          <CmdRef>4</CmdRef>
          <TargetRef>./calendar/</TargetRef>
          <SourceRef>./C\System\Data\Calendar</SourceRef>
          <Cmd>Sync</Cmd>
          <Data>200</Data>
```

```
    </Status>
    <Sync> <!-- an empty Sync is sent from server. -->

      <CmdID>14</CmdID>
      <Source>
        <LocURI>./calendar/</LocURI>
      </Source>
      <Target>
        <LocURI>./C\System\Data\Calendar</LocURI>
      </Target>
    </Sync>
    <Final/>
  </SyncBody>
</SyncML>
```

Final SyncML message from client with status to server's Sync command.

```
<SyncML>
  <SyncHdr>
    <VerDTD>1.0</VerDTD>
    <VerProto>SyncML/1.0</VerProto>
    <SessionID>88</SessionID>
    <MsgID>4</MsgID>
    <Target>
      <LocURI>http://192.168.1.1:6666/</LocURI>
    </Target>
    <Source>
      <LocURI>IMEI:57471724140229</LocURI>
    </Source>
    <Meta>
      <MaxMsgSize>10000</MaxMsgSize>
    </Meta>
  </SyncHdr>
  <SyncBody>
    <Status>
      <CmdID>1</CmdID>
      <MsgRef>3</MsgRef>
      <CmdRef>0</CmdRef>
      <Cmd>SyncHdr</Cmd>
      <TargetRef>IMEI:57471724140229</TargetRef>
      <SourceRef>http://192.168.1.1:6666/</SourceRef>
      <Data>200</Data>
    </Status>
    <Status>
      <CmdID>2</CmdID>
      <MsgRef>3</MsgRef>
      <CmdRef>14</CmdRef>
      <Cmd>Sync</Cmd>
      <TargetRef>./C\System\Data\Calendar</TargetRef>
      <SourceRef>./calendar/</SourceRef>
      <Data>200</Data>
    </Status>
    <Final/>
  </SyncBody>
</SyncML>
```

Final message from SyncML server; no more commands issued.

```
<SyncML xmlns:n0="syncml:SYNCML1.0">
  <SyncHdr>
    <VerDTD>1.0</VerDTD>
    <VerProto>SyncML/1.0</VerProto>
    <SessionID>88</SessionID>
    <MsgID>4</MsgID>
    <Target>
      <LocURI>IMEI:57471724140229</LocURI>
    </Target>
    <Source>
      <LocURI>http://192.168.1.1:6666/</LocURI>
    </Source>
  </SyncHdr>
  <SyncBody>
```

```
      <Status>
        <CmdID>16</CmdID>
        <MsgRef>4</MsgRef>
        <CmdRef>0</CmdRef>
        <TargetRef>http://192.168.1.1:6666/</TargetRef>
        <SourceRef>IMEI:57471724140229</SourceRef>
        <Cmd>SyncHdr</Cmd>
        <Data>212</Data>
      </Status>
      <Final/>
    </SyncBody>
  </SyncML>
```

See Also:
References : www.syncml.org

17.12. UDDI

17.12.1. Concepts

Universal Description, Discovery and Integration (UDDI) is the name of a web-based service that exposes information about a business or other entities and its technical interfaces or APIs. These services are run by UDDI-enabled servers, and can be used by any business that wants to make their information available, as well as anyone who wants to find that information. There is no charge for using the basic services of these sites.

By accessing any of the public sites, anyone can search for information about web services that are made available by or on behalf of a business. This provides a mechanism that allows others to discover what technical programming interfaces are provided for interacting with a business for such purposes as electronic commerce. The benefit to the individual business is increased exposure in an electronic commerce enabled world.

Businesses can register several kinds of simple data to help others answer questions of "who, what, where and how." Simple information about a business - information such as name, business identifiers (D&B D-U-N-S Number(R), etc.), and contact information answers the question "Who." "What" involves classification information including industry codes and product classifications, as well as descriptive information about the services that are available for electronic interchange. Answering the question "Where" involves registering information about the URL or email address (or other address) through which each type of service may be accessed. Finally, the question "How" is answered by registering references to information about specifications that describe how a particular software package or technical interface functions. These references are called '*tModels*' in the documentation.

17.12.2. Dealing with SOAP

UDDI API functions are exposed as SOAP v1.1 messages over the HTTP protocol. In version 1, the `SOAPAction` HTTP Header is required. The value passed in this HTTP Header must be an empty string that is surrounded by double quotes.

Example 17.39. UDDI and Soap

```
POST /UDDI/inquiry HTTP/1.1
Host: www.foo.com
Content-Type: text/xml
Content-Length: nnnn
SOAPAction: ""

.... body follows ....
```

SOAP is used in conjunction with HTTP to provide a simple mechanism for passing XML messages to UDDI-enabled servers using a standard HTTP-POST protocol. Unless specified, all responses will be returned in the normal HTTP response document.

See Also:
For more information about Virtuoso's SOAP Implementation see the SOAP Services section.

17.12.3. Supported API Calls

The UDDI APIs always return a SOAP entity body which contains messages as described in UDDI v1 XML Schema (uddi_1.xsd).

- *Authorization API* - Used to establish authentication mechanism (tokens), dropping already established connections.
  - *discard_authToken:* Used to inform a UDDI enabled server that a previously provided authentication token is no longer valid.
  - *get_authToken:* Used to request an authentication token from a UDDI-enabled server. Authentication tokens are required to use all other APIs defined in the publishers API. This function serves as the program's equivalent of a login request.
  - *get_registeredInfo:* Used to request an abbreviated synopsis of all information currently managed by a given individual.
- *Searching API* The publicly accessible queries are:
  - *find_binding:* Used to locate specific bindings within a registered `businessService`. Returns a `bindingDetail` message.
  - *find_business:* Used to locate information about one or more businesses . Returns a `businessList` message.
  - *find_service:* Used to locate specific services within a registered `businessEntity`. Returns a `serviceList` message.
  - *find_tModel:* Used to locate one or more `tModel` information structures. Returns a `tModelList` structure.
  - *get_bindingDetail:* Used to get full `bindingTemplate` information suitable for making one or more service requests. Returns a `bindingDetail` message.
  - *get_businessDetail:* Used to get full `businessEntity` information for one or more businesses. Returns a `businessDetail` message.
  - *get_businessDetailExt:* Used to get extended `businessEntity` information. Returns a `businessDetailExt` message.
  - *get_serviceDetail:* Used to get full details for a given set of registered `businessService` data. Returns a `serviceDetail` message.
  - *get_tModelDetail:* Used to get full details for a given set of registered `tModel` data. Returns a `tModelDetail` message.
- *Repository Manipulation API*
  - *delete_binding:* Used to remove an existing `bindingTemplate` from the `bindingTemplate`s collection that is part of a specified `businessService` structure.
  - *delete_business:* Used to delete registered `businessEntity` information from the registry.
  - *delete_service:* Used to delete an existing `businessService` from the businessServices collection that is part of a specified `businessEntity`.
  - *delete_tModel:* Used to delete registered information about a `tModel`. If there are any references to a `tModel` when this call is made, the `tModel` will be marked deleted instead of being physically removed.
  - *save_binding:* Used to register new `bindingTemplate` information or update existing `bindingTemplate` information. Use this to control information about technical capabilities exposed by a registered business.
  - *save_business:* Used to register new `businessEntity` information or update existing `businessEntity` information. Use this to control the overall information about the entire business. Of all the save APIs this one has the broadest effect.
  - *save_service:* Used to register or update complete information about a `businessService` exposed by a specified `businessEntity`.
  - *save_tModel:* Used to register or update complete information about a `tModel`.

17.12.4. Authorization Mechanism

The Publishers API describes the messages that are used to control the content contained within a UDDI-enabled server, and can be used by compliant non-operator implementations that adhere to the behaviors described in this programmer's reference specification.

All calls made to UDDI-enabled servers that use the messages defined in the publisher's API will be transported using SSL encryption. UDDI-enabled servers will each provide a service description that exposes a `bindingTemplate` that makes use of HTTPS and SSL to secure the transmission of data.

Authentication

Each of the calls in the publisher's API that change information at a given UDDI-enabled server requires the use of an opaque authentication token. These tokens are generated by or provided by each UDDI-enabled server independently, and are passed from the caller to the UDDI-enabled server in the element named *authInfo*.

These tokens are meaningful only to the UDDI-enabled server that provided them and are to be used according to the published policies of a given UDDI-enabled server.

Each party that has been granted publication access to a given UDDI-enabled server will be provided a token by the site. The methods for obtaining this token are specific to each UDDI-enabled server.

Establishing Credentials

Before any party can publish data within a UDDI-enabled server, credentials and permission to publish must be supplied with the individual operator. Generally, you will only need to interact with one UDDI-enabled server because all data published at any UDDI-enabled server are replicated automatically to all other such servers. Establishing publishing credentials involves providing some verifiable identification information, contact information, and establishing security credentials with the individual server. The specifics of these establishing credentials is server-dependent, and all valid UDDI-enabled servers provide a Web-based user interface through which you can establish an identity and secure permission to publish data.

Every registry implementation that adheres to these specifications establishes its own mechanism for token generation and authentication. The only requirement placed on token generation for use with the publisher's API is that the tokens themselves must be valid string text that can be placed within the *authInfo* XML element. Given that binary-to-string translations are well-understood and in common use, this requirement will not introduce hardships.

Authentication tokens are not required to be valid except at the UDDI-enabled server or implementation from which they originated. These tokens need only have meaning at a single UDDI-enabled server or implementation, and should not be expected to work across sites.

Generating Authentication Tokens

Many implementations are expected to require a login step. The *get_authToken* message is provided to accommodate implementations that desire a login step. Security schemes based on exchanging User ID and password credentials fall into this category. For implementations that desire this kind of security, the *get_authToken* API is provided as a means of generating a temporary authentication token.

Certificate-based authentication and similar security mechanisms do not require this additional step of logging in. Instead, they can pass compatible authentication token information such as a certificate value within the *authInfo* element provided on each of the publisher's API messages. If certificate-based authentication or similar security is employed the use of the *get_authToken* and *discard_authToken* messages is optional.

17.12.5. UDDI API Calls

This section describes the Virtuoso UDDI-related messages. These messages are divided into APIs for authentication, inquiry, and publication.

Authorization API

me_uddi_get_authtoken_dedup

For detailed description and example use of the function, see me_uddi_get_authtoken in the Functions Reference Guide.

me_uddi_get_registeredinfo_dedup

For detailed description and example use of the function, see me_uddi_get_registeredinfo in the Functions Reference Guide.

me_uddi_discard_authtoken_dedup

For detailed description and example use of the function, see me_uddi_discard_authtoken in the Functions Reference Guide.

Inquiry API Functions

The inquiry API messages `find_binding` , `find_business` , `find_service` , and `find_tModel` all accept an optional element named `findQualifiers` . This argument provides a means to allow the caller to override default search behaviors.

The messages in this section represent inquiries that anyone can make of any UDDI-enabled server at any time. These messages all behave synchronously and are required to be exposed via HTTP POST only. Other synchronous or asynchronous mechanisms may be provided at the discretion of the individual UDDI-enabled server or compatible registry.

The publicly accessible queries are:

- *find_binding:* Used to locate specific bindings within a registered `businessService`. Returns a `bindingDetail` message.
- *find_business:* Used to locate information about one or more businesses. Returns a `businessList` message.
- *find_service:* Used to locate specific services within a registered `businessEntity`. Returns a `serviceList` message.
- *find_tModel:* Used to locate one or more `tModel` information structures. Returns a `tModelList` structure.
- *get_bindingDetail:* Used to get full `bindingTemplate` information suitable for making one or more service requests. Returns a `bindingDetail` message.
- *get_businessDetail:* Used to get the full `businessEntity` information for a one or more businesses. Returns a `businessDetail` message.
- *get_businessDetailExt:* Used to get extended `businessEntity` information. Returns a `businessDetailExt` message.
- *get_serviceDetail:* Used to get full details for a given set of registered `businessService` date. Returns a `serviceDetail` message.
- *get_tModelDetail:* Used to get full details for a given set of registered `tModel` data. Returns a `tModelDetail` message.

me_uddi_find_binding_dedup

For detailed description and example use of the function, see me_uddi_find_binding in the Functions Reference Guide.

me_uddi_find_business_dedup

For detailed description and example use of the function, see me_uddi_find_business in the Functions Reference Guide.

me_uddi_find_service_dedup

For detailed description and example use of the function, see me_uddi_find_service in the Functions Reference Guide.

me_uddi_find_tmodel_dedup

For detailed description and example use of the function, see me_uddi_find_tmodel in the Functions Reference Guide.

me_uddi_get_bindingdetail_dedup

For detailed description and example use of the function, see me_uddi_get_bindingdetail in the Functions Reference Guide.

me_uddi_get_businessdetail_dedup

For detailed description and example use of the function, see me_uddi_get_businessdetail in the Functions Reference Guide.

me_uddi_get_businessdetailext_dedup

For detailed description and example use of the function, see me_uddi_get_businessdetailext in the Functions Reference Guide.

me_uddi_get_servicedetail_dedup

For detailed description and example use of the function, see me_uddi_get_servicedetail in the Functions Reference Guide.

me_uddi_get_tmodeldetail_dedup

For detailed description and example use of the function, see me_uddi_get_tmodeldetail in the Functions Reference Guide.

Publishing API Functions

The messages in this section represent inquiries that require authenticated access to an operator site. Each business should initially select one UDDI-enabled server to host their information. Once chosen, information can only be updated at the site originally selected.

The messages defined in this section all behave synchronously and are only callable via HTTP-POST. HTTPS is used exclusively for all the calls defined in this publisher's API.

- *save_binding:* Used to register new `bindingTemplate` information or update existing `bindingTemplate` information. Use this to control information about technical capabilities exposed by a registered business.
- *save_business:* Used to register new `businessEntity` information or update existing `businessEntity` information. Use this to control the overall information about the entire business. Of all the save messages, this one has the broadest effect.
- *save_service:* Used to register or update complete information about a `businessService` exposed by a specified `businessEntity`.
- *save_tModel:* Used to register or update complete information about a `tModel`.
- *delete_binding:* Used to remove an existing `bindingTemplate` from the bindingTemplates collection that is part of a specified `businessService` structure.
- *delete_business:* Used to delete registered `businessEntity` information from the registry.
- *delete_service:* Used to delete an existing `businessService` from the businessServices collection that is part of a specified `businessEntity`.
- *delete_tModel:* Used to delete registered information about a `tModel`. If there are any references to a `tModel` when this call is made, the `tModel` will be marked deleted instead of being physically removed.

me_uddi_save_binding_dedup

For detailed description and example use of the function, see me_uddi_save_binding in the Functions Reference Guide.

me_uddi_save_business_dedup

For detailed description and example use of the function, see me_uddi_save_business in the Functions Reference Guide.

me_uddi_save_service_dedup

For detailed description and example use of the function, see me_uddi_save_service in the Functions Reference Guide.

me_uddi_save_tmodel_dedup

For detailed description and example use of the function, see me_uddi_save_tmodel in the Functions Reference Guide.

me_uddi_delete_binding_dedup

For detailed description and example use of the function, see me_uddi_delete_binding in the Functions Reference Guide.

me_uddi_delete_business_dedup

For detailed description and example use of the function, see me_uddi_delete_business in the Functions Reference Guide.

me_uddi_delete_service_dedup

For detailed description and example use of the function, see me_uddi_delete_service in the Functions Reference Guide.

me_uddi_delete_tmodel_dedup

For detailed description and example use of the function, see me_uddi_delete_tmodel in the Functions Reference Guide.

17.12.6. Examples

Example 17.40. Generic Find

Finds all registry entries for names beginning with 'M':

```
select http_get ('http://example.com/SOAP', null, 'POST',
  'Content-Type: text/xml\r\nSOAPAction: ""',
  '<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
<Body>
<find_business xmlns="urn:uddi-org:api" generic="1.0">
<name>M</name>
</find_business>
</Body>
</Envelope>');
```

Example 17.41. Find By Name 1

Find by name with sort options by name and date, both ascending:

```
select http_get ('http://example.com/SOAP', null, 'POST',
  'Content-Type: text/xml\r\nSOAPAction: ""',
  '<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
<Body>
<find_business xmlns="urn:uddi-org:api" generic="1.0">
<findQualifiers>
<findQualifier>sortByNameAsc</findQualifier>
<findQualifier>sortByDateAsc</findQualifier>
</findQualifiers>
<name></name>
</find_business>
</Body>
</Envelope>');
```

Example 17.42. Find By Name 2

Find by name, sorted by name descending and date ascending:

```
select http_get ('http://example.com/SOAP', null, 'POST',
  'Content-Type: text/xml\r\nSOAPAction: ""',
  '<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
<Body>
<find_business xmlns="urn:uddi-org:api" generic="1.0">
<findQualifiers>
<findQualifier>sortByNameDesc</findQualifier>
<findQualifier>sortByDateAsc</findQualifier>
</findQualifiers>
<name></name>
</find_business>
</Body>
</Envelope>');
```

Example 17.43. Find By Name 3

Find by exact name match, case-sensitive, and sorted by name and date ascending:

```
select http_get ('http://example.com/SOAP', null, 'POST',
  'Content-Type: text/xml\r\nSOAPAction: ""',
  '<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
<Body>
<find_business xmlns="urn:uddi-org:api" generic="1.0">
<findQualifiers>
<findQualifier>exactNameMatch</findQualifier>
<findQualifier>caseSensitiveMatch</findQualifier>
<findQualifier>sortByNameAsc</findQualifier>
<findQualifier>sortByDateAsc</findQualifier>
</findQualifiers>
<name>Microsoft Corporation</name>
</find_business>
</Body>
</Envelope>');
```

Example 17.44. Find By *identifierBag* Element 'DUNS':

```
select http_get ('http://example.com/SOAP', null, 'POST',
  'Content-Type: text/xml\r\nSOAPAction: ""',
  '<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
<Body>
<find_business xmlns="urn:uddi-org:api" generic="1.0">
<findQualifiers>
<findQualifier>exactNameMatch</findQualifier>
<findQualifier>caseSensitiveMatch</findQualifier>
<findQualifier>sortByNameAsc</findQualifier>
<findQualifier>sortByDateAsc</findQualifier>
</findQualifiers>
<identifierBag>
<keyedReference keyName="D-U-N-S" keyValue="08-146-6849"
  tModelKey="UUID:8609C81E-EE1F-4D5A-B202-3EB13AD01823"/>
</identifierBag>
</find_business>
</Body>
</Envelope>');
```

Example 17.45. Find by *tModel*

```
select http_get ('http://example.com/SOAP', null, 'POST',
  'Content-Type: text/xml\r\nSOAPAction: ""',
  '<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
<Body>
<find_business xmlns="urn:uddi-org:api" generic="1.0">
<findQualifiers>
<findQualifier>exactNameMatch</findQualifier>
<findQualifier>caseSensitiveMatch</findQualifier>
<findQualifier>sortByNameAsc</findQualifier>
<findQualifier>sortByDateAsc</findQualifier>
</findQualifiers>
<tModelBag>
<tModelKey>UUID:68DE9E80-AD09-469D-8A37-088422BFBC36</tModelKey>
</tModelBag>
</find_business>
</Body>
</Envelope>');
```

Example 17.46. Find by discovery URL

```
select http_get ('http://example.com/SOAP', null, 'POST',
  'Content-Type: text/xml\r\nSOAPAction: ""',
  '<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
<Body>
<find_business xmlns="urn:uddi-org:api" generic="1.0">
<findQualifiers>
<findQualifier>exactNameMatch</findQualifier>
<findQualifier>caseSensitiveMatch</findQualifier>
<findQualifier>sortByNameAsc</findQualifier>
<findQualifier>sortByDateAsc</findQualifier>
</findQualifiers>
<discoveryURLs>
<discoveryURL>
http://uddi.microsoft.com/discovery?businessKey=D94B25A4-BD6D-4426-AEAC-1087DCC21421
</discoveryURL>
</discoveryURLs>
</find_business>
</Body>
</Envelope>');
```

Example 17.47. Get authentication token

```
select http_get ('http://example.com/SOAP', null, 'POST',
  'Content-Type: text/xml\r\nSOAPAction: ""',
  '<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
<Body>
<get_authToken xmlns="urn:uddi-org:api" generic="1.0">
<userID>dav</userID>
<cred>dav</cred>
</get_authToken>
</Body>
</Envelope>');
```

Example 17.48. Discard Authentication Token

```
select http_get ('http://example.com/SOAP', null, 'POST',
  'Content-Type: text/xml\r\nSOAPAction: ""',
  '<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
<Body>
<discard_authToken xmlns="urn:uddi-org:api" generic="1.0">
<authInfo>71255ffc5b0a22b4699bfcee74cba97f</authInfo>
</discard_authToken>
</Body>
</Envelope>');
```

See Also:
UDDI System Tables

17.13. Exposing Persistent Stored Modules as Web Services

Virtuoso SQL stored procedures and functions can be exposed as SOAP services very simply from Virtuoso, whether they are native Virtuoso or on remote data sources. This powerful ability means that any database servers already existing within an organization can easily become a component in an eBusiness solution using Virtuoso. All you need is a few simple steps that typically take mere minutes to complete:

- **Choose your stored procedure(s) .** The procedures that you want to expose can either be native Virtuoso stored procedures, or remote stored procedures that can be linked in using the Remote Procedures user interface.

- **Choose a virtual directory .** Because SOAP services need to be exposed and accessed via HTTP a Virtuoso virtual directory must be used. Either use the existing SOAP virtual directory or create a new one.
- **Publish procedures to virtual directory .** The user specified as SOAP account on the virtual directory must have execute privileges on the procedures. Use the Publish options on the virtual directory user interface.
- **Test the VSMX output .** Once procedures have been published as SOAP services they are automatically described by WSDL and testable using Virtuoso's VSMX feature.

XML Query Templates provide a direct way to store SQL in an XML file on the Virtuoso server that when executed, i.e. fetched from a web browser, actually returns the results of the query.

The C Interface chapter describes how users can define custom built-in functions, from C or other programming languages, that can be used from within Virtuoso PL. This also means that VSE's can also be published as a Web Service!

17.13.1. Publishing Stored Procedures as Web Services

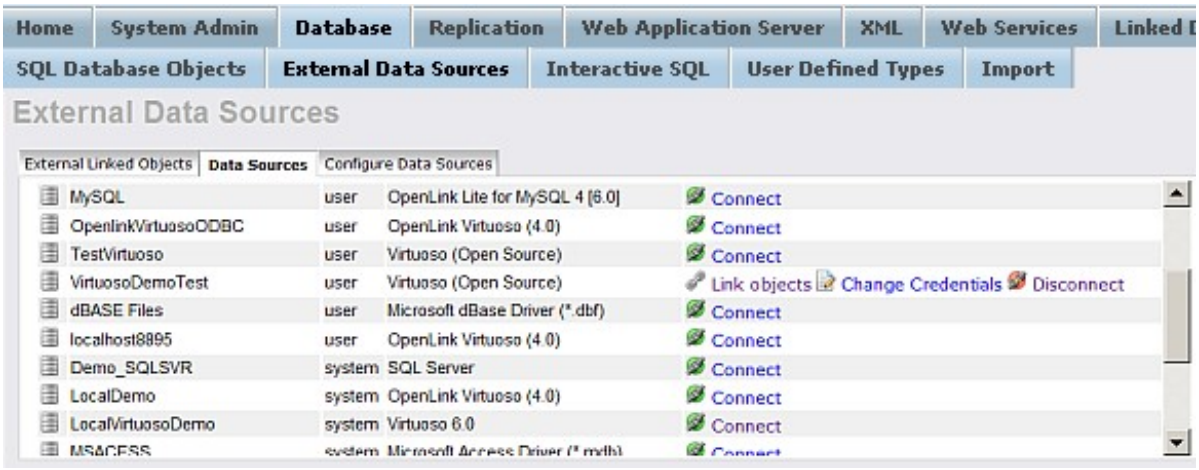
Choosing Stored Procedures to Expose

You can either expose native Virtuoso stored procedures (previously defined or newly created) using the CREATE PROCEDURE statement, or stored procedures from other database types can be linked into Virtuoso using an ODBC datasource.

Virtuoso lists available stored procedures for each catalog in Conductor under: */Database/External Data Sources/External Linked Objects / with checked "Stored Procedures"* .

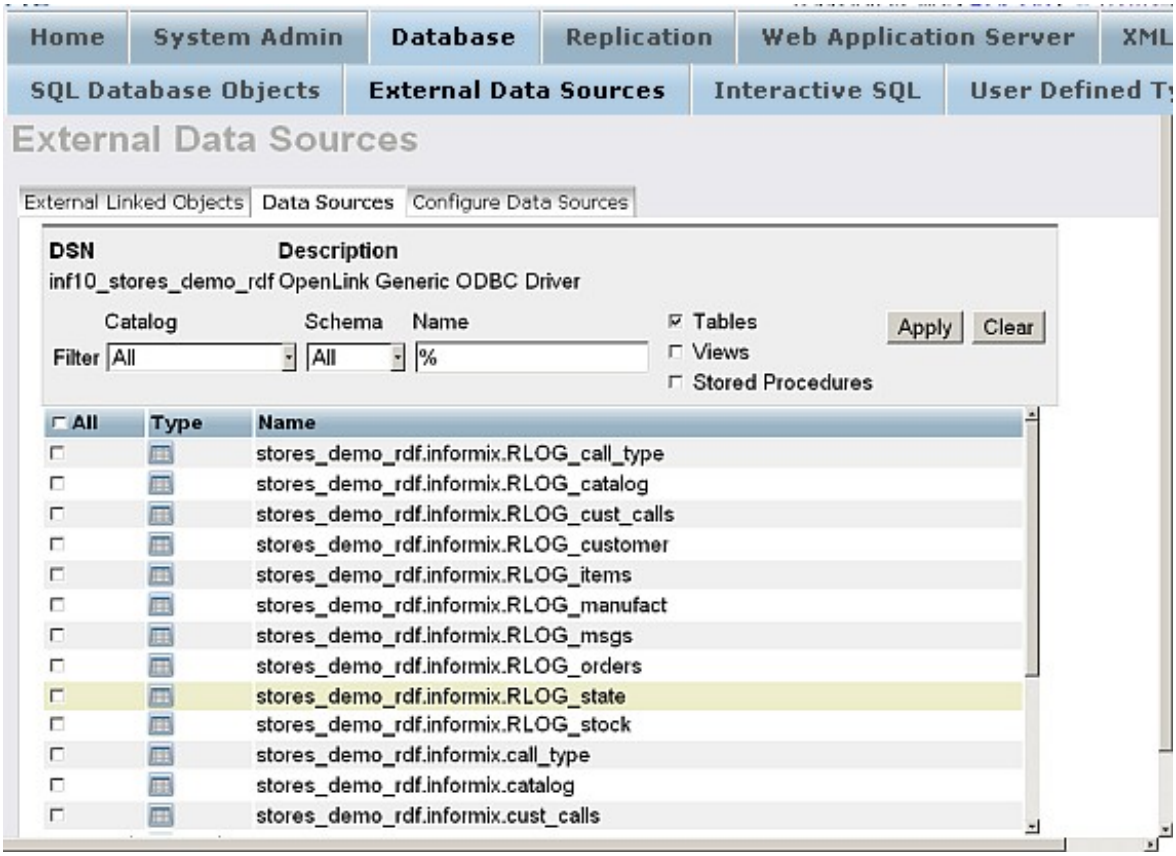
To link a stored procedure from another database system we must first create a valid data source that leads to a connection to that database. Once verified proceed to the Remote Procedures page. Select the "Link objects" link for a data source.

Figure 17.53. Linking Procedures from Remote Data Sources



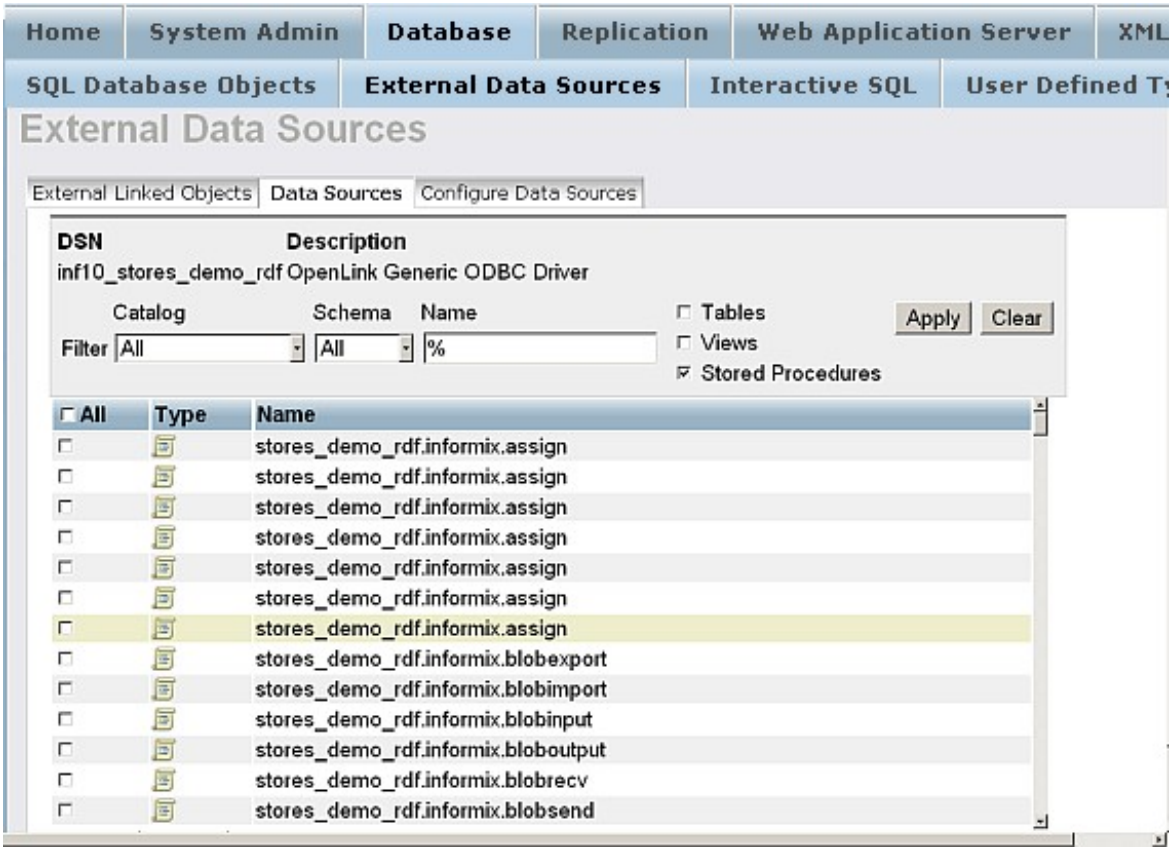
Select the check-box "Store Procedures". Click the "Apply" button. As result will be shown the list of available procedures.

Figure 17.54. Linking Procedures from Remote Datasources



Select the check-boxes for the procedures you want to link and click the "Link" button.

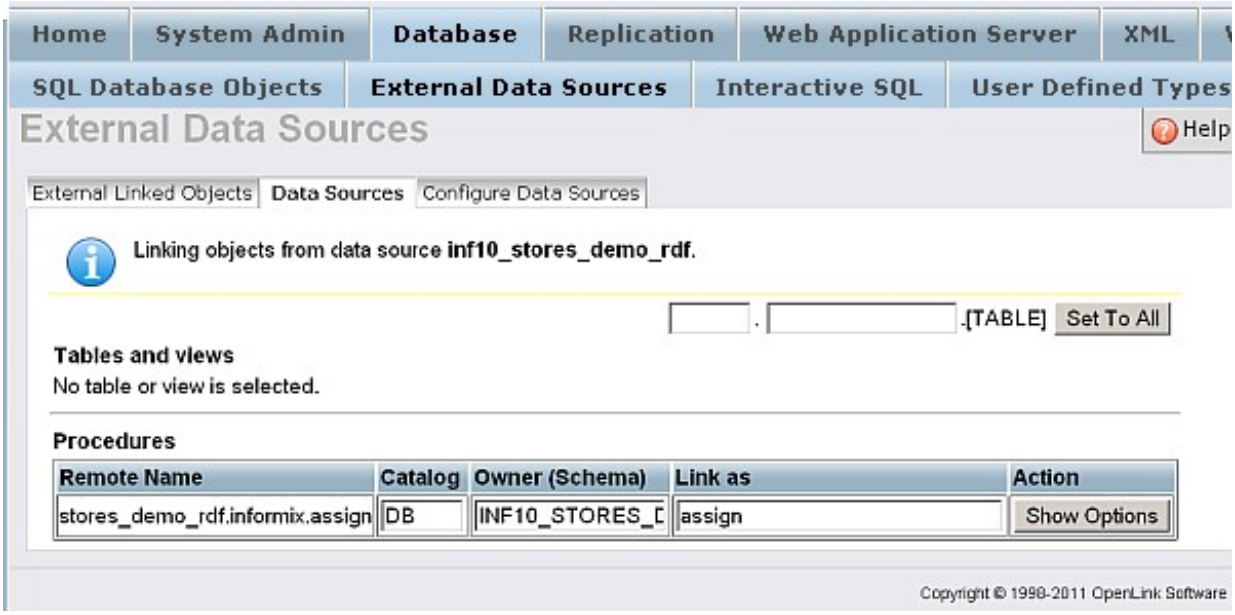
Figure 17.55. Linking Procedures from Remote Datasources



You will be presented with a new page listing the chosen procedures and their data type information. This gives you an opportunity to alter the data type mappings that Virtuoso will use both internally and for any future interactions with the SOAP

server. If you do not want to specify any special type information the details can be left as default.

Figure 17.56. Linking Procedures from Remote Datasources



For each remote procedure you may change how they will be referenced within Virtuoso by making changes to the fields for *Catalog* , *Owner* , *Link as* , and *Description* fields. These fields define how you will find the linked procedure locally to Virtuoso only and do not affect the remote data source.

For each procedure there is an option to *PL Wrapper Requirement* . This option is required if your remote procedure is capable of returning a resultset that you want to process via Virtuoso. Can be *SOAP Execution* , *SQL Execution* or *None* . Also you can specify *Return Type* , *Data Type* , *SOAP Type* .

Once the details are correct press the "Link" button.

See Also:
Linking Remote Procedures

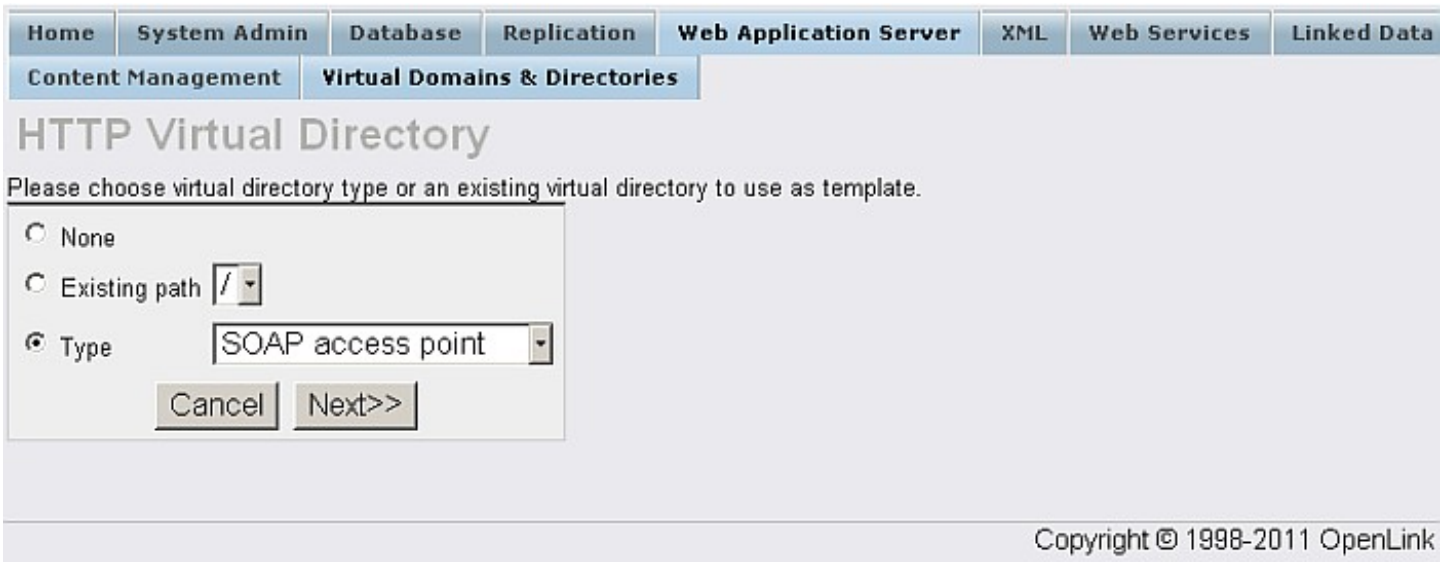
Defining Virtual Directories

Before any procedures native or linked can be exposed as SOAP Services a location in HTTP space must be defined. From Conductor *Web Application Server/Virtual Domains & Directories* you make a new URL Mappings. Click on the *New Directory* link for the {Default Web Site} line to begin defining a new SOAP mapping.

Figure 17.57. Virtual Directories

Select for "Type" from the list the value "SOAP access point" and click the "Next" button.

Figure 17.58. Virtual Directories Mappings

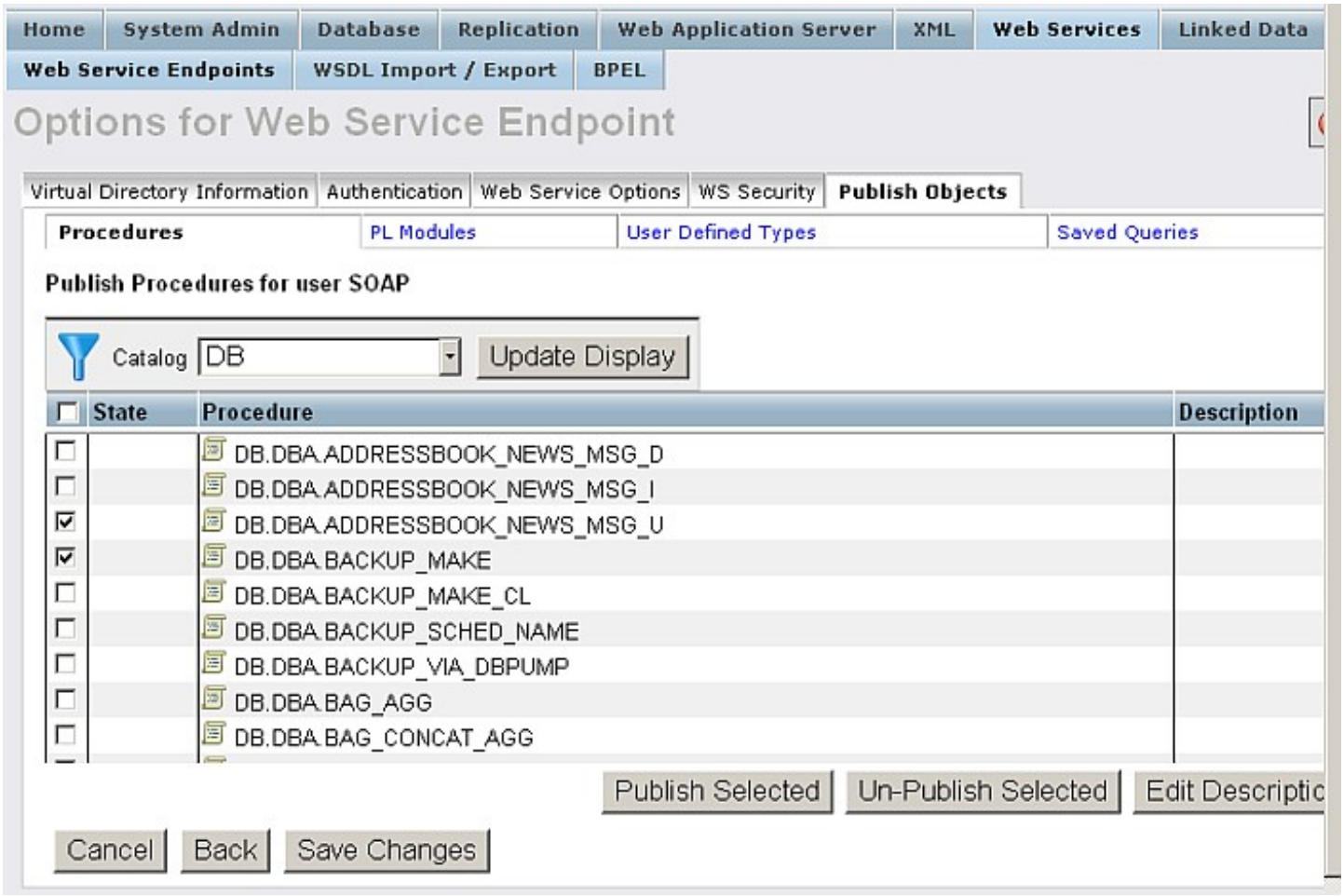


You will then be presented with the following tabs: "Virtual Directory Information", "Authentication", "Web Service Option", "WS Security" and "Publish Objects". Particular options to note are "Virtual Directory Information" and "Publish Objects".

Figure 17.59. Virtual Directories

In *Publish Objects* you can select Virtuoso stored procedures, or remotely linked procedures to be published as SOAP web services. Also you can publish Pl Modules, User Defined Types, or Saved Queries.

Figure 17.60. Publish Objects

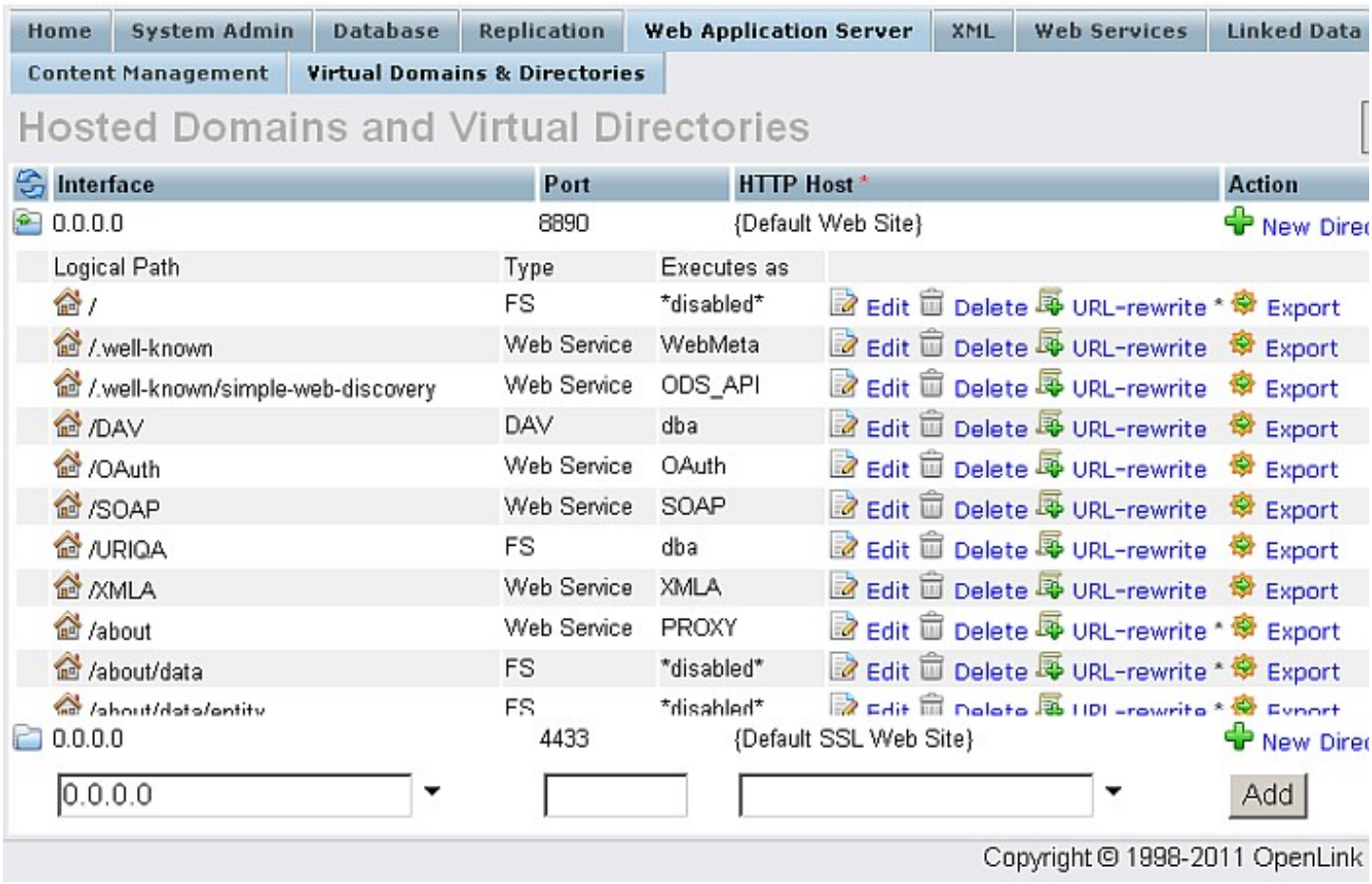


See Also:
Virtual Directories

Publishing Procedures to a Virtual Directory

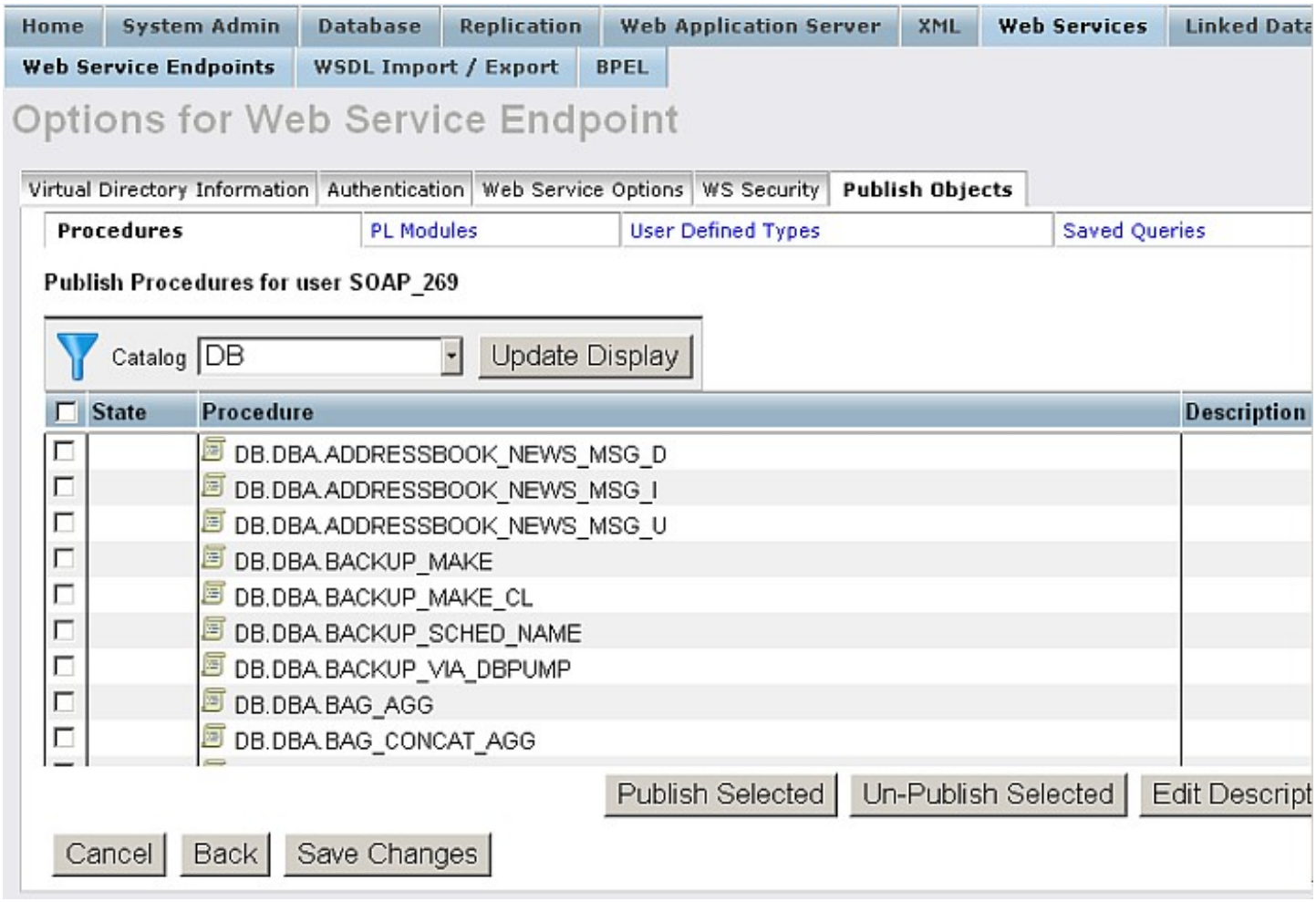
If you already have a virtual directory defined and know what procedures you want to expose as web services you will have to repeat some of the steps in the section above. From Conductor go to *Web Application Server/Virtual Domains & Directories* . Click on the "folder" icon for your {Default Web Site}. You will find the list of previously existing mappings, from which you can select the mapping that you want to edit by pressing on its *Edit* link. Note, the virtual directory should have type "SOAP".

Figure 17.61. Virtual Directories



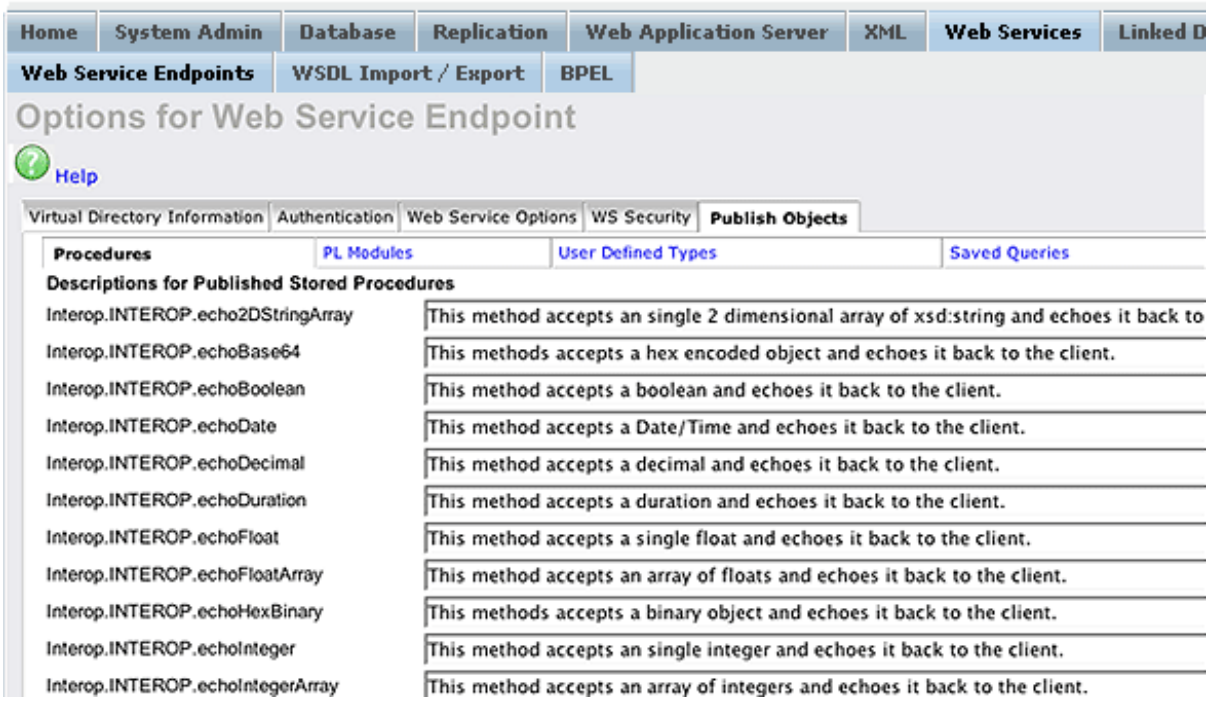
Go to tab "publish Objects" to expose/hide your procedures, Pl Modules, User Defined Types and Saved Queries.

Figure 17.62. Publish Objects



The "Procedures" tab presents the list of available procedures. You can select a catalogue in order to list the procedures you want to publish. When the procedures to be published are selected, you can either click the "Publish Selected" button, or before this to click the "Edit Description" button.

Figure 17.63. Choosing Procedure aPublish



Testing SOAP Services Using VSMX

Virtual directory definitions have a *Logical Path* field, which is reference in URL to find the correct SOAP services. If you connect to Virtuoso on *http://example.com/* , and defined your virtual directory with the logical path of */mysoap* then you will be able to test the following URLs:

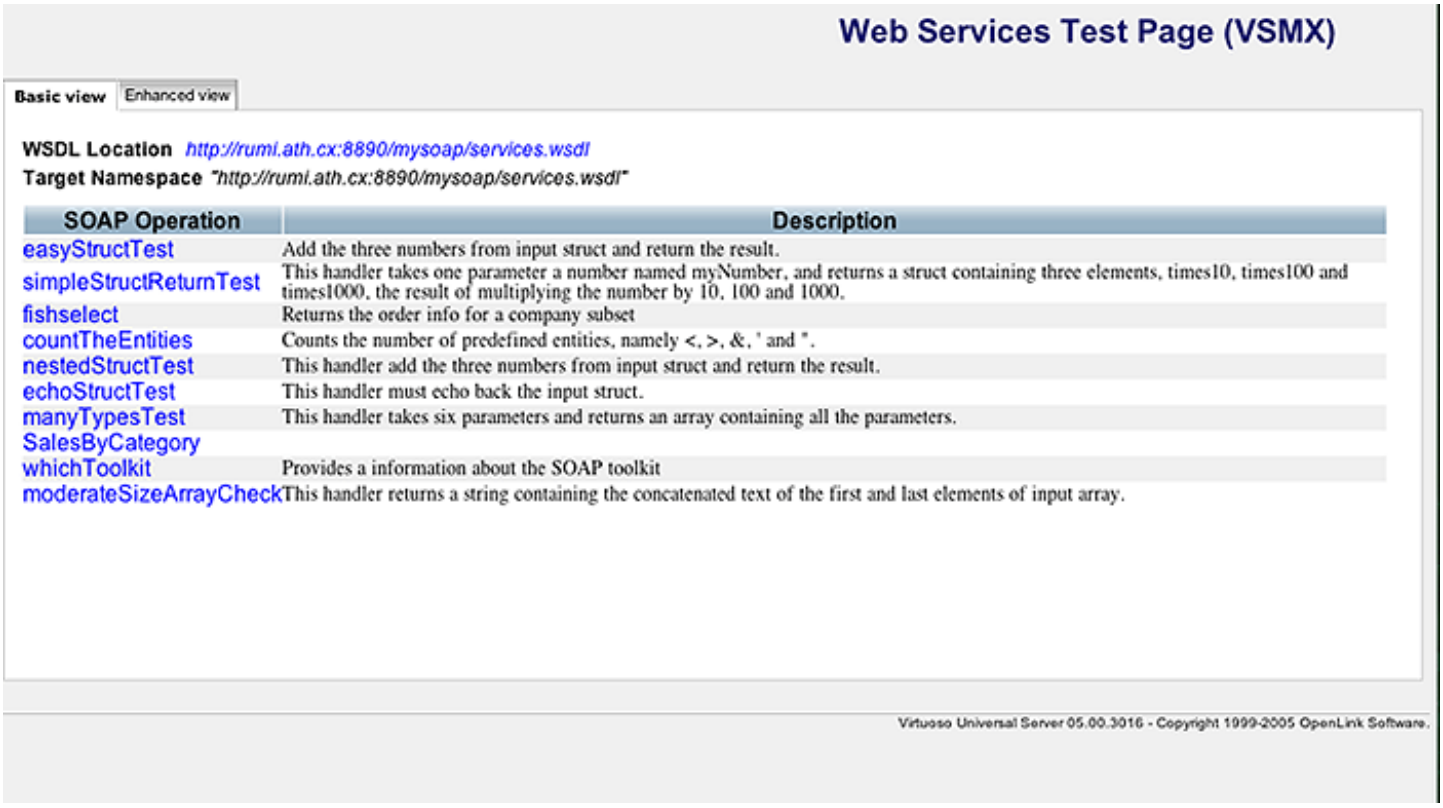
http://example.com/mysoap/services.wsdl

http://example.com/mysoap/services.vsmx

Figure 17.64. Services.wsdl

```
- <definitions targetNamespace="http://rumi.ath.cx:8890/mysoap/services.wsdl" name="SOAP">
  - <types>
    - <schema targetNamespace="http://tempuri.org/">
        <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
      - <complexType name="ArrayOfstring">
        - <complexContent>
          - <restriction base="soapenc:Array">
            - <sequence>
                <element name="item" type="string" minOccurs="0" maxOccurs="unbounded" nillable="true"/>
              </sequence>
              <attributeGroup ref="soapenc:commonAttributes"/>
              <attribute ref="soapenc:arrayType" wsdl:arrayType="string[]"/>
            </restriction>
          </complexContent>
        </complexType>
      - <complexType name="toolkit_info">
        - <all>
            <element name="toolkitDocsUrl" type="string"/>
            <element name="toolkitName" type="string"/>
            <element name="toolkitVersion" type="string"/>
            <element name="toolkitOperatingSystem" type="string"/>
          </all>
        </complexType>
      - <complexType name="ctc">
        - <all>
            <element name="ctLeftAngleBrackets" type="int"/>
            <element name="ctRightAngleBrackets" type="int"/>
            <element name="ctAmpersands" type="int"/>
            <element name="ctApostrophes" type="int"/>
            <element name="ctQuotes" type="int"/>
```

Figure 17.65. Services.vsmx



The WSDL description is a standards-based description of the Web Services available from /mysoap. The VSMX page is a Virtuoso generated test page allowing you to test SOAP services. This feature should improve your development time.

See Also:
VSMX ; SOAP ; WSDL .

17.13.2. XML Query Templates

Virtuoso XML templates allow execution of SQL/XML queries over HTTP to obtain an XML document in response and/or perform some operation in the database using updategrams . XML templates can be executed from within Virtuoso procedure language using the `xml_template()` function. XML templates support two types of action: SQL based or updategram based. SQL query based templates must contain the FOR XML clause used in a SELECT statement and hence cannot update the database. Updates to the database can only occur from an updategram. The XML document returned from calling an XML template can be served either raw, or transformed using XSLT.

XML templates provide quick easy access to results from a SQL query as usual, but now this can be saved to a file. The results are not saved, just the query definition. You can use this feature to rapidly produce dynamic reports that can potentially be rendered in different ways by providing an alternate stylesheet. The report can be refined on the fly by providing parameters for the query. The output is reachable via HTTP directly by providing the URL to the template.

See Also:
The XML Templates Section

XML Templates can also be published just like normal store procedures. This is achieved by using a PL wrapper around the XML template. Then the store procedure is published in the normal way.

Stylesheets transformations with be inactive for published XML templates invoked from SOAP.

See Also:
The Publishing Stored Procedures Section above for a further description of publishing XML Templates.

17.13.3. Publishing VSE's as Web Services

The Virtuoso distribution includes the sample VSE, bif_sample.c. It is thus possible to create a function such as:

```
.....
static caddr_t
bif_hello_world (caddr_t * qst, caddr_t * err_ret, state_slot_t ** args)
{
  return box_dv_short_string ("Hello world.");
}
....
```

Then declare it in the init_func() by adding the following code:

```
...
  bif_define_typed ("hello_world", bif_hello_world, &bt_any);
...
```

The next step is creating a stored procedure that calls this function and you are back to publishing a Virtuoso stored procedure again, as in the above section.

```
create procedure BIF_HELLO_WORLD () { return hello_world (); };
```

See Also:
The C Interface Chapter

17.14. Testing Web Published Web Services

Virtuoso provides a mechanism for testing SOAP messages instantly. This mechanism is the Virtuoso Service Module for XML (VSMX) - an automatically generated test page for published web services. The VSMX file is generated at the same time the WSDL file is generated. A VSMX file is a SOAP operations test page generated based on the descriptions of a WSDL file. The VSMX file has the extension .vsmx, as opposed to the .wsdl extension of the WSDL file, and can be accessed similarly.

VSMX pages give instant access to published SOAP services for testing. Any newly created service will need testing which normally means writing more code to call the service, supply parameters, retrieve the result, display the results in some manor, etc. The VSMX feature of Virtuoso eliminates this otherwise repetitive process by automatically maintaining a test page for SOAP messages: the VSMX file. This greatly speeds up the development and testing cycle. The beauty of this is that this is automated, even Microsoft's ASMX file requires that you code the services descriptions before seeing the results.

Web Service developers would use VSMX to forego the test bed creation step in light of automatic test page generation. Project managers can easily keep track of their developers progress by checking the test page periodically.

Since WSDL descriptions by themselves are not very pleasing to the human eye, the VSMX test page can be used to preview services available on a Virtuoso server, or from a remote source using SOAP/WSDL Proxying . Even experienced developers will find these easier to read than WSDL files.

Each SOAP enabled virtual directory will have a WSDL and VSMX file available. The VSMX file is a test page for the SOAP operations. This test page can be found as simply as the WSDL can be found. From the WSDL chapter we have seen that for every SOAP enabled virtual directory you automatically get the file:

```
http://[host:port]/[SOAP Virtual Directory]/services.wsdl
```

likewise you also get:

```
http://[host:port]/[SOAP Virtual Directory]/services.vsmx
```

You simply point your web browser to this file for the test page. The demo database contains samples that can be found as:

```
http://[host:port]/SOAP/services.vsmx
```

Figure 17.66. VSMX Test Page

The page shows all the operations that are available. Click on the link that corresponds to the operation to test and the services details and test facilities will also appear, allowing you to supply input parameters making the SOAP call.

Once executed the result of SOAP call will be rendered using the Virtuoso XSLT processor and built-in style sheet, and appended to bottom of the services details.

In addition to the simple types: numbers and strings, more complex types such as arrays and structures can be specified as input parameters. For an array, each value must be specified on a separate row in the input text field. For structures the names of structure members will be displayed, following by the equals sign, the values of the structures members can then be input after the equals sign. Arrays of arrays or structures of arrays cannot be used on the test page.

The first comment line of a stored procedure in the format:

```
--## [comment]
```

will be included in the WSDL description of the SOAP message for that procedure when generating the WSDL file.

17.15. BPEL Reference

Introduction. Business Process Execution Language for Web Services (called BPEL4WS or simply BPEL in the rest of this chapter) provides a means of specifying interactions between web services for accomplishing a potentially long running business task.

Used terms .

- **WSDL .** Web Services Description Language as described in the corresponding W3C proposed recommendation. This is a notation for declaring services and the types of data they accept and produce. Also the term may be used to refer to the document containing a WSDL description.
- **partner .** A service or application which interacts with a BPEL process.
- **(BPEL) script .** A document containing BPEL compatible XML constructs.
- **(BPEL process) instance .** An instance of a BPEL process, can be running, suspended, aborted or finished.
- **activity .** An activity is a building block of a BPEL script. Receiving data, invoking other web services, programming language like control structures are all examples of activities.
- **portType .** Container for set of abstract operations see below.
- **operation .** An abstract black-box that have an XML input or/and output. Services potentially supports four types of operations: one-way, request-response, notification and solicit-response; this depends of input and/or output allowed and what is their order. The BPEL4WS uses one-way and request-response operations. Operations are grouped in 'ports' which define what operations a particular service supports.
- **message .** An abstract XML fragment that is used for operation input and output. In particular this is used inside SOAP Envelope and for value of the BPEL variables. Also the WSDL specification specifies how abstract messages will be used as concrete messages using the SOAP protocol and which encoding will be used.

17.15.1. Activities

Common attributes and elements

Every BPEL activity accepts the following standard attributes: 'name', 'joinCondition' and 'suppressJoinFailure'. The 'name' attribute is an optional unique within the script NCName (as defined in the XMLSchema) identifying the activity. The 'joinCondition' is a boolean expression which determine how incoming links' state will be tested, by default this is 'true()' which means the logical OR of the states of incoming links. The 'suppressJoinFailure' is a flag to raise or not exception if join condition fails, by default this is 'no'.

The common elements for every activity are 'target' and 'source'. These elements are used together with links and used to designate which link the current activity depends on or for which one it is a prerequisite. See also section for 'flow' activity below.

receive

This allows the business process to do a blocking wait for a particular message to arrive.

A BPEL process instance is created by a receive activity with the createInstance attribute set to true.

partnerLink
: name of a partner declared in the script from which the process is to receive a message.

portType
: name of the 'port' as declared in corresponding WSDL file.

operation
: name of the operation

variable
: name of the variable to which the received message will be assigned.

createInstance
: used to make instance of the BPEL process and start its execution.

```
<receive partnerLink="ncname" portType="qname" operation="ncname"
    variable="ncname"? createInstance="yes|no"?
    standard-attributes>
    standard-elements
    <correlations>?
        <correlation set="ncname" initiate="yes|no"?/>+
    </correlations>
</receive>
```

reply

allows the business process to send a message in reply to a message that was received through a <receive>

partnerLink
: name of a partner declared in the script to which to send a message.

portType
: name of the 'port' as declared in corresponding WSDL file.

operation
: name of the operation

variable
: name of the variable whose value will be used as output message.

faultName

```
<reply partnerLink="ncname" portType="qname" operation="ncname"
    variable="ncname"? faultName="qname"?
    standard-attributes>
    standard-elements
    <correlations>?
        <correlation set="ncname" initiate="yes|no"?/>+
    </correlations>
</reply>
```

invoke

This allows the business process to invoke a one-way or request-response operation on a portType offered by a partner. When both 'inputVariable' and 'outputVariable' are specified this means request-response operation will be performed. Please note that the operation is defined primarily in the partner's WSDL.

partnerLink
: name of a partner declared in the script to who send a message and optionally receive a response.

portType
: name of the 'port' as declared in corresponding WSDL file.

operation
: name of the operation to invoke

inputVariable
: name of the variable whose value will be used as the message to the partner.

outputVariable
: name of the variable to which the response will be assigned.

```
<invoke partnerLink="ncname" portType="qname" operation="ncname"
    inputVariable="ncname"? outputVariable="ncname"?
    standard-attributes>
    standard-elements
    <correlations>?
        <correlation set="ncname" initiate="yes|no"?
            pattern="in|out|out-in"/>+
    </correlations>
    <catch faultName="qname" faultVariable="ncname"?>*
        activity
```

```
        </catch>
        <catchAll>?
            activity
        </catchAll>
        <compensationHandler>?
            activity
        </compensationHandler>
    </invoke>
```

assign

can be used to update the values of variables with new data. An <assign> construct can contain any number of elementary assignments (copy sub-elements).

```
    <assign standard-attributes>
        standard-elements
        <copy>+
            from-spec
            to-spec
        </copy>
    </assign>
```

from-spec

This represents in <copy> the right part of the assignment.

```
    <from variable="ncname" part="ncname"? query="xpath-expression"?/>
    <from partnerLink="ncname" endpointReference="myRole|partnerRole"/>
    <from variable="ncname" property="qname"/>
    <from expression="general-expr"/>
    <from>literal value</from>
```

to-spec

This represents in <copy> the l-value of the assignment.

```
    <from variable="ncname" part="ncname"? query="xpath-expression"?/>
    <from partnerLink="ncname" endpointReference="myRole|partnerRole"/>
    <from variable="ncname" property="qname"/>
```

throw

generates a fault from inside the business process

faultName
: the fault code to be thrown.

faultVariable

```
    <throw faultName="qname" faultVariable="ncname"? standard-attributes>
        standard-elements
    </throw>
```

terminate

Used to immediately terminate the execution of a business process instance.

```
    <terminate standard-attributes>
        standard-elements
    </terminate>
```

wait

Wait until a given point in time or for a specified duration.

for
: an duration expression as defined in XMLSchema (for example PT10S)

until
: an date time expression as defined in XMLSchema

```
<wait (for="duration-expr" | until="deadline-expr") standard-attributes>
    standard-elements
</wait>
```

empty

insert a "no-op" instruction into a business process

```
<empty standard-attributes>
    standard-elements
</empty>
```

sequence

define a collection of activities to be performed sequentially in lexical order

```
<sequence standard-attributes>
    standard-elementsactivity+
</sequence>
```

switch

Select exactly one branch of activity from a set of choices

case
: the branch which will be executed if 'condition' attribute returns true.

otherwise
: will be executed if all 'case' conditions are evaluated to false.

```
<switch standard-attributes>
    standard-elements
    <case condition="bool-expr">+
        activity
    </case>
    <otherwise>?
        activity
    </otherwise>
</switch>
```

while

Repeat activity while a condition is true.

condition
: an XPath expression which will be evaluated every time before contained activities. If this evaluates to false the loop finishes.

```
<while condition="bool-expr" standard-attributes>
    standard-elementsactivity
</while>
```

pick

This blocks and waits for a suitable message to arrive or for a time-out expire. When one of the events specified in the body of the pick occurs the pick completes. Only one of the activities in the body of the pick will actually take place.

createInstance

This is an alternative of the 'receive' to make a new process instance. And can be expressed as: 'pick' plus 'onMessage' equals to 'receive' activity.

```
<pick createInstance="yes|no"? standard-attributes>
    standard-elements
    <onMessage partnerLink="ncname" portType="qname"
        operation="ncname" variable="ncname"?>+
        <correlations>?
            <correlation set="ncname" initiate="yes|no"?/>+
        </correlations>
        activity
    </onMessage>
    <onAlarm (for="duration-expr" | until="deadline-expr")>*
        activity
    </onAlarm>
</pick>
```

onMessage

This is to wait for an incoming message from given partner.

The attributes are the same as for receive activity

onAlarm

This will register a time-wait object whose expiration will trigger the pick. Obviously no more than one can be specified in a given 'pick'

The attributes are the same as for the wait activity

scope

defines a nested activity with its own associated variables, fault handlers, and compensation handler

variableAccessSerializable

```
<scope variableAccessSerializable="yes|no" standard-attributes>
    standard-elements
    <variables/>?
    <correlationSets/>?
    <faultHandlers/>?
    <compensationHandler/>?
    <eventHandlers/>?
    activity
</scope>
```

flow

Specifies one or more activities to be performed concurrently.

```
<flow standard-attributes>
    standard-elements
    <links>?
        <link name="ncname"/>+
    </links>
    activity+
</flow>
```

compensate

This is used to invoke compensation on an inner scope that has already completed normally. This construct can be invoked only from within a fault handler or another compensation handler

```
<compensate scope="ncname"? standard-attributes>
```

```
    standard-elements
</compensate>
```

compensationHandler

```
<compensationHandler>?
    activity
</compensationHandler>
```

faultHandlers

```
<faultHandlers>?
    <catch faultName="qname"? faultVariable="ncname"?>*
        activity
    </catch>
    <catchAll>?
        activity
    </catchAll>
</faultHandlers>
```

catch

This container will be executed whose 'faultName' attribute value matches the thrown fault.

catchAll

This container will be executed if non of the 'catch' containers are matched, so if no 'catchAll' is specified, the contained scopes will be compensated and the fault will be re-thrown to the outer scope.

eventHandlers

This is a container for events, thus it can contain onMessage or/and onAlarm events. Semantic of the onMessage and onAlarm is same as defined in pick activity, the difference is that here these can be executed asynchronously while the events of the defining scope are running. The events in an event handler section may thus interrupt the execution of the body of the scope. This feature can be used for supporting asynchronous cancel messages or timeouts imposed on a whole sequence of operations..

```
<eventHandlers>?
    <onMessage partnerLink="ncname" portType="qname"
        operation="ncname"
        variable="ncname"?>*

        <correlations>?
            <correlation set="ncname" initiate="yes|no"/>+
        </correlations>
        activity
    </onMessage>
    <onAlarm for="duration-expr"? until="deadline-expr"?>*
        activity
    </onAlarm>
</eventHandlers>
```

exec

This is a specific extension of the Virtuoso BPEL implementation. The exec activity allows executing SQL code from inside a BPEL process without having to define a distinct SOAP service for this.

binding
: only "SQL", "JAVA" and "CLR" are currently permitted.

```
<bpelv:exec binding="SQL">
    insert into dummy values ('hello world');
</bpelv:exec>
```

SQL execution

The additional procedures BPEL.BPEL.setVariableData and BPEL.BPEL.getVariableData allow manipulating BPEL variables. See SQL API for details.

All errors occurring during the SQL execution are translated into BPEL errors.

Java execution

Configuration

First of all, the java support in BPEL4WS is available only for java enabled virtuoso servers. In order to enable java support the following administration steps need to be taken:

- 1. Make java compiler (javac) available for virtuoso server
- 2. Enable "system" call by setting the AllowOSCalls? parameter in virtuoso.ini to 1.
- 3. Add "classlib" directory to CLASSPATH

After the BPEL4WS package has been installed, the server needs to be restarted.

The BPEL4WS package creates the classlib directory in the server's root with BpelVarsAdaptor?.class. This directory and this file are necessary for the proper operation of the Java interface.

Using java code in BPEL4WS

There are two elements which support java execution in BPEL4WS scripts: <bpelv:exec binding="JAVA" import?> (namespace is "http://www.openlinksw.com/virtuoso/bpel") and the Oracle compatible form <bpelx:exec import?> (namespace is "http://schemas.oracle.com/bpel/extension"). These tags are equal.

If the tag contains the "import" attribute the appropriate package will be imported. Example:

```
<bpelx:exec import="java.util.*"/>
```

Otherwise, the included code will be executed. It is suggested to use <![CDATA[statement for representing the java code. Example:

```
<bpelx:exec>
   System.out.println("getVariableData returned: " + getVariableData ("request", "req_payload", "/v:destR

   setVariableData ("res",
         "repl_payload",
         "/v:destResponse/v:country",
         "KZ");
</bpelx:exec>
```

The two "functions" are available for accessing script variables: setVariableData and getVariableData (analogs of the xpath functions with same names).

```
void setVariableData (String var_name, String part, String query_path, Object value);
```

sets the variable. The value to be set is restricted to be string or integer in this version.

```
getVariableData
```

is a full analog of xpath function getVariableData. See details below.

Note, the activity does not commit changes to the database until it finishes successfully, so if in a case of exception the variables are kept untouched. This also means that deadlocks can occur.

The communication errors must be handled and processed in the java code itself.

If some unhandled exception occurs in java code, it will be translated to a BPEL error. See details in the next section.

Errors

Two types of errors can be signalled:

- 1. upload time exceptions
- 2. runtime exceptions

The first type exceptions are related to configuration and java syntax errors. These are as follows:

- 1. [BPELX] The "system" call is disabled, it is needed for use java code in BPEL4WS scripts. * This error is signalled when the "AllowOSCalls" is set to 0. So the engine can not call java compiler. It One must set this parameter in ini file to 1 and restart the server to resolve this issue.
- 2. [BPELX] Compilation of java code for NAME failed. Try to call "javac NAME.java" for details. * The compilation of generated class for the activity has failed. Try to call the suggested operation from command prompt with the same PATH and CLASSPATH global variables to see the javac output.

Runtime errors not handled by java method itself are translated to the BPEL fault in the following form:

```
<bpws:javaFault sqlState="SQLSTATE">
 error message
</bpws:javaFault>
```

this error can be handled by BPEL fault handler. Here is an example:

```
...
    <receive partnerLink="caller" portType="v:dest" operation="check_dest" variable="request" createInsta
    <scope>
      <faultHandlers>
        <catch faultName="bpws:javaFault" faultVariable="error">
          <assign>
            <copy>
              <from variable="error" query="/javaFault"/>
              <to variable="res" part="repl_payload" query="/destResponse/country"/>
            </copy>
          </assign>
        </catch>
      </faultHandlers>
      <bpelv:exec binding="JAVA" name="invokeSomething"><![CDATA[
   System.out.println("Executing Java exec in BPEL");
   System.out.println("getVariableData returned: " + getVariableData ("request", "req_payload", "/v:destR

   setVariableData ("res",
         "repl_payload",
         "/destResponse/country",
         "UK");
   if (true)
     throw (new Exception("test"));
   ]] >
       </bpelv:exec>
     </scope>
    <reply partnerLink="caller" portType="v:dest" operation="check_dest" variable="res"/>
...
```

In this code the handler for java exception is set:

```
...
        <catch faultName="bpws:javaFault" faultVariable="error">
          <assign>
            <copy>
              <from variable="error" query="/javaFault"/>
              <to variable="res" part="repl_payload" query="/destResponse/country"/>
            </copy>
          </assign>
        </catch>
...
```

The handler takes the error message from the variable which holds it ("error") and puts it into the response variable.

BPEL variables accessors

```
void setVariableData (String var_name, String part, String query_path,
            Object value)
```

- changes BPEL variable named var_name. The "part" is a part of the message stored in the variable. "query_path" selects the data in the variable to be changed. The "value" can be only string or integer, so if several subdata need to be changed the setVariableData must be called several times. If the "part" and "query_path" must be ignored their can be passed as empty strings ("").

```
String getVariableData (String var_name, String part, String query)
```

- returns selected data from the part named by "part" argument of variable named by "var_name" argument. If the selection failed the NULL is returned.

Special variables

There are two special variables: "variables" and "xmlnss_pre". They can be changed in the java code, in this case the behaviour of the engine is unpredictable. So, it is strongly recommended do not use these variables.

JavaMail usage example

```
<?xml version="1.0"?>
<process xmlns:jsm="urn:java:sendMail"
        xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
        name="javaSendMail"
        targetNamespace="urn:java:sendMail"
        xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
        xmlns:bpelv="http://www.openlinksw.com/virtuoso/bpel">

  <bpelv:exec binding="JAVA" import="java.io.*"/>
  <bpelv:exec binding="JAVA" import="java.net.InetAddress"/>
  <bpelv:exec binding="JAVA" import="java.util.Properties"/>
  <bpelv:exec binding="JAVA" import="java.util.Date"/>
  <bpelv:exec binding="JAVA" import="javax.mail.*"/>
  <bpelv:exec binding="JAVA" import="javax.mail.internet.*"/>
  <bpelv:exec binding="JAVA" import="com.sun.mail.smtp.*"/>

  <partnerLinks>
    <partnerLink name="caller" partnerLinkType="jsm:dest"/>
  </partnerLinks>

  <variables>
    <variable name="request" messageType="jsm:destRequestMessage"/>
    <variable name="res" messageType="jsm:destResponseMessage"/>
  </variables>
  <sequence>
    <receive partnerLink="caller" portType="jsm:dest" operation="send" variable="request" createInstance=
    <bpelv:exec binding="JAVA" name="sendMail"><![CDATA[
        String  to, subject = null, from = null;
        String mailhost = (String) getVariableData ("request", "req_payload", "/jsm:destRequest/jsm:mailh
        String mailer = "OpenLink Virtuoso[BPEL script]";
        try {
                to = (String) getVariableData ("request", "req_payload", "/jsm:destRequest/jsm:to");
                subject = (String) getVariableData ("request", "req_payload", "/jsm:destRequest/jsm:subje

                Properties props = System.getProperties();
                props.put("mail.smtp.host", mailhost);
                Session session = Session.getInstance(props, null);

                Message msg = new MimeMessage(session);
                if (from != null)
                        msg.setFrom(new InternetAddress(from));
                else
```

```
                msg.setFrom();
            msg.setRecipients(Message.RecipientType.TO,
                                InternetAddress.parse(to, false));
            msg.setSubject(subject);
            msg.setText ((String)getVariableData ("request", "req_payload", "/jsm:destRequest/jsm:tex

            msg.setHeader("X-Mailer", mailer);
            msg.setSentDate(new Date());
            SMTPTransport t = (SMTPTransport)session.getTransport("smtp");
            try {
                t.connect();
                t.sendMessage(msg, msg.getAllRecipients());
            } finally {
                System.out.println("Response: " +
                                        t.getLastServerResponse());
                t.close();
            }

            System.out.println("\nMail was sent successfully.");
            setVariableData ("res",
                    "repl_payload",
                    "/jsm:destResponse/jsm:status",
                    "OK");

        } catch (Exception e) {
            String err_str = "Failed: " + e.toString();
            e.printStackTrace();
            if (e instanceof SendFailedException) {
                MessagingException sfe = (MessagingException)e;
                if (sfe instanceof SMTPSendFailedException) {
                    SMTPSendFailedException ssfe =
                                (SMTPSendFailedException)sfe;
                    System.out.println("SMTP SEND FAILED:");
                    System.out.println(ssfe.toString());
                    System.out.println("  Command: " + ssfe.getCommand());
                    System.out.println("  RetCode: " + ssfe.getReturnCode());
                    System.out.println("  Response: " + ssfe.getMessage());
                } else {
                        System.out.println("Send failed: " + sfe.toString());
                }
                Exception ne;
                while ((ne = sfe.getNextException()) != null &&
                        ne instanceof MessagingException) {
                    sfe = (MessagingException)ne;
                    if (sfe instanceof SMTPAddressFailedException) {
                        SMTPAddressFailedException ssfe =
                                        (SMTPAddressFailedException)sfe;
                        System.out.println("ADDRESS FAILED:");
                        System.out.println(ssfe.toString());
                        System.out.println("  Address: " + ssfe.getAddress());
                        System.out.println("  Command: " + ssfe.getCommand());
                        System.out.println("  RetCode: " + ssfe.getReturnCode());
                        System.out.println("  Response: " + ssfe.getMessage());
                    } else if (sfe instanceof SMTPAddressSucceededException) {
                        System.out.println("ADDRESS SUCCEEDED:");
                        SMTPAddressSucceededException ssfe =
                                        (SMTPAddressSucceededException)sfe;
                        System.out.println(ssfe.toString());
                        System.out.println("  Address: " + ssfe.getAddress());
                        System.out.println("  Command: " + ssfe.getCommand());
                        System.out.println("  RetCode: " + ssfe.getReturnCode());
                        System.out.println("  Response: " + ssfe.getMessage());
                    }
                }
            }
            setVariableData ("res",
                        "repl_payload",
                        "/jsm:destResponse/jsm:status",
                        err_str);
        }

        ]] ></bpelv:exec>
    <reply partnerLink="caller" portType="jsm:dest" operation="send" variable="res"/>
```

```
  </sequence>
</process>
```

and support WSDL file:

```
<?xml version="1.0"?>
<definitions name="javaSendMail"
  targetNamespace="urn:java:sendMail"
  xmlns:jsm="urn:java:sendMail"
  xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <schema attributeFormDefault="qualified" elementFormDefault="qualified"
      targetNamespace="urn:java:sendMail"
      xmlns="http://www.w3.org/2001/XMLSchema">
      <element name="destRequest">
        <complexType>
          <sequence>
            <element name="mailhost" type="string"/>
            <element name="to" type="string"/>
            <element name="subject" type="string"/>
            <element name="text" type="string"/>
          </sequence>
        </complexType>
      </element>
      <element name="destResponse">
        <complexType>
          <sequence>
            <element name="status" type="string"/>
          </sequence>
        </complexType>
      </element>
    </schema>
  </types>
  <message name="destRequestMessage">
    <part name="req_payload" element="jsm:destRequest"/>
  </message>
  <message name="destResponseMessage">
    <part name="repl_payload" element="jsm:destResponse"/>
  </message>
  <portType name="dest">
    <operation name="send">
      <input  message="jsm:destRequestMessage" />
      <output message="jsm:destResponseMessage"/>
    </operation>
   </portType>
   <plnk:partnerLinkType name="dest">
     <plnk:role name="destProvider">
       <plnk:portType name="jsm:dest"/>
     </plnk:role>
   </plnk:partnerLinkType>
</definitions>
```

.NET CLR execution

Configuration

The CLR is available only for CLR enabled Virtuoso servers. In order to configure the server for CLR support in BPEL4WS engine the following administration steps need to be taken:

- 1. Initiate "CLRAssembliesDir" configuration entry ("Directory where .NET CLR assemblies must be stored") by the path where "virt_bpel4ws.dll", "virtclr.dll" etc are stored.

Using the C# code in BPEL4WS

The tag for supporting CLR is <exec binding="CLR" [using | ref]>.

<exec> with "using" attribute relates to "using" directive in C# code.

```
<bpelv:exec binding="CLR" using="System"/>
```

in BPEL file is a substitution of

```
using System;
```

statement in the C# code.

To import assemblies (with a manifest) the <exec> element with "ref" attribute must be used:

```
<bpelv:exec binding="CLR" ref="metad1.dll"/>
<bpelv:exec binding="CLR" ref="d:\\myassemblies\\sms_service.dll"/>
```

If the tag does not contain these attributes then the content of the tag is treated as C# code. It is suggested to use CDATA sections for the code. Example:

```
<bpelv:exec binding="CLR" name="invokeSomething"><![CDATA[

        Console.WriteLine("Executing CLR assembly in BPEL");
        Console.WriteLine("getVariableData returned: " + getVariableData ("request", "req_payload", "/tns

        setVariableData ("res",
                         "repl_payload",
                         "/destResponse/country",
                         "UK");
]] ></bpelv:exec>
```

Two "functions" are available for accessing script variables: setVariableData and getVariableData (analogs of the xpath functions with same names).

```
void setVariableData (String var_name, String part, String query_path, Object value);
```

sets the variable. The value to be set is restricted to be string or integer in this version.

```
Object getVariableData (String var_name, String part, String query)
```

is a full analog of xpath function getVariableData. See details below.

Note, the activity does not commit changes to the database until it finishes successfully, so in a case of exception the variables are kept untouched. This also means that deadlocks can not occur during C# code execution (naturally, this statement is not true if the code contains direct invocation of SQL through the "in-process" .NET provider).

If the some unhandled exception occurred in C# code, it will be translated to a BPEL error. See details in the next section.

Runtime Errors

Runtime errors not handled by CLR itself are translated to the BPEL fault in the following form:

```
<bpws:clrFault sqlState="SQLSTATE">
 error message

</bpws:javaFault>
```

this error can be handled by BPEL fault handler. Here is an example:

```
...
    <scope>
      <faultHandlers>
        <catch faultName="bpws:clrFault" faultVariable="error">
          <assign>
            <copy>
```

```
              <from variable="error" query="/clrFault/text()"/>
              <to variable="res" part="repl_payload" query="/clr:destResponse/clr:status"/>
            </copy>
          </assign>
        </catch>
      </faultHandlers>
      <bpelv:exec binding="CLR" name="Hotmail"><![CDATA[
            String login, passwd;
               login = (String) getVariableData ("request", "req_payload", "/clr:destRequest/clr:login")
               passwd = (String) getVariableData ("request", "req_payload", "/clr:destRequest/clr:passwo
               Console.WriteLine ("user:" + login + " password:" + passwd);

               HotmailUsage usage = new HotmailUsage();
               Console.WriteLine ("Result: " + usage.Call_Hotmail (login, passwd));

               setVariableData ("res",
                       "repl_payload",
                       "/clr:destResponse/clr:status",
                       "OK");
        ]] ></bpelv:exec>
        </scope>
    <reply partnerLink="caller" portType="clr:dest" operation="send" variable="res"/>
...
```

In this code the handler for CLR exception is set:

```
...
        <catch faultName="bpws:clrFault" faultVariable="error">
          <assign>
            <copy>
              <from variable="error" query="/clrFault/text()"/>
              <to variable="res" part="repl_payload" query="/clr:destResponse/clr:status"/>
            </copy>
          </assign>
        </catch>
...
```

The handler takes the error message from the variable which holds it ("error") and puts it into the response variable.

BPEL variable accessors

The name and the signature of BPEL accessors for CLR are fully equal to the java accessors. See java section for details.

Special variables

The special variables for CLR are the same as for Java. See java section for details.

CLR Sample

```
<?xml version="1.0"?>
<process xmlns:clr="urn:clr:Hotmail"
        xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
        name="clrHotmail"
        targetNamespace="urn:clr:Hotmail"
        xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
        xmlns:bpelv="http://www.openlinksw.com/virtuoso/bpel">

  <bpelv:exec binding="CLR" using="HotmailUsageChecker"/>
  <bpelv:exec binding="CLR" ref="l:\\distrib_3_0\\bin\\Hotmail.dll"/>
  <bpelv:exec binding="CLR" using="System"/>

  <partnerLinks>
    <partnerLink name="caller" partnerLinkType="clr:dest"/>
  </partnerLinks>

  <variables>
    <variable name="request" messageType="clr:destRequestMessage"/>
    <variable name="res" messageType="clr:destResponseMessage"/>
  </variables>
```

```
  <sequence>
    <receive partnerLink="caller" portType="clr:dest" operation="send" variable="request" createInstance=
    <scope>
      <faultHandlers>
        <catch faultName="bpws:clrFault" faultVariable="error">
          <assign>
            <copy>
              <from variable="error" query="/clrFault/text()"/>
              <to variable="res" part="repl_payload" query="/clr:destResponse/clr:status"/>
            </copy>
          </assign>
        </catch>
      </faultHandlers>
      <bpelv:exec binding="CLR" name="Hotmail"><![CDATA[
             String login, passwd;
                login = (String) getVariableData ("request", "req_payload", "/clr:destRequest/clr:login")
                passwd = (String) getVariableData ("request", "req_payload", "/clr:destRequest/clr:passwo
                Console.WriteLine ("user:" + login + " password:" + passwd);

                HotmailUsage usage = new HotmailUsage();
                Console.WriteLine ("Result: " + usage.Call_Hotmail (login, passwd));

                setVariableData ("res",
                        "repl_payload",
                        "/clr:destResponse/clr:status",
                        "OK");
        ]]></bpelv:exec>
        </scope>
    <reply partnerLink="caller" portType="clr:dest" operation="send" variable="res"/>
  </sequence>
</process>
```

17.15.2. Protocol Support

The Virtuoso BPEL implementation supports WS Security and WS Reliable Messaging. These protocols may be enabled and configured for individual partner links. The WS-Addressing protocol is automatically used when the partner has such capabilities.

The Partner link options are stored in XML format in the 'BPEL.BPEL.partner_link_init' table in 'bpl_opts' column. See the document format and table description below in this chapter.

WS-Addressing (WSA)

WS-Addressing is transparently handled via WSDL description and request of the partner(s). If the partner has WSA headers exposed then the corresponding operation will be invoked with WSA headers. If the partner uses WSA to call the BPEL service then the service will respond to the partner with WSA headers included.

The benefit of using WSA consists in transparently handled message correlation via "MessageID" and "RelatesTo" headers. Thus no explicit message correlation needs to be specified when the partner is WSA capable (see LoanFlow demo in tutorials to see how this works). Also when the reply is asynchronous the return path can be handled via the "ReplyTo" WSA headers.

As WSA has different revisions that are supported by different implementations (partners), the version of the protocol is configurable via wsOptions/addressing element (see below).

The 'wsa' partner link option can be used in API functions to set or retrieve the value of the wsOptions/addressing.

WS-Security (WSS)

There are several cases where privacy and confidentiality must be observed , this is especially true when business processes involve payment systems, personal information etc. In those cases a partner may, depending on Web Service policy, require secure messages (signed, encrypted or both). Such partners usually do not expose (for many reasons) in WSDL the WSS headers, hence this must be configured by the BPEL administrator per partner.

In order for the BPEL server to be able to sign and encrypt messages keys have to be uploaded into the BPEL user repository. This can be done with the USER_KEY_LOAD() function or using the BPEL GUI (Partner Link properties).

It is important to know the following basics:

- The private key is needed in order to decrypt or sign messages.
- The partner public key is needed to encrypt messages or verify signatures.

These keys have to be described in wsOptions/security/key and wsOptions/security/pubkey in the 'Partner link options' XML document. Note that in the options document are stored only the names of the keys, as the keys themselves are registered in the BPEL user key repository.

Note: Use the 'wss-priv-key' and 'wss-pub-key' partner link option in API functions to set or change the value of them.

The encryption and signing is independently configurable for outbound or inbound messages. This is settable via the following options:

- wsOptions/security/in/encrypt - 'Optional': inbound message MAY be encrypted, 'Mandatory': inbound message MUST be encrypted ('wss-in-encrypt' option in API)
- wsOptions/security/in/signature - same as encryption but for XML signature ('wss-in-signature' option in API)
- wsOptions/security/in/keys - this is a list of keys (key names) that are trusted; only SOAP messages signed with one of these keys will be accepted. If this list is empty, all parties will be accepted unless signature is invalid or message can not be decrypted. ('wss-in-signers' option in API)
- wsOptions/security/out/encrypt - type of the session key for encryption 3DES, AES128, AES192 or AES256. ('wss-out-encrypt-key' option in API)
- wsOptions/security/out/signature - 'Default': the WSS default template (see WS-Security chapter for details), 'Custom': in this case 'function' attribute must refer to a PL procedure returning XML Signature template. ('wss-out-signature-type' and 'wss-out-signature-function' options in API)

WS-Reliable Messaging (WSRM)

The WSRM is applicable for those partners which can be invoked asynchronously. For both inbound or outbound (from the point of view of the process) WSRM can be enabled. The options are settable in wsOptions/delivery/in or wsOptions/delivery/out ('wsrm-in-type' and 'wsrm-out-type' options in API) respectively.

- ExactlyOnce - message will be delivered only once to the destination.
- InOrder - messages will be delivered in the same order as they are sent from the script.

HTTP security

Some partner services may require HTTPS or/and HTTP authentication instead of using WS-Security protocol to ensure SOAP message privacy.

When a partner's endpoint URL contains HTTPS scheme (like https://) the request and response operations will be made with HTTP over SSL (HTTPS protocol). In this case no additional options to the partner links are needed.

In case when a partner service needs a HTTP authentication, the user name and password (for that service) needs to be configured for the given partner link. These are settable via wsOptions/security/http-auth ('http-auth-uid' and 'http-auth-pwd' options in API) element in partner link options. Note that only basic HTTP authentication is supported.

Partner link options format

```
<wsOptions>
    <!-- WSA revision namespace -->

    <addressing version="http://schemas.xmlsoap.org/ws/2003/03/addressing|http://schemas.xmlsoap.org/ws/2
  <!-- WSS options -->

  <security>
      <!-- basic HTTP Authentication -->

      <http-auth username="" password=""/>
      <!-- WSS keys -->

      <key name="[Private Key]"/>
      <pubkey name="[Partner public key]"/>
    <in>
```

```
      <encrypt type="NONE|Optional|Mandatory"/>
      <signature type="NONE|Optional|Mandatory" />
      <keys>
          <!-- trusties -->

          <key name="[Trusted Public Key]"/>
          ...
      </keys>
    </in>
    <out>
        <encrypt type="NONE|3DES|AES128|AES192|AES256"/>
        <signature type="NONE|Default|Custom" function="[PL procedure name]"/>
    </out>
  </security>
  <!-- WSRM options -->

  <delivery>
    <in  type="NONE|ExactlyOnce|InOrder"/>
    <out type="NONE|ExactlyOnce|InOrder" />
  </delivery>
</wsOptions>
```

Configuring the Partner link options

The partner link options can be configured vi BPEL UI (http://host:port/BPELGUI) or programmatically using `plink_get_option()` and `plink_set_option()` functions. It is possible to update the partner link options with SQL update statement using the configuration (described above), but use of the BPEL UI or API functions is recommended.

Dynamic Partner Links

The 'assign' activity allows partner links to be assigned values at run time. This means that partner links can be dynamic and that a process may call partners that were not known at the time of defining the process. Partner links can be defined at run time using the 'EndpointReference' element from the WS-Addressing specification :

```
<xs:element name="EndpointReference" type="wsa:EndpointReferenceType"/>
<xs:complexType name="EndpointReferenceType">
  <xs:sequence>
    <xs:element name="Address" type="wsa:AttributedURI"/>
    <xs:element name="ReferenceProperties" type="wsa:ReferencePropertiesType" minOccurs="0"/>
    <xs:element name="PortType" type="wsa:AttributedQName" minOccurs="0"/>
    <xs:element name="ServiceName" type="wsa:ServiceNameType" minOccurs="0"/>
    <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>
```

Thus it may keep endpoint URL, Port type, Service name and any number of implementation specific options under 'ReferenceProperties'. Also it may keep any other extensible elements and attributes.

The Virtuoso BPEL implementation extends the EndpointReferenceType with element 'wsOptions' under WSA element 'ReferenceProperties' (see previous section for 'wsOptions' element description). Thus it is possible to decide at run time to send a secure reply (WSS) or to use reliable messaging (WSRM). The following excerpt shows BPEL source that assigns WS Security options to a partner link:

```
    ...
    <assign>
        <copy>
            <from>
                <EndpointReference xmlns="http://schemas.xmlsoap.org/ws/2003/03/addressing">
                    <Address>http://securehost/myService</Address>
                    <ReferenceProperties>
                        <wsOptions xmlns="">
                            <addressing version="http://schemas.xmlsoap.org/ws/2003/03/addressing"/>
                            <security>
                                <http-auth username="" password=""/>
                                <key name="myKey"/>
                                <pubkey name="serverKey"/>
```

```
                                <in>
                                    <encrypt type="Optional"/>
                                    <signature type="Optional" />
                                    <keys/>
                                </in>
                                <out>
                                    <encrypt type="AES128"/>
                                    <signature type="Default"/>
                                </out>
                            </security>
                        </wsOptions>
                    </ReferenceProperties>
                </EndpointReference>
            </from>
            <to partnerLink="service"/>
        </copy>
    </assign>
    ...
```

To manipulate the partner link options and endpoint URL we can also use variables declared as element EndpointReference from WS-Addressing schema.

17.15.3. Process lifecycle

A BPEL process may have more than one version during development or maintenance in production environment. This means the process may have a more than one copy on the server where one of these copies is current all others are either obsolete or in the process of being defined . We will call these states further in this section as 'current', 'obsolete' or 'edit' (versions). The link between these versions is the process name which remains the same during process life-cycle.

When a process is compiled for the first time it becomes 'current'. The current version of a process is the one that was most recently compiled without errors. Only a current version of a process can make new instances in response to receiving messages. For a single process name only one version can be current at any one time. The other compiled versions of the process are always obsolete, meaning that new instances cannot be created but that old instances may proceed and complete.

The 'obsolete' version can not crate new process instances, it may only receive messages for its instances which are still running or suspended. In other words the old process instances will continue and any new instances will be instances of the current version.

The 'edit' state means that a new copy of the process is created but it is still not compiled. After successful compilation, a process in the edit state is set into the current state and the formerly current process is set to the the obsolete state.

All process versions may have only one endpoint corresponding to the process. The BPEL administrator MUST take care about endpoints to make sure that clients using the process are compatible with different process versions. This means that if changes in new versions are for example in the logic but not in the process' WSDL then new endpoint is not needed in most cases; furthermore in this case clients will be compatible with new input and output message structure; except in the case where the same messages are used with logically different meaning. So when a new process version involves significant change to messages then the new process should have a new endpoint and WSDL description.

17.15.4. Using virtual directories

To allow a process to make new instances or to receive messages from remote services in asynchronous way it needs an endpoint. Endpoint means an URL that allows to accept and process known SOAP messages.

Every process has a default endpoint which is accessible via the 'http://[host:port]/BPELGUI/bpel.vsp?script=[process_name]' URL. Also it has a default WSDL document describing the process at URL http://[host:port]/BPELGUI/bpel.vsp?script=[process_name]&wsdl. Upon compilation a different virtual directory can be specified for the process. For example if '/Service' is specified the 'http://[host:port]/Service' would be the endpoint URL also in this case the WSDL will be available on 'http://[host:port]/Service?wsdl'.

It is important to note that changing the virtual directory when a process is redefined will force all clients to update their configuration. This is the case when a major changes are made to the process and clients need to be re-linked against new WSDL. So when changes are small there will be no need of a new virtual directory. (See section 'Process life-cycle' above for process versions).

17.15.5. Process archiving

When an instance of a BPEL process is completed it will be archived and purged from system after a configurable time interval. The interval is settable via 'InstanceExpiryDelay' configuration option (see below how to change). The default interval for archiving is one day.

The archive file contains status of the instance which is archived, execution path of the activities when execution is completed, value of the variables and partner links.

The format of the archive file is as follows:

```
<instances>
  <instance
xml:id="[instance id]">
    <status code="[2 - success |3 - error]" error="[error description]"/>
    <execution>
        <node
xml:id="[activity id]" type="[activity type]"/>
        ... more nodes may follow ...
    </execution>
    <variables>
        <variable name="[name of the variable]">... variable value ...</variable>
        ... more variables may follow ...
    </variables>
    <partnerLinks>
      <partnerLink name="[name of the partner link]">
        <EndpointReference>
            <Address>... partner link endpoint ...</Address>
        </EndpointReference>
      </partnerLink>
      ... more partner links may follow ...
    </partnerLinks>
  </instance>
</instances>
```

17.15.6. Configuration parameters

The BPEL engine has global configuration parameters settable via the configuration settings page of its web user interface. These are global and are kept in the BPEL..configuration table as name/value pairs.

- EngineMailAddress - e-mail address to whom will be sent error alerts and reports.
- ErrorAlertSkeleton - e-mail template for error alerts
- ErrorReportSkeleton - e-mail template for error reports
- AlertSubject - Subject for error alert e-mail
- CommonEmailHeader - Header for all BPEL e-mail notifications
- ErrorSubject - Subject for error report e-mail
- InstanceExpiryDelay - how long (hours a completed instance will be kept in the database before purging.
- MailServer - if specified this will be used to send the error alerts and reports. If this is not specified the default (from INI file) mail server will be used.
- Statistics - process statistics flag, if 0 (default) the statistics collection is disabled. When this is 1 the engine will start collecting statistics for new process instances .

17.15.7. Process Statistics

The BPEL process can be set to collect statistics, this can be enabled by turning on the statistics flag (see Configuration section). If the statistics flag is on the following data is available:

- count of instance creations per instance creating partner link
- count/duration/data volume of invokes per invoked synchronous partner link
- count/data volume per asynchronous partner link
- waiting time/count/data volume at non-instance creating receives
- average time to complete for process
- Errors per partner link

- count of created instances
- count of completed instances
- count of instances that could not terminate successfully

17.15.8. Deployment file suitcase format

The relation between a BPEL process and different partner links can be stored in a deployment file. This file contains a description for every partner link mention in the process. The descriptor file simplifies deployment. Entering the URL of this file through the web administration interface will download and register all sources needed for the process compilation.

The format of the deployment file is as follows:

```
    <BPELSuitcase>
        <BPELProcess
xml:id="[process id]" src="[URI of the file containing BPEL script]">
                <partnerLinkBindings>
                        <partnerLinkBinding name="[name of the partner]">
                            <property name="wsdlLocation">[URI of the WSDL for the partner]</property>
                        </partnerLinkBinding>
                        ... more partnerLinkBinding may follow ...
                </partnerLinkBindings>
        </BPELProcess>
    </BPELSuitcase>
```

17.15.9. SQL API

The following API functions are available:

bpel_compile_script_dedup

For detailed description and example use of the function, see bpel_compile_script in the Functions Reference Guide.

bpel_copy_script_dedup

For detailed description and example use of the function, see bpel_copy_script in the Functions Reference Guide.

bpel_get_partner_links_dedup

For detailed description and example use of the function, see bpel_get_partner_links in the Functions Reference Guide.

bpel_instance_delete_dedup

For detailed description and example use of the function, see bpel_instance_delete in the Functions Reference Guide.

bpel_purge_dedup

For detailed description and example use of the function, see bpel_purge in the Functions Reference Guide.

bpel_script_delete_dedup

For detailed description and example use of the function, see bpel_script_delete in the Functions Reference Guide.

bpel_script_obsolete_dedup

For detailed description and example use of the function, see bpel_script_obsolete in the Functions Reference Guide.

bpel_script_source_update_dedup

For detailed description and example use of the function, see bpel_script_source_update in the Functions Reference Guide.

bpel_script_upload_dedup

For detailed description and example use of the function, see bpel_script_upload in the Functions Reference Guide.

bpel_wsdl_upload_dedup

For detailed description and example use of the function, see bpel_wsdl_upload in the Functions Reference Guide.

plink_get_option_dedup

For detailed description and example use of the function, see plink_get_option in the Functions Reference Guide.

plink_set_option_dedup

For detailed description and example use of the function, see plink_set_option in the Functions Reference Guide.

bpel_import_script_dedup

For detailed description and example use of the function, see bpel_import_script in the Functions Reference Guide.

17.15.10. BPEL XPath Functions

bpel_get_var_dedup

For detailed description and example use of the function, see bpel_get_var in the Functions Reference Guide.

bpel_set_var_dedup

For detailed description and example use of the function, see bpel_set_var in the Functions Reference Guide.

xpf_processxquery_dedup

For detailed description and example use of the function, see xpf_processxquery in the Functions Reference Guide.

xpf_processxslt_dedup

For detailed description and example use of the function, see xpf_processxslt in the Functions Reference Guide.

xpf_processxsql_dedup

For detailed description and example use of the function, see xpf_processxsql in the Functions Reference Guide.

17.15.11. Tables

BPEL Engine Tables

```
-- Scripts table, keeps one record per version
create table BPEL.BPEL.script (
        bs_id integer identity,         -- unique id identifying the process
        bs_uri varchar,                 -- obsoleted: script source URI
        bs_name varchar,                -- process name, all versions have same name
        bs_state int,                   -- 0 on, current version, 1 obsolete, 2 edit mode
        bs_date datetime,               -- date of registration
        bs_audit int default 0,         -- audit flag : 1 on, 0 off
        bs_debug int default 0,         -- debug flag
        bs_version int default 0,       -- process version
        bs_parent_id int default null,  -- fk to bs_id of previous process version
        bs_first_node_id int,           -- the first node id in the graph
        bs_pickup_bf varbinary default '\x0',   -- bitmask for resume nodes
        bs_act_num      int,            -- stores the total number of activities
        bs_lpath        varchar default null, -- virtual directory

        -- process statistics
        bs_n_completed int default 0,
```

```
        bs_n_errors int default 0,
        bs_n_create int default 0,
        bs_cum_wait int default 0,
        primary key (bs_id));

-- BPEL and WSDL sources
create table BPEL..script_source
        (
         bsrc_script_id int, -- script id, fk to bs_id of scripts table.
         bsrc_role varchar,  -- one of bpel, bpel-ext, wsdl, deploy, partner-1... partner-n
         bsrc_text long xml, -- source text
         bsrc_url varchar,   -- if this comes from an uri
         bsrc_temp varchar,  -- contains the namespaces info
         primary key (bsrc_script_id, bsrc_role)
)
;

-- Process instances
create table BPEL.BPEL.instance (
        bi_id int identity,             -- global immutable id of instance
        bi_script int,                  -- fk to bs_id from BPEL.BPEL.script
        bi_scope_no int default 0,      -- sequence counter for scope numbers in instance
        bi_state int default 0,
                -- 0, started
                -- 1, suspended (wait for signal)
                -- 2, finished
                -- 3, aborted
        bi_error any,                   -- error
        bi_lerror_handled int,
        bi_last_act     datetime,       -- last activity execution
        bi_started      datetime,       -- start time
        bi_init_oper    varchar,        -- operation that made the instance
        bi_wsa          long xml,       -- WS-Addressing headers
        bi_activities_bf varbinary default '\x0\x0', -- bitmask for each activity is completed or not
        bi_link_status_bf varbinary default '\x0\x0', -- bitmask for link status
        bi_prefix_info varchar default '', -- xpath prefix string
        primary key (bi_id));

-- Initial values (URL etc.) for partner links
create table BPEL.BPEL.partner_link_init (
        bpl_script int,         -- script instance id
        bpl_name varchar,       -- partner link name
        bpl_partner any,        -- url, end point etc serialized
        bpl_role varchar,
        bpl_myrole varchar,
        bpl_type varchar,
        bpl_endpoint varchar,   -- partner service endpoint URL
        bpl_backup_endpoint varchar,
        bpl_wsdl_uri varchar,
        bpl_debug int default 0,-- debug flag
        bpl_opts long xml,      -- partner link options (WS-Security, WS-RM etc.)
        primary key (bpl_script,bpl_name));

-- Runtime values for partner links (run time copy of partner_link_init table)
create table BPEL..partner_link (
        pl_inst int,            -- instance id
        pl_name varchar,        -- partner link name
        pl_scope_inst int,      -- scope instance id
        pl_role int,            -- flag 0 - myRole, 1 - partnerRole
        pl_endpoint varchar,    -- current URL to the partner service
        pl_backup_endpoint varchar, -- second URL to the service for connection error
        pl_debug int default 0, -- debug flag
        pl_opts long xml,       -- partner link options (WS-Security, WS-RM etc.)
        primary key (pl_inst, pl_name, pl_scope_inst, pl_role));

-- Script compilation
create table BPEL.BPEL.graph (
        bg_script_id int,       -- FK to bs_id of BPEL.BPEL.script
        bg_node_id int ,        -- running id in the script, referenced from BPEL.BPEL.waits etc.
        bg_activity BPEL.BPEL.activity, -- UDT representing activity
        bg_childs any,
        bg_parent int,
        bg_src_id varchar,      -- internal use
```

```
        primary key (bg_script_id, bg_node_id));

-- Receive activities waiting for incoming message
create table BPEL.BPEL.wait (
        bw_uid varchar,
        bw_instance integer,  -- instance id
        bw_script varchar,    -- FK reference to bs_name of script table
        bw_script_id int,     -- FK reference to bs_id of script table
        bw_node int,          -- FK reference to bg_node_id of the graph table
        bw_scope  int,
        bw_partner_link varchar, -- the party from which instance waiting a message
        bw_port varchar,         -- the name of the operation which instance wait to receive
        bw_deadline datetime,
        bw_message long varchar default null, -- if instance is occupied and message is already arrived
        bw_state int default 0,               -- flag that bw_message is not null (0 or 1)
        bw_correlation_exp varchar,           -- XPath expression for computing the correlation value fro
        bw_expected_value long varbinary,     -- value of the expected correlation
        bw_message_type int default 0,        -- where to expect the data : 0 - SOAP:Body 1 - SOAP:Header
        bw_start_date datetime,
        primary key (bw_instance, bw_node));

-- Messages which have been arrived but not correlated yet
create table BPEL.BPEL.queue (
        bq_id int identity,     -- unique id
        bq_script int,          -- FK references bs_id from the script table
        bq_ts timestamp,
        bq_state int,           -- state of the Queue item; 0 - not processed
        bq_endpoint varchar,    -- not used
        bq_op varchar,          -- Operation name
        bq_mid varchar,         -- mot used
        bq_message long varchar, -- The incoming message text
        bq_header long varchar,  -- SOAP:Header
        primary key (bq_op, bq_ts)
        );

-- Initial values for SOAP Messages and XMLSchema types
create table BPEL..types_init (
        vi_script int,     -- FK reference to bs_id to the script table
        vi_name   varchar, -- message name, element name etc.
        vi_type   int,     -- 0 - message, 1 - element, 2 - XMLSchema type
        vi_value  long xml,-- Initial value
        primary key (vi_script, vi_name, vi_type)
)
;

-- Matching XPath expressions for the SOAP message parts
create table BPEL.BPEL.message_parts
        (
        mp_script int,      -- FK reference to bs_id to the script table
        mp_message varchar, -- message name
        mp_part varchar,    -- part name
        mp_xp   varchar,    -- location XPath expression
        primary key (mp_script, mp_message, mp_part)
        )
;

-- Operations which are invoked by process (used in invoke activities)
create table BPEL.BPEL.remote_operation (
        ro_script int,          -- FK reference to bs_id to the script table
        ro_partner_link varchar,-- name of the partner link
        ro_role varchar,        -- not used
        ro_operation varchar,   -- operation name
        ro_port_type varchar,   -- port type
        ro_input varchar,       -- input message name
        ro_output varchar,      -- output message name
        ro_endpoint_uri varchar,-- not used
        ro_style int,           -- messages encoding style : 1 - literal, 0 - RPC like
        ro_action varchar default '', -- SOAP Action value
        ro_target_namespace varchar,  -- for RPC encoding the namespace to be used for wrapper elements
        ro_use_wsa int default 0, -- WS-Addressing capabilities flag
        ro_reply_service varchar, -- for one-way operations: reply service name
        ro_reply_port varchar,    -- for one-way operations: reply port type
        primary key (ro_script, ro_partner_link, ro_operation)
```

```
)
;

-- Operations which process defines (can receive and reply)
create table BPEL.BPEL.operation (
        bo_script int,          -- FK reference to bs_id to the script table
        bo_name varchar,        -- operation name
        bo_action varchar,      -- SOAP Action value
        bo_port_type    varchar,-- port type
        bo_partner_link varchar,-- name of the partner link
        bo_input        varchar,-- input message name
        bo_input_xp     varchar,-- XPath expression to match the input message
        bo_small_input  varchar,-- not used
        bo_output       varchar,-- output message name
        bo_style        int default 0,-- messages encoding style : 1 - literal, 0 - RPC like
        bo_init         int,    -- process instantiation flag: 1 - can make new instances
        primary key (bo_script, bo_name, bo_partner_link)
);

-- Predefined endpoint URLs for partner links
create table BPEL.BPEL.partner_link_conf (
        plc_name        varchar,
        plc_endpoint    varchar,
        primary key (plc_name)
)
;

-- Properties
create table BPEL.BPEL.property
(
  bpr_script int,       -- FK reference to bs_id to the script table
  bpr_name varchar,     -- property name
  bpr_type varchar,     -- property type
  primary key (bpr_script, bpr_name)
)
;

-- Aliases
create table BPEL.BPEL.property_alias (
        pa_script       int,            -- FK reference to bs_id to the script table
        pa_prop_id      int identity,
        pa_prop_name    varchar,        -- property name
        pa_message      varchar,        -- message name
        pa_part         varchar,        -- part name
        pa_query        varchar,        -- XPath query to set the property value
        pa_type         varchar,
        primary key (pa_script, pa_prop_name, pa_message))
;

-- Correlation properties
create table BPEL.BPEL.correlation_props (
        cpp_id          int identity (start with 1),
        cpp_script      int,            -- FK reference to bs_id to the script table
        cpp_corr        varchar,        -- correlation name
        cpp_prop_name   varchar,        -- property name
        primary key (cpp_id, cpp_script, cpp_corr, cpp_prop_name))
;

-- Variables
create table BPEL..variables (
        v_inst          int,            -- instance id, FK reference bi_id of the instance table
        v_scope_inst    int,            -- scope instance id; different than 0 for compensation scope
        v_name          varchar,        -- variable name
        v_type          varchar,        -- variable type
        v_s1_value      any,            -- string, numeric
        v_s2_value      varchar,        -- XML entities
        v_b1_value      long varchar,   -- long strings
        v_b2_value      long varchar,   -- XML entities
        primary key (v_inst, v_scope_inst, v_name))
;

-- Links
create table BPEL..links
```

```
        (
          bl_script int,        -- FK reference to bs_id to the script table
          bl_name   varchar,    -- link name
          bl_act_id int,        -- corresponding link activity bit number
          primary key (bl_act_id, bl_script)
        )
;

-- Compensation scopes
create table BPEL..compensation_scope
        (tc_inst int,
         tc_seq  int identity (start with 1),
         tc_scopes long varbinary,
         tc_head_node int,
         tc_head_node_bit int,
         tc_compensating_from int default null,
         primary key (tc_inst, tc_seq)
        )
;

-- Messages are correlated via WS-Addressing
create table BPEL..wsa_messages
        (
        wa_inst int,
        wa_pl   varchar,
        wa_mid  varchar,
        primary key (wa_inst, wa_pl, wa_mid)
        )
;

create table BPEL..lock
        (
        lck int primary key
        )
;

-- Accepted connections which are waiting for reply
create table BPEL..reply_wait
        (
        rw_inst int,
        rw_id int, -- identity (start with 1),
        rw_partner varchar,
        rw_port varchar,
        rw_operation varchar,
        rw_query varchar,
        rw_expect varchar,
        rw_started datetime,
        primary key (rw_inst, rw_id)
        )
;

-- Registered alarm events
create table BPEL..time_wait
        (
          tw_inst       int,
          tw_node       int,
          tw_scope_inst int,
          tw_script     varchar,
          tw_script_id  int,
          tw_sec        int,
          tw_until      datetime,
          primary key (tw_inst, tw_node)
        )
;

-- BPEL message debugging queue
create table BPEL..dbg_message (
        bdm_text long varchar,          -- message text
        bdm_id int identity (start with 1),
        bdm_ts datetime,
        bdm_inout int,                  -- 1 for in, 0 for out
        bdm_sender_inst int,            -- instance id of sender if outbound message
        bdm_receiver int,               -- if inbound, inst id of receiving inst
```

```
        bdm_plink varchar,              -- name of partner link in the script in question
        bdm_recipient varchar,          -- partner link value for outbound message, URL.
        bdm_activity int,               -- activity id of activity that either sent the message or would
        bdm_oper varchar,               -- operation name
        bdm_script int,                 -- process id, FK reference bs_id from script table
        bdm_action varchar,             -- SOAP Action value
        bdm_conn int,                   -- client connection id
        primary key (bdm_id)
)
;

-- BPEL engine configuration
create table BPEL..configuration (
        conf_name       varchar not null,
        conf_desc       varchar,
        conf_value      any, -- not blob
        conf_long_value long varchar,
        primary key (conf_name)
)
;

create table BPEL.BPEL.op_stat
        (
         bos_process int,
         bos_plink varchar,
         bos_op varchar,
         bos_n_invokes int default 0,
         bos_n_receives int default 0,
         bos_cum_wait numeric default 0, -- milliseconds total time wait at the partner link/operation
         bos_data_in numeric default 0,
         bos_data_out numeric default 0,
         bos_n_errors int default 0,
         primary key (bos_process, bos_plink, bos_op)
)
;

create table BPEL.BPEL.error_log
        (
         bel_ts timestamp,
         bel_seq int identity,
         bel_level int, -- bel_level is 1. fatal 2. network, 3 instance.
         bel_notice_sent datetime,  -- time the email was sent, null if none
         bel_text varchar,
         primary key (bel_ts, bel_seq)
)
;

create table BPEL.BPEL.hosted_classes
        (
         hc_script      int,
         hc_type        varchar default 'java',
         hc_name        varchar,
         hc_text        long varbinary, -- compiled class
         hc_path        varchar, -- path to class if it is stored in file system
         hc_load_method varchar,
         primary key (hc_script, hc_type,  hc_name)
)
;
```

17.15.12. Errors

During the BPEL process execution we may consider following types of errors:

- Server failure - This means that the server as a whole has stopped operations and requires manual intervention. This is the case for: out of disk or database corruption.
- Network - This category applies to possibly transient conditions of not being able to contact partners. The server remains in operation for unaffected partners.
- Process Instance - This category applies to process instances getting an unhandled error condition. The process instance is out of service until the condition is resolved. This may indicate a bug in the process itself or a component used by it.

These errors will be logged in the file system (bpel_audit/server_log.txt) . If logging fails, an email is sent to the operator. The errors are also logged in the 'BPEL.BPEL.error_log' table so as to avoid repeatedly sending the same message.

During uploading the BPEL process we may consider following types of errors:

- bpel.xml file contains non-absolute paths and must be changed to absolute in order the uploading to be successful. This is in case user uses import_script api.
- wsdl file contains non-absolute paths at importing other wsdl file or xsd and must be changed to absolute in order the uploading to be successful. This is in case user uses upload process by choosing url/file for bpel and wsdl.
- process name already exists. This means there is already uploaded a process with the given name.
- at least one activity "receive" or "pick" must be declared at script source. Otherwise will be reported error this condition to be accomplished.

Connection Errors

The communication error can be caught by explicit fault handler in the script. In this case the <catch> handler must catch the fault "bpws:communicationFault":

```
<catch faultName="bpws:communicationFault" faultVariable="error">
 ...
</catch>
```

The error variable "error" will contain the following fault structure:

```
<comFault sqlState="xxxxx" message="text-of-message"
  partnerLink="plinkname" activity="name of activity" partnerURI="uri of
partner">
   <message>-- copy of the message being sent when fault occurred -- </message>
</comFault>
```

which can be used for reporting, recovery etc... The "sqlState" attribute contains SQL error state and the "message" stores the first line of SQL error message.

If the script does not handle this fault, the script will be frozen until explicit or implicit restart.

Here is an example of explicit communication handling:

```
<process xmlns:tns="urn:echo:echoService" xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  <partnerLinks>
    <partnerLink name="caller" partnerLinkType="tns:echoSLT"/>
    <partnerLink name="test" partnerLinkType="tns:testSLT" partnerRole="service"/>
  </partnerLinks>
  <variables>
    <variable name="request" messageType="tns:StringMessageType"/>
    <variable name="res" messageType="tns:StringMessageType"/>
    <variable name="n_tries" messageType="tns:IntMessageType"/>
  </variables>
  <sequence name="EchoSequence">
    <assign>
      <copy>
        <from expression="3"/>
        <to variable="n_tries" part="value" query="/value"/>
      </copy>
    </assign>
    <receive partnerLink="caller" portType="tns:echoPT" operation="echo" variable="request" createInstanc
    <while condition="getVariableData ('n_tries', 'value', '/value')
          &gt; 0">
      <sequence>
        <scope>
          <faultHandlers>
            <catch faultName="bpws:communicationFault" faultVariable="error">
              <sequence>
                <bpelv:exec binding="SQL">
                  dbg_obj_print (BPEL.BPEL.getVariableData ('error'));
                  dbg_obj_print (BPEL.BPEL.getVariableData ('n_tries'));
                </bpelv:exec>
```

```
                <assign>
                  <copy>
                    <from variable="error" query="/comFault/@sqlState"/>
                    <to variable="res" part="echoString" query="/echoString"/>
                  </copy>
                  <copy>
                    <from expression="getVariableData ('n_tries',
                          'value', '/value') - 1"/>
                    <to variable="n_tries" part="value" query="/value"/>
                  </copy>
                </assign>
                <wait for="PT10S"/>
              </sequence>
            </catch>
            <catchAll>
              <empty/>
            </catchAll>
          </faultHandlers>
          <sequence>
            <invoke partnerLink="test" portType="tns:SOAPPortType" operation="test" inputVariable="reques
            <assign>
              <copy>
                <from expression="0"/>
                <to variable="n_tries" part="value" query="/value"/>
              </copy>
            </assign>
          </sequence>
        </scope>
      </sequence>
    </while>
    <reply partnerLink="caller" portType="tns:echoPT" operation="echo" variable="res" name="EchoReply"/>
  </sequence>
</process>
```

In this example the process in a case of communication exception makes 3 reconnection retries to the remote service. Each reconnection is made after waiting 10 seconds.

Deadlock Errors

During concurrent execution of several BPEL scripts deadlock conditions can be signalled. That is why Virtuoso BPEL engine contains implicit deadlock detection and retry capability. When the deadlock is detected, the engine tries to wait some time, and retry the transaction. If the number of retries exceeds some maximum number the script execution will be aborted and mail will be sent to the operator with an appropriate message.

The time to wait before retry and the maximum number of retries can be configured on the configuration page of the administration UI.

17.15.13. Samples

Simple echo script

```
<process name="echo" targetNamespace="http://temp.uri" xmlns:tns="http://temp.uri"
    xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/">

    <partnerLinks>
        <partnerLink name="caller" partnerLinkType="tns:echoService"/>
    </partnerLinks>

    <variables>
        <variable name="request" messageType="tns:StringMessageType"/>
    </variables>

    <sequence name="EchoSequence">
        <receive partnerLink="caller" portType="tns:echoPort"
            operation="echo" variable="request"
            createInstance="yes" name="EchoReceive"/>
        <reply partnerLink="caller" portType="tns:echoPort"
            operation="echo" variable="request" name="EchoReply"/>
    </sequence>
```

```
</process>
```

... and corresponding WSDL

```
<definitions targetNamespace="http://temp.uri" xmlns:tns="http://temp.uri"
             xmlns:pl="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns="http://schemas.xmlsoap.org/wsdl/">

   <message name="StringMessageType">
      <part name="echoString" type="xsd:string"/>
   </message>

   <portType name="echoPort">
      <operation name="echo">
         <input message="tns:StringMessageType"/>
         <output message="tns:StringMessageType"/>
      </operation>
   </portType>

   <pl:partnerLinkType name="echoService">
     <pl:role name="svc">
       <pl:portType name="tns:echoPort"/>
     </pl:role>
   </pl:partnerLinkType>

</definitions>
```

Invoking Echo Service

```
 SQL>select xpath_eval ('/echoResponse/echoString/text()',
     xml_tree_doc (
             DB.DBA.soap_client (
             url=>sprintf ('http://example.com:%s/BPELGUI/bpel.vsp?script=file://echo/echo.bpel',serve
             operation=>'echo',
             soap_action=>'echo',
             parameters=> vector ('par', xtree_doc (']]&gt;&lt;echoString>hello world&lt;/echoString>&
             ));
     callret
     VARCHAR
      _______________________________________________________________________________

      hello world

      1 Rows. -- 330 msec.
```

BPEL Functions

getVariableData

```
...
<assign name="assignResult">
  <copy>
    <from expression="concat( 'Hello ', bpws:getVariableData('input', 'payload', '/tns:echovirtRequ
    <to variable="output" part="payload" query="/tns:echovirtResponse/tns:result"/>
  </copy>
</assign>
...
```

count

```
...
<assign name="assignResult">
  <copy>
    <from part="payload" variable="input" query="count(//lines/line)"/>
    <to variable="count"/>
```

```
  </copy>
</assign>
...
```

See Also: Reference Material in the Tutorial:
BP-S-1 Loan Flow demo

See Also: Reference Material in the BPELDemo tutorials:
BPEL4WS VAD package must be installed in order to view these tutorials.

BPELDemo

17.15.14. References

BPEL4WS specification

WS-Addressing

WSDL specification

SOAP specification

XPath specification

17.15.15. BPEL4WS VAD Package installation

Using Conductor

- Step 1. Download BPEL4WS VAD and copy it to a local directory.
- Step 2. Go to Conductor UI: http://host:port/conductor/
- Step 3. Login into the Virtuoso Conductor using the admin account.
- Step 4. Select Systems Admin->Packages Tab.
- Step 5. Under the Install Packaged, section of the UI is a list of VAD packages. If this is the first time you have installed a VAD package you will only see the Virtuoso Conductor. Directly under this list is the option Install a Package. You can either enter or browse to the location of the directory of the BPEL4WS package or enter a DAV location.
- Step 6. Once you have provided the location, select Proceed to upload the package.

Using ISQL

```
SQL>VAD_INSTALL('bpel4ws.vad',0);
 SQL_STATE  SQL_MESSAGE
 VARCHAR  VARCHAR
 _____________________________________________________________________________

 00000    GUI is accessible via http://host:port/BPELGUI
 00000    Quick Start is available from http://host:port/BPELGUI/start.vsp
 00000    No errors detected, installation complete.
 00000    Now making a final checkpoint.
 00000    Final checkpoint is made.
 00000    SUCCESS

 7 Rows. -- 5438 msec.
```

17.16. XSQL

XSQL is an XML-based format for describing simple stored procedures that can parse XML data, query or update database tables and compose XML output. Both input and output XMLs of such procedures are usually standard three-level documents: a top-level `ROWSET` element contains some number of `ROW` elements and every `ROW` contains one element per database field. XSQL lets the application developer avoid writing routine code for parsing and composing such documents: the server translates a short and self-evident XSQL description into a relatively long Virtuoso/PL procedure.

XSQL pages are usually executed from XSLT stylesheets by calling `processXSQL()` XPATH function. When the page is executed, it can access an XML entity that is context entity of the `processXSQL()` call. This entity is used inside the page as an implicit parameter called "context XML".

An XSQL document that describes one procedure is called "XSQL page". A page consists of small directives. Every directive is written as a single XML element from namespace "urn:oracle-xsql" (the typical namespace prefix is "xsql"). Every directive describes one standard operation. The resulting Virtuoso/PL procedure will execute all directives in turn. Directives are of four sorts:

Parameter assignments create and initialize local variables ("page parameters") that can be used in the rest of page.

Data modification requests can insert, delete or update data in database tables.

XML generators can query database or page parameters and produce XML fragments. These fragments form the resulting XML that is returned by the procedure.

DML directives let the author to put arbitrary Virtuoso/PL code in the procedure.

All directives are children of one op-level element called 'xsql:page'. This element can have any number of attributes but no one attribute is used by Virtuoso. These attributes may be used by specialized XSQL editors and standalone XSQL processors that should establish a database connection to read and write data so store connection details as attributes of 'xsql:page'.

The XSQL development cycle consists of editing '.xsql' resources in the file system or Virtuoso DAV. The editing can take place using a regular text editor or a supporting XML editor or some specialized third-party XSQL tool.

17.16.1. XSQL Syntax

Properties of each XSQL directives are specified by XML attributes. These attributes are of different types, mostly SQL expressions, calculateable strings, SQL names and lists of names (e.g., name of a database table or a list of columns of a table).

In some cases, SQL expressions are long and it is not convenient to place them into attribute values. Such expressions are written as text content of the XSQL directive element.

SQL expressions. XSQL allows slightly extended syntax of SQL expressions. If X is a name of page parameter made by xsql:set-page-param then special notation "{@X}" stands for the reference to the parameter value. This notation can be used in any place where a variable name is acceptable. In addition, this notation can be used inside string literals, such a literal is translated into a string concatenation expression; e.g., `'text-head{@X}text-middle{@Y}text-tail'` is converted into `concat ('text-head', cast ({@X} as varchar), 'text-middle', cast ({@Y} as varchar), 'text-tail')`.

Calculateable strings . Some attribute values are strings like resource URIs or XML names to be used in the output generated by a page. They are usually written 'as is' but they can use "{@X}" notation to insert the value of page parameter into the resulting string. The actual value of a calculateable string is compiled only once even if it appears many times in the XML output of the page.

Names of SQL columns . Some attributes are not arbitrary SQL expressions but only SQL column names. These names are written 'as is'. Only unqualified names are allowed, not in form 'table.column'. No "{@X}" is allowed for obvious reasons. If SQL name is case-sensitive or contains nonalphabetical characters then the name should be enclosed in double quotes. For readability, use single quotes to surround attribute value to not mix them with possible double quotes used in SQL name. Whitespace characters are not allowed in these names because these will not make proper names of XML elements.

Names of SQL tables . SQL table names are also written 'as is', with the same rules for double quotes around case-sensitive parts of the name. Both qualified and unqualified names are allowed.

Lists of names of SQL columns . When the value of attribute lists one or more column names, white space characters or commas delimit column names. The list should be space-delimited like 'COL1 COL2 COL3' or comma-delimited like 'COL1, COL2, COL3' but not a mix of them like 'COL1, COL2 COL3'.

17.16.2. XSQL Directives

xsql:delete-request

Deletes the rows listed in the context XML.

```
<xsql:delete-request
  table="table_name"
  key-columns="column_list"
  [ transform="calculateable_URI_string" ]
/>
```

If 'transform' URI is specified then it is used as a name of XSLT stylesheet that is applied to the context XML before the rest of processing. The result of that stylesheet should be in standard ROWSET/ROW form. The result may also contain elements with other names but they will be silently ignored.

After this optional XSLT transformation, the XSQL procedure gets all ROW elements in all top-level ROWSET elements of the source XML. For each such element it parses all subelements whose names match column names listed in the 'key-columns' attribute. It is legal to have a ROW element that does not contain an element that matches a particular column; the missing element is treated as a database NULL. If a subelement of ROW has attribute 'NULL' equal to 'Y' then it is also treated as database NULL.

When required subelements of ROW are parsed, the procedure deletes all rows from the specified table that have all specified column values equal to values specified by ROW subelements. When the delete operation is complete, the procedure parses all subelements of the next ROW element in queue and so on.

```
<xsql:delete-request table='"Demo"."demo"."Customers"' key-columns='"CustomerID"'/>
```

xsql:dml

Executes an arbitrary fragment of Virtuoso/PL code.

```
<xsql:dml>
One or more Virtuoso/PL statements, {@X} syntax is allowed.
</xsql:dml>
```

The XSQL procedure will contain the text of the directive 'as is'; the only change is that `{@X}` notation is replaced with appropriate Virtuoso/PL variables. Note that the use of `<![CDATA[...]]>` XML syntax is very convenient here.

```
<xsql:set-page-param name="X" value="2"/>
<xsql:set-page-param name="Y" value="2"/>
<xsql:dml><![CDATA[
-- This will simply print the string on server's console:
dbg_obj_print ('Hello World\n');
-- This will compose a string '2 * 2 =' and print on console:
dbg_obj_print ('{@X} * {@Y} = ');
-- This will calculate and print a well-known product.
-- Note that attempt to write {@X * @Y} or {@X * Y} results in a syntax error.
dbg_obj_print ({@X} * {@Y}, '\n');
]]>
</xsql:dml>
```

xsql:include-owa

This Oracle-specific directive is not implemented.

xsql:include-param

Puts the value of the specified page parameter into the resulting XML of the page.

```
<xsql:include-param name="page_param_name"/>
```

This writes the value of the specified page parameter as an element whose name is made from the name of parameter. NULL value results in an empty element that has attribute named 'NULL' with value 'Y'.

Note that XML elements ROWSET and ROW have special use in XSQL pages. The use of parameter names ROWSET and ROW is legal but may cause undesired effects.

```
<xsql:set-page-param name="an-int" value="2"/>
<xsql:set-page-param name="an-xml" value="xtree_doc('<sample>text</sample>')"/>
<xsql:set-page-param name="a-null" value="NULL"/>
```

```
<!-- This will form an element <an-int>2</an-int> -->

<xsql:include-param name="an-int"/>

<!-- This will form an element <an-xml><sample>text</sample></an-xml> -->

<xsql:include-param name="an-xml"/>

<!-- This will form an element <a-null NULL="Y"/> -->

<xsql:include-param name="a-null"/>
```

xsql:include-request-params

This directive is not implemented in this version.

xsql:include-xml

This directive is not implemented in this version.

xsql:include-xsql

This directive is not implemented in this version.

xsql:insert-param

Inserts rows listed in XML that is stored in page parameter.

```
<xsql:insert-param
  name="parameter_XML_name"
  table="table_name"
  [ mode="enum" (could be "into", "soft" or "replacing", default is "into") ]
  [ columns="column_list" ]
  [ date-format="string" (allowed but ignored) ]
  [ transform="calculateable_URI_string" ]
/>
```

The directive inserts into a table all data rows listed in the value of a page parameter that is named by 'name' attribute. The destination table is specified by 'table' attribute.

If 'transform' URI is specified then it is used as a name of XSLT stylesheet that is applied to the source data before the rest of processing. The result of that stylesheet should be in standard ROWSET/ROW form. The result may also contain redundant elements but they will be silently ignored.

After this optional XSLT transformation, the XSQL procedure gets all ROW elements in all top-level ROWSET elements of the source. For each such element it parses all subelements whose names match column names listed in the 'columns' attribute; if the 'columns' attribute is not specified then all column names from the destination table are used. It is legal to have a ROW element that does not contain an element that matches a particular column; the missing element is treated as a database NULL. If a subelement of ROW has attribute 'NULL' equal to 'Y' then it is also treated as database NULL.

When required subelements of ROW are parsed, the procedure adds a row to the table. It executes INSERT INTO, INSERT SOFT or INSERT REPLACING statement depending on the value of 'mode' attribute. When the insert operation is complete, the procedure parses all subelements of the next ROW element in queue and so on.

```
<xsql:insert-param name="customer-details" table='"Demo"."demo"."Customers"' />
```

xsql:insert-request

Inserts the rows listed in context XML.

```
<xsql:insert-request
  table="table_name"
  [ mode="enum" (could be "into", "soft" or "replacing", default is "into") ]
  [ columns="column_list" ]
```

```
  [ date-format="string" (allowed but ignored) ]
  [ transform="calculateable_URI_string" ]
/>
```

The directive inserts into a table all data rows listed in context XML. The destination table is specified by 'table' attribute.

If 'transform' URI is specified then it is used as a name of XSLT stylesheet that is applied to the context XML before the rest of processing. The result of that stylesheet should be in standard ROWSET/ROW form. The result may also contain redundant elements but they will be silently ignored.

After this optional XSLT transformation, the XSQL procedure gets all ROW elements in all top-level ROWSET elements of the source. For each such element it parses all subelements whose names match column names listed in the 'columns' attribute; if the 'columns' attribute is not specified then all column names from the destination table are used. It is legal to have a ROW element that does not contain an element that matches a particular column; the missing element is treated as a database NULL. If a subelement of ROW has attribute 'NULL' equal to 'Y' then it is also treated as database NULL.

When required subelements of ROW are parsed, the procedure adds a row to the table. It executes INSERT INTO, INSERT SOFT or INSERT REPLACING statement depending on the value of 'mode' attribute. When the insert operation is complete, the procedure parses all subelements of the next ROW element in queue and so on.

```
<xsql:insert-request table='"Demo"."demo"."Customers"' columns='"CustomerID", "Phone", "Fax"'/>
```

xsql:query

This executes an SELECT statement and writes its result set into the resulting XML in some sort of ROWSET/ROW format.

```
<xsql:query
  [ fetch-size="integer" (allowed but ignored) ]
  [ id-attribute="calculateable_XML_NAME_string" ]
  [ id-attribute-column="SQL_column_name" ]
  [ max-rows="integer_SQL_expn" (default is no limit) ]
  [ null-indicator="boolean" (default value is "no") ]
  [ row-element="calculateable_XML_NAME_string" (default value is "ROW") ]
  [ rowset-element="string" (default value is "ROWSET") ]
  [ skip-rows="integer_SQL_expn" (default is no skip) ]
  [ tag-case="enum" (could be "lower" or "upper", default is "lower") ]
>
Text of SELECT statement, {@X} syntax is allowed.
</xsql:query>
```

When no attributes is specified, the directive executes the SELECT statement and composes an XML fragment that consists of ROWSET element that have one ROW child element per row of the result set of the executed statement. Every ROW has one child element per result set column. The name of each element is made by converting column name to lowercase and element values are serializations of result set fields. The procedure does not create elements for fields with NULL values so an element with no text inside means empty string value but the totally missing element means NULL. In addition, ROW element have so-called "id attribute" whose name is 'num' and value is a number of the row in the result set.

If attribute 'null-indicator' is equal to '1' or 'yes' then elements are created for both non-NULL and NULL field values. Unlike elements that represent empty strings, element that represent NULL will have an attribute with name 'NULL' and value 'Y'.

Attributes 'id-attribute' and 'id-attribute-column' configures the composing of "id attribute". 'id-attribute-column' instructs to use the value specified SQL column as a value of attribute, 'id-attribute' can specify an attribute name other than default 'num'.

Attribute 'tag-case' specifies the character case of elements for columns. This does not affect names for ROW and ROWSET elements.

Element names ROW and ROWSET can be changed to whatever else by specifying attributes 'row-element' and 'rowset-element'. Either or both of these elements can be omitted at all by specifying empty string value for appropriate attributes; this will prevent the result set from proper parsing by some standard tools but may be convenient for special purposes. Note that If 'row-element' is empty string and ROW should not be created then attributes 'id-attribute' and 'id-attribute-column' should not be used.

Attributes 'max-rows' and 'skip-rows' adds TOP N and SKIP N clauses to the specified SELECT statement. They are supported mostly for compatibility. Attribute 'fetch-size' is ignored at all.

Note that Oracle allows the xsql:query to contain any code that produces a result set whereas Virtuoso allows only SELECT statement.

```
<xsql:set-page-param name="usermask" value="'%DAV%'"/>
<xsql:query>
-- This will produce the following XML fragment:
-- <ROWSET>
--   <ROW num="1"><u_id>3</u_id><u_name>administrators</u_name>
--      <u_full_name>WebDAV Administrators</u_full_name><ROW>
--   <ROW num="2"><u_id>3</u_id><u_name>dav</u_name>
--      <u_full_name>WebDAV System Administrator</u_full_name><ROW>
-- </ROWSET>
SELECT U_ID, U_NAME, U_FULL_NAME from SYS_USERS where U_FULL_NAME like {@usermask}
</xsql:query>
```

xsql:ref-cursor-function

This directive is not implemented in this version of Virtuoso Server.

xsql:set-cookie

This Oracle-specific directive is not implemented.

xsql:set-page-param

Declares a page parameter and assigns a value to it.

```
<xsql:set-page-param
  name="XML_name"
  [ ignore-empty-value="boolean" ]
  [ value="SQL_expression" ]
  [ xpath="XPATH_expression" ]
>
SQL expression, {@X} syntax is allowed, but only
for names that were declared above the current one.
</xsql:set-page-param>
```

The declaration should contain either 'value' attribute or 'xpath' attribute or a SQL expression as a text inside element, but not two or three of them simultaneously. The specified expression is calculated and the result is saved as a page variable with name specified by 'name' attribute. The resulting page variable can be used in other SQL expressions and calculateable strings of the page via '{@X}' notation described in XSQL Syntax subsection above.

If the directive uses 'xpath' attribute, the XPATH expression is applied to the context XML of the page. To apply an XPATH expression to some other XML entity, use `xpath_eval()` function in SQL expression specified by 'value' attribute or text inside the element.

If attribute 'ignore-empty-value' is set to 'yes' or '1' and the value of the calculated expression is an empty string then the page parameter is set to NULL. This may be convenient for handling default values that are passed to the page from HTML forms.

It is recommended to have names of parameters compatible with "XML 1.1 unqualified name" syntax. Hence, it is better to use minus sign instead of underscore and avoid using unusual characters like spaces. This become important when you use xsql:include-param because the name of create XML element will be equal to the name of parameter.

```
<!-- string constant '%DAV%' is an SQL expression -->

<xsql:set-page-param name="user-mask" value="'%DAV%'"/>

<!-- SELECT statement is an expression too, but only when enclosed in parenthesis.
  Without parenthesis, it is an SQL operator but not an SQL expression. -->

<xsql:set-page-param name="user-id">
(select U_ID from SYS_USERS where U_FULL_NAME like {@user-mask})
</xsql:set-page-param>

<!-- This copies implicit page argument 'context XML'
  into plain page parameter 'my-context' -->
```

```
<xsql:set-page-param name="my-context" xpath="." />
```

xsql:set-session-param

This Oracle-specific directive is not implemented.

xsql:set-stylesheet-param

This directive is not implemented.

xsql:insert-request

Updates the rows listed in context XML.

```
<xsql:update-request
  key-columns="column_list"
  table="table_name"
  [ columns="column_list" ]
  [ date-format="string" (allowed but ignored) ]
  [ transform="calculateable_URI_string" ]
/>
```

The directive updates a table by changing rows listed in the context XML. The destination table is specified by 'table' attribute.

If 'transform' URI is specified then it is used as a name of XSLT stylesheet that is applied to the context XML before the rest of processing. The result of that stylesheet should be in standard ROWSET/ROW form. The result may also contain redundant elements but they will be silently ignored.

After this optional XSLT transformation, the XSQL procedure gets all ROW elements in all top-level ROWSET elements of the source. For each such element it parses all subelements whose names match column names listed in the 'key-columns' attribute or in the 'columns' attribute; if the 'columns' attribute is not specified then it works as if all column names from the destination table are listed in 'columns'. It is legal to have a ROW element that does not contain an element that matches a particular column; the missing element is treated as a database NULL. If a subelement of ROW has attribute 'NULL' equal to 'Y' then it is also treated as database NULL.

When required subelements of ROW are parsed, the procedure updates all rows in the table that have all values of key columns equal to values listed in ROW. All these rows are updated by values from subelements of ROW. If 'columns' is specified then only named fields are updated; otherwise, the update operation changes all fields of a table.

When the update operation is complete, the procedure parses all subelements of the next ROW element in queue and so on.

```
<xsql:insert-request table='"Demo"."demo"."Customers"' columns='"CustomerID", "Phone", "Fax"'/>
```

Chapter 18. Runtime Hosting

Abstract

Virtuoso can be extended by in-process hosting of a number of different run time environments. This includes: Microsoft's .NET CLR (CLR), the Mono ECMA-CLI (Mono), and the Java Virtual Machine (JVM). Thus, you can create persistent stored modules, SQL functions, and user defined data types using any Microsoft .NET or Mono bound language, Java, and traditional languages such as 'C'/C++.

Objects provided by hosted run times, such as Java and CLR can be directly and transparently accessed from SQL. This also means that application logic in any hosted language can be exposed as a web service. By offering a choice of hosted run times in a cross platform setting, Virtuoso adds a new degree of freedom for the application designer. Applications can now be developed and deployed on either Windows, Linux or Unix, can be written in Java or any .net CLR bound language and can talk to any database via Virtuoso's virtual databasing capabilities.

The Virtuoso Server Extension Interface (VSEI) provides C interfaces and hooks that enable developers to write external functionality that can be bolted on the side of Virtuoso and called as SQL functions.

18.1. Runtime Environments

The Virtuoso server in its standard build is a single executable. To allow for external customizations based on VSEs the Virtuoso server engine becomes a callable library, the main function is re-written to incorporate other libraries and a new executable is created. For example, a Virtuoso server that contains PHP support under Windows consists of:

```
 65K virtuoso-odbc-php-t.exe
5.2M libvirtuoso-odbc-t.dll
1.2M php4ts.dll
```

which replaces the usual `7M virtuoso-odbc-t.exe`. `virtuoso.odbc-php-t.exe` is then executed in the usual way.

The Linux counterpart would be an `8M virtuoso-odbc-php-t` binary.

The Virtuoso executable files provided in the installer are appropriately named to indicate which customizations are supported. e.g. one that contains "php" and "clr" in the file name has been built with PHP and CLR support.

The location of the support libraries (.DLL's) for the various customizations of Virtuoso on Windows must either be in the %PATH%; environment variable or be the same directory as the virtuoso .EXE file. Virtuoso will not be able to start otherwise.

Note:

If Virtuoso is starting as a system service then the SYSTEM PATH must be altered as opposed to the USER PATH. The USER PATH is appropriate to alter if Virtuoso is being started as that user, usually when in debug/foreground mode. This is applicable to any required environment variables, not just PATH.

Hosted environments that require SQL access back to the database can make use of the in-process client mechanism. Hosted Runtimes such as PHP and ASP .Net would ordinarily use an RPC based (ODBC) connection. To improve the efficiency of the system, keeping it all in-process, a special connection can be made with no alteration to the API used in the code for communication.

See Also:

The In-Process Client keeps connections within Virtuoso avoiding unnecessary RPCs.

18.2. CLR, .Net & ASPX Host

The Common Language Runtime (CLR) is the foundation upon which the Microsoft® .NET strategy is built. The CLR provides an execution environment that manages running code and provides services that make software development easier. These services include automatic memory management, cross-language integration, interoperability with existing code and systems, simplified deployment, and a fine grained security system. The .NET platform also brings with it a new programming language called C#. It is very similar to Java and allows developers to make full use of all features available on the .NET runtime.

The CLR can run a variety of different types of applications, such as: console applications, Web server scripts, traditional Win32-based applications, and more.

The CLR provides a language specification for compilers to generate classes and code that can interoperate with other programming languages (The Common Language Specification: CLS) . Any API that is written using a CLS provider language can be used by any language that is a CLS consumer. Compilers generate code in a format called Common Intermediate Language (CIL) which is an intermediate representation of a compiled program and is easy to compile to native code or be compiled using Just-in-Time (JIT) engines. The restrictions placed by the runtime on the CIL byte codes ensures good potential for code optimization by the JIT compiler. The runtime environment provides garbage collection, threading and a comprehensive class library. All these things make code development easier and more appealing by allowing a mixture of languages to be collaborating to produce results, all contained within a managed sandbox to prevent bugs from breaking other running programs.

Windows has compilers that target the virtual machine from a number of languages: Managed C++, JavaScript, Eiffel, Component Pascal, APL, Cobol, Perl, Python, Ruby, Scheme, Smalltalk, Standard ML, Haskell, Mercury and Oberon.

Virtuoso is a CLR host. It is responsible for initializing the runtime, defining the application domain (its runtime boundaries, security settings, etc), and executing user code within such domains. Windows can be used as the development platform, a very developer friendly environment with a rich set of tools, but you will not be restricted to Windows, .NET and IIS to run the assemblies produced. Where .NET is not readily available or desired Virtuoso contains Mono, an open source re-implementation of Microsoft .NET, a vehicle making .NET cross-platform.

The CLR run time is a part of the Virtuoso process. The Mono run time may be run either as in-process or as out-of-process. Hosted applications can make use of the regular Virtuoso .net data provider to access Virtuoso SQL data and procedures. Microsoft ASPX files can also be run directly from Virtuoso either from the filesystem or WebDAV. Each of these capabilities releases you from the Microsoft platforms without compromising your development platform.

18.2.1. Environment Setup

Follow the steps below to configure the CLR runtime environment with Virtuoso:

1. **Install .Net Framework SDK.** The .Net Framework can be downloaded from the Microsoft web site.
2. **Install virtclr.dll into the GAC.** The `virtclr.dll` library supplied with Virtuoso must be installed into the .Net 'Global Assembly Cache (GAC)'.

.Net library components can be installed into the GAC using the command: `gacutil /i <DLL>`. The gacutil utility is supplied with the .Net framework. In this case we would use the command:

```
gacutil /i virtclr.dll
```

3. **Register virtclr with COM.** The virtclr.dll library must be registered as COM object using the 'regasm' utility as follows:

```
regasm virtclr.dll
```

4. **Install virt_http.dll into the GAC.** The `virt_http.dll` library supplied with Virtuoso must be installed into the .Net 'Global Assembly Cache (GAC)'.

```
gacutil /i virt_http.dll
```

Note:

The utilities gacutil and regasm are typically found in `%WINDIR%\Microsoft.NET\Framework\<version>` e.g. `C:\WINDOWS\Microsoft.NET\Framework\v1.0.3705.288`.

Failure to register the components successfully as described will generate "Virtuoso failed to initialize" messages. The Virtuoso installer, however, should automatically register the components for you.

Private assemblies must be deployed within the directory of the containing application and are found during runtime through a process called probing - a mapping from an assembly's identity to a file on disk that contains the manifest.

By default, probing for private assemblies is done from the root directory (application base) of the calling application (Virtuoso in this case) and the subdirectories that follow naming conventions based on assembly name and culture. You can customize this behavior by specifying a privatePath in your application's configuration file. The privatePath is a semi-colon delimited list of directories that the common language runtime will search for private assemblies. These directory names are relative to the application base - they cannot point outside the application base. The directories in the privatePath will be searched after the application base itself. The following configuration file adds a bin directory to the assembly search path:

```
<configuration>
    <runtime>
        <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
            <probing privatePath="bin"/>
        </assemblyBinding>
    </runtime>
</configuration>
```

Configuration files are called: `<application name>.config`.

18.2.2. Testing the Virtuoso .NET Runtime Environment

To test that you have successfully started the Virtuoso server with .NET CLR runtime support make a simple library, import the contained class into Virtuoso and call it.

Using an text editor create a C# source file in the server root directory called sanity.cs, with the following contents:

```
using System;

public class sanity
{
    public static string test(string  name) {
       return "Hello "+ name + ", from Virtuoso";
    }
}
```

This sample needs to be compiled into bytecode assembly before it can be used. Use a command prompt that is suitable set up to find .NET utilities in its path, the .NET Framework SDK installation provides a shortcut in the Start menu to a command prompt that is preconfigured. From the command prompt change directory to the Virtuoso server root containing the C# source file.

```
C:\Program Files\OpenLink\Virtuoso 3.0\bin>csc /target:library sanity.cs
Microsoft (R) Visual C# .NET Compiler version 7.00.9466
for Microsoft (R) .NET Framework version 1.0.3705
Copyright (C) Microsoft Corporation 2001. All rights reserved.
```

Now this library must be introduce to the Virtuoso Server. Using ISQL use the following commands to test the CLR:

```
C:\Program Files\OpenLink\Virtuoso 3.0\bin>isql 1112
Connected to OpenLink Virtuoso
Driver: 03.00.2315 OpenLink Virtuoso ODBC Driver
OpenLink Interactive SQL (Virtuoso), version 0.9849b.
Type HELP; for help and EXIT; to exit.
SQL> DB..import_clr (vector ('sanity'), vector ('sanity'));

Done. -- 300 msec.
SQL> select sanity::test('Rob');
callret
VARCHAR
_______________________________________________

Hello Rob, from Virtuoso

1 Rows. -- 60 msec.
```

Congratulations, you have proven that your Virtuoso server can run .NET classes.

See Also:
The Virtuoso Tutorials, which are installed with the Demo Virtuoso Server (port 1112), contains numerous samples further demonstrating the use of the CLR, .NET and Virtuoso.

See Also:
The Create Assembly Syntax

The External Hosted Create PROCEDURE Syntax

18.3. CLR & Mono

The Mono Project is an open development initiative sponsored by Ximian that is working to develop an open source, Unix version of the Microsoft .NET development and run time platform. Its objective is to enable Unix developers to build and deploy cross-platform .NET Applications. Likewise, this expands the capabilities of Windows programmers, this project opens .NET to non Windows platforms. Mono will implement various Microsoft technologies that have now been submitted to the ECMA for standardization.

The Goal is similar to that of Java, implementing a common virtual machine on multiple operating systems, however you are not restricted to one language.

Like .NET, Mono contains a Common Language Infrastructure (CLI) virtual, machine that contains a class loader, Just-in-time compiler, and a garbage collecting runtime; a class library that can work with any language which works on the CLR; a compiler for the C# language.

Windows has compilers that target the virtual machine from a number of languages: Managed C++, JavaScript, Eiffel, Component Pascal, APL, Cobol, Perl, Python, Ruby, Scheme, Smalltalk, Standard ML, Haskell, Mercury and Oberon. A single object system, threading system, class libraries, and garbage collection system can be shared across all these languages, the CLR.

The CLR and the Common Type System (CTS) enables applications and libraries to be written in a collection of different languages that target the CLR byte code. This means that you could define a class to do algebraic manipulation written in C#, that class can be reused from any other language that supports the CLI. You could create a class in C#, subclass it in C++ and instantiate it in an Eiffel program.

See Also:
The Create Assembly Syntax

The External Hosted Create PROCEDURE Syntax

18.3.1. Environment Setup

Ximian announced the launch of the Mono project, an effort to create an open source implementation of the .NET Development Framework.

Follow the steps below to configure the CLR runtime environment with Virtuoso:

1. **Install Mono.** The Mono package is available from the Mono project home page in the form of an RPM for Linux. The source is also available.
2. **Make mscorlib.dll available to /usr/local/lib.** The mscorlib.dll is installed by Mono and needs to be symlinked from its default location to `/usr/local/lib` .

18.3.2. Testing the Virtuoso Mono Runtime Environment

To test that you have successfully started the Virtuoso server with Mono CLR runtime support make a simple library, import the contained class into Virtuoso and call it.

Using an text editor create a C# source file in the server root directory called sanity.cs, with the following contents:

```
using System;

public class sanity
{
    public static string test(string  name) {
       return "Hello "+ name + ", from Virtuoso";
    }
}
```

This sample needs to be compiled into bytecode assembly before it can be used. Make sure you environment is configured to use Mono, change directory to the Virtuoso server root containing the C# source file. Run:

```
bash$ mcs /target:library sanity.cs
Compilation succeeded
```

Now this library must be introduce to the Virtuoso Server. Using ISQL use the following commands to test the CLR:

```
C:\Program Files\OpenLink\Virtuoso 3.0\bin>isql 1112
Connected to OpenLink Virtuoso
Driver: 03.00.2315 OpenLink Virtuoso ODBC Driver
OpenLink Interactive SQL (Virtuoso), version 0.9849b.
Type HELP; for help and EXIT; to exit.
SQL> DB..import_clr (vector ('sanity'), vector ('sanity'));

Done. -- 300 msec.
SQL> select sanity::test('Rob');
callret
VARCHAR
_______________________________________________

Hello Rob, from Virtuoso

1 Rows. -- 60 msec.
```

Congratulations, you have proven that your Virtuoso server can run .NET classes via Mono Runtime.

See Also:

The Virtuoso Tutorials, which are installed with the Demo Virtuoso Server (port 1112), contains numerous samples further demonstrating the use of the CLR, .NET, Mono and Virtuoso.

See Also:

The Create Assembly Syntax

The External Hosted Create PROCEDURE Syntax

18.4. Embedded Java VM API

The Java VM is an embedded system within Virtuoso that allows the calling of class Java methods and getting class properties. It uses the JAVA JNI API to interact with the JAVA VM.

18.4.1. Correspondence Between Virtuoso & Java VM Threads

At maximum one Java VM will be started on demand. If a function from the Java VM API is called and no JVM is running, one will be started, as required. Since Virtuoso is multithreaded it requires JDK version 1.3 or above in order to make better use of it's multithreading support. If the Java VM is already running the API VSEs attaches the current Virtuoso working thread, if not already attached, as a Java VM thread to the running VM. The Virtuoso worker thread does not automatically detached itself from the Java VM after use, therefore, in order to prevent leaving redundant Virtuoso worker threads being left attached to the Java VM the `java_vm_detach()` VSE should be used.

The following require access to the Java VM:

all Virtuoso JAVA PL API (JVM VSEs)

allocating/deallocating/copying of the Virtuoso/PL Java VM class reference values.

If no subsequent JVM VSEs are called after calling `java_vm_detach()` , the worker thread can still attach to the VM in order to deallocate or copy Java VM class reference values. In order to ensure that the worker thread is properly detached it is advisable to set all the variables that may hold Java VM class reference values to NULL, which deallocates their current value, before calling the `java_vm_detach()` JVM VSE.

Attaching and detaching Virtuoso worker threads is marked as a debug level message in the Virtuoso Event log, so that these messages can be used to debug the process.

18.4.2. Virtuoso/PL <-> Java VM Type Mapping Schema

Since the Java language uses a different set of data types than Virtuoso a type mapping system has been established to allow the passage of data to and from Java. All Java simple types are mapped to a corresponding Virtuoso type as follows:

Table 18.1. Java - Virtuoso Data Type Mapping

Java Type/Class	Virtuoso Internal Type
boolean	smallint
char	smallint
double	double precision
float	real
int	integer
short	integer
long	integer
boolean	smallint
java.lang.String	NVARCHAR
java.util.Date	DATETIME

All other Java objects are represented as a special Virtuoso value, which contains a reference to the Java object in the VM memory space. When such a value is returned as a Virtuoso/PL return value or as a result set column, it calls the `java.lang.Object.toString()` method for the Java VM object it refers to and that result is returned to the calling client.

Arrays are mapped to a Virtuoso vector of their elements. The array handling routines are recursive, so each element may be a Java array or a scalar type.

The above mappings are applied when converting Virtuoso/PL data to Java data as follows:

method parameter values in java_call_method VSE

method parameter values in java_new_object VSE

property values in java_set_property VSE

the instance parameters in java_call_method/java_get_property/java_set_property VSEs

The above mappings are applied in the opposite direction (Java Data -> Virtuoso/PL data) as follows:

method return values in java_call_method VSE

return value of the java_new_object VSE

property values returned from java_get_property VSE

18.4.3. References to Java VM Class Instances in Virtuoso/PL

Java Class instances are represented as a Virtuoso/PL variable values using a Java VM Global Reference. Each time such a Virtuoso/PL variable value is created or copied, it adds a Java VM Global Reference. When it is freed, it removes the Java VM global reference, allowing the Java Garbage collector (GC) to free it. Each of the API VSEs create a Java VM local frame upon it's start, and frees it upon exit. This allows for fast deallocation of the local objects created by mapping Virtuoso/PL native values to Java objects.

18.4.4. Specifying the Correct Java Type When Passing Values from Virtuoso/PL

Each method parameter in the `java_new_object()` /`java_call_method()` can be either the value itself or a two-element vector. When it is a two-element vector it's first element is the JNI type signature string, e.g.: for integer - I, for array of integers - [I], for java.lang.String - Ljava/lang/String. The signature is important because the Java VM JNI API needs it in order to find the appropriate constructor/method/property.

18.4.5. Virtuoso Java PL API VSEs

The API consists of the following VSEs:

`java_call_method()`

`java_set_property()`

`java_get_property()`

`java_load_class()`

`java_new_object()`

`java_vm_attach()`

`java_vm_detach()`

18.4.6. Java Security

Java classes are hosted in one of two modes:

Restricted

Unrestricted

Java class Permissions are managed by security classes that fall into categories as follows listed with it managing class:

File - java.io.FilePermission

Socket - java.net.SocketPermission

Net - java.net.NetPermission

Security - java.security.SecurityPermission

Runtime - java.lang.RuntimePermission

Property - java.util.PropertyPermission

AWT - java.awt.AWTPermission

Reflect - java.lang.reflect.ReflectPermission

Serializable - java.io.SerializablePermission

Restricted classes are not allowed any of the above privileges. Virtuoso returns errors that are returned by the security manager if breaches in security are attempted by a hosted Java class.

By default all Java classes are imported/created/hosted in restricted mode. To create Java class based user defined types that are unrestricted you need to use create type syntax with UNRESTRICTED keyword. The `import_jar()` function can also be used to import classes, its third optional parameter can be used to define the security mode.

 **Note:**

Note:

New behavior since Virtuoso 3.2 provides these two security modes defaulting to restricted. Prior to this Java classes were hosted in unrestricted mode.

Example 18.1. Java Security

The class Write_file, shown below, will attempt to write to a file on the file system. This class will be used to create a user defined type first in unrestricted mode and then in restricted mode to demonstrate how security exceptions are returned.

Source of Write_file.java:

```
import java.io.*;

public class Write_file
{
  public String write ()
    {
      String myFile = "foo";
      File f = new File(myFile);
      DataOutputStream dos;

      try
        {
          dos = new DataOutputStream (new BufferedOutputStream(new FileOutputStream (myFile),128));
          dos.writeBytes("ABC\n");
          dos.flush();
          dos.close();
        }

      catch (IOException ioe)
        {
          System.out.println("writeFile: caught i/o exception");
        }

      return "OK";
    }
}
```

Create the unrestricted type:

```
create type "Write_file" language JAVA external name 'Write_file'
  unrestricted METHOD "write" ()
  returns nvarchar external type 'Ljava/lang/String;' external name 'write';
```

Test calling the method:

```
SQL> select new Write_file().write();
callret
NVARCHAR
_______________________________________

OK
```

Now we want to recreate the type in restricted mode, remembering to drop it first:

```
drop type DB.DBA.write_file;

create type "Write_file" language JAVA external name 'Write_file'
  METHOD "write" ()
  returns nvarchar external type 'Ljava/lang/String;' external name 'write';
```

Test calling the method:

```
SQL> select new Write_file().write();

*** Error 42000: [Virtuoso Driver][Virtuoso Server]JV001: Java exception
occurred : java.security.AccessControlException : access denied
(java.io.FilePermission foo write)
in  __udt_method_call:(BIF),
```

```
<Top Level>
at line 1 of Top-Level:
select new Write_file().write()
```

Another way to import the above class is by use the `import_jar()` such as:

```
import_jar (NULL, 'Write_file', 1) - will import java classes in unrestricted mode.
import_jar (NULL, 'Write_file') - will import java classes in restricted mode.
```

18.5. Virtuoso Server Extension Interface (VSEI) (C Interface)

18.5.1. Virtuoso Server Extension Interface (VSEI)

The Virtuoso Server Extension Interface allows Virtuoso functionality to be extended by including new functions written in other languages such as C. These new functions are SQL callable.

These functions share the same C prototype and use Virtuoso internal APIs to do the following:

Retrieve arguments.

Assign values to output parameters.

Compile and execute SQL/PL statements and fetch results.

Signal SQL errors.

Return values.

A SQL-callable C function is called a Virtuoso Server Extension (VSE). These are external functions integrated into Virtuoso by linking their executable produced with a Virtuoso server in library format, rather than executable. VSEs were formally known as BIFs, which stood for Built-In Functions. Such functions must be exported using the `bif_define()` or `bif_define_typed()` C-functions when initializing the extended Virtuoso server.

These functions will thereafter be invoked on server threads. The functions should be re-entrant and comply to some simple memory management conventions outlined below.

These functions may execute arbitrary C code and call arbitrary APIs, to the extent these are compatible with the host operating system's threading model.

Virtuoso VSEs can be debugged within the normal C debugger by either starting Virtuoso under the debugger in foreground mode or by attaching the debugger to a running process.

Stack consumption should not be excessive: threads normally have 100K of stack on 32 bit platforms. The stack size may however be increased by settings in the virtuoso.ini file.

18.5.2. SQL Run Time Objects

The Virtuoso Server Extension API introduces the following data types:

box
: This is a run-time-typed block of memory which represents any SQL data type, e.g. number, string, array etc. Boxes have a type and length that are retrievable at run time and can be allocated, freed and otherwise manipulated by functions appropriate to each type of box. Boxes may form trees through use of heterogeneous arrays but should not form graphs.

query_t
: This is a compiled query, corresponding to a SQL statement or procedure compilation. The *`query_t`* is made from a SQL string and can thereafter be executed multiple times. This is a read-only object, not affected by execution on any number of threads, analogously to machine code not being affected by being executed.

query_instance_t
: This is a structure representing a query execution state. These are created when executing a *`query_t`* . This is analogous to a stack frame of a C function. It holds all relevant query state, such as cursor positions, intermediate results, column values etc. This is passed to all VSEs so they can have access to environment information such as current transaction, current client etc. The query instance references the *`query_t`* . As a rule, the query instance is specific to a thread. A query instance can be relatively long lived in the case of a cursor, which may live across multiple client-server message exchanges.

state_slot_t
: This is a part of *query_t* that specifies or describes a query time variable, column, parameter, intermediate result etc. This is analogous to an offset in a stack frame, it actually indexes a position inside a query instance. Given the state slot and the instance, it is possible to read or to set a value in the query state. Arguments of VSEs are passed as an array of state. Slots combined with the running query instance give the arguments values, and can be used to set output parameters.

local_cursor_t
: When executing a select statement, the local cursor structure is returned for accessing the result set rows. This is always a forward-only cursor. This can be advanced, column values may be accessed and the cursor may be closed.

18.5.3. Memory Management Rules

All state slots in a query have distinct values. With the exception of a reference parameter, no value is referenced twice. All state slot values can therefore be recursively freed independently of each other.

If a VSE returns data, this data must always be new, i.e. allocated inside the VSE and may not be a copy of or include any of the arguments as a substructure. All return values and arguments must be legitimate boxes and may not share a structure.

18.5.4. Server Main Function

The server main function for a customized Virtuoso server has the following format:

```
static void (*old_ddl_hook) (client_connection_t *cli) = NULL;

static void
ddl_hook (client_connection_t *cli)
{
  if (old_ddl_hook)
    old_ddl_hook (cli);

  /* DDL code (depending on the server being fully initialized
    (ex: create table) ) goes here */
}

static void
init_func (void)
{
  old_ddl_hook = set_ddl_init_hook (ddl_hook);
  /* initialization code (prerequisite for server initialization
     (bif_define, unrelated init code) goes here */
}

int
main (int argc, char *argv[])
{
  VirtuosoServerSetInitHook (init_func);
  return VirtuosoServerMain (argc, argv);
}
```

There are three phases to custom code initialization:

execution of the `main()` function

execution of the `init_func()` function

execution of the `ddl_hook()` function

The `init_func()` function is called before any server initialization functions are called. This is typically a place for defining new VSEs, allocation of synchronization objects (since the server does not have any threads yet), and/or custom code initialization not related to Virtuoso. Here, the `set_ddl_init_hook()` should be called also, if it exists, to register the `ddl_hook()` callback function.

Note:

The `old_ddl_hook()` mechanism - this allows queuing of the ddl_hooks.

The `ddl_hook()` function is called during normal startup just before the roll forward, but after the server's internal structure has been initialized. This is typically a place to execute SQL statements to initialize the extension. The variable *client_connection_t ** is passed to the function to provide the client connection that should be used for SQL execution.

The `main()` function can call `VirtuosoServerSetInitHook()` if there is any Virtuoso-related initialization to be performed, and should then call the `VirtuosoServerMain()` function to start the Virtuoso server. The `VirtuosoServerMain()` function will return control after the server has been shut down.

18.5.5. Compiling & Linking

The files should be compiled for the multi-threaded environment appropriate to the operating system and should be linked accordingly e.g. `-lm` , `-ldl` .

The Virtuoso distribution contains the following libraries/object files:

- **libvirtuoso-* libraries/object files.** The Virtuoso engine. The suffix denotes the kind of remote and threading support : -iodbc for iODBC, -udbc for UDBC, -t for native threads, -f for fiber threads.
- **c_javavm object file.** the Virtuoso Java VM Integration file (no main function). This is the file to be linked in if the custom code executable needs to have Java VM support. To initialize the Java VM integration the C function `void bif_init_func_javavm (void)` should be called from the `VirtuosoServerSetInitHook` hook. This file references the `libjvm` library from the Java JRE, hence, should also be appended to your linker options.
- **c_bif_server_php object file.** The Virtuoso PHP integration support library. This is the file to be linked in if the custom code executable needs to have the PHP support built in. To initialize the PHP engine integration the C++ function `void init_func_php (void)` should be called from the `VirtuosoServerSetInitHook` hook. Note that this file is a C++ object file and needs to be linked in accordingly. This file references the libphp4 library.
- **basec object file.** The Virtuoso .NET CLR Integration file. This is the file to be linked in if the custom code executable needs to have the .NET CLR integration built in. This is a C file.
- **c_dotnet object file.** The Virtuoso .NET CLR Integration file. This is the file to be linked in if the custom code executable needs to have the .NET CLR integration (and nothing but) built in. This is a C++ file. Binaries using this file should initialize the .NET CLR support by calling the C++ function `void bif_init_func_clr (void)` (defined in the basec object file) from the `VirtuosoServerSetInitHook` hook. In order for the .NET CLR support to work correctly the `::CoInitialize` Win32 COM API should also be called before the `VirtuosoServerMain` call.
- **sql_code_javavm object file.** This contains the DDL definitions for the Java VM support. Executables using that binary should call the `sqls_define_javavm` function in their DDL init hook.
- **sql_code_xslt object file.** This contains the DDL definitions of the XSLT stylesheets used in the Java VM & CLI support. Executables using that binary should call the `sqls_define_xslt` function in their DDL init hook.
- **sql_code_clr object file.** This contains the DDL definitions for the .NET CLR integration support. Executables using that binary should call the `sqls_define_clr` function in their DDL init hook.

18.5.6. Functions by Category

General Box Functions

The box, usually marked with the caddr_t data type is the basic representation of any SQL data in Virtuoso. All boxes have a run time data type, with a name beginning with `DV_` . All boxes have a 3 byte run time length which allows for up to 16 MB of contiguous array size in SQL data.

The further interpretation of the content of the box is determined by the type tag. The length is always an exact byte length, although the actual length is rounded up to the next suitably aligned value. The length and tag of a box must never be changed while the box is allocated but the content is freely writable. The tag and length reside immediately under the pointer of the box, so that a box, with the appropriate type cast will pass as a C array or string.

Numbers are generally represented as boxes. There is an exception for small integers, which are always distinguishable from pointers. Thus the range from -10000 to 10000 are not allocated as boxes holding the value but can be passed directly. This is hidden however and the programmer need not be concerned about this except sometimes when debugging.

The byte order in boxes depends on the platform.

The most important types are:

DV_SHORT_STRING - The box contains a null terminated string of char. The box length is 1 + the count of characters, including the final 0 in the count.

DV_LONG_STRING - same as DV_SHORT_STRING.

DV_LONG_INT, TAG_BOX - The box is sizeof (long) long, with the long as element 0, in the appropriate byte order.

DV_SINGLE_FLOAT - sizeof (float) bytes, containing the float.

DV_DOUBLE_FLOAT - sizeof (double) bytes, contains the double.

DV_NUMERIC - opaque, contains a decimal floating point. The `numeric_t` functions can be used, see appropriate documentation.

DV_BIN - Binary string, no terminating 0 counted in the length.

DV_DATETIME - datetime, opaque, dt_ functions can be used, see appropriate documentation.

DV_BLOB_HANDLE - Blob handle, opaque but convertible to string if short enough.

DV_ARRAY_OF_POINTER - Heterogeneous array, sizeof (caddr_t) * n_elements, first all elements are box pointers. The length is in bytes, so divide by sizeof (caddr_t).

DV_DB_NULL - a box of 0 bytes (header only), represents the SQL NULL value.

Box Functions

```
box_t dk_alloc_box ( uint32 bytes ,
                     int tag );
```

`dk_alloc_box()` allocates a box of the given size and type. The initial contents are undefined.

```
int dk_free_box ( box_t box );
```

`dk_free_box()` frees a box allocated by `dk_alloc_box()` . The argument may not be any other pointer.

```
int dk_free_tree ( box_t box );
```

`dk_free_tree()` is like `dk_free_box()` but will free recursively, following through DV_ARRAY_OF_POINTER boxes.

```
uint32 box_length ( box_t box2 );
```

```
#define box_tag(box) \
        (*((dtp_t *) &(((unsigned char *)(box))[-1])))
```

These return the length and the tag of a box.

```
long unbox ( box_t n );
```

```
box_t box_num ( long n );
```

```
box_t box_dv_short_string ( char *string
                            );
```

```
box_t box_double ( double d );
```

```
box_t box_float ( float f );
```

```
#define unbox_num(n) unbox(n)
#define unbox_float(f) (*((float *)f))
#define unbox_double(f) (*((double *)f))
#define unbox_string(s) ((char *)s)
```

The above functions and macros convert between C data types and boxes. `box_dv_short_string()` takes a `char *` to any null terminated string and allocates a string box of appropriate size. This itself looks like a null terminated string but has the box header with the run time length and type under the pointer.

```
box_t box_copy ( box_t box );
```

box_copy() returns an identical size box with the same type and contents.

box_t **box_copy_tree** (box_t *box*);

box_copy_tree() performs a recursive copy, traversing DV_ARRAY_OF_POINTER references.

int **box_equal** (box_t *b1* ,
box_t *b2*);

Given two arbitrary boxes, returns true if they are recursively equal.

Example 18.2. Box Examples

Below is the code for box_copy_tree:

```
box_t
box_copy (box_t box)
{
  dtp_t tag;
  uint32 len;
  box_t copy;

  if (!IS_BOX_POINTER (box))
    return box;

  tag = box_tag (box);
  if (box_copier[tag])
    return (box_copier[tag] (box));
  len = box_length (box);
  copy = dk_alloc_box (len, tag);
  memcpy (copy, box, (uint32) len);
  return copy;
}

box_t
box_copy_tree (box_t box)
{
  box_t *copy;
  dtp_t tag;

  if (!IS_BOX_POINTER (box))
    return box;

  tag = box_tag (box);
  copy = (box_t *) box_copy (box);
  if (tag == DV_ARRAY_OF_POINTER || tag == DV_LIST_OF_POINTER)
    {
      uint32 inx, len = BOX_ELEMENTS (box);
      for (inx = 0; inx < len; inx++)
        copy[inx] = box_copy_tree (((box_t *) box)[inx]);
    }

  return (box_t) copy;
}
```

Note:

The IS_BOX_POINTER check at the start will detect the unboxed, 'bare' small integers which are actually not allocated and can be returned by value. Only then can box_tag be used to find the type.

The DV_TYPE_OF macro should be used instead of box_tag when the type is unknown to avoid de-referencing a small integer.

Also note BOX_ELEMENTS , which is box_length () / sizeof (caddr_t). This is practical for iterating over arrays.

See Also

See Also

The VSEI Functions .

18.5.7. VSEI Definition

```
typedef caddr_t (*bif_t) (caddr_t *qst, caddr_t *error_return, state_slot_t ** arguments);
void bif_define (char *name, bif_t bif);
void bif_define_typed (char * name, bif_t bif, bif_type_t *bt);
```

These functions associate a function pointer to a VSE name. The typed variant allows associating a value type used when inferring SQL meta-data if the result is returned to a client. The type can be one of the following externs:

extern *bif_type_t* bt_varchar;
extern *bif_type_t* bt_any;
extern *bif_type_t* bt_integer;
extern *bif_type_t* bt_double;
extern *bif_type_t* bt_float;
extern *bif_type_t* bt_numeric;
extern *bif_type_t* bt_convert;
extern *bif_type_t* bt_timestamp;
extern *bif_type_t* bt_time;
extern *bif_type_t* bt_date;
extern *bif_type_t* bt_datetime;
extern *bif_type_t* bt_bin;

If a VSE accesses indexes either by its own internal code or by executing Virtuoso/PL statements, there becomes a potential for deadlocks. To prevent deadlocks, the Virtuoso/PL compiler must be informed of potential index usage inside the VSE. Special deadlock-safe code can be created for its needs. The `bif_set_uses_index()` function should be used after `bif_define()` or `bif_define_typed()` in such cases.

The potential for deadlocking is always present if the VSE executes Virtuoso/PL code or uses XPath/XSLT functions. Other functions of Virtuoso's C interface are deadlock-safe since they perform no database access.

18.5.8. SQL Exceptions

```
caddr_t srv_make_error (char *code, char *msg);
void sqlr_error (char *code, char *msg,...);
void sqlr_resignal (caddr_t err);
```

An error object is a three element array of type `DV_ARRAY_OF_POINTER` , consisting of the number 3, the SQL state and the message. The control flow in case of errors signalled inside VSEs is a longjmp to an outer context, typically that of the calling stored procedure or top level query. The condition is there handled or sent to the next level up, ultimately to the ODBC, JDBC or Web client. Executing a SQL statement inside a VSE always returns and never exits the VSE by longjmp. Thus the VSE gets a first look at all SQL errors caused by statements executed by it.

sqlr_error is the normal function for signaling a SQL state. It takes a 5 character SQL state, a printf format string and optional arguments, a la printf.

sqlr_resignal is used to throw a condition to the next level handler. This is typically done when executing a query which returns an error and the error is sent up to the caller of the VSE.

srv_make_error makes the error structure. The expression

sqlr_resignal (srv_make_error ("12345", "message")); is equivalent to sqlr_error ("12345", "message");

Note

srv_make_error does not take the printf-type arguments.

By convention a NULL pointer indicates no error. sqlr_resignal (NULL) is an error.

The macros:

```
#define ERR_STATE(err)  (((caddr_t*) err)[1])
#define ERR_MESSAGE(err)  (((caddr_t*) err)[2])
```

can be used to read an error returned by a statement.

18.5.9. Executing SQL

```
query_t * sql_compile (char *string2, client_connection_t * cli,
                                caddr_t * err, int store_procs);
void qr_free (query_t * qr);

client_connection_t * qi_client (caddr_t * qi);
```

These functions allow executing SQL from VSEs. First the SQL statement needs to be compiled with sql_compile. The statement may take value parameters and may be a DDL or DML statement, including select, update, procedure call, table creation etc.

The query_t returned can be used multiple times on any number of simultaneous threads. if an application repeatedly performs the same queries the text can be compiled once and reused at infinitum.

qr_free will free a query returned by sql_compile.

Example 18.3. Example

```
{
  caddr_t err = NULL;
  query_t * qr  = sql_compile ("select * from SYS_USERS", qi_client (qst), &err, 0);
  ...
  if (err)
  exit (-1);
  qr_free (qr);
}
```

```
caddr_t qr_rec_exec (query_t * qr, client_connection_t * cli,
    local_cursor_t ** lc_ret, query_instance_t * caller, stmt_options_t * opts,
    long n_pars, ...);
```

Once a query is compiled it can be executed and fetched. This function executes a query in the context of a VSE. The execution is on behalf of the same user and in the same transaction as the VSE. This is only possible in the context of a VSE, not at top level in the main program, for example.

The first argument is the compiled query to execute. The second is the client connection, obtained by qi_client from the qst argument of the VSE. The lc_ret, if non NULL will get a be set to a newly allocated local_cursor_t * that allows fetching rows from the result set. This only applies to a select statement. The caller is the qst argument of the VSE, The opts can be NULL. The n_args is the count of query parameters, 0 if no parameters are passed.

The return value is an error, suitable for sqlr_resignal. A NULL value means success.

It should be double-checked if the query access or potentially may access any tables or indexes. If it may do this, the VSE must be described as deadlock-unsafe by calling bif_set_uses_index() after bif_define() or bif_define_typed(). If qr_rec_exec access any tables or views, and the call of VSE from Virtuoso/PL code is compiled as deadlock-safe, the whole server may be halted.

If parameters are passed, a group of 3 actual parameters follows for each ? in the query being executed. In each such group the first is the name of the parameter, of the form ":n", where n is the position of the parameter, starting at 0, so ":0" corresponds to the 1st ? and ":11" to the 12th. The second in the group of 3 is the value, usually a box pointer. The third is the type, one of QRP_INT, QRP_STRING or QRP_RAW.

QRP_INT means that the value will be converted to a box as by box_num. QRP_STRING means that the value will be converted to a string as by box_dv_short_string. In either case the value is allocated and freed as part of the execution. QRP_RAW means that an arbitrary box is passed as is. If so, this box will be freed in the process and MUST NOT BE REFERENCED AGAIN in the VSE. if the statement is a select, lc_ret should be specified and should be the address of a local_cursor_t * variable, where the

cursor can be returned.

```
long lc_next (local_cursor_t * lc);
caddr_t lc_nth_col (local_cursor_t * lc, int n);
void lc_free (local_cursor_t * lc);
```

These functions allow reading through a result set. The local_cursor_t * must have come from qr_rec_exec.

lc_next will move the cursor one row forward. The first call after the exec places the cursor on the first row. A 0 return value indicates that the cursor is at end. if 0 is returned at the first call, the result set had zero rows. The data member lc_error may be set and should be checked after calls to this function. See examples. The value will be suitable for sqlr_resignal if copied (box_copy_tree).

The lc_nth_col returns the value of the nth column of the current row. The index is 0 based. The value is an arbitrary box pointer and is READ ONLY, to be copied (box_copy_tree) if the application needs to keep it around. The value will stay readable until the next lc_next or lc_free. Use DV_TYPE_OF et al to determine the type of the value.

lc_free frees the cursor and any resources associated to it. This has no effect on the transaction.

The bif_my_select function returns an array with one element for each row of the SYS_KEYS table. The rows are themselves arrays containing the column values.

18.5.10. Adding New Languages And Encodings Into Virtuoso

There are too many languages to be able to support them all by default so Virtuoso is user extensible in this respect. The built-in 'x-any' language supports most languages to a degree, but it is not the optimum solution for some specific languages or if you want to perform a words' normalization to make text search more effective. To make Virtuoso extensible, language-specific functions are organized into *language handlers* , and handlers are organized in hierarchical trees. Every handler contains pointers to such functions as "count words in given string", "call given callback once for every word in the string" etc.

XML documents and SQL procedures may identify languages by their names, for example by value of *xml:lang* attribute, `content_language` argument of built-in functions, or by `__lang` option etc... Every language handler defines up to two names of the language it supports, one matching ISO 639 regulations (e.g. 'en'), and one matching RFC 1766 (e.g. 'en-UK'). When Virtuoso finds a match to the language name specified, it searches through the an internal hash-table. If the name is unknown, the 'x-any' handler will be returned as a default.

Custom language handlers should contain a pointer to a more generic handler, e.g. to the handler, Handler may have NULLs stored instead of pointers to required functions, these NULLs will be replaced with pointers to generic handler's functions automatically when the custom handler will be activated.

See Also:

lh_get_handler

lh_load_handler

There are two trees of language handlers in current version of Virtuoso. "Main" tree starts from 'x-any' root and contains handlers of languages used in documents, another tree starts from 'x-ftq-x-any' root contains handlers of Free Text query ('ftq') languages. The difference is in handling of wildcard characters: query string 'hello, wo*ld' consists of two "words", 'hello' and 'wo*ld', and 'x-ftq-x-any' will properly locate them, but 'x-any' handler will report three words -- 'hello', 'wo' and 'ld', because it knows nothing about special meaning of '*' in query strings. That is why every handler may contain a pointer to a handler of its own query language.

In addition to plain language handlers, it's possible to add handlers of "encoded language" They are useful if you have large number of documents in some particular encoding and speed of free text indexing is critical for your applications. While usual handlers deal with Unicode data, and it requires data to be decoded before processing, functions of "encoded language" handler may accept buffers of encoded text, eliminating decoding.

See Also:

elh_get_handler

elh_load_handler

The OpenLink Virtuoso Server distribution contains sources of sample language handler for 'en-UK' language. The difference between 'x-any' handler and this one is in handling of abbreviations and numbers. 'en-UK' handler will read phrase '$3.54 per sq.inch.' as the sequence of words '3.54', 'per' and 'sq.inch', instead of sequence '54', 'per', 'sq' and 'inch' that 'x-any' will read. The generic 'x-any' handler has no specific rules for dealing with the "decimal point" because in many scripts "decimal comma" is used, thus '3.54' will be processed as pair of words '3' and '54', but '3' will be ignored in many cases as noise word due to its 1-character length.

In addition to the language extension interface, Virtuoso provides an *eh_load_handler* function to add new encodings, but it should be used solely for multi-character encodings which cannot be supported through the usual Virtuoso International Character Support . If an encoding was created by the CHARSET_DEFINE function, Virtuoso can build special lookup tables for very fast text translation from Unicode to the encoding, thus you are not likely to gain in performance by writing your own C code, but some applications will know nothing about your encoding because they will check only the SYS_CHARSETS system table.

18.6. VSEI Plugins

Virtuoso functionality can be enhanced through external libraries by loading shared objects or DLLs. The new functions are written in a language of the developer's choice and compiled to produce a shared library appropriate to the operating system. The path to the shared library must be declared in the Virtuoso INI file and the server restarted before it can be used.

The Virtuoso INI file uses a [Plugins] configuration section for listing shared libraries for the server to load upon startup. The layout is as follows:

```
[Plugins]
LoadPath = <module path> (example : /home/virtuoso/hosting)
Load1 = <module type>, <module name> (example : hosting_perl.so)
Load2 = <module type 2>, <module name 2>
..
LoadN = <module type N>, <module name N>
```

Virtuoso reads the Load1, Load2, ... LoadN lines from the [Plugins] section and attaches them according to their type.

`<module path>` is the directory containing shared modules for use with Virtuoso. (e.g. `/home/virtuoso/hosting`)

`<module type>` specifies the type of module that is to be loaded, and hence how Virtuoso is to use it. So far only the "Hosting" type exists.

`<module name>` is the file name of the modules shared library or object. (e.g. `hosting_perl.so`)

The "Hosting" type defines entry points for initialization of the runtime hosting environment, destruction of the user environment and execution of a file or string containing commands in the hosted language. It also returns a list of file extensions that it is capable of processing. Virtuoso dynamically defines memory-resident (no disk image) HTTP server handlers for each specified type. For example, the Perl hosting plugin supports 'pl' extension. Hence, upon initialization of the hosting plugin, Virtuoso defines the `__http_handler_pl(..)` function according to the API for file type handlers in the Virtuoso HTTP server. With this handler in place, each hit on a .pl file (file system or WebDAV) with appropriate execute permissions will cause the HTTP server to execute the code within it and return the result instead of simple the file contents.

The handler will call the __hosting_http_handler VSE with a special set of parameters to represent the HTTP environment correctly.

There's a VSE to call each of the hosting modules:

`__hosting_http_handler ( in` *`extension`* `varchar, in` *`content`* `varchar, [in` *`params`* `varchar, ] [in` *`lines`* `any, ] [inout` *`filename_head_ret`* `varchar, ] [in` *`options`* `any, ] [out` *`diag_ret`* `varchar]) returns varchar`

such that:

extension
: Selects plugin by supported extension handler. e.g. pass 'pl' for the perl plugin.

content
: If <filename_head_ret> is NULL or unspecified this will be the name of the file containing the code to execute in the hosted environment, otherwise this will be program code to execute as a string.

params
: (optional) A string containing the HTTP parameters as encoded in the HTTP request body.

lines
: (optional) A vector (array) containing the HTTP request headers, in the same format as those passed to VSPs.

filename_head_ret
: (optional) On input this is the "name" to put on the command text passed in <content>. If this is NULL it means that <content> holds the path and filename containing the commands. On output this contains the HTTP headers of the HTTP response generated by the plugin, if any and in HTTP mode.

options
: (optional) A vector holding name/value pairs of strings passed as options to the plugin. The Perl plugin sets the other options as environment variables before calling the perl code.

  There is a "`__VIRT_CGI `" = "`0` "/"`1` " option to control whether the plugin operates in HTTP mode (`__VIRT_CGI=1`) i.e. implements the CGI interface and treats the output as an HTTP response.

diag_ret
: (optional) Returns various diagnostics messages returned while the code is running if any. The perl plugin sets this to the collected STDERR messages.

The function will return a varchar containing the HTTP body if in HTTP mode (`__VIRT_CGI = 1`) or the messages printed to STDOUT.

Virtuoso will normally call to memory each plugin as required, and expel it when finished. This behavior can be controlled by the INI file parameter:

```
[HTTPServer]
PersistentHostingModules = 1/0 default 0
```

Setting `PersistentHostingModules` to "1" prevents Virtuoso from removing the interpreters from the HTTP threads after each request.

Example 18.4. Using the Perl Plugin

Executing Perl code directly:

```
select __hosting_http_handler ('pl', 'print "hello world"; ', '', vector (), 'helloworld.pl');

returns : hello world
```

Executing a perl script file (perl/test_print.pl in the Virtuoso working directory):

```
/perl/test_print.pl
-------------------
#!/usr/bin/perl
print "hello world file";

select __hosting_http_handler ('pl', 'perl/test_print.pl');

returns : hello world file
```

Note:
The hosting_perl hosting module uses the perl `tie()` function to "tie" up the STDIN, STDOUT, STDERR, exit() and %ENV perl objects. Untying any of these may lead to unpredictable results.

Chapter 19. Internet Services

Abstract

This section discusses Virtuoso's support for DAV, email, usenet news, mime and other internet standards.

The WebDAV repository lets Virtuoso store and index content within the database. This can then be accessible directly via the HTTP protocol.

This chapter describes the MIME extensions that have been implemented in Virtuoso.

This covers examples of use of MIME for composition of MIME messages and content for mail transit of HTTP output.

19.1. WebDAV Server

Virtuoso supports the WebDAV protocol, which is an extension of HTTP for cooperative work on the Internet. DAV resources can be of any mime type, including binary types. The DAV resources are stored in the Virtuoso database as large objects, they are not in the file system and can only be accessed through the DAV protocol. Direct SQL access to the DAV tables is also possible, so there's a set of Virtuoso/PL procedures that acts as DAV API to let server-side applications access DAV. Virtuoso DAV can be

extended by virtual collections. Instead of accessing DAV tables, DAV server can retrieve data from applications, thus an application can generate documents on demand and these documents will be available via DAV as well as plain DAV resources. Moreover, resources can be submitted directly to the application via DAV. Virtuoso DAV provides settable access rights and ownership of resources. Access rights as such are not covered by the DAV specification but Virtuoso implements both ACLs (access control lists) and a Unix file system like scheme for ownership and permissions. SQL accounts enabled for DAV are valid owners of DAV resources. A resource has both a user owner and a group owner, plus an optional access control list that can grant privileges to users and roles alike. A user can have DAV-only access, SQL/ODBC-only access, both of them or neither. User account information is stored in relational tables and can be manipulated from SQL or through a Web UI.

Virtuoso DAV can store metadata about resources. These metadata are extracted from resources automatically, and can be edited by users. In addition, users can place public and personal 'tags' on resources to categorize data according for personal needs without interference between users. Virtuoso DAV has a powerful and scalable search engine that let application locate resources that match given list of criteria. The search can process both plain DAV resources and data published by applications in virtual collections. Search criteria can check for resource properties, content, metadata and tags.

Certain special processing is provided for the following types of DAV resources:

Any textual content types are automatically indexed in a free text index.

Any XML content types are indexed in the same free text index as other text plus can be queried with the *xcontains* SQL predicate.

Some well-known types of documents are parsed in order to extract metadata, such as author of MS Office document or musical genre of MP3 audio file or resolution of an image.

VSP/VSPX pages - DAV resources can be dynamic web pages written in VSP or VSPX. Requesting such a page with GET or POST will execute the logic on the page. This is very convenient way of building web applications.

DAV resources are stored in an ordinary relational table. The text and XML query features used with DAV are separately usable from SQL on any table with the appropriate indexes and are thus not intrinsically related to DAV.

WebDAV (Web Distributed Authoring and Versioning) is a communication protocol for the Internet implemented as an extension to HTTP 1.1. The WebDAV specification was published by the Internet Engineering Task Force (IETF) in February 1999.

Most operating systems have support for accessing data hosted on DAV servers.

DAV was designed to provide more methods for handling server resources. In addition to the usual HTTP methods such as GET, POST, HEAD, PUT, DELETE, OPTIONS are methods for making directories (or collections), a lock mechanism, copying of resources and collections, predefined and userd-defined properties of resources and collections.

DAV consists of HTTP extensions, often with a message body containing XML. DAV also provides a basic schema for resource metadata by allowing arbitrary XML properties to be attached to resources.

Virtuoso supports the following HTTP methods:

Table 19.1. Features List - Virtuoso Web DAV HTTP Method Support

HTTP Method	Description
HEAD	Meta method for examining server properties or network behavior.
GET	Retrieves documents from the server
POST, PUT	Posts or puts documents on the server
MKCOL	Creates a collection
MOVE	For moving resources and/or collections within the server
DELETE	Removes resources or collections of resources from the server
LOCK, UNLOCK	Locks or unlocks a resource or collection to protect from modification by others
PROPFIND, PROPPATCH	Retrieves and sets properties for collections and/or resources. In addition to arbitrary user-defined properties, Virtuoso supports two sets of predefined properties: standard DAV properties (type, creation time, modification time, size etc.) and Virtuoso-specific DAV properties (access control, tags, extensions for virtual collections.)
MGET (experimental)	Retrieves RDF metadata of a document according to URIQA rules
	Updates RDF metadata of a document according to URIQA rules and appropriate RDF Schemas

HTTP Method	Description
MPUT/MDELETE (experimental)	

See Also: External References

IETF RFC 2616 regarding HTTP 1.1

IETF RFC 2518 regarding WebDAV Specification

19.1.1. DAV User Accounts

Any non-disabled SQL account with the U_DAV_ENABLE column set to non-zero is a valid DAV account. The administration user interface provides a check box for enabling DAV access and will make a default home collection etc. Alternately, regular SQL can be used for setting the DAV flag on. The DAV API or protocol requests can be used for creating collections and resources for the user.

Any non-disabled SQL account with the U_DAV_ENABLE column set to non-zero is a valid DAV account. The administration user interface provides a check box for enabling DAV access and will make a default home collection etc. Alternately, regular SQL can be used for setting the DAV flag on. The DAV API or protocol requests can be used for creating collections and resources for the user.

There is an automatically created initial DAV enabled account called "dav". This has general administration privileges over all DAV.

There are four predefined DAV accounts.

User "dav" is the most powerful DAV account.

DAV group "administrators" usually consists of "dav" only; membership in this group does not give any special privileges but this group is assigned by default to resources that are owned by "dav". Thus group access rights to resources owned by "dav" can be used solely by users that are added to "administrators".

User "nobody" is a special account that acts as owner of all resources that have no real owner person or owner application. It is impossible to log on as "nobody" so it's impossible to use owner permissions of the resource.

Group account "nogroup" always consists of only one user "nobody"; User "nobody" can not become a member of any other group. It is impossible to add other users to "nogroup" or add "nogroup" to some role.

Note:

The WebDAV admin user "dav" can be deleted. However, after a server restart the initial setting of this account will be added again.

Important

It is strongly recommended that the Virtuoso administrator change the default account password of the DAV administrative user after installation.

Example 19.1. Manipulating WebDAV Users By Hand

```
USER_CREATE ('user', 'userpassword',
  vector ('SQL_ENABLE', 0, 'DAV_ENABLE', 1, 'PERMISSIONS', '110100000RR', 'DISABLED', 0));
```

This will create a new account named 'user' with password 'upwd', default permissions for new resources and collections created by the account will be '110100000RR' (equivalent of UNIX 'rw-r----' permissions, recursive free-text indexing and metadata extraction) and the account is enabled and ready to use.

```
DAV_ADD_USER ('user', 'userpassword', '110100000RR', 0, '/DAV/home/user/',
  'Full User Name', 'user@example.com', 'dav', 'davpassword');
```

this will do almost the same but it will describe the user in more details and require no DBA privileges -- only DAV password. In addition, DAV_ADD_USER () can both create a new user and grant DAV permissions to an already existing SQL user.

The following statement is used to grant role 'administrators' to the 'user' account.

```
GRANT ADMINISTRATORS TO "user";
```

or

```
USER_GRANT_ROLE ('user', 'administrators');
```

The following will disable the account 'user' until 'DISABLED' option is returned to zero.

```
USER_SET_OPTION ('user', 'DISABLED', 1);
```

DAV access permissions can be revoked permanently:

```
DAV_DELETE_USER ('user' , 'dav', 'davpassword');
```

See Also:
WS & DAV System Tables

19.1.2. WebDAV Authentication

Virtuoso WebDAV offers two types of authentication dependent on the connecting clients abilities. These are:

Basic (Clear Text) Authentication - sends passwords over the connection in clear text. Clear text passwords can be intercepted and read so should be avoided or used only if you encrypt passwords through SSL.

Digest Authentication - passwords are always transmitted in an MD5 hash.

Basic Authentication

Basic Authentication is a widely used, industry-standard method for collecting user name and password information. The following steps outline how a client is authenticated using Basic authentication:

1. The client browser displays a dialog box for a user to enter a user name and password (his/her credentials).
2. The client browser then attempts to establish a connection to the server using the user's credentials. The clear text password is Base64-encoded before it is sent over the network.
3. If a user's credentials are rejected, the client may re-display the authentication dialog box to re-enter the user's credentials. Failing to supply correct details will terminate the connection, reporting an error to the user.
4. When Virtuoso verifies that the user name and password are valid, a connection is established.

The advantage of Basic authentication is that most clients support it. The disadvantage is that it transmits passwords in an unencrypted form. Simple network monitoring can easily reveal your password. Basic authentication is not recommended unless you are confident that the connection between the user and Virtuoso is secure.

Note:
Base64 encoding is not encryption. A Base64-encoded password can be easily intercepted by a network sniffer and easily decoded.

Digest Authentication

Digest authentication provides a security improvement over Basic authentication in how a user's credentials are sent across the network. Digest authentication transmits credentials across the network as an MD5 hash, or message digest, where the original username and password cannot be deciphered from the hash. Digest authentication relies on the HTTP 1.1 protocol as defined in RFC 2617, which not all browsers support.

The following steps outline how a client is authenticated using Digest authentication:

Figure 19.1. Digest Authentication

1. The client requests a file or connection from Virtuoso.
2. Virtuoso challenges the request, informing that client: Digest is in use, what the realm name is.
3. The client prompts the user for credentials. The client creates an MD5 hash of the credentials and the realm name and resubmits the request, this time supplying the MD5 hash.
4. If Virtuoso approves the credentials then the resource or connection is granted to the client, and the data is returned.

19.1.3. WebDAV Symbolic Links

Virtuoso supports a special type of WebDAV resources, a redirect reference resource, named WebDAV links. This is to extend the WebDAV service to allow multiple access paths to existing resources.

As with conventional HTTP, the redirect reference simply responds to the client with an HTTP/1.1 302 (Found) status code, redirecting the client to a different resource, the target specified in the redirect resource, using the `Location:` header. This behavior is equivalent to UNIX symbolic links. A redirect reference makes it possible to access the target resource indirectly through any URI mapped to the redirect reference resource. The integrity is not guaranteed for associated redirect reference resources.

WebDAV links do not provide a way to circumvent WebDAV security or virtual directories. The target resource must be directly available to the client being redirected.

WebDAV links are achieved by adding a special WebDAV property, '`redirectref`', whose value must contain the reference target URL.

WebDAV link targets are not limited to the WebDAV repository, and may target any HTTP accessible resource.

The WebDAV links can be made with DAV API function `DAV_PROP_SET()` , or can be done via the Content Management interface of the Admin UI. In the admin UI select WebDAV/WebDAV Services/Content Management and press on the *Create Link* button to create a link. In the form choose the target, permissions, owner and enter the name of the link. Pressing the *Add* button will create the new link in the current WebDAV folder.

Example 19.2. WebDAV Links Programmatically

Assuming that we are creating the link as the administrator using the default WebDAV administrator username and password, creating a link from `/a/b.html` to `/c/d/f.html` could be as follows:

```
SQL> select DAV_RES_UPLOAD ('/DAV/a/b.html','','','110100000NN', 'dav', 'nobody', 'dav', 'dav');

SQL> DAV_PROP_SET ('/DAV/a/b.html', 'redirectref', '/c/d/f.html', 'dav', 'dav');
```

The target, '`/c/d/f.html`' could be any HTTP URL. In this case it will be a resource on the same HTTP server.

See Also:

DAV Add and Update functions and DAV Manipulation functions .

WebDAV links behaviour in resource manipulation requests. When some link is moved the target will have the same properties as source, ie. it will be a link. When a copy operation is made the target will have content of the reference i.e. it will be

a resource, not a link. Delete operation on link will remove only the redirect reference resource, not the referenced target itself. Content upload requests will change the content of the referenced target.

19.1.4. Access Right Permissions of Web Resources

The WebDAV resources have two sorts of access right permissions. 'Classical' UNIX filesystem style permissions let assign different permissions for owner user, owner group and public access. Access Control Lists (ACLs) let assign permissions in more flexible Windows style but they are less convenient for simple tasks. Both set of permissions can be applied to the same resource or collection. In any case, each resource or collection (directory/folder) can have defined Read, Write and Execute permissions. The write permission applies to operations which perform content or property change or locking as PUT, PROPPATCH, MOVE, destination URI in COPY, LOCK, UNLOCK, DELETE, MPUT and MUPDATE. The read permission applies to read operations as GET, POST, PROPFIND, HEAD, source URI in COPY and MGET; read permission is also required for any write operation.

See Also

RFC2518 for more details of methods/operations

The execute permission applies only to the active content stored in the WebDAV domain. If a VSP or VSPX page stored in a WebDAV domain has execute permissions then retrieval of that page will perform execution of active content. Note also a special flag to the virtual directory which can override the execution flags for active pages. (for the details how this flag can be set see 'options' in VHOST_DEFINE() function). If such a flag is set to the virtual directory, then all active pages under that directory (direct and indirect children) will be treated as execution flag of the resource is set. In almost any case, user should have both read and execute permissions to access active content.

Resources with the following extensions are treated as executable content: .vsp, .vspx, .xml if this has the XML template properties set. Any other extension is also considered executable if there is a corresponding WS.WS."__http_handler_<extension>" PL procedure.

The UNIX style permissions can be set for user, group or public access. ACL consists of records called Access Control Entries (ACEs). Every ACE allow or deny some sorts of access to an individual user or to a group. Resource owner or administrator can add an "explicit" ACE to the ACL of particular resource or "recursively" add "implicit" ACEs to every resource and subcollection of some collection. The server checks permissions in the following order:

1. match the user part of UNIX-style permissions to the specific operation, if user is the owner of the resource.
2. match the user group part of UNIX-style permissions to the specific operation, if user belongs to a group which owns the resource.
3. match the public part of UNIX-style permissions to the specific operation.
4. If Access Control List is not empty, scan ACEs from the beginning of the list to the end. The loop stops at the first ACE that mentions either the user in question or one of the roles granted to it. Depending on the type of the ACE, the access is either granted or denied and the rest of list has no effect.
5. If ACL contains no appropriate ACE then the access is denied.

According to these rules, the order of records in ACL is important. ACEs are ordered following two rules: "deny" ACEs has higher priority than "allow" ACEs; "local" rules has higher priority that "global":

1. All explicit ACEs are placed in a group before any inherited ACEs.
2. Within the group of explicit ACEs, access-denied ACEs are placed before access-allowed ACEs.
3. Inherited ACEs are placed in the order in which they are inherited. ACEs inherited from the child object's parent come first, then ACEs inherited from the grandparent, and so on up the tree of objects.
4. For each level of inherited ACEs, access-denied ACEs are placed before access-allowed ACEs.

New resources and collections take their initial permissions from the user default permissions mask, U_DEFAULT_PERMS of SYS_USERS. (see: WebDAV Users Administration). In the case of a resource created as public or a collection mask of '110110110' (equivalent of UNIX 'rw-rw-rw'), then everybody can read and write it.

Example 19.3. WebDAV Permissions

Consider a resource with the following permissions:

```
'111110100'
```

This permission string is equivalent to UNIX 'rwxrw-r--'. The resource can be executed, read and written to by the owner, group

members can read and write to it, and non authenticated (public) users or users not belonging to the group group can only read it.

Every resource or collection has two additional flags in permissions, that instruct the server whether resources should be indexed for free-text search and whether resources should be parsed to extract metadata:

Free-text indexing flag can take one of three values: "N", "R" and "T". If a resource has this flag set to R or T then the resource will be free-text indexed. If a collection has this flag set to T then resources directly contained within the collection will be free-text indexed. If a collection has this flag set to R then resources contained within the collection will be free-text indexed, and the setting will be applied to all members and collections underneath recursively. New resources and collections acquire this setting from their parent collection.

Similarly, metadata extraction flag can take one of three values: "N", "R" and "M". If a resource has this flag set to R or M then its metadata are extracted. If a collection has this flag set to T then metadata are extracted from resources directly contained within the collection. If a collection has this flag set to R then metadata are extracted from resources contained within the collection, and the setting will be applied to all members and collections underneath recursively. New resources and collections acquire this setting from its parent collection.

19.1.5. DAV and RDF Metadata

When a Virtuoso server has a URIQA default host setting, it will make metadata extracted from DAV resources available for querying via SPARQL. All metadata for public readable DAV resources are stored in a system graph. The graph IRI is composed from the DAV path of the resource and the URIQA default host name.

see DefaultHost in the URIQA section of the virtuoso.ini file, described in the URIQA section of the documentation for details on configuration.

The automatic maintenance of the SPARQL queryable metadata can be disabled and enable using the function DB.DBA.DAV_REPLICATE_ALL_TO_RDF_QUAD. An argument of 1 enables this and a 0 disables this. The setting stays in effect until it is changed with the same function. For new database, the feature is on by default. Old databases are by default upgraded to have a SPARQL queryable DAV metadata graph upon startup if the URIQA default host name is defined.

If the URIQA default host name changes, the RDF graph can be updated by simply re-enabling the feature. This will adjust the graph and resource IRI's.

If the URIQA default name of the host is example.com, then, the graph will be http://example.com/DAV .

The IRI's of DAV resources will be like http://example.com/DAV/docsrc/XMLDOM.xml , meaning that these are directly usable from a user agent for accessing the resource.

```
SQL> sparql select ?s ?o from <http://example.com/DAV> where {?s <http://www.openlinksw.com/schemas/DAV#o
s                                                                                o
VARCHAR                                                                          VARCHAR
_______________________________________________________________________________

http://example.com/DAV/docsrc/2pc.xml                                        mailto:somebody@example.do
http://example.com/DAV/docsrc/Virtdocs.spp                                   mailto:somebody@example.do

SQL> sparql select ?p ?o from <http://example.com/DAV> where {<http://example.com/DAV/docsrc/2pc.xml> ?p
p                                                                                o
VARCHAR                                                                          VARCHAR
_______________________________________________________________________________

http://purl.org/dc/terms/created                                                 2006-05-23 15:10:32
http://purl.org/dc/terms/modified                                                2006-05-23 15:10:32
http://www.openlinksw.com/schemas/DAV#ownerUser                                  mailto:somebody@example
http://purl.org/dc/terms/extent                                                  7850
```

The examples above show how simple SPARQL queries can be used to retrieve information about DAV resources.

The properties supported for all public readable resources are:

http://purl.org/dc/terms/created - The creation date as SQL datetime.

http://purl.org/dc/terms/modified - Modification time as SQL datetime.

http://www.openlinksw.com/schemas/DAV#ownerUser - The contents of u_e_mail in sys_users for the SQL account owning the resource. This has the protocol prefix mailto:, as in mailto:somebody@example.com.

http://purl.org/dc/terms/extent The size of the resource in bytes as a SQL integer.

http://www.openlinksw.com/schemas/DAV#tag - There is one triple for each public tag of the DAV resource. The value is the string of the tag as a SQL string.

http://www.w3.org/1999/02/22-rdf-syntax-ns#type - The RDF schema for MIME-type of of the DAV resource. If the resource is not recognized as one of the below, this predicate will be omitted.

Table 19.2. RDF Schema by MIME-type

MIME- ype	RDF Schema
application/bpel+xml	http://www.openlinksw.com/schemas/WSDL#
application/doap+rdf	http://www.openlinksw.com/schemas/doap#
application/foaf+xml	http://xmlns.com/foaf/0.1/
application/google-kinds+xml	http://www.openlinksw.com/schemas/google-kinds#
application/license	http://www.openlinksw.com/schemas/OplLic#
application/mods+xml	http://www.openlinksw.com/schemas/MODS#
application/msexcel	http://www.openlinksw.com/schemas/Office#
application/mspowerpoint	http://www.openlinksw.com/schemas/Office#
application/msproject	http://www.openlinksw.com/schemas/Office#
application/msword	http://www.openlinksw.com/schemas/Office#
application/msword+xml	http://www.openlinksw.com/schemas/Office#
application/opml+xml	http://www.openlinksw.com/schemas/OPML#
application/pdf	http://www.openlinksw.com/schemas/Office#
application/rdf+xml	http://www.openlinksw.com/schemas/RDF#
application/rss+xml	http://purl.org/rss/1.0/
application/wsdl+xml	http://www.openlinksw.com/schemas/WSDL#
application/x-openlink-image	http://www.openlinksw.com/schemas/Image#
application/x-openlink-photo	http://www.openlinksw.com/schemas/Photo#
application/x-openlinksw-vad	http://www.openlinksw.com/schemas/VAD#
application/x-openlinksw-vsp	http://www.openlinksw.com/schemas/VSPX#
application/x-openlinksw-vspx+xml	http://www.openlinksw.com/schemas/VSPX#
application/xbel+xml	http://www.python.org/topics/xml/xbel
application/xbrl+xml	http://www.openlinksw.com/schemas/xbrl#
application/xddl+xml	http://www.openlinksw.com/schemas/XDDL#
application/zip	http://www.openlinksw.com/schemas/Archive#
text/directory	http://www.w3.org/2001/vcard-rdf/3.0#
text/eml	http://www.openlinksw.com/schemas/Email#
text/html	http://www.openlinksw.com/schemas/XHTML#
text/wiki	http://www.openlinksw.com/schemas/Wiki#

Additional predicates may exist as a result of resource type specific metadata extraction.

19.1.6. Special Attributes of Web Resources

The Virtuoso WebDAV implementation provides a set of special attributes (properties) for the resources to manipulate the retrieval of XML documents. Properties can be set generate dynamic content based on XML/SQL queries. Special properties can

also be applied to WebDAV folders for storing XML data in a special pre-parsed persistent XML format.

The following special properties are supported:

- **xml-stylesheet.** must contain a valid URL to an XSLT style sheet. Upon request of an XML document with this property set, the WebDAV server will automatically perform the transformation of the XML document and will send the result of transformation to the user-agent instead of the original XML source. This property is only settable for documents having MIME type text/xml.
- **xml-sql.** must contain a valid XML/SQL query (see also: FOR XML statements). When this property is set the server will execute the query and the XML result will be sent to the client. Note that if xml-stylesheet is also supplied for this resource the result will be transformed and then sent to the client.
- **xml-sql-root.** specifies the root element name of automatically generated XML resources when xml-sql is specified. Otherwise it has no effect.
- **xml-sql-dtd.** can be specified as a string value 'on' or valid URL. When the value is 'on' the DTD declaration will be implied in the automatically generated XML resource. Otherwise when the URL is supplied this URL will be included in the DTD declaration of the automatically generated XML resource. If xml-sql is not specified this property has no effect.
- **xml-sql-schema.** specifies the URI for the XML Schema. This schema URI will be included in the XML header, to allow client-side schema validation. If xml-sql is not specified this property has no effect.
- **xper.** can be specified for a collection (folder). When this property is set on a WebDAV collection, the direct resource members will be stored and updated as persistent XML. Note that existing resources will not be changed until they are updated. The reversal of this property for collections will not change the resources at the same time, they will be reverted to the text/xml storage on first update operation. Resources already stored as XML persistent documents also have the xper property set, but manipulation of this property must not be used to revert the storage.

19.2. URIQA Semantic Web Enabler

Virtuoso supports the URIQA (URI Query Agent) extension of HTTP WebDAV protocol. URIQA adds three new methods to HTTP in order to retrieve, add and remove RDF metadata about a given subject. The subject is identified by its URI. If the subject is a DAV resource then URIQA will usually reuse the DAV URI of the resource. If the subject is not a resource but something else (physical entity, imaginary thing or vocabulary item) then URIQA can be used to process metadata about the subject even if the subject itself can not be accessed via HTTP.

URIQA-specific HTTP methods are called MGET (to retrieve existing metadata), MPUT (to add or update RDF triples) and MDELETE (to remove some or all triples). A single URIQA request usually deals with a single subject that is specified by request URI. The MGET response, however, can return metadata about more than one RDF subject, e.g., the request about a book can return both data about the book itself plus some data about persons who are known as authors of the book.

In addition to URIQA-specific HTTP methods, Virtuoso implements a semantic web service interface that allows plain HTTP clients to access metadata using traditional GET or POST HTTP methods.

The Virtuoso URIQA implementation allows flexible configuration using an ordered list of request handlers. Every handler has a pattern for URIs; if the URI in the request does not match the pattern then the handler is ignored, otherwise a callback function of the handler is called to process the request. The default configuration of Virtuoso server will try three sorts of actions.

If the URI points to a resource located at the server then the first handler returns DAV metadata of the resource.

If the URI points to a resource that is outside the server, and URI is of sort 'http://...' then Virtuoso will send a URIQA web service request to the remote server specified in URI in hope that it will return at least something, the response is passed to the client unchange.

Any other URI results in an error.

Note:

URIQA is not yet a stable standard. Virtuoso implements draft of URIQA proposal from Nokia, dated 2004. As URIQA will evolve, future versions of Virtuoso will implement updated versions of the specification. There is no warranty that future implementations will be compatible with the current one.

See Also: External References

The Nokia URI Query Agent Model

19.2.1. URIQA HTTP Methods

All three methods have a set of HTTP header parameters to specify the precise URI of the subject. HTTP does not require that every resource is accessible via a single valid URI, so many equivalent URLs can point to same resource and the result of typical HTTP request does not change if one of equivalent URLs is replaces with some other. Unlike HTTP GET, HTTP PUT etc., metadata methods may return different results for different URLs even if these URLs are equivalent for other methods. URIQA rules are very simple.

If the URIQA request header contains 'URIQA-uri' parameter line then the value of this parameter is used and any other URI data are ignored.

If the URIQA request header contains 'Host' parameter line then the URL from the first line of the request is patched to contain host name specified by 'Host' parameter, no matter whether the original URL contains host or what host name or network interface or port is user by client to connect to the server.

If the URIQA request header does not contain 'URIQA-uri' or 'Host' line then the URL from the first line of the request is used 'as is', extended by host name from 'DefaultHost' URIQA configuration parameter if needed.

Example 19.4. Examples of MGET Requests

The following requests are all equivalent:

Request 1. 'URIQA-uri' is used, the rest does not matter.

```
MGET /foo HTTP/1.1
Host: example.com
URIQA-uri: http://example.com/foo
```

Request 2. 'URIQA-uri' is missing, 'Host' is used, the host name www.example.com is ignored.

```
MGET http://www.example.com/foo HTTP/1.1
Host: example.com
```

Request 3. The URI from the first line is used verbatim. This is unsafe, because proxy servers can alter the URI, e.g. by adding port number.

```
MGET http://example.com/foo HTTP/1.1
```

Request 4. The URI from the first line is used, but host name is retrieved from 'DefaultHost' URIQA configuration parameter. If the parameter is set to example.com then the request is equivalent to previous.

```
MGET /foo HTTP/1.1
```

MGET Method

MGET request contains a subject URI and the response consists of RDF/XML representation of an RDF graph with metadata about the subject. In many cases, the returned graph is a Concise Bounded Description of the resource or something similar, but it can be of any sort.

There are no integrity rules. E.g., if a response for request about subject A contains some data about B then the request about B may return same or different data, or even report that B does not exists. If URI refers to non-existing resource or even to a non-existing server or protocol then the response can be an 'not found' error or an empty graph or even a non-empty graph, depending on the handler that processed the request.

Usually MGET request consists of only subject URI specification, but it can contain any other parameters such as an authentication or even the HTTP request body with extra data for some particular handler. For Virtuoso DAV resources, MGET will need read permission on the subject resource, because the resulting RDF is retrieved from 'http://local.virt/DAV-RDF' property of the resource.

MPUT Method

MPUT request contains an HTTP header that describe a subject URI and contains Content-Length, and the body must be an RDF/XML that consist of triples that should be added. The server will try to add new RDF triples from the body to the description of the subject. In some cases, the server will replace obsolete triples with triples from the body, e.g., if some RDF Schema is in use that states for a predicate that it can not have more than one value for any given subject.

There are no integrity rules. If MPUT request with subject A submits data about resource B then the updated data may become visible via MGET request with subject A and stay unchanged if retrieved directly by MGET with subject B. For instance, the default request handler for DAV will update only 'http://local.virt/DAV-RDF' DAV property of the subject resource not touching any DAV properties of resources named in the request.

A client application can not use MPUT with subject URI that refers to a non-DAV Virtuoso resource, because disk-resident resources do not have DAV properties, including DAV metadata properties. MPUT can refer to nonexisting Virtuoso DAV resource only if the name of this resource has been already locked for uploading of the resource. The most reliable way, however, is to upload the resource first and update metadata only after the uploading. There are two reasons to do operations in this sequence. First of all, Virtuoso can automatically extract some metadata from the content of uploaded resource and if MPUT happens after the upload then MPUT data can properly overwrite automatically extracted values. An additional reason is that resource uploading will set the MIME-type of the resource and may associate some RDF Schemas with the resource; hence MPUT can properly update some triples instead of storing multiple values for some predicate that should have only one value according to RDF Schema.

For Virtuoso DAV resources, MPUT will need both read and write permissions on the subject resource, because 'http://local.virt/DAV-RDF' property of the resource is first retrieved and then updated.

MDELETE Method

MDELETE request contains an HTTP header that describe a subject URI and may contain the body. If present then the body must be an RDF/XML that consist of triples that should be deleted. If the body is totally missing then MDELETE removes all metadata associated with the subject URI.

There are no integrity rules. If MDELETE request with subject A removes triples about resource B then these triples may stay visible if retrieved directly by MGET with subject B. For instance, the default request handler for DAV will update only 'http://local.virt/DAV-RDF' DAV property of the subject resource not touching any DAV properties of resources named in the request.

For Virtuoso DAV resources, MPUT will need both read and write permissions on the subject resource, because 'http://local.virt/DAV-RDF' property of the resource is first retrieved and then updated.

19.2.2. URIQA Web Service

Virtuoso provides the '/uriqa/' web service for clients that do not support URIQA-specific methods. Instead of passing URI and method name in HTTP parameter lines, web service calls pass them as part of web service URI. The beginning of the path can be any, starting from '/uriqa/' or '/URIQA/'. The following two requests are to retrieve metadata about 'http://example.com/foo'.

```
GET /uriqa?uri=http%3a%2f%2fexample%2ecom%2ffoo HTTP/1.1

GET /uriqa?uri=http%3a%2f%2fexample%2ecom%2ffoo&method=MGET HTTP/1.1
```

The following request header is for MPUT

```
GET /uriqa?uri=http%3a%2f%2fexample%2ecom%2ffoo&method=MPUT HTTP/1.1
```

The URIQA web service does not need complicated rules for URI passing because the request can not be significantly changed by any proxy. The value of the 'uri' parameter should be an absolute URI.

19.2.3. URIQA Section in Virtuoso Configuration File

By default,the Virtuoso server acts only as URIQA proxy, i.e. it redirects incoming requests to other servers without trying to return metadata about DAV resources or other data stored on the server itself. To let URIQA retrieve local metadata, the Virtuoso server should know names that can be used by clients to access it. Virtuoso configuration file, e.g., virtuoso.ini, can contain these

names as parameters in "[URIQA]" section

"DefaultHost" is the "canonical" server name that is used to identify the service. It should be either server name including domain name, or an IP address in standard notation, if the server does not have any name. If Virtuoso default HTTP port is not equal to 80 then the port should be mentioned, e.g. "www.example.com:8088".

"LocalHostNames" lists all names that can be used to access the server, such as server names with and without domain name, IP addresses in Internet and intranets etc. The list is comma-delimited string of names. If an URIQA client can reside on server's box, e.g. for debugging purposes, then it may be worth to add names "localhost, localhost.localdomain, 127.0.0.1" to this list.

"LocalHostMasks" is similar to "LocalHostNames" but it lists patterns for names in SQL "like" operator style. If Virtuoso listens at multiple ports and it is the only URIQA enabled service on the machine then it can be convenient to specify "LocalHostMasks = www.example.com:%" instead of "LocalHostNames = www.example.com:8088, www.example.com:8089, www.example.com:8090 ...".

"Fingerprint" is a string that identifies a group of servers that shares same metadata, such as servers that replicate each other. It is an error if two servers have the same fingerprint string and one of them tries to redirect a URIQA request to another instead of prepare an response locally. Such behavior indicates configuration error, and the use of fingerprints help administrator to get a meaningful diagnostics, because suspicious URIQA requests become signed by all intermediate Virtuoso proxies. If this parameter is not specified then a random unique string is created and stored in the database, so you don't have to specify this parameter for typical installations.

"DynamicLocal" is a flag (1 or 0), when it is on and the host part of the IRI matches the Host header of the HTTP request in context or the DefaultHost if outside of HTTP context, then this is replaced with local: before looking up the IRI ID. Even if DynamicLocal is not on and the `local:` prefix occurs in the IRI string being translated to IRI_ID, the translating the IRI_ID back to the IRI name will depend on the context as described as follows: When returning IRI's from id's, this prefix is replaced by the Host header of the HTTP request and if not running with HTTP, with the DefaultHost from this section. The effects of DynamicLocal = 1 can be very confusing since many names can refer to the exact same thing. For example, if the DefaultHost is http://dbpedia.org, `iri_to_id ('http://dbpedia.org/resource/Paris') = iri_to_id ('local:///resource/Paris)` is true and so is `'http://dbpedia.org/resource/Paris' = id_to_iri (iri_to_id ('local:///resource/Paris'))` These hold in a SQL client context, i.e. also when connected through RDF frameworks like Jena or Sesame. When running a SPARQL protocol request, the Host: header influences the behavior, likewise when using web interactive SQL in Conductor.

These configuration parameters are "sticky". If they're found in configuration file then they are preserved in the database registry. If configuration file has changed then new values will be used after server restart. If database dump is replayed on a server whose configuration file does not contain these parameters then values from dump will stay in the registry. If database dump is replayed on a server whose configuration file contains other values then values from dump will stay in the registry till server restart.

19.2.4. URI Matching Rules

A simple installation does not require any special configuration of URIQA except specifying server names in the [URIQA] section of configuration file (virtuoso.ini). However complex applications may need from URIQA more than simple retrieval of metadata of DAV resources. Like HTTP virtual hosts, URIQA may require different processing for different URIs, so Virtuoso offers appropriate tools.

When the URIQA server gets an URI to process, it reads the system table WS.WS.URIQA_HANDLER to find out the procedure that can access metadata about some range of URIs. This table is defined as follows:

```
create table WS.WS.URIQA_HANDLER
(
  UH_ID integer not null primary key,
  UH_ORDER integer not null,
  UH_NAME varchar not null unique,
  UH_MATCH_COND varchar not null,
  UH_MATCH_ENV any,
  UH_HANDLER varchar not null,
  UH_HANDLER_ENV any
)
create index URIQA_HANDLER_ORDER_NAME on WS.WS.URIQA_HANDLER (UH_ORDER, UH_NAME)
;
```

The server scans the table in order of ascending values in UH_ORDER column, and checks whether the request URI matches the condition specified by UH_MATCH_COND and UH_MATCH_ENV. As soon as an appropriate row is found, a function with name specified by UH_HANDLER is called with parameters that describe the request plus any extra application-specific data as stored in UH_HANDLER_ENV. The function should either compose a response and set a flag to 1 or do nothing and set a flag to 0. If 1 is set then the processing of the request is complete, otherwise the server resumes table scan.

At server startup, up to three records are automatically added into WS.WS.URIQA_HANDLER.

First record has UH_ORDER equal to 100. It tells the server that if an URI has server name equal to one of names listed in "LocalHostNames" configuration parameter then metadata should be retrieved from local DAV of the server.

Second record is very similar, it also has UH_ORDER equal to 100, but uses SQL 'like' operator instead of '='. It tells the server that if an URI has server name like one of masks listed in "LocalHostMasks" configuration parameter then metadata should be retrieved from local DAV of the server.

The third record has UH_ORDER equal to 999, and tells the server to act as URIQA proxy if the requested URI starts with "http" protocol name.

Applications can add more lines to the table to handle different sorts of URIs via different application specific functions. The name of function should begin with "WS.WS.URIQA_HANDLER_", the rest is as specified by UH_HANDLER of the row. The signature of function should be

```
function WS.WS.URIQA_HANDLER_myexample (
  inout op varchar,   -- operation name, 'MGET', 'MPUT' or 'MDELETE';
  inout uri varchar,  -- request URI;
  inout split any,    -- request URI split by WS.WS.PARSE_URI into parts;
  inout body any,     -- the body of the request;
  inout params any,   -- get_keyword style vector of parameters of the request;
  inout lines any,    -- vector of lines of HTTP request header;
  inout app_env any,  -- any application-specific data from UH_HANDLER_ENV;
  inout is_final integer -- status flag. Function sets the flag to 1 to report that the request response
  ) re0turns any              -- returns a status vector, see below.
```

Status vector describes either the reason why the request has failed, or the success status. It consists of four elements:

SQL_STATE as five-char string, "00000" if success;

DAV error code as an integer, if the operation has failed due to DAV error, 0 if success or an error other than DAV;

HTTP status as three-digit string, such as "200" for "OK" or "404" for "not found";

Brief description of an error, such as HTTP response status ("OK", "not found" etc.) or SQL_MESSAGE, "OK" if success;

In case of DAV error, elements 3 and 4 can be set to NULL to generate proper values automatically.

Examples are:

```
vector ('00000', 0, '200', 'OK');
vector ('URIQA', 0, '500', 'The remote URIQA server returned an invalid header');
vector ('URIQA', -1, '404', 'Invalid URI; Ill formed or missing path to the resource');
vector ('URIQA', -12, null, null);
```

The current version of Virtuoso supports the following names of matching operations for use in UH_MATCH_COND:

"schema =" -- URI schema name should be equal to UH_MATCH_ENV;

"server =" -- URI server name (including port, if specified) should be equal to UH_MATCH_ENV;

"server like" -- URI server name (including port, if specified) should be "like" to UH_MATCH_ENV;

"server in" -- URI server name (including port, if specified) should be member of UH_MATCH_ENV vector of strings;

"server like in" -- URI server name (including port, if specified) should be "like" to one of members of UH_MATCH_ENV vector of strings;

"default" -- Any URI will match so any request is passed to the handler if not handled before.

19.3. Mail Delivery & Storage

19.3.1. The SMTP Client

smtp_send_dedup

For detailed description and example use of the function, see smtp_send in the Functions Reference Guide.

19.3.2. POP3 Server

The Virtuoso POP3 Server implementation supports the following commands as defined in RFC - 1939:

DELE
LIST
NOOP
PASS
QUIT
RETR
RSET
STAT
TOP
UIDL
USER

The POP3 Server listening port is configured in the HTTP section of the virtuoso.ini.

POP3Port = xxx, in HTTP section.

If the port is not defined in the configuration file then the POP3 server subsystem will be disabled.

Users and passwords of the POP3 Server are described in the system view WS.WS.SYS_DAV_USER over SYS_USERS. These users can be administered from the users section of the Virtuoso Administration User Interface.

The system table DB.DBA.MAIL_MESSAGE is used to stored the messages.

19.3.3. Storing Email in Virtuoso

A generic mail delivery driver enables SMTP servers to deliver email to be stored in either Virtuoso or non Virtuoso SQL database.

Mail storage can occur in a number of ways:

1. Replacing the default generic SMTP mail handler (the program "procmail")
2. Replacing the mail storage settings for individual users Replacing Default SMTP Mail Handler Sendmail

Replacing procmail As Default Mail Handler In Sendmail Configurations

When this configuration is in use, the mails for all of your "sendmail" mail recipients are written to a SQL Database table.

1. Copy the file "odbc_mail.default.ini" to "/etc/odbc_mail.ini"

   
   Important
   Make sure that /etc/odbc_mail.ini is NOT GROUP/WORLD writable. virt_mail will fail to run if it is.
2. Copy or symbolically link the file "virt_mail" to "/usr/bin/virt_mail"
3. Edit /etc/odbc_mail.ini and change the login settings to match your current database installation.

   Note: If you don't have or want to use procmail, comment out the "Fallback" setting in the "[Deliver]" section.
4. Edit the file "/etc/sendmail.cf" as described below:

   Change

   ```
   Mlocal, P=/usr/bin/procmail, F=lsDFMAw5:/|@qSPfhn9, S=10/30,
     R=20/40,T=DNS/RFC822/X-Unix, A=procmail -Y -a $h -d $u
   ```

   To:

   ```
   Mlocal, P=/usr/bin/virt_mail, F=lsDFMA5:/|@qSPhn9, S=10/30,
   ```

```
R=20/40, T=DNS/RFC822/X-Unix, A=virt_mail -c /etc/odbc_mail.ini -l $u -s $g
```

NOTE: The changes to the F= setting involves removing both the 'w' and 'f' flags.

The removal of the 'w' flag affects lookups in /etc/passwd, which are no longer required if all mail delivery goes into a SQL Database. The default ini file is set up to maildrop via procmail, which will perform the check correctly.

Replacing The Mail Storage Settings For Individual Users

Using Sendmail

1. Copy odbc_mail.default.ini to ~the_user_name/odbc_mail.ini

   Important
   Make sure that /etc/odbc_mail.ini is NOT GROUP/WORLD writable. virt_mail will fail to run if it is.
2. At the end of the file ~the_user_name/.procmailrc put something like:

   ```
   :0:
   | /usr/bin/virt_mail -c .odbc_mail.ini -l the_user_name
   ```

   replacing the_user_name with the user you are setting up.

   Note: the '-l ..' parameter is used to relate the local recipient to the database user for which the maildrop is done. See the remarks in the odbc_mail.default.ini file.
3. Adjust the parameters in .odbc_mail.ini to match your configuration

   Important
   disable Fallback delivery in the .odbc_mail.ini file.
   Comment out the "Fallback =" entry in the .odbc_mail.ini or set it to something that does not involve procmail. This may lead to drop loops, otherwise.

Using Qmail

1. Copy odbc_mail.default.ini to ~the_user_name/.odbc_mail.ini

   Important
   Make sure that the .odbc_mail.ini file is NOT GROUP/WORLD writable. virt_mail will fail to run if it is.
2. If you are setting up a .qmail, you can simply do the following:

   ```
   | /usr/bin/virt_mail -m qmail -c .odbc_mail.ini
   ```

   If you are setting up .qmail-default or .qmail-<some_alias_name>, make sure you adjust the RemovePrefix accordingly in the odbc_mail.ini.

   This also works if a single user is receiving mail for an entire (virtual) domain.

   If you want to configure qmail so that user 'db' gets all mail for example.com, do the following:

   a. create a new user db in /etc/passwd etc.
   b. remove example.com from /var/qmail/control/locals
   c. add to /var/qmail/controls/virtualdomains:

      ```
      example.com:db
      ```

   d. adjust /var/qmail/users/assign accordingly:

      ```
      =db:db:<uid>:<gid>:<home>:::

      +db-:db:<uid>:<gid>:<home>:-::
      ```

      or run qmail-pw2u < /etc/passwd > /var/qmail/users/assign then run qmail-newu and restart qmail
   e. in ~db/.qmail-default, put:

```
| /usr/bin/virt_mail -m qmail -c .odbc_mail.ini
```

f. in .odbc_mail.ini, set:

```
RemovePrefix=db-
```

Now, mail to info@example.com will be delivered to the qmail alias db-info@example.com and is stored into the database for user 'info'.

3. Adjust the parameters in .odbc_mail.ini to match your configuration

Using Courier

1. Copy odbc_mail.default.ini to ~the_user_name/.odbc_mail.ini

Important

Make sure that the .odbc_mail.ini file is NOT GROUP/WORLD writable. virt_mail will fail to run if it is.

2. If you are setting up a .courier, you can simply do the following:

```
| /usr/bin/virt_mail -mcourier -c .odbc_mail.ini
```

If you are setting up .courier-default or .courier-<some_alias_name>, make sure you adjust the RemovePrefix accordingly in the .odbc_mail.ini.

Note: although courier is very similar to qmail in this respect, it is different from qmail in how it handles exit codes. If you use -mqmail while running under courier, you'll get the wrong exit codes, so mail is bounced instead of retried.

Note also that if delivering to multiple recipients in a .courier file, make sure the virt_mail is specified first. This is because if the virt_mail fails with a temporary error, the other recipients will get another drop when courier re-attempts to deliver the mail.

Right:

\w

```
| /usr/bin/virt_mail -mcourier -c .odbc_mail.ini
./Maildir
```

Wrong:

```
./Maildir
| /usr/bin/virt_mail -mcourier -c .odbc_mail.ini
```

3. Adjust the parameters in .odbc_mail.ini to match your configuration

Using EXIM

Here are code snippets for Exim that perform maildrops into the odbc database.

```
## IN TRANSPORT SECTION

# Delivers into the database
odbc:
  driver = pipe
  command = /usr/bin/virt_mail \
        -c /etc/odbc_mail.ini \
        -s "${if def:return_path{$return_path}{MAILER-DAEMON}}" \
        -l "$local_part"
  user = USERNAME
  return_path_add
  delivery_date_add
  prefix =
  suffix =
  temp_errors = 73 : 74 : 75
  return_fail_output

# NOTE: Make sure the USERNAME in the 'user = USERNAME' setting matches
# the owner of /etc/odbc_mail.ini, because this file must have mode 0600.
```

```
# Consider creating a new user account for this delivery only.
# You should specify 'user = root' here only if Fallback delivery is
# configured in /etc/odbc_mail.ini (for procmail fallback delivery
# for instance)

## IN DIRECTORS SECTION

# Attempts delivery of all mail into the database
to_db:
  driver = smartuser
  transport = odbc
  require_files = /etc/odbc_mail.ini
  #
  # uncomment line below to deliver all mail to db-XXX into the database, for any
  # value of XXX. For this to work, set "RemovePrefix = db-" in
  # the [Translate] section in /etc/odbc_mail.ini
  #prefix = db-
```

Mail System Tables

The electronic mail accounts are regular SQL accounts.

Id (MM_ID) of the message is unique per account/folder (folder can be 'Inbox' etc.)

See Also:
Mail system tables

Pop3 Client

The Virtuoso POP3 Client implementation can retrieve messages from any POP3 server.

pop3_get_dedup

For detailed description and example use of the function, see pop3_get in the Functions Reference Guide.

Commands

UIDL - get only UIDL's of messages.

DELETE - delete messages after downloading.

Example 19.5. Examples

1. To get 10 KB. messages from the server 'openlinksw.com' POP3 port 110, user name 'user_1', password 'pass_1'

```
Pop3_get ('openlinksw.com:110', 'user_1', 'pass_1', 10000);
```

2. To get and delete 5 KB. messages from the server 'openlinksw.com' POP3 port 110, user name 'user_1', password 'pass_1'

```
Pop3_get ('openlinksw.com:110', 'user_1', 'pass_1', 5000, 'DELETE');
```

3. To get UIDL's of 100 kb. messages from the server 'openlinksw.com' POP3 port 110, user name 'user_1', password 'pass_1'

```
Pop3_get ('openlinksw.com:110', 'user_1', 'pass_1', 100000, 'UIDL');
```

19.3.4. The Virtuoso Mail Sink

The Virtuoso Sink is used to store messages received by Windows 2000 IIS SMTP Service into the MAIL_MESSAGES table in Virtuoso. It consists of an executable, run as a service, and a VBscript for registering a COM object with the IIS, interfacing it with the Virtuoso Sink

The User ID(s) in MAIL_MESSAGES to whom the message belongs to is determined by parsing the **To:** field in the mail message into a list of recipients. For each entry in the list, if an @-sign is found, the user ID will consist of the characters to the left of it, otherwise the whole entry is used.

Registry Entries for the Virtuoso Sink.

```
<<<<<<<< VirtuosoSink.reg
REGEDIT4

[HKEY_LOCAL_MACHINE\SOFTWARE\OpenLink Software\VirtuosoSink]
"User"="dba"
"Password"="dba"
"DSN"="Virtuoso"
"ConnectionCount"=dword:00000001
>>>>>>>> VirtuosoSink.reg
```

These are the registry entries used by the Sink service. ConnectionCount is the number of connections to Virtuoso to be pooled.

Installation

Use these commands to install the Virtuoso Sink:

VirtuosoSink /RegServer Registers the COM Object VirtuosoSink.SMTP into the registry

VirtuosoSink /Service Creates the Service in Manual start mode

cscript smtpreg.vbs /add 1 OnArrival Virtuoso_SMTP_Sink VirtuosoSink.SMTP "mail from=*" - Registers the COM Object as a mail sink with the IIS SMTP service.

See Also:

The MSDN page for more information about registering sinks.

Sink Operation

When the service starts it opens a pool of connections to Virtuoso and waits for a COM event. When the COM callback gets called, the service invokes a stored procedure and passes the message text to it. The procedure then stores the message into the database. In event of lost connection to the database, the service would try once to re-establish it and re-execute the stored procedure call. Errors and successful message routings are entered into the Windows 2000 Application Log.

19.4. NNTP Newsgroups

19.4.1. NNTP Client

nntp_get_dedup

For detailed description and example use of the function, see nntp_get in the Functions Reference Guide.

19.4.2. Commands and Examples

If the requested messages don't exist, `nttp_get()` returns NULL.

Get a list of all groups from the server 'news.openlinksw.com', port 119:

Example 19.6. nntp_get()

```
nntp_get ('news.openlinksw.com:119', 'list');
```

This call returns an array of the form Array ((<group 1> varchar, <last message> integer, <first message> integer, <posting allowed> varchar) (<group 2> . . . .) . . .)

ARTICLE, HEAD, BODY, STAT To get the bodies of all messages in the group 'openlink.public.virtuoso':

```
nntp_get ('news.openlinksw.com:119', 'body', 'openlink.public.virtuoso');
```

This call returns an array of the form Array ((<message number> integer, <body of message> blob) . . .)

To get the article (head + body) of messages numbered from 5 to 10 in the group 'openlink.public.virtuoso':

```
nntp_get ('news.openlinksw.com:119', 'article', 'openlink.public.virtuoso', 5, 10);
```

This call returns an array of the form Array ((<message number> integer, <body of message> blob) . . .)

To get the headers of messages numbered from 7 to the end of the 'openlink.public.virtuoso' group:

```
nntp_get ('news.openlinksw.com:119', 'head', 'openlink.public.virtuoso', 7);
```

To get the status of all messages in the group 'openlink.public.virtuoso':

```
nntp_get ('news.openlinksw.com:119', 'stat', 'openlink.public.virtuoso',);
```

This call returns an array of the form Array ((<message number> integer, <message ID> varchar) . . .)

To get the status of the 'openlink.public.virtuoso' group:

```
nntp_get ('news.openlinksw.com:119', 'group', 'openlink.public.virtuoso');
```

This call returns an array of the form Array (<number of messages> integer, <first message> integer, <last message> integer)

19.5. NNTP Server

19.5.1. Enabling the NNTP Server

The Virtuoso News Server listening port is configured in the HTTP section of the virtuoso.ini file.

```
[HTTPServer]
NewsServerPort   = 119
```

If the port is not defined Virtuoso will disable the News Server subsystem.

19.5.2. NNTP Server Commands

The server supports the following commands:

article [MessageID|Number]
body [Number]
group newsgroup
head [Number]
help
last
list
mode reader
newgroups yymmdd hhmmss
next
post
xover [range]
stat [MessageID|Number]

See Also:

News System Tables

19.5.3. Add Groups to NNTP Server

To add a new newsgroup you must insert a row into the table DB.DBA.NEWS_GROUPS. Below is an example of an insert statement that you could use to do this:

```
insert into DB.DBA.NEWS_GROUPS (NG_NAME, NG_DESC, NG_UP_INT, NG_CLEAR_INT,
        NG_POST, NG_UP_TIME, NG_OUT_GROUP, NG_NUM, NG_FIRST, NG_LAST,
        NG_SERVER, NG_SERV_PORT, NG_CREAT, NG_UP_MESS, NG_PASS)
values ('openlink.public.virtuoso', 'virtuoso news group' ,
```

```
update interval, clear interval, 1, now(), 'openlink.public.virtuoso',
0, 0, 0, 'news.openlinksw.com', 110, now(), 0, 0);
```

If the group you want to add is local, change

```
news.openlinksw.com
```

to

```
localhost
```

, change the port to 0, and the

```
external name
```

to

```
''
```

(that is, two single quotes).

See also the Newsgroups Administration section of the Visual Server Administration Interface.

19.6. MIME & Internet Messages

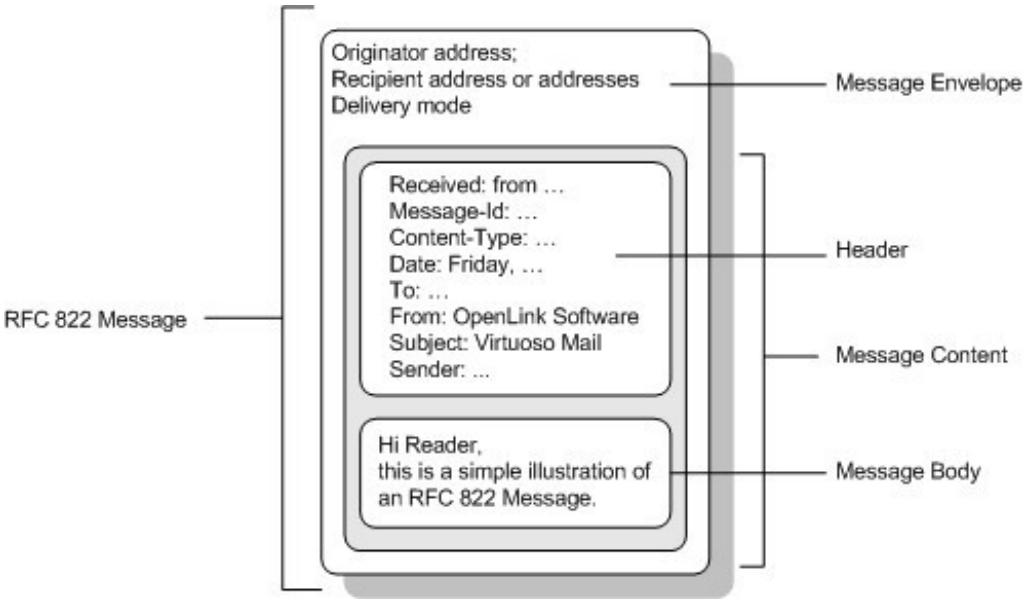
19.6.1. About Simple Internet (RFC 822) Messages

RFC 822 messages have two major parts:

- **Message envelope.** The message envelope contains all the information needed to accomplish transmission and delivery of the message. This information includes the e-mail address of the message's creator fi also known as the originator. This string matches the information in the Sender: header, if this header is present. The envelope is created by a user agent (such as MS Outlook) and is meaningful only to the message transfer agents (MTAs) that move the message on the path to its destination.
- **Message contents.** The contents make up the object to be delivered to the recipient. Message contents consist of lines of ASCII text. This text is arranged in the classic "memo" format, in which the message contains one or more introductory headers and a body.

This structure can be seen in the following illustration:

Figure 19.2. The structure of an Internet mail



Example 19.7. Simple Internet Mail Message

As you can see in the following sample of a message's contents, the format described in RFC 822 produces messages readable with little difficulty by humans.

The first few lines, from the first instance of "Received" to "Precedence", are headers. These lines define the recipients, the sender, the date, and other information involved with message transmission.

Following the headers is a blank line. This is marked by the consecutive occurrence of the four characters: CR, LF, CR, LF. After this blank line starts the body of the message. In the following example, only the final few lines make up the message body.

```
Received: from techsupp@openlinksw.co.uk
Message-Id: <v1214040cad6a13935723@>
Mime-Version: 1.0
Content-Type: text/plain; charset="us-ascii"
Date: Mon, 4 Jun 1998 09:43:14 -0800
To: customer.services@openlinksw.co.uk
From: OpenLink Technical Support <techsupp@openlinksw.co.uk>
Subject: Happy Reading
Precedence: bulk

Hope you are enjoying this chapter,
Technical Support
```

See Also:
RFC 822 at www.rfc-editor.org

19.6.2. MIME Messages - Extension to Simple Internet Messages

MIME (Multipurpose Internet Mail Extensions) grew out of a need to encapsulate messages within messages. It includes multipart messages comprising a variety of file types such as images, audio, and video. MIME does all this while following all the standard SMTP and RFC 822 mail rules. MIME messages can be constructed to transport mail over any mail transport system that is compliant with SMTP. MIME is able to transmit objects with varying ranges of complexity in a way that allows any MIME-compliant user agent (UA) to faithfully process them and hand them off to an appropriate application. The multiple parts are arranged so that the parts requiring the least sophisticated UA are at the beginning of the message. In fact, most MIME UAs include courtesy text when constructing messages to give users of non-MIME UAs an indication of the message content. This courtesy text is inserted ahead of any MIME parts.

MIME is consistent with Internet mail protocols using headers and bodies. It allows for transmission of 7-bit printable US-ASCII characters and maximum 1000-character lines in message bodies over all Internet mail transports. It has become the most widely used extension to the simple e-mail standard. It is also used as a transport mechanism in Web pages.

Each content body part is made up of header fields and content. The headers convey specific information about the content for the message recipient. The content can be essentially any serialized stream of bytes, such as binary data or HTML. When necessary, the content is encoded so that the entire body complies with RFC 822. If the content is encoded, MIME defines the header Content-Transfer-Encoding to specify the mechanism. Other details are sometimes included, such as the Content-Disposition, which indicates to the recipient whether the content is to be treated simply as an attachment, or whether it is to be rendered inline with other content in other body parts.

Example 19.8. A simple MIME message sample

Mime message including a picture stored as a file GIF format. Because .gif files use 8-bit bytes, and the RFC 822 format requires messages to contain only US-ASCII text, the picture data must be encoded. For the recipient to correctly decode and display the picture, it needs information about which encoding mechanism was used. The following example shows part of a MIME header that identifies that the content is a .gif file, that it is encoded using the standard base64 algorithm, and that it is to be treated by the e-mail client as an attachment.

```
Content-Type: image/gif;
     name="picture.gif"
Content-Transfer-Encoding: base64
Content-Disposition: attachment;
     filename="picture.gif"
```

```
[encoded content here]
...
```

MIME accomplishes this simplifying and rebuilding of complex files by encoding a file and transporting it as a message body, or a series of messages with component parts of the file. A MIME-compliant user agent (UA) on the receiving end can decode the message, presenting it to the reader or providing it to another application as a file. A UA that is not MIME-compliant will be able to process a MIME-encoded mail message, but may not be able to decode the message.

MIME defines a message format that allows for:

- Textual message bodies in character sets other than US-ASCII.
- Non-textual message bodies.
- Multipart message bodies.
- Textual header information in character sets other than US-ASCII.

Example 19.9. Sample MIME Message

MIME uses headers and separators to tell a UA what processing is required to re-create the message. An example with no encoded body parts is:

```
From: OpenLink Support <techsupp@openlinksw.co.uk>
MIME-Version: 1.0
Content-Type: multipart/mixed;
        boundary="XXXXboundary text"

This is a multipart message in MIME format.

--XXXXboundary text
Content-Type: text/plain

here is the body text

--XXXXboundary text
Content-Type: text/plain;
Content-Disposition: attachment;
        filename="test.txt"

this is the attachment text

--XXXXboundary text--
```

This example shows the use of a MIME message to send a text message and an attached text file. Both are body parts of this message.

The "MIME-Version:" header tells the receiving UA to treat this as a MIME message.

The "Content-Type:" header specifies "multipart/mixed". This tells the receiving UA that this message has parts separated by the string argument defined in "boundary=". A MIME-compliant UA will only display or otherwise process content following the specified "boundary=" text strings. The actual boundaries are constructed using the "boundary=" string, prepended by "--". The final body part is followed by the "boundary=" string with the "--" both prepended and appended.

In the preceding example, the courtesy message "This is a multipart message in MIME format." will not be displayed or otherwise processed by the MIME UA since it does not follow a "boundary=" string. A UA that does not support MIME will display it, and at least this part of the message will be readable no matter what features the reader supports. If our example had encoded parts, they would make no sense to the human reader using a non-MIME-compliant UA, but at least the courtesy message would give the user a hint as to why.

There are two message body parts in our example. Each body part has headers of its own, in addition to the overall message headers. Each body-part begins with the boundary string. If there were no headers in the body parts, then the blank line that must follow headers in RFC 822 messages would follow the boundary string. The first body part is a plain text message. It is the message the sender might have typed into a UA. Its single "Content-Type:" header identifies it as "text/plain", meaning US-ASCII characters are used exclusively and any UA should be able to display this body part. Text/plain is the default content type.

The second body part in this example contains the file attachment. Since the file attachment is an ASCII text file, it is sent with no encoding and its content-type is given as text/plain. The "Content-Disposition: attachment" header has an attribute, "filename=", which specifies a suggested name for the file. This header specifies that this body part is to be treated as a file rather than being displayed to the user and is to be saved on local storage under the suggested file name.

MIME Headers

MIME headers appear at the beginning of a MIME message as well as within the separate body parts. Some MIME headers can be used both as message headers and in MIME body parts. Some additional headers are defined for use only in body parts.

The following headers are defined in MIME:

- **MIME-Version.** Required header indicating that this message is to use the rules of MIME. "MIME-Version: 1.0" is the only currently defined MIME-Version header allowed. The MIME-Version header is a top-level header only and does not appear in body parts unless the body part is an encapsulated, fully formed message of content-type message/rfc822, which might have its own MIME-Version header.
- **Content-Type.** Content-Type headers are used to specify the media type and subtype of data in the body of a message and to fully specify the native representation of such data. This header embodies much of the power of MIME. The IETF can add new official content types. Additionally, private content-type values can be defined by anyone. Such private content types have values of "x-something" or "X-something", where "something" can take on any value.
- **Content-Transfer-Encoding.** Content-Transfer-Encoding headers can have two different meanings. If the value is "base64" or "quoted-printable", then the header indicates the encoding used for this body part. If the value is "7bit", "8bit", or "binary", then the header indicates that there is no encoding and that this value indicates only the type of content this body part contains. The default is "7bit". It should be noted that "8bit" and "binary" are not guaranteed to be properly handled by all Internet (SMTP) MTAs valid in Internet mail. Eight bit content is not valid in Internet mail headers. Provision is made for private Content-Transfer-Encoding headers. These have values that begin with "x-" or "X-". These are for specialized cases where the users have the tools to decode or otherwise process a specific "x-" encoding.
- **Content-ID.** Content-ID headers are world-unique values that identify body parts, individually or as groups. They are necessary at times to distinguish body parts and allow cross-referencing between body parts.
- **Content-Description.** Content-Description headers are optional and are often used to add descriptive text to non-textual body parts.
- **Content-Disposition.** Content-Disposition headers provide information about how to present a message or a body part. When a body part is to be treated as an attached file, the Content-Disposition header will include a file name attribute.

There are additional headers that are applied in specialized situations, such as Content-Base and Content-Location. All of the "Content-xxx" headers have defined sub-headers, fields, and/or attributes. Headers that begin with "Content-" are the only headers that have defined meaning in body parts. All other body part headers can be ignored and might actually be removed by message transfer agents (MTAs).

MIME_TREE - MIME parser

mime_tree_dedup

For detailed description and example use of the function, see mime_tree in the Functions Reference Guide.

Processing HTTP PUT of type "multipart/form-data"

When the Virtuoso server receives a PUT request of type "multipart/form-data" from the client agent, then it passes the HTTP text to the MIME_TREE function internally and adds any MIME subpart as an element pair ("name", "value") of the "params" parameter of the vsp page being specified in the URI. It also adds an additional "params" pair for each HTTP request MIME part named "attr-name" and whose value is an array of all MIME header fields of that part.

Example 19.10. Example:

Consider the following HTTP request:

```
PUT handler.vsp
Content-Type: "multipart/form-data"; boundary="--end_part"
```

```
----end_part
Content-Type: image/gif
Content-Disposition: form-data; name=upload_control; filename="some image.gif"

GIF...
----end_part
Content-Type: text/plain
Content-Disposition: form-data; name=textarea

The description
----end_part--
```

Virtuoso parses that and produces the following params content when calling handler.vsp:

```
( "upload_control", "GIF....",
  "attr-upload_control", ( "Content-Type", "image/gif",
                "Content-Disposition", "form-data", "name",
                "upload_control", "filename", "some image.gif"),
  "textarea", "The description",
  "attr-textarea", ("Content-Type", "text/plain",
                "Content-Disposition", "form-data", "name",
                "textarea")
)
```

This allows for vsp's to handle uniformly "x-www-form/url-encoded" and "multipart/form-data" PUTS and to have full access to the MIME headers if needed.

19.6.3. S/MIME Support

S/MIME is a specification for secure electronic mail. S/MIME stands for Secure/Multipurpose Internet Mail Extensions and was designed to add security to e-mail messages in MIME format. The security services offered are authentication (using digital signatures) and privacy (using encryption).

The S/MIME specification consists of two documents: S/MIME Message Specification (RFC 2311) and S/MIME Certificate Handling (RFC 2312) . Both of these are Internet Drafts. The S/MIME community has submitted these to the IETF. The goal is to form a working group and produce an Internet standard.

All certificates and private keys are read and stored as PEM encoded strings. If the server is compiled without SSL support then dummy versions of smime_sign, smime_verify, pem_certificates_to_array and get_certificate_info are available so that existing SQL code would not be broken. Currently the Virtuoso server supports S/MIME signing and S/MIME signature verification. These are done through the following 2 functions:

`smime_verify()`

`smime_sign()`

A useful utility function for S/MIME support is:

`pem_certificates_to_array()`

19.7. FTP Services

Virtuoso provides both FTP client and server functionality. The Virtuoso FTP client functions allow for programmatic access to ftp servers from Virtuoso/ PL to list, submit and retrieve files from an FTP server. The Virtuoso FTP server provides FTP access to the Virtuoso WebDAV repository using the same authentication and permissions system as WebDAV, and an configurable anonymous user access.

19.7.1. FTP Client

Virtuoso provides three functions that mimic their FTP command counterparts. These are:

`ftp_get()` - to retrieve a file from an FTP server.

`ftp_put()` - to submit a file to an FTP server.

`ftp_ls()` - list the contents of a directory on an FTP server.

The Virtuoso client uses free ports within the range 20000 - 30000 for active mode connections. This is configurable using the parameters: `FTPServerMinFreePort` and `FTPServerMaxFreePort` in the `HTTPServer` section of the Virtuoso INI file.

19.7.2. FTP Server

Virtuoso can provide FTP access to its WebDAV repository. This FTP server functionality relies on the same credentials and permissions that WebDAV uses. The server is enabled by listing the `FTPServerPort` parameter in the `HTTPServer` section of the Virtuoso INI file. This parameter must be accompanied by a port number to listen at.

```
[HTTPServer]
FTPServerPort = 21
```

Will instruct Virtuoso to listen for FTP requests on the default FTP port of 21.

```
[HTTPServer]
...
FTPServerTimeout = 600
...
```

To control timeout of connections use "FTPServerTimeout" ini parameter. The default time out is 600 sec. This is only for command connection.

The Virtuoso FTP server can be instructed to create a session log file using the parameter:

```
[HTTPServer]
...
FTPServerLogFile = ftpserver
...
```

If specified Virtuoso will produce an FTP server log file with the date appended to the name given in the parameter and the files extension as ".log". The log file is rotated daily. It will contain the following information:

`ftpserverDDMMYYYY.log`:-

Client Host Name

Authorized User

Time

User Command

Server Response Code

Bytes Transferred

An example of which is:

```
hostname anonymous [22/Oct/2003:15:21:43 +0300] "PASS user@domain.com" 230 0
hostname anonymous [22/Oct/2003:15:23:11 +0300] "LIST" 226 162
hostname dav [22/Oct/2003:15:25:00 +0300] "PASS <hidden>" 230 0
```

Virtuoso can allow special "anonymous" user access to the FTP Server by supplying the following Virtuoso INI file parameter:

```
[HTTPServer]
...
FTPServerAnonymousLogin = 0
...
```

The anonymous user is not a real user, it has no SQL or DAV login ability. The anonymous user can only access collections or resources that are set to public. The password of the anonymous user is trivially checked to contain the "@" symbol and is shown in plain-text for the "PASS" command detailed in the log file. Valid DAV users passwords are never revealed in the FTP log file. By default anonymous login is denied.

```
[HTTPServer]
...
FTPServerAnonymousHome = /DAV/dir_name/
...
```

You can control the anonymous user home dir by "FTPServerAnonymousHome" parameter in the ini fail. By default home dir for anonymous user is DAV root dir.

The Virtuoso server supports the usual variety of commands such as:

```
USER    PASS    CWD     CDUP    HELP
QUIT    PORT    PASV    TYPE    NOOP
STOR    RNFR    RNTO    MODE    RETR
DELE    RMD     MKD     PWD     SYST
SIZE    LIST    NLST    ABOR
```

By default the server uses free ports within the range 20000 - 30000 for passive mode connections. This is configurable using the parameters: `FTPServerMinFreePort` and `FTPServerMaxFreePort` in the `HTTPServer` section of the Virtuoso INI file.

The FTP server root directory is the home directory of the authenticated DAV user.

See Also:
RFC-959

RFC-2389

19.8. VSP Guide

19.8.1. Introduction

Virtuoso Server Pages are the equivalent of a stored procedure in a Web page that is compiled when it is first read by the Virtuoso server. Virtuoso detects when the '.vsp' file is modified and recompiles the procedure. Since VSP is essentially Virtuoso PL in a Web page you can do anything that PL can, either directly or from interaction with the user. VSP gives you the advantage of not having to worry about making connections to the database. You also avoid the overhead of RPCs because the HTTP server is built into Virtuoso. When you write a VSP page the connection is automatic since you are already in Virtuoso.

VSP is server script and is therefore executed in the server as it is encountered on the page. For this reason client (JavaScript) and server script cannot directly interact but can supplement each other. You can call JavaScript inside a VSP loop, for example, to manipulate something that already exists on the page but you cannot pass variables by reference from VSP directly to JavaScript or vice versa.

Page flow control can be managed using FORMs. The state of the page is defined in form fields such as INPUT boxes and TEXTAREA boxes and then passed to the next form or page using POST.

19.8.2. Simple HTML FORM usage

We start with a small example that shows the source of a page including a FORM with data from the user being sent when a submit button is pressed. We then examine the elements and attributes of this simple form that are important to us at this stage.

Basic Forms

Example 19.11. Simple Forms

```
<HTML>
  <HEAD>
    <TITLE>Simple FORM demo</TITLE>
  </HEAD>
  <BODY>
  <FORM METHOD="POST" ACTION="formdemo_receiver.vsp">
    <P>Test form, type some info and click Submit</P>
    <INPUT TYPE="TEXT" NAME="myInput" />
    <INPUT TYPE="SUBMIT" NAME="submit" VALUE="Submit" />
  </FORM>
```

```
  </BODY>
</HTML>
```

The METHOD attribute of a FORM TAG in a VSP page can be either GET or POST. The GET method allows the form submission to be contained completely in a URL; this can be advantageous because it permits bookmarking in browsers, but it also prevents form data from containing non ASCII characters such as accented letters and special symbols and restricts the amount of form data that can be handled. The GET method is l mited by the maximum length of the URL that the server and browser can process. To be safe, any form whose input might contain non-ASCII characters or more than 100 characters should use METHOD="POST".

With the POST method, the form input is submitted as an HTTP POST request with the form data sent in the body of the request. Most current browsers are unable to bookmark POST requests, but POST does not entail the character encoding and length restrictions imposed by GET.

The ACTION attribute of FORM specifies the URI of the form handler. This will usually be another web page that performs some action based on the data that is sent from the originating form. The URI could point to the same page as the data originated and for pages that perform a well-defined small set of functions it usually does. When a page needs to manage multiple states there needs to be some flow control that can determine how the page was reached; for example, to differentiate whether it arrived at as a result of someone clicking on the submit button or it is the first time the page has been visited.

Exchanging Values in Forms

Now we add some VSP to check the values of the parameters in the form. VSP markup is typically contained in <?vsp ... ?> blocks.

Example 19.12. Forms and Values

```
<HTML>
  <HEAD>
    <TITLE>Simple FORM demo</TITLE>
  </HEAD>
  <BODY>
  <P>Last value sent:

  <?vsp
    http(get_keyword('myInput', params, 'no value'));
   ?>

</P>

  <FORM METHOD="POST" ACTION="formdemo.vsp">
    <P>Test form, type some info and click Submit</P>
    <INPUT TYPE="TEXT" NAME="myInput" />
    <INPUT TYPE="SUBMIT" NAME="submit" VALUE="Submit" />
  </FORM>
  </BODY>
</HTML>
```

This is the same example as above but now it uses the same page for the form handler and displays the parameters each time. Clicking the Submit button takes you to the same page and displays whatever you typed in the field last time.

The VSP block uses two nested functions. The `http()` function allows you to send data to the HTTP client, the browser. What we send to the browser is the result of the `get_keyword()` function, which has three parameters: *`search_for`*, *`source_array`*, and *`default_val`*. It searches for the keyword-value pair (keyword=value) where the keyword matches the *`search_for`* parameter (in this case 'myInput') in the array passed in the *`source_array`* parameter. It returns the value if one is found; otherwise returns the *`default_val`* parameter in the function, in this case 'no value'. The *`params`* argument is a special array that contains all page parameters from the previous FORM state.

Conditional Processing

Now we extend this further to add some conditional control so that if a value was entered we can respond directly to it. We will also use a variable this time, which must be declared first.

Example 19.13. Conditional Processing Using IF

```
<HTML>
  <HEAD>
    <TITLE>Simple FORM demo</TITLE>
  </HEAD>
  <BODY>

  <?vsp
        declare _myInput varchar;

        _myInput := get_keyword('myInput', params, '');

    if (_myInput <> '')
        {       http('<P>Hello, ');
                http(_myInput);
                http('</P>');
        }
        else
        {       http('<P>Please enter your name</P>');
        }
   ?>

  <FORM METHOD="POST" ACTION="formdemo.vsp">
    <INPUT TYPE="TEXT" NAME="myInput" VALUE="" />
    <INPUT TYPE="SUBMIT" NAME="submit" VALUE="Submit" />
  </FORM>
  </BODY>
</HTML>
```

Further Page Control

We now extend this to control the whole content of the page. In this example we see that VSP and HTML can be interleaved.

Example 19.14. Page Control

```
<HTML>
  <HEAD>
    <TITLE>Simple FORM demo</TITLE>
  </HEAD>
  <BODY>

  <?vsp
        declare _myInput varchar;
        declare Mode varchar;

        _myInput := get_keyword('myInput', params, '');
        Mode := get_keyword('submit', params, '');

    if (Mode = 'Submit')
        {
   ?>
    <P>Hello, <?vsp http(_myInput); ?>
        </P>

<FORM METHOD="POST" ACTION="demo4.vsp">
<INPUT TYPE="HIDDEN" NAME="myInput" VALUE="" />
<INPUT TYPE="SUBMIT" NAME="submit" VALUE="Again" />
</FORM>

  <?vsp
        }
          else
        {
```

```
    ?>

  <P>Please enter you name</P>
  <FORM METHOD="POST" ACTION="demo4.vsp">
    <INPUT TYPE="TEXT" NAME="myInput" />
    <INPUT TYPE="SUBMIT" NAME="submit" VALUE="Submit" />
  </FORM>

  <?vsp
    }
  ?>
  </BODY>
</HTML>
```

We start by setting the mode based on whether the Submit button has been pressed. When the mode has changed a different version of the page is sent to the browser. In the new version, the Again button the appears, to return you to the previous state when pressed.

Communicating Parameters Between Pages

Now we will use two pages to do the same job as in the demo above.

Example 19.15. Using more than one page

Page 1

```
<HTML>
  <HEAD>
    <TITLE>Multi Page Demo</TITLE>
  </HEAD>
  <BODY>
  <P>Please enter you name</P>
  <FORM METHOD="POST" ACTION="demo5_2.vsp">
    <INPUT TYPE="TEXT" NAME="myInput" />
    <INPUT TYPE="SUBMIT" NAME="submit" VALUE="Submit" />
  </FORM>
  </BODY>
</HTML>
```

Page 2

```
<HTML>
  <HEAD>
    <TITLE>Multi Page Demo</TITLE>
  </HEAD>
  <BODY>
  <P>The value you entered was:
  <?vsp
    http(get_keyword('submit', params, 'no data'));
  ?>
  </P>
  <FORM METHOD="POST" ACTION="demo5_1.vsp">
    <INPUT TYPE="SUBMIT" NAME="submit" VALUE="Back" />
  </FORM>
  </BODY>
</HTML>
```

Using JavaScript to Control Forms

JavaScript is a programming language that can be used in the browser and is useful for client-side programming. It is useful to be able to do some work on the client machine before making another round trip to the server for more processing. JavaScript is also useful for making things more appealing to the Web page viewer.

JavaScript can be made to respond to events within the browser such as when the mouse is moved over a link, a graphic or a button or when the mouse is clicked on some part of the page. This can be achieved by using event handlers within the HTML

tags and placing JavaScript code in their content. Common event handlers are *onMouseOver* , *onMouseClick* , *onMouseOut* , *onChange* , and the like.

A simple but useful example of this would be to simplify one of the previous examples by placing a handler on the text box so that you do not have to press the submit button to send the form to the server:

```
<FORM METHOD="POST" ACTION="demo5_2.vsp" NAME="demo5_2">
  <INPUT TYPE="TEXT" NAME="myInput" onChange="document.demo5_2.submit()" />
  <INPUT TYPE="SUBMIT" NAME="submit" VALUE="Submit" />
</FORM>
```

19.8.3. Interacting with the Database

This section describes manipulating data within Virtuoso from VSP. Unless the required tables already exist, new ones will need to be created. This example will involve a simple table of people and a series of pages for adding, editing, viewing, and deleting its entries.

Creating a Table

Tables should be created so that their entries can be uniquely identified. This is very important so that if we need to edit or delete one particular entry we can distinguish it from other entries. A *primary key* is how a database enforces unique rows, by refusing to allow duplicate data to be inserted. It is up to the user to choose a column in the table to act as a primary key. Sometimes one or more of the columns of data are naturally unique either singularly or in composite; other times it is necessary to add a column to contain unique codes for each row.

See Also:

Primary Keys

Here is the definition of the simple table that will be used:

```
CREATE TABLE DB.DBA.DEMO_PEOPLE (
  EMAIL VARCHAR(255) PRIMARY KEY,
  FORENAME VARCHAR(100),
  SURNAME VARCHAR(100)
);
```

The email address has been selected as a primary key.

Basic Form Input Page

After the table has been created; for example via Virtuoso's iSQL utility; it will need some data. For this we create a "New Person" page. This page uses form inputs and some VSP code to determine whether an insert button was pressed. If the insert button is pressed then the page takes submitted values from the POST and uses them to construct an SQL statement that inserts a new row into the table. This is demonstrated below:

```
<HTML>
  <HEAD>
    <TITLE>New Person Page</TITLE>
  </HEAD>
  <BODY>
  <?vsp

    declare _email, _forename, _surname varchar;

    _email := get_keyword('email', params, '');
    _forename := get_keyword('forename', params, '');
    _surname := get_keyword('surname', params, '');

    -- insert new person if we came from the insert button
    if ('' <> get_keyword('ins_button', params, ''))
    {
      INSERT INTO DB.DBA.DEMO_PEOPLE(EMAIL, FORENAME, SURNAME)
        VALUES(_email, _forename, _surname);
    }
  ?>
  <P>Please enter the details of new person:</P>
```

```
  <FORM METHOD="POST" ACTION="demo_people_add.vsp">
  <TABLE>
    <TR><TH>Email:</TH><TD><INPUT TYPE="TEXT" NAME="email" /></TD></TR>
    <TR><TH>Forename:</TH><TD><INPUT TYPE="TEXT" NAME="forename" /></TD></TR>
    <TR><TH>Surname:</TH><TD><INPUT TYPE="TEXT" NAME="surname" /></TD></TR>
  </TABLE>

  <INPUT TYPE="SUBMIT" NAME="ins_button" VALUE="Insert" />
  </FORM>

  </BODY>
</HTML>
```

The underscores were added to this example to keep the param variables and page variables visibly distinguishable.

Displaying Table Data in a VSP Page

Now that some data exists in the table we need a way to display it. The FOR ... DO construct is used to construct the insides of an HTML table:

```
<HTML>
  <HEAD>
    <TITLE>The People Page</TITLE>
  </HEAD>
  <BODY>
  <P>The Peoples' Details</P>

  <TABLE>
    <TR><TH>Email</TH><TH>Forename</TH><TH>Surname</TH></TR>
  <?vsp
    FOR (SELECT EMAIL, FORENAME, SURNAME FROM DB.DBA.DEMO_PEOPLE) DO
    {
  ?>
    <TR><TD><?=EMAIL?></TD><TD><?=FORENAME?></TD><TD><?=SURNAME?></TD></TR>
  <?vsp
    }
  ?>
  </TABLE>
  </BODY>
</HTML>
```

Simple Form Delete Page

The page above can easily be extended to allow deletion. For each row an 'action' link is added. The action Remove link hardwires a form GET on the URL. This is then intercepted by the IF condition looking for the *remove* parameter.

```
<HTML>
  <HEAD>
    <TITLE>The People Page With Deletion</TITLE>
  </HEAD>
  <BODY>
  <?vsp
    declare deleteme varchar;

    deleteme := get_keyword('remove', params, '');
    if ('' <> deleteme)
      DELETE FROM DB.DBA.DEMO_PEOPLE WHERE EMAIL = deleteme;
  ?>

  <FORM METHOD="POST" ACTION="demo_people_view2.vsp">
  <P>The Peoples' Details</P>

  <TABLE>
    <TR><TH>Email</TH><TH>Forename</TH><TH>Surname</TH>
      <TH>Action</TH></TR>
  <?vsp
    FOR (SELECT EMAIL, FORENAME, SURNAME FROM DB.DBA.DEMO_PEOPLE) DO
    {
  ?>
    <TR><TD><?=EMAIL?></TD><TD><?=FORENAME?></TD><TD><?=SURNAME?></TD>
      <TD><A HREF="?remove=<?=EMAIL?>">Remove</A></TD></TR>
```

```
  <?vsp
    }
  ?>
  </TABLE>
  </FORM>
  </BODY>
</HTML>
```

Simple Form Edit Page

The last step is to have a way to edit rows of the table. To do this, we combine everything that we have so far and use the SQL UPDATE statement to update the row. The EMAIL column is not made updateable since this is the primary key.

```
<HTML>
  <HEAD>
    <TITLE>The People Page With Deletion</TITLE>
  </HEAD>
  <BODY>
  <FORM METHOD="POST" ACTION="demo_people_view3.vsp">
  <?vsp
    declare deleteme, editme, edt_email, edt_forename, edt_surname,
            save_email, save_forename, save_surname varchar;

    deleteme := get_keyword('remove', params, '');
    if ('' <> deleteme)
      DELETE FROM DB.DBA.DEMO_PEOPLE WHERE EMAIL = deleteme;

    if ('' <> get_keyword('save_button', params, ''))
    {
      save_email := get_keyword('email', params, '');
      save_forename := get_keyword('forename', params, '');
      save_surname := get_keyword('surname', params, '');

      update DB.DBA.DEMO_PEOPLE
        SET FORENAME = save_forename, SURNAME=save_surname
        WHERE EMAIL = save_email ;
    }

    editme := get_keyword('edit', params, '');
    if ('' <> editme)
    {
      SELECT EMAIL, FORENAME, SURNAME
        INTO edt_email, edt_forename, edt_surname
        FROM DB.DBA.DEMO_PEOPLE WHERE EMAIL = editme;
  ?>
  <TABLE>
    <TR><TH>Email:</TH><TD><INPUT DISABLED TYPE="TEXT" NAME="email" VALUE="<?=edt_email?>" /></TD></TR>
    <TR><TH>Forename:</TH><TD><INPUT TYPE="TEXT" NAME="forename" VALUE="<?=edt_forename?>" /></TD></TR>
    <TR><TH>Surname:</TH><TD><INPUT TYPE="TEXT" NAME="surname" VALUE="<?=edt_surname?>" /></TD></TR>
  </TABLE>
  <INPUT TYPE="SUBMIT" NAME="save_button" VALUE="Save" />
  <?vsp
    }
  ?>

  <P>The Peoples' Details</P>

  <TABLE>
    <TR><TH>Email</TH><TH>Forename</TH><TH>Surname</TH>
      <TH>Action</TH></TR>
  <?vsp
    FOR (SELECT EMAIL, FORENAME, SURNAME FROM DB.DBA.DEMO_PEOPLE) DO
    {
  ?>
    <TR><TD><?=EMAIL?></TD><TD><?=FORENAME?></TD><TD><?=SURNAME?></TD>
      <TD><A HREF="?remove=<?=EMAIL?>">Remove</A> <A HREF="?edit=<?=EMAIL?>">Edit</A></TD></TR>
  <?vsp
    }
  ?>
  </TABLE>
  </FORM>
  </BODY>
```

```
</HTML>
```

19.8.4. The Forums Application

The "Forums" Application is a World Wide Web Application for posting, reading and searching of messages developed under the Virtuoso VDBMS with a wide usage of Virtuoso Server Pages (VSP) and server-side XSL-T transformation.

Messages in the forums are classified to forums and sub-forums by interest. Posting is only allowed for registered users. Registration is performed via a registration form. Every registered user can create new a theme, post new messages to an existing theme or reply to an existing message. Unregistered users can only search, browse, and read existing themes and messages.

Principles

The application is based on VSPs, XML and XSLT transformations. The VSPs are used to produce XML documents that are transformed to HTML using server side XSLT. The design and appearance of the application depends solely on XSLT style sheets. This allows us to divide the development into two distinct parts: layout and design, and functionality of the application.

Session management is based on URL manipulation and persistent HTTP session variables. The messages are stored in Database as XML documents with a free-text index applied over them.

Navigation

The application consists of four main pages:

1. *home.vsp* - the main page introduces the forums with the following information:

   Forums: name of each forum with link to the relevant sub-forums.
   Total: total number of messages for this forum.
   New: new messages for this forum within the last day.
   Last: number of the last message inserted in the forum.
   Total users: count of registered users.
   Options: login if the user is already registered in the forums.
   Registration: add a new user.
   Search: search in the messages.
2. *subforums.vsp* - sub-forums of the current forum with the following information:

   Subforum: name of each sub-forum with links to relevant themes.
   Total: total number of messages for this forum.
   New: new messages for this forum within the last day.
   Last: number of the last message inserted in the forum.
   Options: login if the user is already registered in the forums.
   Registration: add a new user
   Search: search in the messages.
   Forums path: links to the home page and to the forum to which the current sub-forums belong.
3. *forum.vsp* - themes of the current sub-forum with the following information:

   Theme: name of each theme with links to its messages.
   Total: total number of messages for this theme.
   New: new messages for this theme within the last day.
   Last: number of the last message inserted in the theme.
   Options: login if the user is already registered in the forums.
   Registration: add a new user.
   Search: search in the messages.
   Forums path: links to the home page, to the forum and to the sub-forum to which the current themes belong.
4. *thread.vsp* - messages of the current theme with the following information:

Message: name of each message with a link to its properties. When the link is activated the same page is presented, but with the tree of messages for which the current message is the parent.

Author: the name of the author of the current message.

Date: posting date of the message.

Options: login if the user is already registered in the forums.

Registration: add a new user.

Search: search in the messages.

Forums path: links to the home page, to the forum and to the sub-forum to which the current message belongs. Also for the current message, the parent message's name is presented. As users move lower in the tree, they can go back using this path.

Remarks

The basic principles of the application's implementation are:

The result of a VSP execution is a well-formed XML document.

Use the Virtuoso server to do server-side XSLT transformations with appropriate style sheets whenever needed.

Use URL manipulations for session management.

Use post-processing functions to provide commonly used parameters (such as user ID) as session variables.

If users are not logged in they can access all pages of the site, but if they want to insert a new theme or create a new message, they have to log in. When users attempt to create or insert, they will be prompted with the login page. When they log in, the forums application will take them directly to the form for inserting messages or themes. If users are not logged in, the name 'anonymous' is displayed, instead of the e-mail address that would be displayed if they were logged in.

The Search Page

Users can search in three ways:

Theme title: titles of message that are titles of themes.

Message title: titles of message that have the current theme as parent.

Message body: bodies of messages that have different themes as parents.

Search results contain information about how many hits were found, and for each hit the following:

Message title: the title of the current message. When you search in message bodies, the message titles are displayed.

Time: date the message or theme was inserted.

Author: author of the message or theme.

The search page provides the interface for searching contents of forums including messages and titles.

```
<?vsp
  declare id, acount, aresults integer;
  declare aquery, awhat, askiped, search_res, sid, uid, url, usr varchar;

  -- > at this point we instruct server to do server-side XSLT transformation
        -- >    over resultant document
  -- > The transformation will be done before sending the document to the user-agent
  -- >    and after page execution is done.
        -- > To provide flexible file location we use a registry setting for XSLT
        -- >    style sheets
  http_xslt(sprintf ('file:%s/search.xsl', registry_get ('app_forums_xslt_location')));

  -- > because the application does URL poisoning for session management
  -- > we must retrieve the request parameters:

  -- > the session ID
  sid      := get_keyword('sid', params, '0');
  -- > the query text
  aquery   := get_keyword('q',params,'');
  -- > the query locator (for what we searching)
  awhat    := get_keyword('wh',params,'t');
  -- > how many records to skip
  askiped  := atoi (get_keyword('sk',params,'0'));
```

```
  -- > how many results to return
  aresults := atoi (get_keyword('rs',params,'10'));
  -- > hits count
  acount   := atoi (get_keyword('c',params,'0'));

  url := 'thread.vsp';

  -- > also we get the user ID from the session variables
  uid := connection_get ('pid');
  usr := connection_get ('usr');

  -- > now we are ready and call the stored procedure that returns the result from search
  search_res := FORI_SEARCH_RES (aquery, awhat, askiped, aresults, acount);
?>

<!-- now we produce the result as well-formed XML document -->

<?xml version="1.0"?>
<page>
<sid><?=sid?></sid>
<usr><?=usr?></usr>
<url><?=url?></url>
<nav_2><?vsp http(FORI_SEARCH_FORM (sid, aquery, awhat, askiped, aresults, acount)); ?></nav_2>
<css_1/>
<squery><?=aquery?></squery>
<swhat><?=awhat?></swhat>
<sskiped><?=askiped?></sskiped>
<sresults><?=aresults?></sresults>
<scount><?=acount?></scount>
<?vsp http (search_res); ?>
<?vsp http (FORI_SEARCH_NAVIGATION (
  sprintf('search.vsp?q=%s&wh=%s&rs=%d&c=%d&sid=%s&', aquery, awhat, aresults, acount, sid),
         acount, askiped, aresults)); ?>
</page>
```

Search Page Analysis

First we declare the variables used inside the page. In VSP, variables can be defined at any time, but it is generally good practice is to declare them near the top.

We call `http_xslt()` with a file URL parameter. This instructs the Virtuoso server to do XSLT transformation on the server side before sending output to the client, and after execution of the page. Hence we will produce an XML document.

After this we need to get the input parameters session_id, query text, how many records to skip, and how many records to display. We do this by calling `get_keyword()` and passing it the 'params' array. Every VSP page has *`params`* , *`path`* , and *`lines`* as input parameters. For each call we supply a default value in case the variable has not been used. For some of the parameters we need an integer, but varchars are always returned, so we use `atoi()` on the result to convert it to an integer.

Next we retrieve the persistent variables *`user id`* and *`user name`* . We do this by calling `connection_get()` with the session variable name.

After this preparation, we are ready to perform the searching by calling the PL stored procedure `FORI_SEARCH_RES()` . This procedure returns the XML entities that contain the results from the search.

Once we have results, we want to produce the XML document that will be used in the XSLT transformation. We do this part of the work using the shortcuts to the `http_value()` function: '<?= ?>' pairs, and also in some places '<?vsp ?>' to call the `FORI_SEARCH_FORM()` and `FORI_SEARCH_NAVIGATION()` procedures and output their results.

The main benefits to this approach are:

design independency of VSP content: if we need a different design we only need to change the XSLT style sheet;

different style sheets can represent different themes. If we pass an additional 'theme' parameter, for example, we can associate a new theme with a different style sheet. This way we have a fast approach to customizing the look of the page;

browser independence: because we do server-side XSLT transformation we do not need the browser to support XSLT or XML at all.

If we comment out the line that instructs Virtuoso to perform the XSLT transformation (`http_xslt()`) we will get the following document:

```
<?xml version="1.0"?>
<page>
<sid>0</sid>
<usr>anonymous</usr>
<url>thread.vsp</url>
<nav_2><search_form>
<hidden>
<hidden_input type="hidden" name="sid" value="0" />
<hidden_input type="hidden" name="sk" value="0" />
<hidden_input type="hidden" name="rs" value="10" />
<hidden_input type="hidden" name="c" value="0" />
</hidden>
<select name="wh">
<option value="t" selected="1">theme title</option>
<option value="mt">message title</option>
<option value="mb">message body</option>
</select>
</search_form>
</nav_2>
<css_1/>
<squery></squery>
<swhat>t</swhat>
<sskiped>0</sskiped>
<sresults>10</sresults>
<scount>0</scount>
<search_result>
<no_hits/>
</search_result>
<navigation pages="0">
</navigation>
</page>
```

If we re-enable the XSLT transformation the user agent will receive the following HTML content:

```
<html><head>
<style type="text/css">
a:hover{color:#a2a2a2}
.id{font-size:12px;font-family:arial,sans-serif;font-weight:bold;color:#004C87}
.ie{font-size:12px;font-family:verdana,sans-serif;color:#FFFFFF}
.ir{font-size:14px;font-weight:bold;font-family:verdana,sans-serif;color:#FFFFFF}
.if{font-size:12px;text-decoration:none;font-family:verdana,sans-serif;
   font-weight:bold;color:#E1F2FE}
.iname{font-size:12px;font-weight:bold;font-family:verdana,sans-serif;color:#FFFFFF}
.ipath{font-size:12px;text-decoration:none;font-weight:bold;
   font-family:verdana,sans-serif;color:#004C87}
.inew {font-size:12px;text-decoration:none;font-weight:bold;
   font-family:verdana,sans-serif;color:#FFC600}
.text {font-size:12px;text-decoration:none;font-family:Arial,sans-serif;color:#004C87}
</style>
</head>
  <body>
    <TABLE WIDTH="100%" BGCOLOR="#004C87" CELLPADDING="0" CELLSPACING="0" BORDER="0">
     <TR>
      <form action="search.vsp" method="post">
        <TD WIDTH="20%" VALIGN="top">
          <IMG SRC="i/logo_n.gif" HEIGHT="49" WIDTH="197" BORDER="0"></TD>
        <TD WIDTH="40%" ALIGN="center">
          <input type="text" name="q" size="36" value=""></TD>
        <TD WIDTH="25%" ALIGN="center">
           <select name="wh">
              <option value="t" selected>theme title</option>
              <option value="mt">message title</option>
              <option value="mb">message body</option>
           </select> <input type="hidden" name="sid" value="0">
           <input type="hidden" name="sk" value="0">
           <input type="hidden" name="rs" value="10">
           <input type="hidden" name="c" value="0">
        </TD>
        <TD WIDTH="15%">
```

```
            <input type="image" name="search" src="i/search.gif" border="0"></TD>
            </form>
          </TR>
        </TABLE>
        <TABLE WIDTH="100%" BGCOLOR="#02A5E4" CELLPADDING="0" CELLSPACING="0" BORDER="0">
          <TR>
            <TD>
                <IMG SRC="i/str.gif" HEIGHT="12" WIDTH="35">
                <a class="id" href="home.vsp?sid=0">Home</a>
            </TD>
            <TD HEIGHT="22" class="iname" ALIGN="right">anonymous </TD>
          </TR>
          </TABLE>
          <TABLE BGCOLOR="#E1F2FE" ALIGN="center" WIDTH="100%" CELLPADDING="0" CELLSPACING="0" BORDER="0">
            <TR>
              <TD COLSPAN="3">
                 <IMG SRC="i/c.gif" HEIGHT="12" WIDTH="1">
              </TD>
            </TR>
            <TR>
             <TR BGCOLOR="#0073CC">
              <TD WIDTH="60%" HEIGHT="24" class="ie"> message title</TD>
              <TD WIDTH="20%" class="ie">time</TD>
              <TD WIDTH="20%" class="ie">author</TD>
            </TR>
          <TR>
            <TD COLSPAN="3">
              <IMG SRC="i/c.gif" HEIGHT="2" WIDTH="1">
            </TD>
          </TR>
          <TR>
            <TD align="left" class="id" COLSPAN="3">No hits found</TD>
          </TR>
          <TR>
            <TD HEIGHT="18" colspan="3" BGCOLOR="#0073CC" class="id">
            </TD>
          </TR>
        </TR>
        <TR>
          <TD COLSPAN="3">
            <IMG SRC="i/c.gif" HEIGHT="2" WIDTH="1">
          </TD>
        </TR>
        <TR>
          <TD COLSPAN="3" BGCOLOR="#0073CC">
            <IMG SRC="i/c.gif" HEIGHT="1" WIDTH="1">
          </TD>
        </TR>
      </TABLE>
    </body>
  </html>
```

The page sources are available in the default distribution under the samples directory.

See Also:

For more information about the functions used see: http_xslt() , http() , http_value() .

For more information about VSP in general go the VSP Section .

19.8.5. Setting up server-side Cross-Origin Resource Sharing (CORS) with Virtuoso

User agents (e.g., Web browsers) have traditionally restricted scripts within web pages by a Same Origin Policy, which allowed scripts to make requests only to resources within the same domain from which the scripts themselves originated. This restriction is meant to protect the user and their computer from "Trojan horse websites" which may appear to be safe, but which then make unsafe HTTP requests to other, invisible sites. This restriction also protects the second website from potential "Denial of Service" and other attacks, whether accidental or intentional.

This policy has the unfortunate side-effect of also preventing client-side Web applications served from one website ("Origin") from retrieving data from another website ("Origin").

Cross-Origin Resource Sharing (CORS) is a mechanism intended to enable safer client-side cross-origin requests, primarily focused on data.

How does CORS work?

Authentication and session-management information methods are extended in several ways:

- Enforcement by User Agent
  - A server providing a resource can include an

    Access-Control-Allow-Origin

    HTTP response header, with a value of the request's triggering script's site of origin (that is, the site which provided the script which made the request for the resource), to indicate whether access to the resource's contents may be allowed. The user agent validates that the value in this header matches the actual origin of the script which made the request.
  - User agents can use a "pre-flight request" to discover whether a cross-origin resource is prepared to accept requests from a given script origin, using a complex method (which we will not detail here). Again, the response is validated by the user agent.
- Enforcement by Server-side Application
  - Server-side applications can refer to the

    Origin

    HTTP request header to discover whether the user agent deemed it a cross-origin request. Here, the server-side application enforces limitations (e.g., returning nothing, partial results, or full results) on the cross-origin requests that they are willing to service at all.

CORS Setup for Virtuoso servers

With Virtuoso 6 and later (specific earliest versions as noted below), CORS support may be configured at the server-level or enabled through application logic (scripting, PL, etc.).

When working with older versions of Virtuoso, CORS support cannot be configured at the server-level, but it may be enabled within application logic (scripting, PL, etc.).

Application-level CORS Setup

Any Virtuoso PL (VSP)-based application can implement CORS checking through well-known HTTP functions `http_request_header()` and `http_header()` . This method will work with any version of Virtuoso. For instance:

```
<?vsp
    IF (http_request_header (lines, 'Origin', NULL) = 'http://host.org')
      {
          http_header ('Access-Control-Allow-Origin: http://host.org\r\n');
      }
   ELSE
      {
         RETURN;
      }
-- Additional code here ---

?>
```

Applications running in other hosted environments (Perl, Python, PHP, ASP.NET, etc.) may also use their specific scripting options to add and/or check relevent headers.

Server-level CORS Setup

Note: These instance/server-level configuration instructions require Virtuoso Open Source (VOS) 6.1.3 or later , or Virtuoso Commercial Edition 6.2.3129 or later .

1. In the Virtuoso Conductor, go to

   Web Application Server -> Virtual Directories & Directories

   .

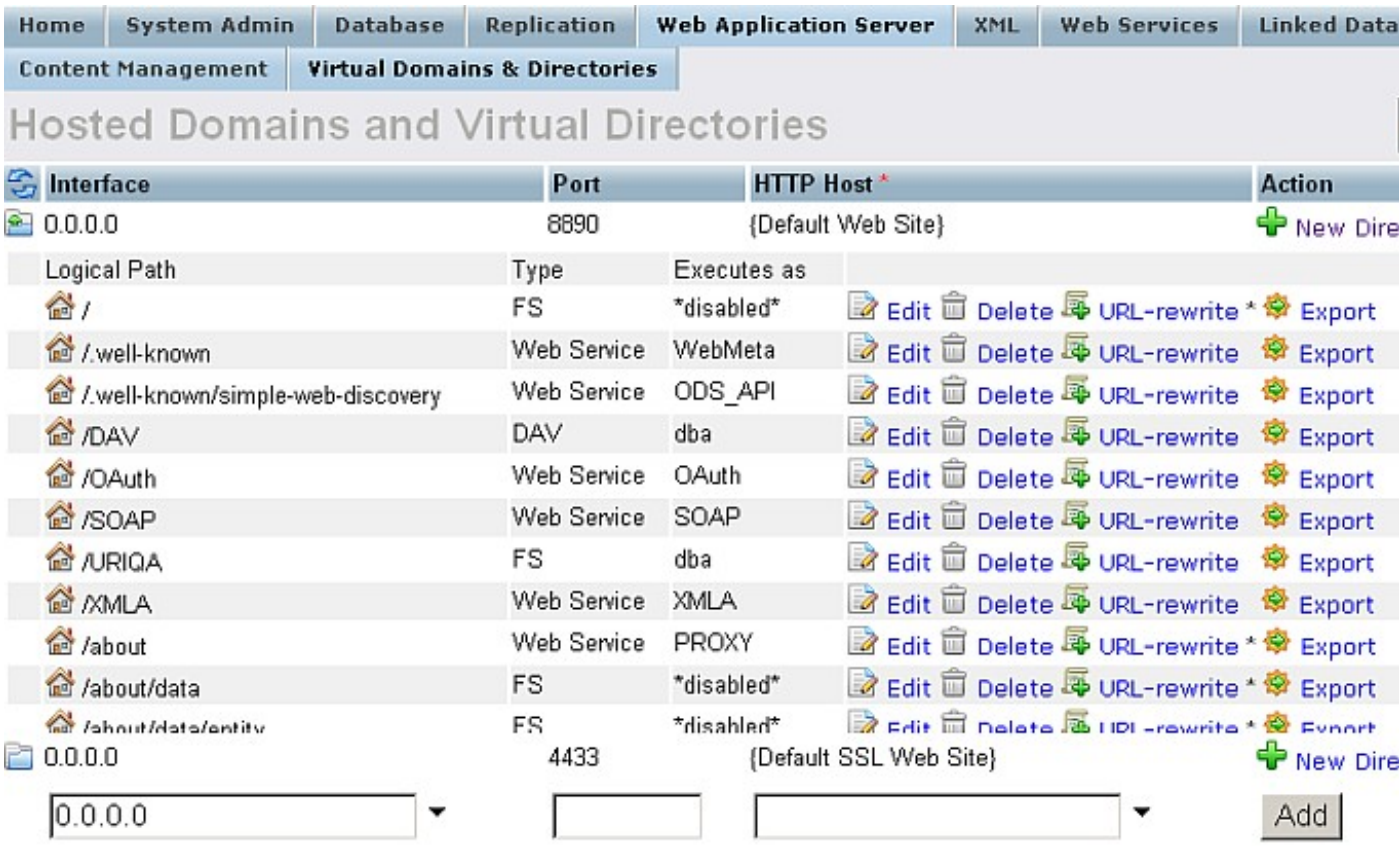
Figure 19.3. Server-side Cross-Origin Resource Sharing (CORS) Example Setup

2. Expand the

   Interface

   store.

Figure 19.4. Server-side Cross-Origin Resource Sharing (CORS) Example Setup



3. Click

   New Directory

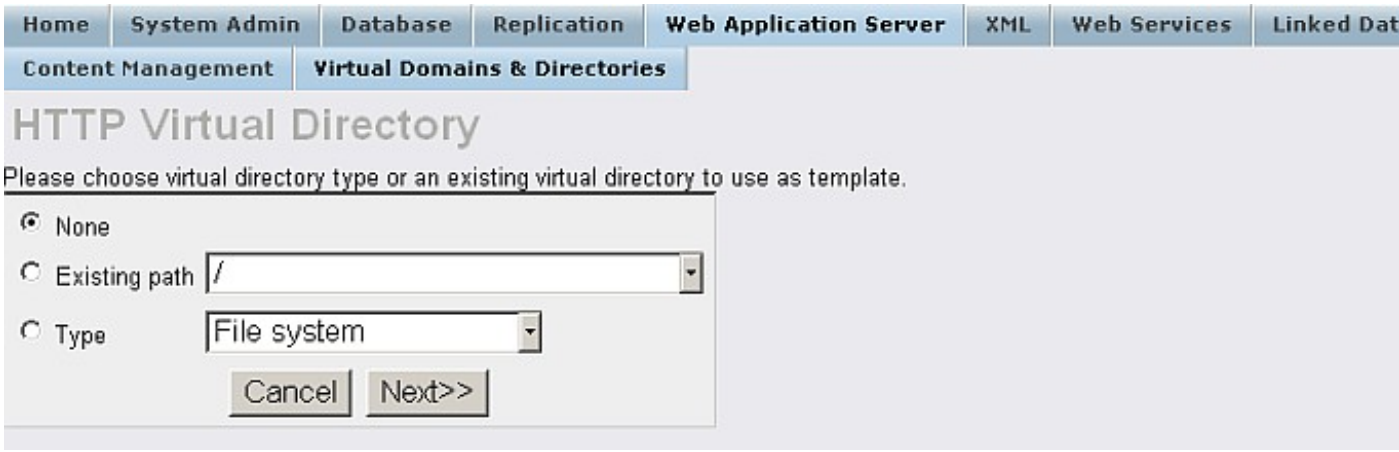
.

4. Specify the desired

   Virtual Directory Type

   , or choose an existing virtual directory to use as a template.

   Figure 19.5. Server-side Cross-Origin Resource Sharing (CORS) Example Setup



5. Click

   Next

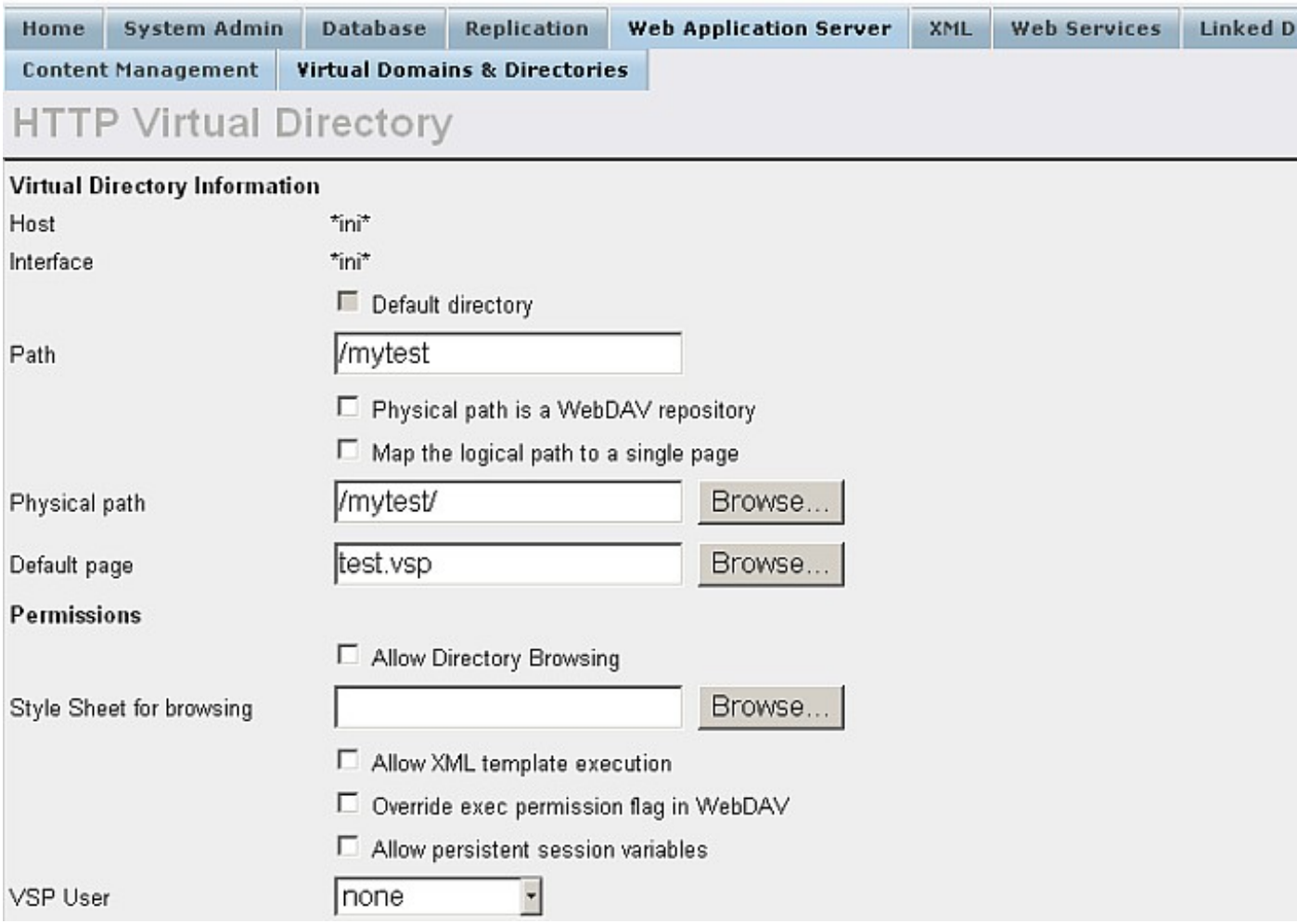
   .

6. Specify the

   Directory Path

   value.

   Figure 19.6. Server-side Cross-Origin Resource Sharing (CORS) Example Setup

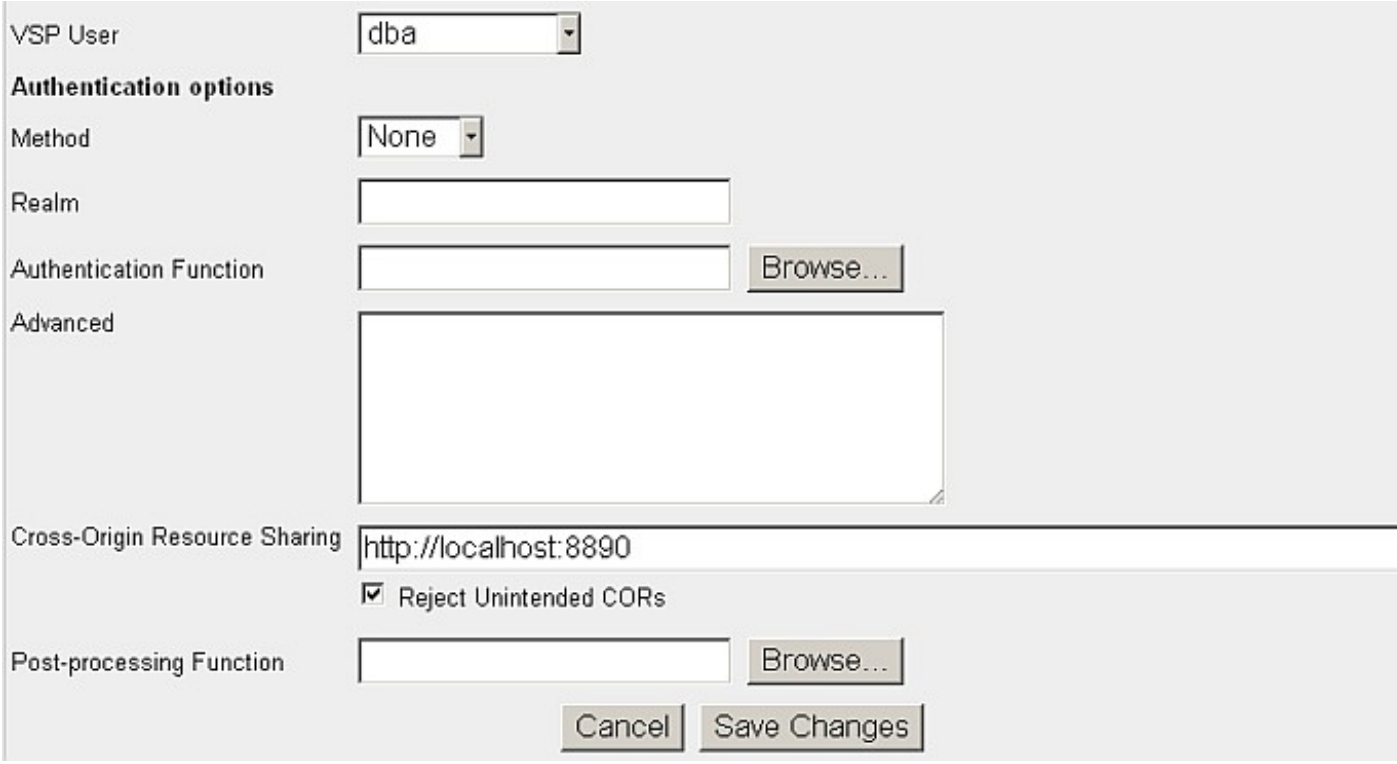


7. Set the

   CORS

   options.

Figure 19.7. Server-side Cross-Origin Resource Sharing (CORS) Example Setup



- *Cross-Origin Resource Sharing*

  : Contains a single wildcard asterisk, i.e., "

  *

  ", or a space-delimited list of HTTP server URIs, e.g., "

  http://example.com:8080 http://blah.example.com http://foo.example.com

  ". Scripts originating on the listed HTTP servers are authorized to retrieve the specified resource(s); the wildcard means scripts from any HTTP server will be authorized. For this example, enter the following single URI:

  ```
  http://demo.openlinksw.com
  ```

- *Reject Unintended CORs*

  checkbox: When ticked (and the application does not overwrite headers), unmatched Origins will be rejected by sending an empty response. For this example, tick this box.

8. Click Save changes.

Example Usage with cURL

Example 1

1. Suppose the example setup above is performed, and http://demo.openlinksw.com/ is in the CORS list.
2. In this case, the request below will return an empty response:

```
$ curl -i  http://demo.openlinksw.com/mytest/test.vsp
HTTP/1.1 200 OK
Server: Virtuoso/06.02.3128 (Win32) i686-generic-win-32  VDB
Connection: Keep-Alive
Content-Type: text/html; charset=ISO-8859-1
Date: Thu, 28 Oct 2010 09:27:54 GMT
Accept-Ranges: bytes
Content-Length: 0
```

Example 2

1. Suppose the example Setup above is performed, and http://demo.openlinksw.com/ is in the CORS list.
2. Also, suppose the curl command includes a proper Origin value, e.g.:

```
-H "Origin: http://demo.openlinksw.com"
```

3. In this case, the request below will return a response including the retrieved content, etc.

```
$ curl -i -H "Origin: http://demo.openlinksw.com"  http://demo.openlinksw.com/mytest/test.vsp
HTTP/1.1 200 OK
Server: Virtuoso/06.02.3128 (Win32) i686-generic-win-32  VDB
Connection: Keep-Alive
Content-Type: text/html; charset=ISO-8859-1
Date: Thu, 28 Oct 2010 09:40:21 GMT
Access-Control-Allow-Origin: http://demo.openlinksw.com
Accept-Ranges: bytes
Content-Length: 7
```

Example 3

1. Suppose the Example Setup above is performed, but reject is off (i.e., "Reject Unintended CORs" check-box is not ticked).
2. In this case, the request below will return a response that lacks Access-Control-Allow-Origin:

```
$ curl -i  http://demo.openlinksw.com/mytest/test.vsp
HTTP/1.1 200 OK
Server: Virtuoso/06.02.3128 (Win32) i686-generic-win-32  VDB
Connection: Keep-Alive
Content-Type: text/html; charset=ISO-8859-1
Date: Thu, 28 Oct 2010 09:45:01 GMT
Accept-Ranges: bytes
Content-Length: 7
```


See Also:

Origin header proposal for CSRF and click-jacking mitigation

CORS In Action

Cross-domain Ajax with Cross-Origin Resource Sharing

Guide to Secure Implementation of Cross Origin Requests in HTML5

19.9. LDAP

The Lightweight Directory Access Protocol (LDAP) is a protocol for accessing online directory services. It runs directly over TCP and can be used to access a standalone LDAP directory service or a directory service maintained by X.500. A directory is type of database that stores information that is read more often that written to and so services are typically geared for high volume read access and offer simpler transaction support than general purpose databases. The LDAP directory service model is based on entries. An entry is a collection of attributes that has a "distinguished name" (DN). The DN is used to refer to the entry unambiguously. Each of the entry's attributes has a type and one or more values. The types are typically mnemonic strings, like "cn" for common name, or "mail" for email address. The values depend on what type of attribute it is: an email attribute might contain the string value "foo@bar.org". A jpegPhoto attribute would contain a photograph in binary JPEG format.

LDAP directory entries are arranged in a hierarchical tree-like structure that may reflect political, geographic and/or organizational boundaries. Hence, entries representing countries appear at the top of the tree, below them are entries representing states or national organizations, then, entries representing people, printers, documents, anything else...

LDAP provides methods for authentication. Directories can be made accessible to the general public or protected however the case may be.

LDAP is made for finding people and resources on a network. It provides a completely different functionality from web searches such as yahoo or webcrawler, which work simply by text matching and often return many thousand's of entries. Using an LDAP directory to locate something however, if you know the approximate location of where it is, such as what organization and country it is in, then you can do a search and return much fewer entries. LDAP is a vendor independent open protocol. The fact that organizations can alter it to their own needs is key. Also, large companies can use it as the basis for their own more complex

directory servers.

There are common misconceptions about the role of LDAP. LDAP is not intended as a replacement for local databases. They are not built to be added and modified too easily and thus would not work in a situation such as an airline reservation system. Also, LDAP is not meant to be a replacement for DNS. DNS is a specialized well used service on the Internet for matching IP addresses to real names. However databases and DNS and LDAP like most Internet services work very well in collaboration.

Virtuoso has the ability to act as an LDAP client, using built-in functions, and has been tested as a back end for the OpenLDAP server.

19.9.1. LDAP Client

Virtuoso provides four functions for accessing an LDAP Directory Service:

`ldap_search()`
`ldap_delete()`
`ldap_add()`
`ldap_modify()`

You can call each function independently. Each invocation of a function creates a separate LDAP session which is closed at the end of the operation.

URLs beginning with

```
ldaps://
```

invoke a non-conditional SSL operation. Such URLs cannot be used with the *try_tls* flag set. If the URL does not contain a port number then Virtuoso will use the default port number 389 for <ldap://> and port number 636 for <ldaps://>.

The *username* and *password* optional parameters are used to bind LDAP connections using basic authentication. If the operation is permitted to all users then you can leave these parameters out.

19.9.2. LDAP Server

Virtuoso is not an LDAP server but can easily provide a data back-end to one, such as OpenLDAP. This section explains how to incorporate OpenLDAP with Virtuoso on the Linux platform. For other platforms the installation of OpenLDAP and configuration of the ODBC data source may differ.

1. download the OpenLDAP distribution from OpenLDAP.org .
2. Build OpenLDAP with the --enable-sql option.
3. You may need the iODBC library installed, which can be obtained from iODBC.org .
4. install the LDAP server using: make install
5. copy the following text into the slapd.conf

```
>>>>
include         /usr/local/etc/openldap/schema/core.schema

pidfile         /usr/local/var/slapd.pid
argsfile        /usr/local/var/slapd.args

database        sql
suffix          "c=BG"
rootdn          "cn=root,o=opl,c=US"
rootpw          secret
dbname          1111
dbuser          dba
dbpasswd        dba
subtree_cond    "UPPER(ldap_entries.dn) LIKE CONCAT('%',UPPER(?))"
upper_func      UPPER

access to attr=userPassword
  by self write
  by anonymous auth
  by dn="cn=root,o=opl,c=BG" write
```

```
          by * none

        access to *
          by self write
          by anonymous auth
          by * read
        <<<<
```

6. Configure the ODBC data source named '1111' for your running Virtuoso server.
7. Apply the virt_back.sql script against a running virtuoso server, for example with isql.
8. If you can also apply the virt_testdb.sql to insert some demo schema and entries into the DB.
9. Export ODBCINI variable: export ODBCINI=<location of your odbc.ini>.
10. Start the slapd executable with debug option: 'slapd -d5' to ensure that all settings are correct.

The virt_back.sql Script

```
create table ldap_oc_mappings (
        id              integer identity primary key,
        name            varchar (64) not null,
        keytbl          varchar (64) not null,
        keycol          varchar (64) not null,
        create_proc     varchar (255),
        delete_proc     varchar (255),
        expect_return   integer not null
);

create unique index ldap_oc_mappings_name on ldap_oc_mappings (name);

create table ldap_attr_mappings (
        id              integer identity primary key,
        oc_map_id       integer not null references ldap_oc_mappings(id),
        name            varchar (255) not null,
        sel_expr        varchar (255) not null,
        from_tbls       varchar (255) not null,
        join_where      varchar (255),
        add_proc        varchar (255),
        modify_proc     varchar (255),
        delete_proc     varchar (255),
        param_order     integer not null,
        expect_return   integer not null
);

create table ldap_entries (
        id              integer identity primary key,
        dn              varchar (255) not null ,
        oc_map_id       integer not null references ldap_oc_mappings(id),
        parent          integer not null ,
        keyval          integer not null
);

create unique index ldap_entries_oc_map_id_keyval on ldap_entries (oc_map_id, keyval);

create unique index ldap_entries_dn on ldap_entries (dn);

create table ldap_referrals
(
        entry_id        integer not null references ldap_entries(id),
        url             varchar not null
);

create index entry_idx on ldap_referrals(entry_id);

create table ldap_entry_objclasses
(
        entry_id        integer not null references ldap_entries(id),
        oc_name         varchar(64)
);

create index entry_idx_entry_id on ldap_entry_objclasses(entry_id);
```

The virt_testdb.sql Script

```
create table authors_docs (
        pers_id integer not null ,
        doc_id integer not null,
        primary key  (pers_id, doc_id)
);

create table documents (
        id integer identity not null ,
        abstract varchar (255) null ,
        title varchar (255) null ,
        body binary (255) null,
        primary key  (id)
);

create table institutes (
        id integer identity  not null ,
        name varchar (255) not null,
        primary key  (id)
);

create table persons (
        id integer identity  not null ,
        name varchar (255) null,
        sname varchar (255) null,
        email varchar (255) null,
        passw varchar (255) null,
        primary key  (id)

);

create table phones (
        id integer identity  not null ,
        phone varchar (255) not null ,
        pers_id integer not null,
        primary key  (id)

);

insert into institutes (id,name) values (1,'');
insert into institutes (id,name) values (2,'opl');

insert into persons (id,name,sname,email,passw)
  values (1,'Mitko Iliev', 'Iliev', 'imitko@openlinksw.com', 'abc');
insert into persons (id,name,sname,email,passw)
  values (2,'George Kodinov', 'Kodinov', 'gkodinov@openlinksw.com', 'cde');
insert into persons (id,name,sname,email,passw)
  values (3,'Zdravko Tashev', 'Tashev', 'ztashev@openlinksw.com', 'efg');

insert into phones (id,phone,pers_id) values (1,'032-947020',1);
insert into phones (id,phone,pers_id) values (2,'032-633710',1);
insert into phones (id,phone,pers_id) values (3,'032-947020',2);

insert into documents (id,abstract,title) values (1,'abstract1','book1');
insert into documents (id,abstract,title) values (2,'abstract2','book2');

insert into authors_docs (pers_id,doc_id) values (1,1);
insert into authors_docs (pers_id,doc_id) values (1,2);
insert into authors_docs (pers_id,doc_id) values (2,1);

-- LDAP mappings
insert into ldap_oc_mappings (id,name,keytbl,keycol,create_proc,delete_proc,expect_return)
  values (1,'inetorgperson','persons','id','{call create_person(?)}','{call delete_person(?)}',0);

insert into ldap_oc_mappings (id,name,keytbl,keycol,create_proc,delete_proc,expect_return)
  values (2,'document','documents','id','{call create_document(?)}','{call delete_document(?)}',0);

insert into ldap_oc_mappings (id,name,keytbl,keycol,create_proc,delete_proc,expect_return)
  values (3,'organization','institutes','id','{call create_org(?)}','{call delete_org(?)}',0);

insert into ldap_attr_mappings
    (id, oc_map_id, name, sel_expr, from_tbls, join_where, add_proc,
```

```
                modify_proc, delete_proc, param_order, expect_return)
  values (1,1,'cn','persons.name','persons',NULL,'{call set_person_name(?,?)}', NULL,NULL,0,0);

insert into ldap_attr_mappings
    (id, oc_map_id, name, sel_expr, from_tbls, join_where, add_proc,
    modify_proc, delete_proc, param_order, expect_return)
  values (2,1,'telephoneNumber','phones.phone','persons,phones', 'phones.pers_id=persons.id',
          '{call add_phone(?,?)}', NULL,'{call delete_phone(?,?)}',0,0);

insert into ldap_attr_mappings
    (id,oc_map_id,name,sel_expr,from_tbls,join_where,add_proc,modify_proc,
    delete_proc,param_order,expect_return)
  values (3,1,'sn','persons.sname','persons',NULL,
          '{call set_person_sname(?,?)}', NULL,NULL,0,0);

insert into ldap_attr_mappings
    (id,oc_map_id,name,sel_expr,from_tbls,join_where,add_proc,modify_proc,
    delete_proc,param_order,expect_return)
  values (10,1,'mail','persons.email','persons',NULL,
          '{call set_email(?,?)}', NULL,NULL,0,0);

insert into ldap_attr_mappings
    (id,oc_map_id,name,sel_expr,from_tbls,join_where,add_proc,modify_proc,
    delete_proc,param_order,expect_return)
  values (10,1,'userPassword','persons.passw','persons',NULL,
          '{call set_pwd (?,?)}', NULL,NULL,0,0);

insert into ldap_attr_mappings
    (id,oc_map_id,name,sel_expr,from_tbls,join_where,add_proc,modify_proc,
    delete_proc,param_order,expect_return)
  values (4,2,'description','documents.abstract','documents',NULL,
          '{call set_doc_abstract(?,?)}', NULL,NULL,0,0);

insert into ldap_attr_mappings
    (id,oc_map_id,name,sel_expr,from_tbls,join_where,add_proc,modify_proc,
    delete_proc,param_order,expect_return)
  values (5,2,'documentTitle','documents.title','documents',NULL,
          '{call set_doc_title(?,?)}', NULL,NULL,0,0);

insert into ldap_attr_mappings
    (id,oc_map_id,name,sel_expr,from_tbls,join_where,add_proc,modify_proc,
    delete_proc,param_order,expect_return)
  values (7,3,'o','institutes.name','institutes',NULL,
          '{call set_org_name(?,?)}', NULL,NULL,0,0);

insert into ldap_attr_mappings
    (id,oc_map_id,name,sel_expr,from_tbls,join_where,add_proc,modify_proc,
    delete_proc,param_order,expect_return)
  values (8,1,'documentDN','ldap_entries.dn','ldap_entries,documents,authors_docs,persons',
          'ldap_entries.keyval=documents.id AND ldap_entries.oc_map_id=2 AND
                authors_docs.doc_id=documents.id AND authors_docs.pers_id=persons.id',
                '{call make_doc_link(?,?)}',NULL,'{call del_doc_link(?,?)}',0,0);

insert into ldap_attr_mappings
    (id,oc_map_id,name,sel_expr,from_tbls,join_where,add_proc,modify_proc,
    delete_proc,param_order,expect_return)
  values (9,2,'documentAuthor','ldap_entries.dn','ldap_entries,documents,
          authors_docs,persons','ldap_entries.keyval=persons.id AND ldap_entries.oc_map_id=1
                AND authors_docs.doc_id=documents.id AND authors_docs.pers_id=persons.id',
                '{call make_author_link(?,?)}',NULL,'{call del_author_link(?,?)}',0,0);

-- entries
insert into ldap_entries (id,dn,oc_map_id,parent,keyval)
values (1,'c=US',3,0,1);

insert into ldap_entries (id,dn,oc_map_id,parent,keyval)
values (2,'o=opl,c=US',3,1,2);

insert into ldap_entries (id,dn,oc_map_id,parent,keyval)
values (3,'cn=Mitko Iliev,o=opl,c=US',1,2,1);

insert into ldap_entries (id,dn,oc_map_id,parent,keyval)
values (4,'cn=George Kodinov,o=opl,c=US',1,2,2);
```

```
insert into ldap_entries (id,dn,oc_map_id,parent,keyval)
values (5,'cn=Zdravko Tashev,o=opl,c=US',1,2,3);

insert into ldap_entries (id,dn,oc_map_id,parent,keyval)
values (6,'documentTitle=book1,o=opl,c=US',2,2,1);

insert into ldap_entries (id,dn,oc_map_id,parent,keyval)
values (7,'documentTitle=book2,o=opl,c=US',2,2,2);

-- testDB support procedures

CREATE PROCEDURE add_phone (in _pers_id int, in _phone varchar(255))
{
  INSERT INTO phones (pers_id,phone) VALUES (_pers_id,_phone);
};

CREATE PROCEDURE create_person (inout _keyval integer)
{
  INSERT INTO persons (name) VALUES ('');
  _keyval := (SELECT MAX(id) FROM persons);
};

CREATE PROCEDURE delete_person (in _keyval integer)
{
  DELETE FROM phones WHERE pers_id=_keyval;
  DELETE FROM authors_docs WHERE pers_id=_keyval;
  DELETE FROM persons WHERE id=_keyval;
};

CREATE PROCEDURE create_org (out _keyval integer)
{
  INSERT INTO institutes (name) VALUES ('');
  _keyval:=(SELECT MAX(id) FROM institutes);
};

CREATE PROCEDURE create_document (out _keyval integer)
{
  INSERT INTO documents (title) VALUES ('');
  _keyval:=(SELECT MAX(id) FROM documents);
};

CREATE PROCEDURE delete_org (in _keyval integer)
{
  DELETE FROM institutes WHERE id=_keyval;
};

CREATE PROCEDURE delete_document (in _keyval integer)
{
  DELETE FROM authors_docs WHERE doc_id=_keyval;
  DELETE FROM documents WHERE id=_keyval;
};

CREATE PROCEDURE delete_phone (in _keyval integer, in _phone varchar(64) )
{
  DELETE FROM phones WHERE pers_id=_keyval AND phone=_phone;
};

CREATE PROCEDURE set_person_name (in _keyval integer, in _new_name varchar(255))
{
  UPDATE persons SET name=_new_name WHERE id=_keyval;
};

CREATE PROCEDURE set_person_sname (in _keyval integer, in _new_name varchar(255))
{
  UPDATE persons SET sname=_new_name WHERE id=_keyval;
};

CREATE PROCEDURE set_email (in _keyval integer, in _mail varchar(255))
{
  UPDATE persons SET email=_mail WHERE id=_keyval;
};
```

```
CREATE PROCEDURE set_pwd (in _keyval integer, in _mail varchar(255))
{
  UPDATE persons SET passw=_mail WHERE id=_keyval;
};

CREATE PROCEDURE set_org_name (in _keyval integer, in _new_name varchar(255))
{
  UPDATE institutes SET name=_new_name WHERE id=_keyval;
};

CREATE PROCEDURE set_doc_title ( in _keyval integer, in _new_title varchar(255)  )
{
  UPDATE documents SET title=_new_title WHERE id=_keyval;
};

CREATE PROCEDURE set_doc_abstract (in _keyval integer, in _new_abstract varchar(255)  )
{
  UPDATE documents SET abstract=_new_abstract WHERE id=_keyval;
};

CREATE PROCEDURE make_author_link (in _keyval integer, in _author_dn varchar(255)  )
{
  DECLARE _per_id integer;
  _per_id := (SELECT keyval FROM ldap_entries
      WHERE oc_map_id=1 AND dn=_author_dn);
  IF (_per_id IS NOT NULL)
      INSERT INTO authors_docs (doc_id,pers_id) VALUES (_keyval,_per_id);
};

CREATE PROCEDURE make_doc_link (in _keyval integer, in _doc_dn varchar(255)  )
{
  DECLARE _doc_id integer;
  _doc_id := (SELECT keyval FROM ldap_entries
      WHERE oc_map_id=2 AND dn=_doc_dn);
  IF (_doc_id IS NOT NULL)
      INSERT INTO authors_docs (pers_id,doc_id) VALUES (_keyval,_doc_id);
};

CREATE PROCEDURE del_doc_link (in _keyval integer, in _doc_dn varchar(255)  )
{
  DECLARE _doc_id integer;
_doc_id := (SELECT keyval FROM ldap_entries
      WHERE oc_map_id=2 AND dn=_doc_dn);
  IF (_doc_id IS  NOT NULL)
      DELETE FROM authors_docs WHERE pers_id=_keyval AND doc_id=_doc_id;
};

CREATE PROCEDURE del_author_link (in _keyval integer, in _author_dn varchar(255)  )
{
  DECLARE _per_id integer;
  _per_id := (SELECT keyval FROM ldap_entries
      WHERE oc_map_id=1 AND dn=_author_dn);
  IF (_per_id IS NOT NULL)
      DELETE FROM authors_docs WHERE doc_id=_keyval AND pers_id=_per_id;
};
```

Chapter 20. Free Text Search

Abstract

Virtuoso provides a compact and efficient free text indexing capability for text and XML data. A free text index can be created on any character column, including wide and long data.

The contains SQL predicate allows content based retrieval of textual data. This predicate takes a column and a text expression and is true if the pattern of words in the text expression occurs in the column value. There must exist a previously created text index of the column. The text expression can contain single words and phrases connected by boolean connectives or the proximity operator. Words can contain wildcards but must begin with at least three non-wildcard characters if a wildcard is to be used. While it is enough to declare a free text index on a column and then just use the contains predicate for many applications, Virtuoso offers a range of options for tailoring how the indexing works.

If a certain application specific order of search results is desired more frequently than others, it is possible to specify a single or multipart key in the order of which hits will be returned from contains searches. Both ascending and descending order of the key is supported. To restart a search in the middle it is possible to specify a starting and ending key value. This works if the results are generated in the order of the application specific doc ID .

If non-text criteria are often used to filter or sort results of contains searches, it is possible to cluster these non-text data inside the free text index for faster retrieval. It is often substantially faster to retrieve the extra data from inside the text index than to get them from the row referenced by the text index. Such data are called offband data , since they are not actually text but are stored similarly to text.

It is possible to pre-process the text before it is indexed or unindexed. This feature can be used for data normalization and/or for adding content from other than the primary text field being indexed into the index. One example is adding the names of all newsgroups where an article appears to the index when indexing a news article. Thus when retrieving articles based on text and newsgroup, group can be used to very efficiently filter out the hits that are not in the group, even if the text indexed does not itself contain the group name. Another application of the same technique is adding text from multiple columns into the same index.

If the column being indexed is XML data, this can be declared and enforced by the text index. XML data will be indexed specially to support efficient XPATH predicate evaluation with the xcontains predicate.

Text Triggers is a feature that allows the storage of a large body of free text queries and automatically generating hits when documents matching the criteria are added to the index. This is useful for personalized data feeds, user profiles, content classification etc, which Virtuoso can send the results to in an email message. The conditions can be either free text expressions or XPATH expressions for XML content.

The text index can be kept synchronous with the data being indexed, so that the index is updated in the same transaction as the data. The other possibility is to maintain the text index asynchronously as a scheduled task (batch mode), which can execute up to an order of magnitude faster. The asynchronous mode of operation offers substantially higher performance if changes of multiple entries are processed in one batch index refresh.

20.1. Basic Concepts

A text index is created with the CREATE TEXT INDEX statement. This creates a number of stored procedures and triggers which will transparently manage the text index. A text index is dropped by dropping the generated words table, called <table>_<column>_WORDS, where <table> and <column> are the table and column over which the index is made.

Example 20.1. Creating a Text Index

```
CREATE TABLE FTT (ID INTEGER, FILE varchar,  DT LONG VARCHAR );
CREATE TEXT INDEX ON FTT (DT);
```

This is the simplest case of making a text index. This process will add an extra column to the table being indexed which it will use to reference rows from the new text index. If there already exists an integer primary key then this will be used and no new column will be added. Such a column may not be 0 or negative.

Once the index is made the contains query can be used to retrieve rows:

```
insert into ftt (id, dt) values (1, 'foo');
select from ftt where contains (dt, 'foo');
```

The contains predicate is a normal SQL predicate and can be used together with other predicates in the where clause. Contains may however not figure inside an OR or NOT. Hence:

```
select * from ftt where contains (dt, 'foo or bar ');
```

is OK but

```
select * from ftt where contains (dt, 'foo ') or contains (dt, 'bar');
```

is not.

20.2. Creating Free Text Indexes

20.2.1. The CREATE TEXT INDEX statement

Define and optionally initialize a text index on a column.

```
create_freetext_index
```

```
        : CREATE TEXT [XML] INDEX ON q_table_name '(' column ')'
        [WITH KEY column]
        [NOT INSERT]
        [CLUSTERED WITH '(' column_commalist ')' ]
        [USING FUNCTION]
        [LANGUAGE STRING]
    [ENCODING STRING]
        ;
```

XML - The XML keyword specifies that the data is to be indexed as XML, hence element names and attributes will be processed separately for use with the XCONTAINS predicate.

The *q_table_name* is a qualified table name on which the index is created. The generated procedures and auxiliary tables will be with the owner and qualifier of this table.

The *column* must be a column of the above table, of the VARCHAR, NVARCHAR, LONG VARCHAR, LONG NVARCHAR, LONG XML or XMLTYPE data type. The column may additionally have the IDENTIFIED BY option if the content is XML. This will be used to provide a base URI for traversing relative references. The XML option for the index has to be specified if the content is LONG XML or XMLTYPE.

WITH KEY column - This allows optionally specifying a uniquely identifying column which will be used as a foreign key for referencing the table from the text index. If this is not specified and there is a single part integer primary key, this primary key is used as the key. If there is no suitable primary key and the option is not present, an integer column is added and a sequence object is used to supply distinct values.

When specified, the column must be non-NULL and its run time value must either be an integer or a composite (See composite data type). The length of the values of this column is crucially important since it is repeated for each distinct word of each row. This column is called the free text document id column in the rest of this documentation.

NOT INSERT - If present, specifies that the index is not filled when created. The VT_INDEX_<table> function is still created but not run. You can run this manually or scheduled at an appropriate time. If batching is enabled then the index will be filled up at that time.

CLUSTERED WITH - The column list must consist of columns of the table. Their values are stored in the text index so that the values can be more efficiently located for filtering than if they had to be retrieved from the table itself. The combined length of the columns should be relatively small, not much over 200 bytes for this to be effective. There is no hard upper limit but long blobs are not advisable.

USING FUNCTION - This allows specifying a hook function for indexing an unindexing a document. The index hook is called before processing the words of the column to be indexed. This can be used to index extra text in addition to the column value or to modify the text before indexing. If the hook function returns 1 the column is not additionally processed. If hook returns 0 the column is processed normally, in addition to the words the hook may have inserted. See the section on hook functions. The hook functions are always named <table>_<column>_INDEX_HOOK and <table>_<column>_UNINDEX_HOOK, in the owner and qualifier of the table, where <table> is the unqualified name of the table.

LANGUAGE - The string literal following this keyword may be a language name. See Internationalization for details.

ENCODING - The string literal following this keyword may be an encoding name.

Creating the text index may take a long time. All this time is in 'atomic' mode, so no other database activity is allowed during it and no log is generated. If there is an error, e.g. out of disk the created index is dropped and the error is reported. A checkpoint should be made after the index is complete. If the index should be created under a different qualifier or the generated tables altered after the index is created the NOT INSERT option should be used to delay filling the index.

The CREATE TEXT INDEX statement will automatically make a unique index based on the free text document id, if this is not the PK of the table. If one wishes to modify this index, it can be found and dropped with DROP INDEX and reconstructed, but make sure that the reference in SYS_VT_INDEX matches and that the new index has the same name as the previous version.

A freshly created text index is in synchronous mode. This means that that changes to the table are immediately reflected on the index as they occur. This is done through a set of automatically generated triggers. If large changes to data will be performed, the batch mode is far more efficient.

See Also:

Although it is recommended to use the methods described above, the vt_create_text_index function can also be used

20.2.2. Choosing An Application Specific Document ID

The free text index conceptually works by making an index entry for each distinct word of each indexed column value which references back to the row containing the data being indexed. Therefore the table must have a unique ID that will be stored in conjunction with each distinct word in the indexed column in the text index. For space efficiency this should be as short as possible. If nothing else is specified the CREATE TEXT INDEX statement makes such a unique column and fills it automatically from a sequence producing unique numbers. If a single part integer primary key exists then this key is used as the free text index document ID. Note However that the values <= 0 are prohibited. This is however not always optimal, hence the application may specify what column is used to identify the row for text indexing. Such a unique column is referred to as the *Free Text Document ID* .

Suppose that a table contains news articles that should most frequently be retrieved latest first, in descending order of a datetime field. This can be achieved by just selecting the matching articles and sorting them with a SQL ORDER BY clause but this can be very inefficient. The reason for this is that all hits will first have to be found, then sorted and only then can the first hit be returned to the user. Further, the sort key will have to be retrieved from the table, causing a random access for each text hit. The sorting can be totally avoided if the document ID that is used to refer to the table from the index is itself ordered by date. This has several advantages:

- To retrieve the n latest, one just takes the n first hits produced by the contains search, no sorting required.
- To get the next n hits, one repeats the search but now specifying that the start ID is the ID of the last row of the previous set. No sorting and no scrollable cursors are required and the first hits can be returned before generating all hits. This is specially useful if the search criteria match many articles.

This has a disadvantage in that a longer document ID will have to be stored for each distinct word of each distinct article. This may result in a 60% increase in the index size but largely offsets the penalties of sorting. One should however exercise the utmost care in making this ID as short as possible. The maximum length of the ID is 30 bytes, but with this length the storage is extremely wasteful, so an ID with fields adding up to some 10 bytes is much better.

We will note that the document ID can be an aggregate of several scalars. In the news article example, it could be a datetime, ID number pair. This is so because the datetime typically would not be unique and the ID is required itself to be unique.

However, rather than storing the datetime and an integer article number, it is advisable to compress the datetime into a number, e.g. a count of minutes after a given date. This maintains the temporal order to within a minute and takes less than half the space taken by the datetime with all its fractions, time zones etc.

For handling multi-part ID's like scalars there is a special data type, composite. Thus, if an application specific document ID is not an integer, it must be a composite totaling less than 30 bytes of content divided among its members.

20.2.3. The composite Data Type

A composite is like a heterogeneous array, except that it is limited in length, may be stored as a column value for a column declared as ANY, and may be a key part in a SQL index. Thus, comparison is defined for composites as follows:

- Composites are equal if all parts are equal.
- A composite is less than another if the first part of it which is not equal to the corresponding part of the other composite is less than that part.
- If a composite has less parts than another and all of its parts are equal to the corresponding parts of the longer composite, then the shorter is considered less.

The collation of composites is just like that of strings, except that in the place of characters, arbitrary run time typed scalars are compared.

When strings are compared inside composites, they are compared as binary, without any specific collation. Normal numeric coercion applies to comparison of composites. Narrow and wide strings are compared with binary collation. If two elements are of different types, e.g. a number and a string and are compared, the data type will decide the outcome. For example any integer is always less than any string.

The composite SQL function makes a composite. It takes a variable number of arguments and returns a composite. The composite_ref function takes a composite and a zero based index and returns the value. The serialized length of a composite is limited to 255 characters. If a composite is used in a free text index it is limited to 30 characters.

- composite()
- composite_ref()

See Also:

See the Data Types section for the storage requirement of each data type.

20.2.4. Free Text Index Examples

```
composite (1, 2) = composite (1, 2)  is true
composite (1, 2) < composite (1, 3) is true.
composite (1, 0) > composite (1) is true.

composite_ref (composite (1, 2), 1) = 2
```

Example of a Composite Application Specific Document ID

The below code creates a table for news articles and defines a text index with a composite document id.

```
create table article(author_name varchar(255),
                     description long varchar,
                     pub_date datetime,
                     id integer,
                     title varchar(255),
                     dtid any not null,
                     primary key(id));

create text index on article (description)
        with key dtid clustered with (pub_date, author_name);
```

Note:

The dtid must be unique, non null and must be a composite since it is used as a custom text index id.

Next we define a mapping between dates and integers. These will be minutes consecutive of 1990-1-1 0:00. With 525600 minutes per 365 day year we will not run out of values for a long time, the positive integer limit being 2**31.

```
create procedure date2short (in dt datetime)
{
  return (1440 * datediff ('day', {d'1990-1-1'}, dt)
        + hour (dt) * 60 + minute (dt));
}


create procedure short2date (in n integer)
{
  return (dateadd ('minute', mod (n, 1440),
        dateadd ('day', n / 1440, {d'1990-1-1'})));
}
```

Now we can insert an article:

```
insert into article (id, drtid, description, pub_date)
        values (1, composite (date2short ({dt '2001-1-15 12:44'}), 1),
                'sample news article', {dt '2001-1-15 12:44'});
```

Note:

The composite is the date2short of the datetime and the id. The dtid must be specified and cannot be generated by a trigger, since the free text index related triggers must have access to the value.

The text index will be in synchronous mode by default so we can now query the data:

```
select id from article where contains (description, 'sample');
select id from article where contains (description, 'sample', descending);
```

The first query will return the oldest hits first, the second the newest first. Note that inserting in ascending order of the document ID is incomparably more efficient than in descending order. There is no great speed difference between reading in ascending or descending order.

If no application specific ID were specified the order would reflect the insertion order. Note that in this example articles do not have to be received in publication order, although insertions will naturally tend to follow this.

Now since the ID has an application semantic, we can use it for filtering based on date:

Consider:

```
select id from article where contains (description, 'sample', descending,
        start_id, composite (date2short ({dt'2001-1-5'})));
```

Since the search goes in descending order of id and starts at an id beginning with the numeric value corresponding to 2001-1-5 0:00, we may only get hits where the id date component is less than this, newest first.

There is no time penalty for the start_id option. This is therefore incomparably faster than the query:

```
select id from article where contains (description, 'sample', descending)
and pub_date < {dt '2001-1-5'};
```

See Also:

The reference section for contains for a definition of these options.

20.2.5. Pre-processing and Extending the Content Being Indexed

Let us consider the news application. Assume now a many to many relationship between articles and numbered news channels.

```
create table article_channel (
        a_id integer references article,
        c_id integer,
        primary key (a_id, c_id);
```

Assume further that free text search criteria be combined to channel membership tests.

This could be expressed as follows

```
select * from article
        where contains (description, 'sample')
        and exists
                (select 1 from article_channel
                where a_id = id and c_id = ?);
```

We have a random access per each hit to a table with at least the population of the article table for each hit. The situation is yet worse if there is an OR of multiple channel id's to which the article may belong.

To optimize this, we may choose to add an extra word for each channel in which the article appears. Likewise, we may add the text of the title of the article to the text being index.

This can be done with the index hook feature.

```
create text index on article (description)
        with key dtid clustered with (pub_date, author_name)
        using function ;
```

We then define the hook functions

```
create procedure
        article_description_index_hook (inout vtb any, inout d_id any)
{
```

```
  for (select c_id from article_channel
        where a_id = composite_ref (d_id, 0))
  do
    {
      vt_batch_feed (vtb, sprintf ('ch%d', c_id), 0);
    }
  vt_batch_feed (vtb, coalesce ((select title from article
        where dtid = d_id), ''), 0);
  return 0;
}
```

This function gets all channel id's where the article appears and adds the word ch<nnnn> where <nnnnn> is the channel id. Thus to look for 'xx' on channel 1 or 10 one can use the text expression xx and (ch1 or ch10). Additionally, the text of the title is added to the text being indexed. Note that the d_id supplied is the free text document id and that the second part of it is the article id.

To reverse the effect, the unindex function works as follows:

```
create procedure
        article_description_unindex_hook (inout vtb any, inout d_id any)
{
  for (select c_id from article_channel
        where a_id = composite_ref (d_id, 0))
  do
    {
      vt_batch_feed (vtb, sprintf ('ch%d', c_id), 1);
    }
  vt_batch_feed (vtb, coalesce ((select title
        from article where dtid = d_id), ''), 1);
  return 0;
}
```

These hooks accept 2 inout parameters, the so called vt batch and the free text document ID of the row at hand.

The function returns 1 to indicate that it has processed all words of the row to be indexed and 0 to indicate that it expects the default text to be processed by the caller as normally. Returning 1 is useful for example if extra word normalization is applied by the hook.

If the resulting index is used by `xcontains()` special predicate then hook functions should not alter the indexing of XML documents. `xcontains()` reads both free-text index and the actual document in order to locate particular fragments and it may miss search hits or get false hits if free-text index of a column does not match to the actual content of the column. It is still safe to call `vt_batch_feed` more than once during a single call of a hook function: first call for an unmodified XML document in the column plus calls for additional data.

If offband columns are declared then any call of a hook function should either return 0 or call `vt_batch_feed_offband()` before returning a non-zero value. If this condition is violated for a document then NULL is returned instead of correct offband value for the document without signalling any error.

The vt batch is an opaque data structure that accumulates words that will be added to the text index entry for a given row. The `vt_batch_feed` function adds words to the batch, the first argument is the vt batch, the second is the text and the third is a flag 0 for insert and 1 for delete. The text to be associated to the d_id in the index is the concatenation of all the text supplied by successive calls to `vt_batch_feed` . Word proximity is defined as if all text were a single string in the order of calling `vt_batch_feed` .

It is possible to partially alter the rtext associated with an existing document. This could be done when adding channels to an article which already exists. This could be done with the procedure:

```
create procedure
        ch_add_article (in cid integer, in aid integer)
{
  declare vtb, _dtid, cname any;
  if (exists (select 1 from article_channel
        where c_id = cid and a_id = aid))
    return;

  insert into article_channel (c_id, a_id) values (cid, aid);
```

```
  select dtid into _dtid from article where id = aid;

  vtb := vt_batch (1);
  vt_batch_d_id (vtb, _dtid);
  vt_batch_feed (vtb, sprintf ('ch%d', cid), 0);
  vt_batch_process_db_dba_article (vtb);
}
```

This first checks if the article is already on the channel, and if not, it makes a vt batch, gets the free text document id of the article, associates it to the vt batch and then adds a single word, ch<nnnn>. The vt batch is applied by calling the generated procedure vt_batch_process_<table>_<column> with the vt batch as only argument.

If multiple documents should be processed in one batch, it is possible to call vt_batch_d_id multiple times to feed data about multiple documents. In this case the successive document ids must be given in ascending order. The batch can be processed (applied to the words table) by calling T_BATCH_PROCESS_<table>_<column>, generated by the index creation.

The sensitive columns of the UPDATE trigger generated are the free text document id, the CLUSTERED WITH columns and the main text column. If more columns are needed for hook functions etc., the triggers should be manually edited.

If an explicit integer document ID column is specified, its value may not be 0 or negative.

See Also

vt_batch , vt_batch_feed , vt_batch_feed_offband , vt_batch_d_id .

20.2.6. Hit Scores

When a document satisfies a text search expression a score is computed to reflect the closeness of the match. This is a positive integer, with a higher value meaning a closer match. The scores are only comparable between results of the same query against the same database. Moreover, the scoring rules are different for different versions of Virtuoso server, due to the progress in information retrieval theory. Thus the only really safe thing to do with scores is to sort the list of hits by descending score to produce more relevant hits first. In addition, the CONTAINS predicate allows specifying a lower limit to the score of produced hits so a smart application can reduce a number of retrieved documents by filtering out the less relevant documents.

While the real scoring rules of the server are too sophisticated to be listed here, the basic concept is simple: hits for restrictive terms are most important than hits for generic terms, frequent hits in same document are more important than occasional, hits that are probably concentrated in same sentence are more important than hits here and there in a long text:

The score of a single word term is proportional to the frequency of the word in the document.

The score of a phrase is proportional to both the frequency of the phrase in the document and the number of words in the phrase.

The score of a proximity term 'A NEAR B' depends on the distance between A and B in words - greater distances result in smaller scores. E.g. the score is 100 for zero distance, 80 for 5 word distance, 50 for distance 10, 20 for distance 20; then it slowly decreases to 1 for distance 100. If the distance is 100 words or greater then the condition A NEAR B is considered not to be satisfied. If the A-B pair occurs several times throughout the document the scores of each pair are added. Virtuoso searches for shortest possible pairs, so if there is more than one word A around B then the nearest A is selected for scoring.

Score of a term like A AND B AND C... AND Z is a minimum of scores of A, B,... Z plus some bonuses. Longer list of AND'ed terms get higher score. A special bonus for term A AND B is added if hits of subterm A are close to the hits of subterm B.

Score of a term like A OR B OR C... OR Z is a maximum of scores of A, B,... Z, with some additional corrections. Longer list of OR'ed terms get lower score. A special bonus for term A OR B is added if hits of subterm A are close to the hits of subterm B.

Score of a term like A AND NOT B is equal to the score of subterm A. The subterm B is used for filtering out redundant hits but it does not affect the scoring.

The XCONTAINS predicate can also return scores. These scores are somewhat similar to scores made by CONTAINS but rules for them are too complicated to be explained here.

20.2.7. Word Ranges

This feature allows returning the positions of matches of a query inside the indexed text. This is done by creating a virtual column which gets as its value an array of arrays, one array for each top level term of the text query expression. The component arrays in turn contain word positions, expressed as the ordinal number of the first and last word of each match of the term in question. The ordinal numbers are counted from 0, skipping noise words.

Example 20.2. Word Ranges

```
select dbg_obj_print (r), * from ftt
        where contains (dt, 'foo', ranges, r);
```

Prints the array

```
(L(0 0 ) )
```

to the server standard output, indicating that the match starts and ends at the 0'th word, inclusive.

This feature can be used to show specific portions of matching documents in applications. This is internally used as part of the xcontains predicate for XML text. Also see the function search_excerpt ().

20.2.8. Using Offband Data for Faster Filtering

When evaluating a select where there is a contains predicate and filtering conditions on columns of the table on which the text index is defined it is useful to store the most frequently used columns in the free text index instead of the table itself.

The rationale is that in order to access the filtering data the engine will do a merge join with the text index table instead of a nested loop join with the actual table. Further note that if the columns to be accessed are not in the index that begins with the free text document id actually 2 random accesses will be needed: 1. to get the primary key based on the document id and 2. to get the filtering criterion based on the primary key. It is vastly more efficient to do a merge join in the text index to get frequently needed non-text filtering or sorting keys.

If the text index is maintained in background mode the offband data will also be maintained with a delay. This should not be a problem however since this is no more delayed than the text data itself.

If a select with a contains does not reference any columns from the indexed table besides the document id, then no access to the actual table will be generated in the compiled query. Likewise, if only columns found in the index used to link the document id to the table are referenced, only that index will be accessed. A special case of the latter situation is where the document id is the primary key itself. This will speed up retrieving the row for free text hits.

Let us consider a query for getting articles where the author name is at a specific value:

```
select id from article where contains (description, 'sample')
        and author_name = 'John Pumpkin';
```

This can be alternately written as

```
select id from article
        where contains (description, 'sample', offband, author_name)
        and author_name = 'John Pumpkin';
```

In the latter case the author_name will be retrieved from the text index, saving 2 random accesses, one to the index on dtid and the other to the table itself per each free text hit.

The notation is different because the semantic is slightly different. The author_name in the latter case is the name at the time of indexing the article and in the latter case it is the name at the time of evaluating the query. There can be a difference if the index is maintained with a delay. For most applications this is however irrelevant. offband data should not be used for often changing, transactional data.

Now consider

```
select * from (select top 10  dtid from article
        where contains (description, 'sample', offband, author_name)
        order by author_name) a, article b where a.dtid = b.dtid;
```

The derived table select the 10 first articles matching the text condition in order of author_name. This does not itself access the article table at all. The outer select will then select the full row for these 10 articles. This is possible since the inner select only references dtid, which is the free text index document id and author_name which is declared an offband column in the text index.

This does less disk access than

```
select top 10 * from article
        where contains (description, sample', offband, author_name)
        order by author_name;
```

In this case, all matches are fetched, including the row in the article table and all are sorted and the top 10 are returned. This differs from the first by the fact that this accesses the article table for each of the text hits, not only the top 10. This can easily make a 3x speed difference when running in memory and much greater when I/O is involved, not to mention the adverse impact of more I/O on the working set.

20.2.9. Order of Hits

Rows from a select where there is a contains predicate and no exact match of the primary key of the table are produced in the order of the document id of the free text index if there is no ORDER BY. If there is an ORDER BY matching an index, the query is evaluated in the order of that index and each consecutive row in the order of the index is compared against the free text expression. This is practically never desirable.

If results are desired in the order of the free text document id, there MUST BE NO ORDER BY. The DESCENDING option of contains should be used to produce the reverse order, see contains reference section. If another sorting order is desired, an ORDER BY can be used but to avoid using an index the ordering columns should be expressions or ordinal numbers of result columns.

Therefore:

```
select * from article where contains (description, 'sample')
  order by id;
```

Will have the effect of traversing the table in the order of id and checking each row for free text match. This is practically NEVER good.

To produce the results in order of id instead of dtid it is better to write

```
select id, * from article where contains (description, 'sample')
  order by 1;
```

or

```
select * from article where contains (description, 'sample')
  order by id + 0;
```

20.2.10. Noise Words

Noise words are often occurring words which can be skipped to save space in the indexing, such as 'the', 'of', and' etc. These are ignored when they occur in queries or documents to be indexed. The set of noise words is configurable and is read from the noise.txt file, in the server's working directory, at server start up. Words mentioned in that file will be ignored for both indexing and querying.

The file noise.txt consists of control lines and text lines. A text line is just a string of one or more words to be declared as noise. Please keep them shorter than 1000 characters.

Control lines are those starting with "Language:" or "Encoding:" (case is important) "Language: lang-id" tells the system to use rules for language "lang-id" for subsequent text lines, until either another "Language:" control line or end of file. Similarly, "Encoding: enc-id" tells the system to use rules for encoding "enc-id". Control lines are always in plain ASCII, no matter which encoding is active for text lines. By default, the server default language and "UTF-8" encoding will be used.

The simplest way of composing noise.txt is to place every word on a separate line and save the file in UTF-8 encoding; this will work fine for most European languages.

Example 20.3. Example

```
AND
OF
THIS
THE
```

Noise words seem to be case-insensitive, but this is not so. If you enter a word on a text line, up to four noise words will be registered:

the word exactly as it was entered;

an uppercased form of this word, if it is defined for active language;

an lowercased form of this word, if it is defined for active language;

a capitalized form, with one (or more) first chars in upper case and the rest in lower case.

An error is signalled for a free text query consisting exclusively of noise words.

It is important to understand that changes in noise.txt may invalidate free text indexes that were filled with the previous version of noise.txt. For example, if a text contains a phrase 'A B C' and after indexing the word 'B' is added to the noise.txt then 'contains' predicate will properly search for words 'A' and 'C' but will fail to find the phrase 'A B C' or 'A C' due to differences in counting of word positions. The 'xcontains' predicate is even more sensitive to changes in word positions, because any change in word counting will corrupt the index for element names. In addition, "persistent XML" documents may contain pre-calculated word positions for all elements and these positions may become out of sync with positions in free text index, so it is best not to change noise.txt if the database contains any free text indexes on persistent XMLs.

20.3. Querying Free Text Indexes

20.3.1. CONTAINS predicate

Returns TRUE if a free text indexed column matches a text expression.

Syntax

```
contains_pred:
        contains (column, expr, opt_or_value ....)

opt_or_value:
          DESCENDING
          | START_ID ',' scalar_exp
          | END_ID ',' scalar_exp
          | SCORE_LIMIT ',' scalar_exp
          | RANGES ',' variable
          | OFFBAND column

variable: IDENTIFIER
```

The *column* must refer to a column for which there exists a free text index. The *expr* must be a narrow or wide string expression whose syntax matches the rules in 'Text Query Syntax'. The *START_ID* is the first allowed document ID to be selected by the expression in its traversal order, e.g. least or equal for ascending and greatest or equal for descending. *END_ID* is the last allowed ID in the traversal order. For descending order the START_ID must be >= END_ID for hits to be able to exist. For ascending order the START_ID must be <= END_ID for hits to be able to exist.

DESCENDING specifies that the search will produce the hit with the greatest ID first, as defined by integer or composite collation. RANGES specifies that the query variable following the RANGES keyword will be bound to the word position ranges of the hits of the expression inside the document. The variable is in scope inside the enclosing SELECT statement.

SCORE_LIMIT specifies a minimum score that hits must have or exceed to be considered matches of the predicate.

OFFBAND specifies that the following column will be retrieved from the free text index instead of the actual table. For this to be possible the column must have been declared as offband with the CLUSTERED WITH option of the CREATE TEXT INDEX statement.

See Also:

`contains()`

The XCONTAINS Predicate .

20.3.2. Comments

Order - If the select statement containing the contains predicate does not specify an exact match of the primary key of the table having the contains predicate, then the contains predicate will be the 'driving' condition, meaning that rows come in ascending or descending order of the free text document ID.

The *DESCENDING* keyword specifies the descending order of the free text index document ID and has nothing to do with a possible ORDER BY of the enclosing statement. Even if there is an ORDER BY in the enclosing statement the DESCENDING keyword of contains has an effect in the interpretation of the STRT_ID and END_ID contains options.

If there is a full equality match of the primary key of the table, this will be the driving predicate and contains will only be used to check if the text expression matches the single row identified by the full match of the primary key.

The contains predicate may not appear outside of a select statement and may only reference a column for which a free text index has been declared. The first argument must be a column for which there is such an index. The text expression may be variable and computed, although it must be constant during the evaluation of the select statement containing it.

The contains predicate must be a part of the top level AND of the WHERE clause of the containing select. It may not for example be a term of an OR predicate in the select but can be AND'ed with an OR expression.

20.3.3. Text Expression Syntax

```
expr ::= proximity_expr
        expr AND expr
        | expr OR expr
        | expr AND NOT  expr
        | '(' expr ')'

word_expr ::=
          word
        | '"' phrase '"'

proximity_expr ::=
          word_expr
        | proximity_expr NEAR word_expr

word ::=
        <word char>*

phrase ::=
          word
        | phrase <whitespace> word

word_char ::=  alphanumeric characters, '*',  ISO Latin accented characters.
```

A word is a sequence of word characters. A phrase is a sequence of words separated by white spaces and enclosed in double quotes. If a word contains a wildcard character it must be quoted with double quotes.

Note:

An expression may not consist of all negative terms, e.g. (not a) and (not b) is not a valid expression but 'c and not a and not b' is a valid expression.

Note that the NEAR connective may not be used between AND'ed or OR'ed terms. It can be used to combine words or phrases.

Example 20.4. Querying Free Text Indexed Columns

```
select count (*) from docs
where contains (text, '"virtual database"')
```

returns the count of documents with one or more occurrences of "virtual" immediately followed by "database".

```
'performance and (tuning or optimization)'
```

specifies documents with performance and either 'tuning' or optimization' in any respective positions.

```
'graphics and not (graphics near user near interface)'
```

matches documents with the word graphics more than 100 words away from 'user' or 'interface'.

```
'"sql interfac*"'
```

matches documents with SQL followed by a word beginning with 'interfac'.

```
'"dragon*" and not "once upon a time"'
```

matches documents with words beginning with 'dragon' and not containing the phrase 'once upon a time'.

20.4. Text Triggers

The text trigger mechanism allows implementing a broad range of content tracking functionality. The idea is storing free text or XPATH queries in association to a text indexed column of a table. When the content of the table changes through inserts or updates, the new data is matched against a base of stored queries and hits are marked into a separate table. The data being tracked may either be plain text or XML. In the event of XML, both free text and XPATH queries can be stored.

The benefit of the text trigger system as opposed to other forms of periodic content tracking is that the incoming data itself indexes a base of stored queries instead of a base of stored queries repeatedly indexing the database. This means that only the changes are compared to the stored queries and that queries that could not even in principle match will not be tried. This results in a qualitatively better performance and scalability than repeatedly running a batch of queries over updated data and thus makes possible personalized information filtering applications that would be impractical with other approaches.

20.4.1. Creating Text Triggers

The **CREATE TEXT TRIGGER** statement creates a set of tables and procedures named after the table and column being watched. The **TT_QUERY_<xx>** table contains the set of queries, the **TT_HIT_<xx>** table records the matches and the TT_USER_<xx> table can be used to map stored queries to specific users that should be notified.

Syntax:

```
CREATE TEXT TRIGGER ON <table> [(<data_column>)]
DROP TEXT TRIGGER ON <table> [(<data_column>)]
```

The <data_column> is optional and must be a text indexed column.

The table to be watched by a text trigger should have a free text indexed column. This may or may not be XML data.

The <table> must be text indexed before creating text trigger on it.

Example 20.5. Creating a Text Trigger

```
create table ftt (id integer not null primary key, dt long varchar);
create text xml index on ftt (dt);
create text trigger on ftt;
```

Adding queries to the text trigger

```
TT_QUERY_ftt ('virtuoso and server and international',
  1, 'Virtuoso international support', 'iam@foo.bar');
```

this adds a query which will filter only documents matching words 'virtuoso', 'server' and 'international', named 'Virtuoso international support' of the user with Id equal to 1 with e-mail notification.

```
TT_XPATH_QUERY_ftt ('/chapter[@label = ''XI'']',
  2, 'Chapter XI changes', '');
```

this will add an XPATH query which will filter only XML documents matching Chapter XI, named 'Chapter XI changes' of the user with Id equal to 2 without e-mail notification.

```
insert into ftt values (1, 'virtuoso server international');
insert into ftt values (2, 'virtuoso international');

select TTH_D_ID, TTH_T_ID, TTH_U_ID, TTH_TS from ftt_dt_HIT;
```

will produce

```
TTH_U_ID    TTH_D_ID         TTH_T_ID    TTH_TS
_____________________________________________

1           1                1           2001-01-17 12:35:30
```

Meaning that row from 'ftt' with id equal to 1 matches query with TT_ID equal to 1 defined for user with ID equal to 1. Also the hit is registered on '2001-01-17 12:35:30'.

20.4.2. Created Database Objects

Text trigger hits table. Text trigger stores hits on documents matching a condition within this table.

```
<table_name>_<data_column>_HIT (
  TTH_U_ID INTEGER,   -- references User table TTU_U_ID
  TTH_D_ID ANY,       -- references matching document id
  TTH_T_ID INTEGER,   -- references matching query TT_ID
  TTH_TITLE VARCHAR,  -- user application specific
  TTH_URL VARCHAR,    -- user application specific
  TTH_TS TIMESTAMP,   -- time of registering a hit
  TTH_NOTIFY VARCHAR, -- e-mail address for notification
  PRIMARY KEY (TTH_U_ID, TTH_TS, TTH_D_ID, TTH_T_ID)
)
```

Text trigger queries table, where the query definition procedure stores the user specific queries.

```
<table_name>_<data_column>_QUERY (
  TT_WORD VARCHAR,  -- the most effective word for searching
  TT_ID INTEGER,    -- id of query
  TT_QUERY VARCHAR, -- text of the query, in case of XPATH query this
                               -- column contains a serialized value
  TT_CD VARCHAR,    -- user data
  TT_COMMENT VARCHAR, -- Human readable label with general purpose
  TT_XPATH VARCHAR,   -- text of XPATH query
  TT_PREDICATE VARCHAR, -- Virtuoso/PL function hook
  PRIMARY KEY (TT_WORD, TT_ID)
)
```

One query can add one or more rows to the queries table.

Text trigger users table, in it query definition procedure add a reference between query and user.

```
<table_name>_<data_column>_USER (
  TTU_T_ID INTEGER, -- references text query TT_ID
  TTU_U_ID INTEGER, -- references application specific users table ID
  TTU_NOTIFY VARCHAR, -- e-mail address list for notification
  TTU_COMMENT VARCHAR, -- Human readable label of query definition.
  PRIMARY KEY (TTU_T_ID, TTU_U_ID)
)
```

Note:

Note:

In the case of e-mail notification to the main addresses the string passed as address list must be in the following format: '<u1@foo.com>, <u2@foo.bar>'.

Procedures that are used for adding text queries.

```
TT_QUERY_<table_name> (in query_text varchar, in user_id integer,
  in label varchar, in notification_address varchar)
```

Used for adding XPATH queries.

```
TT_XPATH_QUERY_<table_name> (in xpath_query_text varchar , in user_id integer,
in label varchar, in notification_address varchar)
```

Used to send notifications to the users. This procedure is usually called by the server event scheduler.

```
TT_NOTIFY_<table_name> ();
```

20.5. Generated Tables and Internals

20.5.1. Generated Tables and Procedures

vt_create_text_index makes a separate table for storing the text index information and separate procedures and triggers for maintaining this data. These are automatically dropped if the original table is dropped.

The updates to the indexed column are recorded in a separate update tracking table. This table, named VTLOG_<qualifier>_<owner>_<table> contains a row for each row in the indexed table that has been changed since the text index was last updated.

The text index is stored in a table named <table>_<column>_WORDS. The generated tables are made under the qualifier that is current at the time of their creation. The owner is the creating user.

20.5.2. The procedures are:

```
VT_INDEX_<qualifier>_<owner>_<table>  (in flag integer)
```

This re-indexes the table. A flag of 0 makes the index, a flag of 1 deletes data found in the table from the index.

```
VT_INC_INDEX_<qualifier>_<owner>_<table>  (in flag integer) ()
```

This function refreshes the index using the change tracking information in the VTLOG_ table.

20.5.3. Tables and Procedures Created By Text Triggers

```
- [TARGET_TABLE_NAME]_[DATA_COLUMN_NAME]_HIT

  columns

  TTH_D_ID          INTEGER   - document ID, references unique id of [TARGET_TABLE]
  TTH_T_ID          INTEGER   - query ID, references QUERY table
  TTH_U_ID          INTEGER   - user ID, references USER table
  TTH_TS            TIMESTAMP - date and time of retrieval
  TTH_NOTIFY        VARCHAR   - e-mail address of user for notification
  TTH_TITLE         VARCHAR   - not used (can be filled with user-defined trigger)
  TTH_URL           VARCHAR   - not used (can be filled with user-defined trigger)

- [TARGET_TABLE_NAME]_[DATA_COLUMN_NAME]_QUERY

  columns

  TT_ID             INTEGER   - Unique ID of query
  TT_QUERY          VARCHAR   - query text
  TT_WORD           VARCHAR   - the best word for query
  TT_COMMENT        VARCHAR   - Description
  TT_CD             VARCHAR   - user data (not used)
```

```
  TT_PREDICATE     VARCHAR   - not used
  TT_XPATH         VARCHAR   - XPATH expression.  If specified,
                                        the test is this XPATH predicate,
                                        see separate section.

- [TARGET_TABLE_NAME]_[DATA_COLUMN_NAME]_USER

  columns

  TTU_U_ID         INTEGER - unique user ID can reference SYS_USERS.
  TTU_T_ID         INTEGER - query ID, references QUERY table
  TTU_COMMENT      VARCHAR - Description
  TTU_NOTIFY       VARCHAR - e-mail address of user for notification
```

Note that the queries are available given a table wide query identifier and an 'entry point' word. Such a word is a word that must occur in the document in order for the query to have a possibility of matching the document, The query text and other attributes are denormalized so that the primary key is the word, id pair when the id itself is unique. Note that in the case of an AND of words, the least frequent of the words will be used as unique entry point of the query, so that it will not be tried on documents that do not contain this word. However, a query with OR'ed terms may have several such words, hence the possibility of multiple rows in the query table for the same query.

The TT_USER_<xx> table maps from the query to a user. The idea of this is to allow a single query to have multiple users. Consider an application which allows creating personalized information filtering profiles. It is to be expected that multiple users would store the same profiles. Therefore the link between the user and the query is entitized as this table. The user specific comment and notification mode are thus stored here, not with the query. The notification mode itself is application dependent. The user id is an application dependent id that can be used to reference application user entities. Some applications may use this whereas other applications will have all queries on a single user.

When a hit is noticed an entry is made into the TT_HIT_<xx> table. One row is inserted for each unique document id, query id, user id combination for which the document matches the query and there is a link to a user from the query through TT_USER_<xx>. A query with no row in TT_USEER_<xx> is an integrity error. The number of times the pattern is found in each document or its free text hit score has no effect on the hit insertion.

One may note that defining application specific triggers on the hit table can be used to add immediate application reactions to incoming data.

The free text triggers are matched against the new content immediately before the content is inserted into the free text index. Therefore the batch mode setting affects the time of matching. In all situations, the matching takes place after the data is inserted but before the free text index is updated. If text index maintenance is in synchronous (non-batched) mode, the text trigger match and hit generation is in the same transaction as the update to the content being watched.

Procedures

For queries definition

```
tt_query_[TARGET_TABLE_NAME] (
            in [query text] varchar,           - query expression
            in [user_id] integer,              - user id references SYS_USERS or SYS_DAV_USER
            in [comment] varchar,              - description
            in [e-mail or empty] varchar)      - e-mail address for user notification
```

or hits registration (used inside text index procedures)

```
vt_hits_[TARGET_TABLE_NAME] (inout [batch] any, inout [words array] any)
```

Triggers

for hits removal after document delete occurred

```
[TARGET_TABLE_NAME]_FTT_D
```

Examples

```
-- create a table
create table T1 (id integer, dt varchar, primary key (id));

-- define text index
create text index on T1 (dt);

-- create text trigger
create text trigger on T1;

-- define an query
tt_query_T1 ('xyz and abc', 1, 'This is a test query', null);

-- do some inserts
insert into T1 (id, dt) values (1, 'xyz');
insert into T1 (id, dt) values (2, 'xyz abc');
insert into T1 (id, dt) values (3, 'abc');
update T1 set dt = 'xyz qwe abc' where id = 2;

select TTH_TS, dt from T1, T1_dt_HIT where id = TTH_D_ID order by TTH_TS desc;
-- produces following
TTH_TS      dt
BINARY      VARCHAR
_______________________________________________________________________________

2000-10-24 18:25:53  xyz qwe abc
2000-10-24 18:25:53  xyz qwe abc
```

20.6. Removing A Text Index

A text index is dropped by dropping the words table with DROP TABLE. This will drop all triggers, procedures and auxiliary tables. The words table is in the qualifier and owner of the indexed table and is named <table>_<column>_WORDS.

Example 20.6. Example

```
drop table DB.DBA.XML_TEXT_XT_TEXT_WORDS;
```

-- drops the text index created in the vt_create_text_index example

20.7. Removing A Text Trigger

Used to drop text trigger definition on text indexed table. The operation also drop all tables created by create text trigger statement.

Syntax:

```
DROP TEXT TRIGGER ON <table> [(<data_column>)]
```

Example 20.7. Removing A Text Trigger

```
drop text trigger on ftt;
```

will drop the text trigger definition from table ftt.

Or using the stored procedure:

20.7.1. vt_drop_ftt_dedup

For detailed description and example use of the function, see vt_drop_ftt in the Functions Reference Guide.

20.8. Internationalization & Unicode

The text being indexed and the text query expression may both be wide strings. The word boundaries used to cut the text in words in both queries and index maintenance may depend on a language declared for the text index.

The default language has white space and punctuation as word delimiters and will recognize Unicode ideographic characters as self standing. A single non-ideographic character will always be considered noise and not indexed.

Non-ASCII Unicode values are converted to UTF8 before being stored into the word table as narrow strings. Narrow 8 bit strings are stored in the words table as is.

See Also:
The LANGUAGE option in CREATE TEXT INDEX .

20.9. Performance

For indexing large volumes it is critical to run the indexing process over large volumes of data. This is accomplished by using the batch update mode. This is activated with the VT_BATCH_UPDATE procedure.

When this mode is on the index will be updated after a settable interval, doing all the updates accumulated since the last batch in a single go. The size of a single batch is configurable in the virtuoso.ini file using the FreeTextBatchSize option. This is the number of bytes of text which will be processed as one sweep over the index. A value of 1MB is often suitable. Even All changes are processed in a batch at the set interval. If there are more characters of text to index / unindex than the batch size, the operation is broken into several transactions, each processing about FreeTextBatchSize bytes worth of text. This improves concurrency and cuts down on locking.

The command

```
DB.DBA.vt_batch_update ('DB.DBA.ARTICLE', 'ON', 1);
```

turns on the batch mode for the article table with a 1 minute delay between index refreshes. The table name must be fully qualified and is case sensitive. The correct case is seen in the administration interface tables list of the isql tables command etc.

20.9.1. Restrictions

If the free text document ID is an integer, which is encouraged for compactness, the values 0 and negative are reserved.

20.10. Free Text Functions

20.10.1. vt_batch_dedup

For detailed description and example use of the function, see vt_batch in the Functions Reference Guide.

20.10.2. vt_batch_d_id_dedup

For detailed description and example use of the function, see vt_batch_d_id in the Functions Reference Guide.

20.10.3. vt_batch_feed_dedup

For detailed description and example use of the function, see vt_batch_feed in the Functions Reference Guide.

20.10.4. vt_batch_feed_offband_dedup

For detailed description and example use of the function, see vt_batch_feed_offband in the Functions Reference Guide.

20.10.5. vt_batch_update_dedup

For detailed description and example use of the function, see vt_batch_update in the Functions Reference Guide.

20.10.6. vt_is_noise_dedup

For detailed description and example use of the function, see vt_is_noise in the Functions Reference Guide.

20.11.

Text index log table. In case of batch update mode this table is used to store a log of actions over text indexed table.

```
VTLOG_<table> (
  VTLOG_<document_id_col> ANY NOT NULL PRIMARY KEY,
                        -- references text indexed table by document id
  SNAPTIME DATETIME,
                        -- time of insert/update/delete action
  DMLTYPE VARCHAR (1),
                        -- type of log 'I' 'U' 'D' for insert, update or delete
  VT_DISTINCT_WORDS LONG VARBINARY,
                        -- in case of update or delete are stored words which should be deleted
  VT_OFFBAND_DATA LONG VARCHAR
                        -- in case of update or delete are stored offband data should be deleted
)
```

Text index batch procedure. This procedure log, process and stores already filled _vt_batch, caused storing of words in index table.

```
VT_BATCH_PROCESS_<table> (inout _vt_batch any)
```

Text trigger hook function. This function, for existing text trigger definition, applies the filtering queries (if defined) to the _strings, and if the _vt_batch matches any of them then add new record in HITS table (see Text trigger)

```
VT_HITS_<table> (inout _vt_batch any, inout _strings any);
```

Text index words table. This table maintains distinct words collected from all documents from text indexed table.

```
<table>_<column>_WORDS (
  VT_WORD VARCHAR,  -- distinct word
  VT_D_ID ANY,  -- referencing the first matching
                        -- document id in text indexed table
  VT_D_ID_2 ANY,  -- referencing the last matching document
                        -- id in text indexed table
  VT_DATA VARCHAR,  -- string with document id's and word
                        -- positions where the word is matched
  VT_LONG_DATA LONG VARCHAR, -- the same as VT_DATA but in a
                        -- case of large amount of data
  PRIMARY KEY (VT_WORD, VT_D_ID)
)
```

Text indexing procedure, using for clearing and creating the text index. If flag is equal to 1 then index data will be cleared, else if equal to 0 then index data will generated. Note that for large tables generating the index can run for a long time and the server will go into atomic mode.

```
VT_INDEX_<table> (in _flag integer)
```

Procedure for incremental update of text index. In case of batch mode update this procedure must be called to process the entries in log table and refresh text index data. Before calling this function the contains/xcontains predicate may not match the newest inserted, updated or deleted documents. This function is also registered for scheduled action if the vt_batch_update() function is called with refresh interval greater than zero. (See vt_batch_update function)

```
VT_INC_INDEX_<table> ()
```

Text indexing and unindexing hook procedures. These are user-defined procedure which can access additional related data and perform preprocessing and call vt_batch_feed inside. These procedures receive the free text id as argument and can use this to retrieve data related to the row being indexed. If the text index is created with 'USING FUNCTION' clause then internally generated procedures and functions will include calls to these. If these procedures return 1 then the caller will skip filling a vt_batch, assuming the hook function already filled it, otherwise the caller will proceed as if there had been no hook.

The difference between these two functions is that ..._INDEX_HOOK will be called upon insertion of new data or after update, but .._UNINDEX_HOOK will be called after delete or before update on the text indexed table.

Note:

This function is USER-DEFINED, the user can create it before or after creating a text index.

In the second case the text index MUST be create with the 'NOT INSERT' option. After the hook is defined, the index can be filled with with 'VT_INDEX_<table>(0)' procedure.

```
<table>_<column>_INDEX_HOOK (inout _vt_batch any, inout d_id any)
<table>_<column>_UNINDEX_HOOK (inout _vt_batch any, inout d_id any)
```

Example 20.8. Free Text

```
create table fth (id integer not null primary key, dt varchar, c1 varchar);

create procedure fth_dt_index_hook (inout vtb any, inout d_id integer)
{
  declare data any;
  data := coalesce ((select concat (dt, ' ', c1)
        from fth where id = d_id), null);
  if (data is null)
    return 0;
  vt_batch_feed (vtb, data, 0);
  return 1;
}

create procedure fth_dt_unindex_hook (inout vtb any, inout d_id integer)
{
  declare data any;
  data := coalesce ((select concat (dt, ' ', c1)
        from fth where id = d_id), null);
  if (data is null)
    return 0;
  vt_batch_feed (vtb, data, 1);
    return 1;
}

insert into fth values (1, 'abc', 'one');

create text index on fth (dt) with key id using function;
```

test the text index:

```
select id from fth where contains (dt, 'abc');
select id from fth where contains (dt, 'one');
```

Both select statements will return 1 because the content is concatenated with an additional column.

Note that in the case of using additional columns, they should be added as offband data to the text index, otherwise update them will not affect the index.

```
<table>_<column>_QUERY table (see Text trigger definition)
<table>_<column>_USER table (see Text trigger definition)
<table>_<column>_HIT table (see Text trigger definition)
TT_QUERY_<table>_<column> procedure (see Text trigger definition)
```

Table for text indexes system information

```
SYS_VT_INDEX (
  VI_TABLE VARCHAR,  -- Fully qualified text indexed table name
  VI_INDEX VARCHAR,  -- Index name
  VI_COL VARCHAR,  -- Data column name
  VI_ID_COL VARCHAR,  -- Document id column name
  VI_INDEX_TABLE VARCHAR,  -- fully qualified name of table with words
                           -- (See: <table>_<column>_WORDS table)
  VI_ID_IS_PK INTEGER,  -- if the document id is specified by user or
```

```
                        -- used primary key then it equal to 1
VI_ID_CONSTR VARCHAR,  -- serialized value with id column(s) names
VI_OFFBAND_COLS VARCHAR,  -- serialized value of offband data columns names
VI_OPTIONS VARCHAR,  -- reserved
VI_LANGUAGE VARCHAR,  -- language which applied to the document contents
PRIMARY KEY (VI_TABLE, VI_COL))
```

Chapter 21. TPC C Benchmark Kit

Abstract

The Virtuoso TPC C Kit is a database benchmark written in C and SQL stored procedures using the CLI or ODBC API. It is modeled after the industry standard Transaction Processing Performance Council (TPC) C benchmark and can be used as a component in an official benchmark but does not in itself constitute a complete test driver.

See Also:

A complete description of the benchmark and its metric can be found at the TPC Web Site . This document assumes the reader is familiar with the general content of the benchmark.

The C and SQL source of the benchmark is contained in the sample directory of the Virtuoso tree. The tpcc executable is built by running make after choosing the target (see makefile).

21.1. Building the Test Database

To build a 1 warehouse test database (approximately 100 MB), go through the following procedure:

Start the database server.

Assuming the server is listening at the default port of 1111 on the local host execute:

```
isql 1111 dba dba tpccddk.sql
tpcc 1111 dba dba I 1
```

to create a 1 warehouse database. This may take some time. As long as the file gets longer everything is OK. You may follow the progress with interactive SQL.

Once the tpcc program exits you can load the stored procedures used by the benchmark. Assuming the server is listening at the default port of 1111 and that the dba password is the default `dba', type:

```
../isql 1111 dba dba tpcc.sql
```

This will exit when the procedures are loaded, typically a few seconds.

To complete the initialization make a checkpoint to freeze the initial database state:

```
./isql 1111
SQL> checkpoint;
Done.
SQL> exit;
```

The database is now ready for use.

21.2. Using the Test Program

The tpcc program simulates one user making random transactions according to the specified mix of:

```
10 new order transactions, each with 10 order lines
10 payment transactions, 60% by name
1 delivery transaction
1 stock level transaction
1 order status transaction, 60% by name.
```

Each instance of the test program has a home warehouse on which it does most of its operations. If there are more than one operation the test program will give a supply warehouse different from the local warehouse to 10% of new order lines.

The test is started with:

```
tpcc <database> <username> <password> r
        <n-10-new-order-sets> <local warehouse> <n warehouses>
```

e.g.

```
./tpcc 1111 dba dba r 10
```

for 10 sets of the transaction mix on warehouse 1 in a 1 warehouse database or,

```
./tpcc 1111 r 100 12 100
```

to perform 100 sets of transactions on warehouse 12 in a 100 warehouse database.

The program reports a tpmC rate after performing each set of 10 new orders and the related support transactions. A statistic will be printed every 10 sets of 10 new orders showing the time spent on each of the benchmark transactions done during the last run of 100 new orders, 100 payments 10 deliveries, stock levels and order status queries.

The throughput will increase during the first minutes of the run to level off at the attained rate.

21.3. Tuning Parameters and Number of Users

You may run several instances of tpcc, each representing one user. You will see CPU utilization improve as users are added since there are more possibilities of interleaving I/O and CPU.

The amount of RAM (number_of_buffers: in wi.cfg) is the single most important factor influencing throughput. Setting this to about half the system RAM is usually good. One will remember that each buffer takes 8.5K of actual RAM. One should be careful not to cause the server process to swap.

Striping should be used if there are multiple independent disks, one stripe per physically independent volume. Each stripe should have its own I/O queue. If there is a RAID, then striping is less beneficial. Also one should have multiple handles per files, see FDSPerFile in the configuration file.

21.4. Omissions, Exceptions from the Definition

Running the benchmark by the book is a complex and costly process which requires hardware and software that is not commonly available.

To measure tpmC rates that are directly comparable with published figures the benchmark must comply with the scaling rule of a maximum of 12.5 tpmC per warehouse. Therefore to measure 1250 tpmC, one must have a database of 100 warehouses, approximately 10 GB.

Obtaining a good figure will require the maximum RAM configuration of the platform in question.

One will further remember the 180 day rule which states that the disk configuration quoted in the pricing section must be large enough to accommodate 180 8 hour days worth of new orders coming in at the reported rate.

The number of users will also be large, e.g. 1000 users for the 1200 tpmC result, based on 10 terminals per warehouse.

Almost all published results have been obtained with a transaction monitor, typically Tuxedo.

Note that the driver in this kit initiates a new transaction as soon as the previous one is completed. The correct behavior is to generate transactions at a basically constant rate, load being increased by adding clients and a new warehouse every 10 clients.

See Also:

The complete benchmark specification is available at the TPC Web Site .

21.5. Sample Configuration

This section describes how to set up disks and I/O for a sample run. To begin with, the scaling rule is 12.5 tpmC per warehouse. This means that in order to measure 1000 tpmC you must have a 1000 / 12.5 = 81 warehouses. These take about 100 MB apiece.

The benchmark's working set consists of the STOCK and CUSTOMER tables of each warehouse and of the ITEM table of the database. Other tables are accessed more or less sequentially, i.e. inserts to end or deletes from start. There is a particular distribution of hits for the STOCK and CUSTOMER rows of each warehouse, leading to a specific working set within each.

The 160 day rule requires a disk configuration sufficient for accumulating 160 days worth of transactions at the reported rate. For practical reasons we will ignore this rule here. To just run the benchmark for the required 20 minutes we will need about twice the space of the initial data. Let's assume we have an initial database of 8 GB and have another 16 GB for working space, a total of 24 GB. This is 6 4 GB disks or 12 2 GB ones.

Let's now look at the relationship between CPU and disk usage. The New Order transaction, which mostly defines the benchmark's working set consists of an average of 10 updates to the STOCK table, which mostly cause disk reads, 10 reads of the ITEM table which is always in cache, 10 ascending ORDER_LINE inserts, 1 ascending ORDERS insert and 1 CUSTOMER read plus a DISTRICT update and WAREHOUSE read.

If this happens entirely in RAM this takes about 10 milliseconds on a 200 MHz Pentium Pro, 13 on an Ultra SPARC and so on. Which is roughly as long a one random disk seek takes.

Because the scaling rule limits RAM to cover only 10% to 20% of the working set, the STOCK updates will miss the cache most of the time. This with a transaction mix of New Orders only, we would need about 8 disk seeks to be in progress concurrently in order to feed one CPU. The other transactions are either more local or repeat the New Order access profile. Thus we end up needing about 5 concurrent 10 msec disks for one Power PC 604 and almost double for a 200 MHz Pentium Pro.

For our 24 GB configuration we may as well divide it over as many stripes as we have disks. For 6 4 GB disks, we would have:

```
[Striping]
Segment1 = 24G,  /disk1/tpcc-1-1.db = q1, /disk2/tpcc-1-2.db = q2,  /disk3/tpcc-1-3.db = q3,  /disk4/tpcc
```

Assuming we have file systems. Note the I/O queue names q1...q6, meaning that each stripe gets processed on a separate thread for I/O.

See

Disk Configuration for more on this.

Your Virtuoso may or may not support raw devices. If it does, they are specified here.

For the other configuration parameters, we will have sufficient RAM configured for the DBMS, counting 8.5K of RAM for each buffer. For 512 MB of RAM, we would have about 25000 buffers. The maximum dirty parameter is more tricky. A low number causes unnecessary writing and a high value causes the number of clean buffers at any time being too low, causing an imperfect match of the read working set. The read-only set is only the ITEM table, about 20 MB.

Note:

Note that about half of the available RAM can efficiently be allocated to the database, allocating all RAM may result in swapping due to the OS's disk caching. In terms of kernel tuning, if one can decrease the OS cache, one may increase the RAM utilizable by the DBMS without the OS's disk cache getting in the way.

The Stock Level transaction reads lines written by recent New Order transactions, which are likely to be in RAM and still dirty. The reads and updates of CUSTOMER are random and generally speaking what is read is also likely to be or have been updated.

We could start with a guess of 70% maximum dirty, i.e. a value of 70000 for 100000 buffers.

The checkpoint remap should be as large as possible and the unremap quota should be small. The benchmark does practically no sequential reads and therefore does not care about disk locality. No limit on remapping means that a checkpoint can be made in the time it takes to flush the disk cache. This is done in the background, so that the atomic checkpoint time is limited to the time it takes to write out those buffers that became dirty while the first sweep was in progress.

Thus we could have:

```
MaxCheckpointRemap = 2000000
UnremapQuota = 3000
```

21.6. Other Factors

Benchmarks are run with a transaction monitor, usually Tuxedo. This has not been discussed here. Multiprocessor questions have not been addressed either. Virtuoso off the box should scale to about 4 CPU's on any appropriate multithreaded, multiprocessor OS. Past 4 CPU's the returns will diminish.

Operating systems have different caching policies which must be taken into account. If an OS does read ahead, that's OK. Generally OS intelligence is harmful and should be turned off. For example, AIX reacts to its disk write queue being full by turning off the writing process until it has flushed enough of its own file cache. This instead of blocking the writing thread and leaving the rest of the process to run.

We may release more information on OS tuning in the future.

21.7. TPC C Procedures

21.7.1. Introduction

This document goes through the TPC C sample and explains how and why the transaction procedures are written as they are.

This describes the five transactions in the benchmark and points out how the procedures are written and which features of the language are used where. See the sample/tpcc.sql and sample/tpcctrx.c files along with this commentary.

See Also:

For a formal benchmark definition, see the documentation at the TPC Web Site .

21.7.2. New Order

- Passing parameters
- Using positioned operations
- Using read for update
- Order of locking
- Application-defined SQL STATE

The new_order procedure implements this transaction. It accepts the warehouse, district and customer data and the item, quantity and supply warehouse id for up to ten order lines.

The transaction profile requires this to update the stock level for each order line, add a row to ORDERS and NEW_ORDER and a row to ORDER_LINE for each order line. This also reads the customer, updates the district and reads the warehouse. This all needs to take place as one transaction with a high integrity requirement.

The procedure begins by updating the stock levels. This is the part with lowest locality and thus most likely to cause I/O and least likely to cause lock contention. Most of the transaction's real time will be spent inside ol_stock. If the order lines are sorted in order of item id, new order transactions will never deadlock on the stock level part. This will maximize the number of concurrent new orders on one warehouse.

```
CREATE PROCEDURE OL_STOCK (
        INOUT   OL_I_ID          INTEGER,
        IN      OL_SUPPLY_W_ID   INTEGER,
        IN      OL_QTY           INTEGER,
        OUT     AMOUNT           FLOAT)
```

```
{
    IF (OL_I_ID = -1) RETURN;

    DECLARE S_DATA,
        S_DIST_01, S_DIST_02, S_DIST_03, S_DIST_04, S_DIST_05,
        S_DIST_06, S_DIST_07, S_DIST_08, S_DIST_09, S_DIST_10 VARCHAR;
    DECLARE S_QUANTITY INTEGER;
    DECLARE I_PRICE FLOAT;

    WHENEVER NOT FOUND GOTO NO_ITEM;

    SELECT I_PRICE INTO I_PRICE FROM ITEM WHERE I_ID = OL_I_ID;

    DECLARE S_CUR CURSOR FOR
        SELECT
            S_QUANTITY, S_DATA,
            S_DIST_01, S_DIST_02, S_DIST_03, S_DIST_04, S_DIST_05,
            S_DIST_06, S_DIST_07, S_DIST_08, S_DIST_09, S_DIST_10
        FROM
            STOCK
        WHERE
              S_I_ID = OL_I_ID
          AND S_W_ID = OL_SUPPLY_W_ID;

    WHENEVER NOT FOUND GOTO NO_STOCK;

    OPEN S_CUR (EXCLUSIVE);

    FETCH S_CUR INTO
        S_QUANTITY, S_DATA,
        S_DIST_01, S_DIST_02, S_DIST_03, S_DIST_04, S_DIST_05,
        S_DIST_06, S_DIST_07, S_DIST_08, S_DIST_09, S_DIST_10;

    IF (S_QUANTITY < OL_QTY)
        S_QUANTITY := S_QUANTITY - OL_QTY + 91;
    ELSE
        S_QUANTITY := S_QUANTITY - OL_QTY;

    UPDATE STOCK SET S_QUANTITY = S_QUANTITY WHERE CURRENT OF S_CUR;

    AMOUNT := OL_QTY * I_PRICE;

    RETURN;

NO_STOCK: SIGNAL ('NOSTK', 'NO STOCK ROW FOUND.');
NO_ITEM:  SIGNAL ('NOITM', 'NO ITEM ROW FOUND.');
}
```

This procedure first reads the I_PRICE from ITEM and update the S_QUANTITY in STOCK. The price of the order line is returned as output parameter amount.

 **Note**

- Use inout parameters if possible. The inout (call by reference) is the fastest way of passing parameters.
- When reading a table with the intention of updating the row afterwards, use a cursor and positioned update.
- Use the EXCLUSIVE option in opening the cursor. This causes the read to put an exclusive lock on the row, which eliminates deadlocks caused by a shared read followed by an update. This deadlocks if there are more than one readers at the time of the update.
- Use `whenever not found' and signal to signal exceptions (no item or stock line).

When the stock level for all order lines has been updated this reads the customer data.

The bottleneck in terms of serialization is the read-update of the district, where the order gets an O_ID. This must be done as late as possible but has to precede the inserts, since these will use the O_ID. Note the exclusive cursor again.

To finish the transaction new_order insert ORDERS, NOW_ORDER and ORDER_LINE. These are basically in ascending order for each district and have high locality. Note the call by reference (inout) for ol_insert.

21.7.3. Payment

The payment transaction reads and updates the customer. The customer may either be identified by its last name or its C_ID. In the case the last name this chooses the middle row of the set of customers sharing the same last name, ordered by first name. Note the select count and the cursor.

The transaction profile does not offer possibilities of optimization.

21.7.4. Delivery

The delivery transaction reads and deletes a line from NEW_ORDER and updated the corresponding ORDERS and ORDER_LINE rows. The rules allow committing the transaction after processing each order. The client calls this procedure ten times with a different d_id parameter once every ten new orders. It is better to have the 1 to 10 loop in the client in order to keep locks on for the least time possible.

We use a cursor to read NEW_ORDER. Note the open no_cur (exclusive, prefetch 1). The prefetch 1 means we only intend to get one row. This prevents the cursor from prefetching more rows, which would be useless here.

Otherwise the transaction does not leave room for optimization.

21.7.5. Order Status

This transaction picks the last order of a given customer. It uses a select in descending order to locate it.

This is a read only transaction. Note the use of SQL_CONCURRENCY in the client code for specifying historical read mode. This eliminates locking.

Note

The ORDER BY clause must list all key parts of the index to be used, all with either ASC or DESC ordering in order to specify that index.

This transaction could be implemented in principle using the ORDERS primary key, O_W_ID, O_D_ID, O_ID. This would read in descending order until finding a row with a given C_IC. There being 3000 customers per district this could cause an average 1500 lines to check before finding the right one. Since the distribution of O_C_ID is not even the above is not entirely correct but close enough. The other possibility is having a separate index, O_W_ID, O_D_ID, O_C_ID, O_ID for this transaction. The trade-off being 1500 serial reads against 10 random insert (10 new order per one order status) we choose to have the extra index.

21.7.6. Stock Level

This is a complex read-only transaction. This finds all distinct items which have been ordered within the last n orders from a district having a stock level lower than a given threshold. The SQL statement:

```
SELECT
    COUNT (DISTINCT S_I_ID) INTO N_ITEMS
FROM
    ORDER_LINE, STOCK
WHERE
        OL_W_ID = W_ID
    AND OL_D_ID = D_ID
    AND OL_O_ID < LAST_O
    AND OL_O_ID >= LAST_O - 20
    AND S_W_ID = W_ID
    AND S_I_ID = OL_I_ID
    AND S_QUANTITY < THRESHOLD;
```

is the fastest way of getting this. Note that the ORDER_LINE is the driving table of join (leftmost in FROM). Also not the use of historical read in the client (SQL_CONCURRENCY option).

21.8. DDL Statements

```
CREATE TABLE WAREHOUSE (
    W_ID                INTEGER,
    W_NAME              CHARACTER (10),
```

```
    W_STREET_1          CHARACTER (20),
    W_STREET_2          CHARACTER (20),
    W_CITY              CHARACTER (20),
    W_STATE             CHARACTER (2),
    W_ZIP               CHARACTER (9),
    W_TAX               NUMERIC,
    W_YTD               NUMERIC,
    PRIMARY KEY (W_ID)
);

CREATE TABLE DISTRICT (
    D_ID                INTEGER,
    D_W_ID              INTEGER,
    D_NAME              CHARACTER (10),
    D_STREET_1          CHARACTER (20),
    D_STREET_2          CHARACTER (20),
    D_CITY              CHARACTER (20),
    D_STATE             CHARACTER (2),
    D_ZIP               CHARACTER (9),
    D_TAX               NUMERIC,
    D_YTD               NUMERIC,
    D_NEXT_O_ID         INTEGER,
    PRIMARY KEY (D_W_ID, D_ID)
);

CREATE TABLE CUSTOMER (
    C_ID                INTEGER,
    C_D_ID              INTEGER,
    C_W_ID              INTEGER,
    C_FIRST             CHARACTER (16),
    C_MIDDLE            CHARACTER (2),
    C_LAST              VARCHAR,
    C_STREET_1          CHARACTER (20),
    C_STREET_2          CHARACTER (20),
    C_CITY              CHARACTER (20),
    C_STATE             CHARACTER (2),
    C_ZIP               CHARACTER (9),
    C_PHONE             CHARACTER (16),
    C_SINCE             VARCHAR,
    C_CREDIT            CHARACTER (2),
    C_CREDIT_LIM        NUMERIC,
    C_DISCOUNT          NUMERIC,
    C_BALANCE           NUMERIC,
    C_YTD_PAYMENT       NUMERIC,
    C_CNT_PAYMENT       NUMERIC,
    C_CNT_DELIVERY      NUMERIC,
    C_DATA_1            CHARACTER (250),
    C_DATA_2            CHARACTER (250),
    PRIMARY KEY (C_W_ID, C_D_ID, C_ID)
);

CREATE INDEX C_BY_LAST ON CUSTOMER (C_W_ID, C_D_ID, C_LAST, C_FIRST);

CREATE TABLE HISTORY (
    H_C_ID              INTEGER,
    H_C_D_ID            INTEGER,
    H_C_W_ID            INTEGER,
    H_D_ID              INTEGER,
    H_W_ID              INTEGER,
    H_DATE              DATE,
    H_AMOUNT            NUMERIC,
    H_DATA              CHARACTER (24),
    PRIMARY KEY (H_DATE, H_C_ID)
);

CREATE TABLE NEW_ORDER (
    NO_O_ID             INTEGER,
    NO_D_ID             INTEGER,
    NO_W_ID             INTEGER,
    PRIMARY KEY (NO_W_ID, NO_D_ID, NO_O_ID)
);

CREATE TABLE ORDERS (
```

```
    O_ID                INTEGER,
    O_D_ID              INTEGER,
    O_W_ID              INTEGER,
    O_C_ID              INTEGER,
    O_ENTRY_D           DATE,
    O_CARRIER_ID        INTEGER,
    O_OL_CNT            INTEGER,
    O_ALL_LOCAL         NUMERIC,
    PRIMARY KEY (O_W_ID, O_D_ID, O_ID)
);

CREATE INDEX O_BY_C_ID ON ORDERS (O_W_ID, O_D_ID, O_C_ID, O_ID)

CREATE TABLE ORDER_LINE (
    OL_O_ID             INTEGER,
    OL_D_ID             INTEGER,
    OL_W_ID             INTEGER,
    OL_NUMBER           INTEGER,
    OL_I_ID             INTEGER,
    OL_SUPPLY_W_ID      INTEGER,
    OL_DELIVERY_D       DATE,
    OL_QUANTITY         NUMERIC,
    OL_AMOUNT           NUMERIC,
    OL_DIST_INFO        CHARACTER (24),
    PRIMARY KEY (OL_W_ID, OL_D_ID, OL_O_ID, OL_NUMBER)
);

CREATE TABLE ITEM (
    I_ID                INTEGER,
    I_IM_ID             INTEGER,
    I_NAME              CHARACTER (24),
    I_PRICE             NUMERIC,
    I_DATA              CHARACTER (50),
    PRIMARY KEY (I_ID)
);

CREATE TABLE STOCK (
    S_I_ID              INTEGER,
    S_W_ID              INTEGER,
    S_QUANTITY          NUMERIC,
    S_DIST_01           CHARACTER (24),
    S_DIST_02           CHARACTER (24),
    S_DIST_03           CHARACTER (24),
    S_DIST_04           CHARACTER (24),
    S_DIST_05           CHARACTER (24),
    S_DIST_06           CHARACTER (24),
    S_DIST_07           CHARACTER (24),
    S_DIST_08           CHARACTER (24),
    S_DIST_09           CHARACTER (24),
    S_DIST_10           CHARACTER (24),
    S_YTD               NUMERIC,
    S_CNT_ORDER         NUMERIC,
    S_CNT_REMOTE        NUMERIC,
    S_DATA              CHARACTER (50),
    PRIMARY KEY (S_I_ID, S_W_ID)
);
```

21.9. Stored Procedures

```
--
--  tpcc.sql
--
--  Implementation of the TPC C Benchmark transactions as stored procedures.
--
--  These are for use with the tpcc test driver (tpcc) in the Virtuoso sample
--  directory.  See TPCC.DOC in the virtuoso documentation bundle for comments
--  and instructions.
--
--  Copyright (C) 1998-2024 OpenLink Software.
--  All Rights Reserved.
--
--  The copyright above and this notice must be preserved in all
```

```
--  copies of this source code.  The copyright above does not
--  evidence any actual or intended publication of this source code.
--
--  This is unpublished proprietary trade secret of OpenLink Software.
--  This source code may not be copied, disclosed, distributed, demonstrated
--  or licensed except as authorized by OpenLink Software.
--

--  slevel - The transaction procedure for the Stock Level transaction.
--
--  This is executed as an autocommitting history read transaction. The number
--  of STOCK rows where quantity is below th threshold.  The rows are taken
--  from the last 20 orders on a warehouse / district combination.

CREATE PROCEDURE SLEVEL (
    IN W_ID             INTEGER,
    IN D_ID             INTEGER,
    IN THRESHOLD        INTEGER)
{
    DECLARE LAST_O, N_ITEMS INTEGER;

    SELECT D_NEXT_O_ID INTO LAST_O
    FROM DISTRICT
    WHERE
            D_W_ID = W_ID
        AND D_ID = D_ID;

    SELECT COUNT (DISTINCT S_I_ID)
    INTO N_ITEMS
    FROM ORDER_LINE, STOCK
    WHERE
            OL_W_ID = W_ID
        AND OL_D_ID = D_ID
        AND OL_O_ID < LAST_O
        AND OL_O_ID >= LAST_O - 20
        AND S_W_ID = W_ID
        AND S_I_ID = OL_I_ID AND S_QUANTITY < THRESHOLD;

    RESULT_NAMES (N_ITEMS);

    RESULT (N_ITEMS);
}

--
-- c_by_name, call_c_by_name
-- Examples on retrieving CUSTOMER by last name.
-- Functionality open coded in actual transaction procedures.
--

CREATE PROCEDURE C_BY_NAME (
    IN  W_ID            INTEGER,
    IN  D_ID            INTEGER,
    IN  NAME            VARCHAR,
    OUT ID              INTEGER)
{
    DECLARE N, C_COUNT INTEGER;

    DECLARE C_CUR CURSOR FOR
    SELECT C_ID
    FROM CUSTOMER
    WHERE
            C_W_ID = W_ID
        AND C_D_ID = D_ID
        AND C_LAST = NAME
    ORDER BY C_W_ID, C_D_ID, C_LAST, C_FIRST;

    SELECT COUNT (*) INTO C_COUNT
    FROM CUSTOMER
    WHERE
            C_W_ID = W_ID
        AND C_D_ID = D_ID
        AND C_LAST = NAME;
```

```
    N := 0;
    OPEN C_CUR;
    WHENEVER NOT FOUND GOTO NOTFOUND;

    WHILE (N <= C_COUNT / 2) {
        FETCH C_CUR INTO ID;
        N := N + 1;
    }
    RETURN;

NOTFOUND:
    SIGNAL ('CNF', 'CUSTOMER NOT FOUND BY NAME');
    RETURN;
}

CREATE PROCEDURE CALL_C_BY_NAME (
    IN W_ID             INTEGER,
    IN D_ID             INTEGER,
    IN C_LAST           VARCHAR)
{
    DECLARE C_ID INTEGER;

    C_BY_NAME (W_ID, D_ID, C_LAST, C_ID);
}

--
-- payment
-- This procedure implements the Payment transaction.
--

CREATE PROCEDURE BC_C_DATA (
    INOUT C_NEW         VARCHAR,
    INOUT C_DATA        VARCHAR)
{
    RETURN (
        CONCATENATE (C_NEW, SUBSEQ (C_DATA, LENGTH (C_NEW), LENGTH (C_DATA))));
}

CREATE PROCEDURE PAYMENT (
    IN _W_ID            INTEGER,
    IN _C_W_ID          INTEGER,
    IN H_AMOUNT         FLOAT,
    IN _D_ID            INTEGER,
    IN _C_D_ID          INTEGER,
    IN _C_ID            INTEGER,
    IN _C_LAST          VARCHAR)
{
    DECLARE
        _C_DATA, _C_FIRST, _C_MIDDLE, _C_STREET_1,  _C_STREET_2, _C_CITY,
        _C_STATE, _C_ZIP, _C_PHONE, _C_CREDIT, _C_CREDIT_LIM, _C_DISCOUNT,
        _C_BALANCE, _C_SINCE, _C_DATA_1, _C_DATA_2 VARCHAR;

    DECLARE
        N, _W_YTD, _D_YTD, _C_CNT_PAYMENT INTEGER;

    DECLARE
        _D_STREET_1, _D_STREET_2, _D_CITY, _D_STATE, _D_ZIP, _D_NAME VARCHAR;

    DECLARE
        _W_STREET_1, _W_STREET_2, _W_CITY, _W_STATE, _W_ZIP, _W_NAME,
        SCREEN_DATA VARCHAR;

    IF (_C_ID = 0) {
        DECLARE NAMECNT INTEGER;
        WHENEVER NOT FOUND GOTO NO_CUSTOMER;

        SELECT COUNT(C_ID) INTO NAMECNT
        FROM CUSTOMER
        WHERE
                C_LAST = _C_LAST
            AND C_D_ID = _D_ID
            AND C_W_ID = _W_ID;
```

```
    DECLARE C_BYNAME CURSOR FOR
    SELECT C_ID
    FROM CUSTOMER
    WHERE
            C_W_ID = _C_W_ID
        AND C_D_ID = _C_D_ID
        AND C_LAST = _C_LAST
    ORDER BY
        C_W_ID, C_D_ID, C_LAST, C_FIRST;

    OPEN C_BYNAME (EXCLUSIVE);

    N := 0;
    WHILE (N <= NAMECNT / 2) {
        FETCH C_BYNAME   INTO _C_ID;
        N := N + 1;
    }

    CLOSE C_BYNAME;
}

DECLARE C_CR CURSOR FOR
SELECT
    C_FIRST, C_MIDDLE, C_LAST, C_STREET_1, C_STREET_2, C_CITY,
    C_STATE, C_ZIP, C_PHONE, C_CREDIT, C_CREDIT_LIM, C_DISCOUNT,
    C_BALANCE, C_SINCE, C_DATA_1, C_DATA_2, C_CNT_PAYMENT
FROM
    CUSTOMER
WHERE
        C_W_ID = _C_W_ID
    AND C_D_ID = _C_D_ID
    AND C_ID = _C_ID;

OPEN C_CR (EXCLUSIVE);

FETCH C_CR INTO
    _C_FIRST, _C_MIDDLE, _C_LAST,
    _C_STREET_1, _C_STREET_2, _C_CITY, _C_STATE, _C_ZIP,
    _C_PHONE, _C_CREDIT, _C_CREDIT_LIM,
    _C_DISCOUNT, _C_BALANCE, _C_SINCE, _C_DATA_1, _C_DATA_2,
    _C_CNT_PAYMENT;

_C_BALANCE := _C_BALANCE + H_AMOUNT;

IF (_C_CREDIT = 'BC') {
    UPDATE CUSTOMER
    SET C_BALANCE = _C_BALANCE,
        C_DATA_1 = BC_C_DATA (
            SPRINTF ('%5d%5d%5d%5d%5d%9f',
                _C_ID, _C_D_ID, _C_W_ID, _D_ID, _W_ID, H_AMOUNT),
            _C_DATA_1),
        C_CNT_PAYMENT = _C_CNT_PAYMENT + 1
    WHERE CURRENT OF C_CR;
    SCREEN_DATA := SUBSEQ (_C_DATA_1, 1, 200);
} ELSE {
    UPDATE CUSTOMER
    SET C_BALANCE = _C_BALANCE, C_CNT_PAYMENT = _C_CNT_PAYMENT + 1
    WHERE CURRENT OF C_CR;
    SCREEN_DATA := ' ';
}

DECLARE D_CUR CURSOR FOR
SELECT D_STREET_1, D_STREET_2, D_CITY, D_STATE, D_ZIP, D_NAME, D_YTD
FROM DISTRICT
WHERE
        D_W_ID = _W_ID
    AND D_ID = _D_ID;

OPEN D_CUR (EXCLUSIVE);

FETCH D_CUR INTO
    _D_STREET_1,  _D_STREET_2, _D_CITY, _D_STATE, _D_ZIP,  _D_NAME, _D_YTD;
```

```
    UPDATE DISTRICT SET D_YTD = _D_YTD + H_AMOUNT WHERE CURRENT OF D_CUR;

    CLOSE D_CUR;

    DECLARE W_CUR CURSOR FOR
    SELECT  W_STREET_1, W_STREET_2, W_CITY, W_STATE, W_ZIP, W_NAME, W_YTD
    FROM WAREHOUSE
    WHERE W_ID = _W_ID;

    OPEN W_CUR (EXCLUSIVE);
    FETCH  W_CUR INTO
        _W_STREET_1, _W_STREET_2, _W_CITY, _W_STATE, _W_ZIP, _W_NAME, _W_YTD;

    UPDATE WAREHOUSE SET W_YTD = _W_YTD + H_AMOUNT;

    DECLARE H_DATA VARCHAR;
    H_DATA := _W_NAME;

    INSERT INTO HISTORY
        (H_C_D_ID, H_C_W_ID, H_C_ID, H_D_ID, H_W_ID, H_DATE, H_AMOUNT, H_DATA)
    VALUES
        (_C_D_ID, _C_W_ID, _C_ID, _D_ID, _W_ID, NOW (), H_AMOUNT, H_DATA);

    RESULT (
        _C_ID, _C_LAST, NOW (), _W_STREET_1, _W_STREET_2, _W_CITY, _W_STATE,
        _W_ZIP, _D_STREET_1, _D_STREET_2, _D_CITY, _D_STATE, _D_ZIP,
        _C_FIRST, _C_MIDDLE, _C_STREET_1, _C_STREET_2, _C_CITY, _C_STATE,
        _C_ZIP, _C_PHONE, _C_SINCE, _C_CREDIT, _C_CREDIT_LIM, _C_DISCOUNT,
        _C_BALANCE, SCREEN_DATA);

    RETURN;

NO_CUSTOMER:
    SIGNAL ('NOCUS', 'NO CUSTOMER IN PAYMENT.');
}

-- ol_stock - Part of the New Order transaction - Set the stock level for
-- an order line.  Compute the price and return it in amount.
--
-- Note - Open the cursor on STOCK as exclusive to avoid deadlocks.
-- Use positioned update on STOCK for speed.
--
-- Fetch the s_dist_01 - 10 columns from STOCK even though they are not used.
-- The test specification requires this. The operation is measurably faster if these
-- are omitted.-- The ORDER LINE is inserted later for better lock concurrency.

CREATE PROCEDURE OL_STOCK (
    IN    _W_ID                 INTEGER,
    IN    D_ID                  INTEGER,
    INOUT _OL_I_ID              INTEGER,
    IN    _OL_SUPPLY_W_ID       INTEGER,
    IN    QTY                   INTEGER,
    OUT   AMOUNT                FLOAT,
    INOUT S_DIST_01             VARCHAR,
    INOUT S_DIST_02             VARCHAR,
    INOUT S_DIST_03             VARCHAR,
    INOUT S_DIST_04             VARCHAR,
    INOUT S_DIST_05             VARCHAR,
    INOUT S_DIST_06             VARCHAR,
    INOUT S_DIST_07             VARCHAR,
    INOUT S_DIST_08             VARCHAR,
    INOUT S_DIST_09             VARCHAR,
    INOUT S_DIST_10             VARCHAR,
    INOUT DIST_INFO             VARCHAR)
{
    IF (_OL_I_ID = 0) RETURN;

    DECLARE _S_DATA VARCHAR;
    DECLARE _S_QUANTITY, _S_CNT_ORDER, _S_CNT_REMOTE INTEGER;

    WHENEVER NOT FOUND GOTO NO_ITEM;

    DECLARE _I_NAME VARCHAR;
```

```
    SELECT
        I_PRICE, I_NAME
    INTO
        AMOUNT, _I_NAME
    FROM
        ITEM
    WHERE I_ID = _OL_I_ID;

    DECLARE S_CUR CURSOR FOR
    SELECT
        S_QUANTITY, S_DATA, S_CNT_ORDER, S_CNT_REMOTE,
        S_DIST_01, S_DIST_02, S_DIST_03, S_DIST_04, S_DIST_05,
        S_DIST_06, S_DIST_07, S_DIST_08, S_DIST_09, S_DIST_10
    FROM STOCK
    WHERE
            S_I_ID = _OL_I_ID
        AND S_W_ID = _OL_SUPPLY_W_ID;

    WHENEVER NOT FOUND GOTO NO_STOCK;

    OPEN S_CUR (EXCLUSIVE);

    FETCH S_CUR INTO
        _S_QUANTITY, _S_DATA, _S_CNT_ORDER, _S_CNT_REMOTE,
        S_DIST_01, S_DIST_02, S_DIST_03, S_DIST_04, S_DIST_05,
        S_DIST_06, S_DIST_07, S_DIST_08, S_DIST_09, S_DIST_10;

    IF (_S_QUANTITY < QTY)
        _S_QUANTITY := _S_QUANTITY - QTY + 91;
    ELSE
        _S_QUANTITY := _S_QUANTITY - QTY;

    IF (_W_ID <> _OL_SUPPLY_W_ID)
        _S_CNT_REMOTE := _S_CNT_REMOTE + 1;

    UPDATE STOCK
    SET
        S_QUANTITY = _S_QUANTITY,
        S_CNT_ORDER = _S_CNT_ORDER + 1,
        S_CNT_REMOTE = _S_CNT_REMOTE
    WHERE CURRENT OF S_CUR;

    IF (D_ID = 1)       DIST_INFO := S_DIST_01;
    ELSE IF (D_ID = 2)  DIST_INFO := S_DIST_02;
    ELSE IF (D_ID = 3)  DIST_INFO := S_DIST_03;
    ELSE IF (D_ID = 4)  DIST_INFO := S_DIST_04;
    ELSE IF (D_ID = 5)  DIST_INFO := S_DIST_05;
    ELSE IF (D_ID = 6)  DIST_INFO := S_DIST_06;
    ELSE IF (D_ID = 7)  DIST_INFO := S_DIST_07;
    ELSE IF (D_ID = 8)  DIST_INFO := S_DIST_08;
    ELSE IF (D_ID = 9)  DIST_INFO := S_DIST_09;
    ELSE IF (D_ID = 10) DIST_INFO := S_DIST_10;

    RESULT (_I_NAME, _S_QUANTITY, 'G', AMOUNT, AMOUNT * QTY);

    AMOUNT := QTY * AMOUNT;

    RETURN;

NO_STOCK:
    SIGNAL ('NOSTK', 'NO STOCK ROW FOUND.');
NO_ITEM:
    SIGNAL ('NOITM', 'NO ITEM ROW FOUND.');
}

--
-- ol_insert - Part of New Order transaction. Insert an ORDER LINE.
--
-- Note use of inout parameters even though they are not modified here.
-- This saves copying the values.

CREATE PROCEDURE OL_INSERT (
```

```
    INOUT W_ID                  INTEGER,
    INOUT D_ID                  INTEGER,
    INOUT O_ID                  INTEGER,
    IN    OL_NUMBER             INTEGER,
    INOUT OL_I_ID               INTEGER,
    INOUT OL_QTY                INTEGER,
    INOUT OL_AMOUNT             FLOAT,
    INOUT OL_SUPPLY_W_ID        INTEGER,
    INOUT OL_DIST_INFO          VARCHAR,
    INOUT TAX_AND_DISCOUNT      FLOAT)
{
    IF (OL_I_ID = -1) RETURN;

    OL_AMOUNT := OL_AMOUNT * TAX_AND_DISCOUNT;

    INSERT INTO ORDER_LINE (
        OL_O_ID, OL_D_ID, OL_W_ID, OL_NUMBER, OL_I_ID, OL_SUPPLY_W_ID,
        OL_QUANTITY, OL_AMOUNT, OL_DIST_INFO)
    VALUES (
        O_ID, D_ID, W_ID, OL_NUMBER, OL_I_ID, OL_SUPPLY_W_ID,
        OL_QTY, OL_AMOUNT, OL_DIST_INFO);
}

--
-- cust_info - part of New Order transaction. Return customer info.
--
--

CREATE PROCEDURE CUST_INFO (
    IN    W_ID INTEGER,
    IN    D_ID INTEGER,
    INOUT _C_ID INTEGER,
    INOUT _C_LAST VARCHAR,
    OUT    _C_DISCOUNT FLOAT,
    OUT    _C_CREDIT VARCHAR)
{
    WHENEVER NOT FOUND GOTO ERR;

    SELECT
        C_LAST, C_DISCOUNT, C_CREDIT INTO _C_LAST, _C_DISCOUNT, _C_CREDIT
    FROM
        CUSTOMER
    WHERE
            C_W_ID = W_ID
        AND C_D_ID = D_ID
        AND C_ID = _C_ID;
    RETURN;
ERR:
    SIGNAL ('NOCUS', 'NO CUSTOMER');
}

-- new_order - Top level procedure of New Order transaction.
-- Take a fixed 10 order lines as individually named parameters
-- to stay easily portable.

CREATE PROCEDURE NEW_ORDER (
    IN _W_ID            INTEGER,
    IN _D_ID            INTEGER,
    IN _C_ID            INTEGER,
    IN O_OL_CNT         INTEGER,
    IN O_ALL_LOCAL      INTEGER,
    IN I_ID_1           INTEGER,
    IN S_W_ID_1         INTEGER,
    IN QTY_1            INTEGER,
    IN I_ID_2           INTEGER,
    IN S_W_ID_2         INTEGER,
    IN QTY_2            INTEGER,
    IN I_ID_3           INTEGER,
    IN S_W_ID_3         INTEGER,
    IN QTY_3            INTEGER,
    IN I_ID_4           INTEGER,
    IN S_W_ID_4         INTEGER,
    IN QTY_4            INTEGER,
```

```
        IN I_ID_5           INTEGER,
        IN S_W_ID_5         INTEGER,
        IN QTY_5            INTEGER,
        IN I_ID_6           INTEGER,
        IN S_W_ID_6         INTEGER,
        IN QTY_6            INTEGER,
        IN I_ID_7           INTEGER,
        IN S_W_ID_7         INTEGER,
        IN QTY_7            INTEGER,
        IN I_ID_8           INTEGER,
        IN S_W_ID_8         INTEGER,
        IN QTY_8            INTEGER,
        IN I_ID_9           INTEGER,
        IN S_W_ID_9         INTEGER,
        IN QTY_9            INTEGER,
        IN I_ID_10          INTEGER,
        IN S_W_ID_10        INTEGER,
        IN QTY_10           INTEGER)
{
    DECLARE
        OL_A_1, OL_A_2, OL_A_3, OL_A_4, OL_A_5,
        OL_A_6, OL_A_7, OL_A_8, OL_A_9, OL_A_10 INTEGER;
    DECLARE _C_DISCOUNT, _D_TAX, _W_TAX, TAX_AND_DISCOUNT FLOAT;
    DECLARE DATETIME DATE;
    DECLARE  _C_LAST, _C_CREDIT VARCHAR;
    DECLARE _O_ID INTEGER;

    DECLARE
        I_NAME, S_DIST_01, S_DIST_02, S_DIST_03, S_DIST_04, S_DIST_05,
        S_DIST_06, S_DIST_07, S_DIST_08, S_DIST_09, S_DIST_10 VARCHAR;
    DECLARE
        DISTI_1, DISTI_2, DISTI_3, DISTI_4, DISTI_5,
        DISTI_6, DISTI_7, DISTI_8, DISTI_9, DISTI_10 VARCHAR;

    DATETIME := NOW ();

    -- DECLARE RESULT ROW FROM OL_STOCK. OPTIONAL.;
    RESULT_NAMES (I_NAME, QTY_1, DISTI_1, OL_A_1, OL_A_2);

    OL_STOCK (
        _W_ID, _D_ID, I_ID_1, S_W_ID_1, QTY_1, OL_A_1, S_DIST_01, S_DIST_02,
        S_DIST_03, S_DIST_04, S_DIST_05, S_DIST_06, S_DIST_07, S_DIST_08,
        S_DIST_09, S_DIST_10, DISTI_1);
    OL_STOCK (
        _W_ID, _D_ID, I_ID_2, S_W_ID_2, QTY_2, OL_A_2, S_DIST_01, S_DIST_02,
        S_DIST_03, S_DIST_04, S_DIST_05, S_DIST_06, S_DIST_07, S_DIST_08,
        S_DIST_09, S_DIST_10, DISTI_2);
    OL_STOCK (
        _W_ID, _D_ID, I_ID_3, S_W_ID_3, QTY_3, OL_A_3, S_DIST_01, S_DIST_02,
        S_DIST_03, S_DIST_04, S_DIST_05, S_DIST_06, S_DIST_07, S_DIST_08,
        S_DIST_09, S_DIST_10, DISTI_3);
    OL_STOCK (
        _W_ID, _D_ID, I_ID_4, S_W_ID_4, QTY_4, OL_A_4, S_DIST_01, S_DIST_02,
        S_DIST_03, S_DIST_04, S_DIST_05, S_DIST_06, S_DIST_07, S_DIST_08,
        S_DIST_09, S_DIST_10, DISTI_4);
    OL_STOCK (
        _W_ID, _D_ID, I_ID_5, S_W_ID_5, QTY_5, OL_A_5, S_DIST_01, S_DIST_02,
        S_DIST_03, S_DIST_04, S_DIST_05, S_DIST_06, S_DIST_07, S_DIST_08,
        S_DIST_09, S_DIST_10, DISTI_5);
    OL_STOCK (
        _W_ID, _D_ID, I_ID_6, S_W_ID_6, QTY_6, OL_A_6, S_DIST_01, S_DIST_02,
        S_DIST_03, S_DIST_04, S_DIST_05, S_DIST_06, S_DIST_07, S_DIST_08,
        S_DIST_09, S_DIST_10, DISTI_6);
    OL_STOCK (
        _W_ID, _D_ID, I_ID_7, S_W_ID_7, QTY_7, OL_A_7, S_DIST_01, S_DIST_02,
        S_DIST_03, S_DIST_04, S_DIST_05, S_DIST_06, S_DIST_07, S_DIST_08,
        S_DIST_09, S_DIST_10, DISTI_7);
    OL_STOCK (
        _W_ID, _D_ID, I_ID_8, S_W_ID_8, QTY_8, OL_A_8, S_DIST_01, S_DIST_02,
        S_DIST_03, S_DIST_04, S_DIST_05, S_DIST_06, S_DIST_07, S_DIST_08,
        S_DIST_09, S_DIST_10, DISTI_8);
    OL_STOCK (
        _W_ID, _D_ID, I_ID_9, S_W_ID_9, QTY_8, OL_A_9, S_DIST_01, S_DIST_02,
```

```
        S_DIST_03, S_DIST_04, S_DIST_05, S_DIST_06, S_DIST_07, S_DIST_08,
        S_DIST_09, S_DIST_10, DISTI_9);
    OL_STOCK (
        _W_ID, _D_ID, I_ID_10, S_W_ID_10, QTY_10, OL_A_10, S_DIST_01,
        S_DIST_02, S_DIST_03, S_DIST_04, S_DIST_05, S_DIST_06, S_DIST_07,
        S_DIST_08, S_DIST_09, S_DIST_10, DISTI_10);

    CUST_INFO (_W_ID, _D_ID, _C_ID, _C_LAST, _C_DISCOUNT, _C_CREDIT);

    DECLARE  D_CUR CURSOR FOR
    SELECT
        D_TAX, D_NEXT_O_ID
    FROM
        DISTRICT
    WHERE
            D_W_ID = _W_ID
        AND D_ID = _D_ID;

    WHENEVER NOT FOUND GOTO NOWARE;

    OPEN D_CUR (EXCLUSIVE);
    FETCH D_CUR INTO _D_TAX, _O_ID;
    UPDATE DISTRICT SET D_NEXT_O_ID = _O_ID + 1 WHERE CURRENT OF D_CUR;
    CLOSE D_CUR;

    INSERT INTO ORDERS
        (O_ID, O_D_ID, O_W_ID, O_C_ID, O_ENTRY_D, O_OL_CNT, O_ALL_LOCAL)
    VALUES (_O_ID, _D_ID, _W_ID, _C_ID, DATETIME, O_OL_CNT, O_ALL_LOCAL);

    INSERT INTO NEW_ORDER
        (NO_O_ID, NO_D_ID, NO_W_ID)
    VALUES (_O_ID, _D_ID, _W_ID);

    SELECT W_TAX INTO _W_TAX FROM WAREHOUSE WHERE W_ID = _W_ID;

    TAX_AND_DISCOUNT := (1 + _D_TAX + _W_TAX) * (1 - _C_DISCOUNT);

    OL_INSERT (
        _W_ID, _D_ID, _O_ID, 1, I_ID_1, QTY_1, OL_A_1,  S_W_ID_1, DISTI_1,
        TAX_AND_DISCOUNT);
    OL_INSERT (
        _W_ID, _D_ID, _O_ID, 2, I_ID_2, QTY_2, OL_A_2,  S_W_ID_2, DISTI_2,
        TAX_AND_DISCOUNT);
    OL_INSERT (
        _W_ID, _D_ID, _O_ID, 3, I_ID_3, QTY_3, OL_A_3,  S_W_ID_3, DISTI_3,
        TAX_AND_DISCOUNT);
    OL_INSERT (
        _W_ID, _D_ID, _O_ID, 4, I_ID_4, QTY_4, OL_A_4,  S_W_ID_4, DISTI_4,
        TAX_AND_DISCOUNT);
    OL_INSERT (
        _W_ID, _D_ID, _O_ID, 5, I_ID_5, QTY_5, OL_A_5,  S_W_ID_5, DISTI_5,
        TAX_AND_DISCOUNT);
    OL_INSERT (
        _W_ID, _D_ID, _O_ID, 6, I_ID_6, QTY_6, OL_A_6,  S_W_ID_6, DISTI_6,
        TAX_AND_DISCOUNT);
    OL_INSERT (
        _W_ID, _D_ID, _O_ID, 7, I_ID_7, QTY_7, OL_A_7,  S_W_ID_7, DISTI_7,
        TAX_AND_DISCOUNT);
    OL_INSERT (
        _W_ID, _D_ID, _O_ID, 8, I_ID_6, QTY_8, OL_A_8,  S_W_ID_8, DISTI_8,
        TAX_AND_DISCOUNT);
    OL_INSERT (
        _W_ID, _D_ID, _O_ID, 9, I_ID_9, QTY_9, OL_A_9,  S_W_ID_9, DISTI_9,
        TAX_AND_DISCOUNT);
    OL_INSERT (
        _W_ID, _D_ID, _O_ID, 10, I_ID_10, QTY_10, OL_A_10,  S_W_ID_10, DISTI_10,
        TAX_AND_DISCOUNT);

    END_RESULT ();
    RESULT (_W_TAX, _D_TAX, _O_ID, _C_LAST, _C_DISCOUNT, _C_CREDIT);
    RETURN;

NOWARE:
```

```
    SIGNAL ('NOWRE', 'WAREHOUSE OR DISTRICTNOT FOUND.');
}

-- delivery_1 - Top level procedure for the Delivery transaction
--
-- This is called 10 times by the client in each delivery transaction.
-- The rules allow Delivery to be implemented as up to 10 separately committed
-- transactions. This is done to minimize lock duration.

CREATE PROCEDURE DELIVERY_1 (
    IN W_ID             INTEGER,
    IN CARRIER_ID       INTEGER,
    IN D_ID             INTEGER)
{
    DECLARE NO_CUR CURSOR FOR
    SELECT
        NO_O_ID
    FROM
        NEW_ORDER
    WHERE
            NO_W_ID = W_ID
        AND NO_D_ID = D_ID;

    DECLARE DATETIME DATE;
    DECLARE _O_ID, _C_ID INTEGER;
    DECLARE OL_TOTAL FLOAT;
    DATETIME := NOW ();

    OPEN NO_CUR (EXCLUSIVE, PREFETCH 1);
    FETCH NO_CUR INTO _O_ID;
    DELETE FROM NEW_ORDER WHERE CURRENT OF NO_CUR;
    CLOSE NO_CUR;

    DECLARE O_CUR CURSOR FOR
    SELECT
        O_C_ID
    FROM
        ORDERS
    WHERE
            O_W_ID = W_ID
        AND O_D_ID = D_ID
    AND O_ID = _O_ID;

    OPEN O_CUR (EXCLUSIVE);
    FETCH O_CUR INTO _C_ID;
    UPDATE ORDERS SET O_CARRIER_ID = CARRIER_ID WHERE CURRENT OF O_CUR;
    CLOSE O_CUR;

    DECLARE OL_CUR CURSOR FOR
    SELECT
        OL_AMOUNT
    FROM
        ORDER_LINE
    WHERE
            OL_W_ID = W_ID
        AND OL_D_ID = D_ID
        AND OL_O_ID = _O_ID;

    WHENEVER NOT FOUND GOTO LINES_DONE;
    OL_TOTAL := 0.0;
    OPEN OL_CUR (EXCLUSIVE);
    WHILE (1) {
        DECLARE TMP INTEGER;
        FETCH OL_CUR INTO TMP;
        OL_TOTAL := OL_TOTAL + TMP;
        UPDATE ORDER_LINE SET OL_DELIVERY_D = DATETIME WHERE CURRENT OF OL_CUR;
    }
LINES_DONE:
    UPDATE CUSTOMER
    SET
        C_BALANCE = C_BALANCE + OL_TOTAL,
        C_CNT_DELIVERY = C_CNT_DELIVERY + 1
    WHERE
```

```
            C_W_ID = W_ID
        AND C_D_ID = D_ID
        AND C_ID = _C_ID;

    RETURN _O_ID;
}

-- ostat - Top level procedure for the Order Status transaction.
--
--

CREATE PROCEDURE OSTAT (
    IN _W_ID            INTEGER,
    IN _D_ID            INTEGER,
    IN _C_ID            INTEGER,
    IN _C_LAST          VARCHAR)
{
    DECLARE _C_FIRST, _C_MIDDLE, _C_BALANCE VARCHAR;
    DECLARE
        _O_ID, _OL_I_ID, _OL_SUPPLY_W_ID, _OL_QUANTITY, _O_CARRIER_ID,
        N INTEGER;
    DECLARE _OL_AMOUNT FLOAT;
    DECLARE _OL_DELIVERY_D, _O_ENTRY_D VARCHAR;

    IF (_C_ID = 0) {
        DECLARE NAMECNT INTEGER;
        WHENEVER NOT FOUND GOTO NO_CUSTOMER;

        SELECT COUNT (*)
        INTO NAMECNT
        FROM CUSTOMER
        WHERE
                C_LAST = _C_LAST
            AND C_D_ID = _D_ID
            AND C_W_ID = _W_ID;

        DECLARE C_BYNAME CURSOR FOR
        SELECT
            C_BALANCE, C_LAST, C_MIDDLE, C_ID
        FROM CUSTOMER
        WHERE
                C_W_ID = _W_ID
            AND C_D_ID = _D_ID
            AND C_LAST = _C_LAST
        ORDER BY
            C_W_ID, C_D_ID, C_LAST, C_FIRST;

        OPEN C_BYNAME;

        N := 0;
        WHILE (N <= NAMECNT / 2) {
            FETCH C_BYNAME INTO _C_BALANCE, _C_FIRST, _C_MIDDLE, _C_ID;
            N := N + 1;
        }

        CLOSE C_BYNAME;
    }  ELSE {
        SELECT
            C_BALANCE, C_FIRST, C_MIDDLE, C_LAST
        INTO
            _C_BALANCE, _C_FIRST, _C_MIDDLE, _C_LAST
        FROM
            CUSTOMER
        WHERE
                C_W_ID = _W_ID
            AND C_D_ID = _D_ID
            AND C_ID = _C_ID;
    }

    WHENEVER NOT FOUND GOTO NO_ORDER;
    SELECT
        O_ID, O_CARRIER_ID, O_ENTRY_D
    INTO
```

```
        _O_ID, _O_CARRIER_ID, _O_ENTRY_D
    FROM
        ORDERS
    WHERE
            O_W_ID = _W_ID
        AND O_D_ID = _D_ID
        AND O_C_ID = _C_ID
    ORDER BY
        O_W_ID DESC, O_D_ID DESC, O_C_ID DESC, O_ID DESC;

    DECLARE O_LINE CURSOR FOR
    SELECT
        OL_I_ID, OL_SUPPLY_W_ID, OL_QUANTITY, OL_AMOUNT, OL_DELIVERY_D
    FROM
        ORDER_LINE
    WHERE
            OL_W_ID = _W_ID
        AND OL_D_ID = _D_ID
        AND OL_O_ID = _O_ID;

    WHENEVER NOT FOUND GOTO LINES_DONE;

    OPEN O_LINE;
    RESULT_NAMES (
        _OL_SUPPLY_W_ID, _OL_I_ID, _OL_QUANTITY, _OL_AMOUNT, _OL_DELIVERY_D);
    WHILE (1 = 1) {
        FETCH O_LINE INTO
            _OL_I_ID, _OL_SUPPLY_W_ID, _OL_QUANTITY, _OL_AMOUNT,
            _OL_DELIVERY_D;

        RESULT (
            _OL_SUPPLY_W_ID, _OL_I_ID, _OL_QUANTITY, _OL_AMOUNT,
            _OL_DELIVERY_D);
    }

LINES_DONE:
    END_RESULT ();

    RESULT_NAMES  (
        _C_ID, _C_LAST, _C_FIRST, _C_MIDDLE, _O_ENTRY_D, _O_CARRIER_ID,
        _C_BALANCE, _O_ID);

    RESULT (
        _C_ID, _C_LAST, _C_FIRST, _C_MIDDLE, _O_ENTRY_D, _O_CARRIER_ID,
        _C_BALANCE, _O_ID);

    RETURN;

NO_CUSTOMER:
    SIGNAL ('NOCUS', 'NO CUSTOMER IN ORDER STATUS');

NO_ORDER:
    RETURN 0;
}
```

Chapter 22. Using Virtuoso with Tuxedo

Abstract

BEA Tuxedo provides the framework, or middleware, for building scalable multi-tier client/server applications in heterogeneous (dissimilar), distributed environments that extend from the Web to the Enterprise. Using BEA Tuxedo, users can develop, manage, and deploy distributed applications independently of the underlying hardware, operating system, network, and database environment.

The current document covers linkage between Virtuoso server and Tuxedo by ATMI (Application-to-Transaction Monitor Interface) only.

In contrast to classic 2-tier client/server configuration of SQL servers, the Tuxedo brings 3-tier paradigm (clients, services, resource managers).

At the foundation of BEA Tuxedo ATMI is a proven, reliable transaction processor, also known as a transaction processing (TP) monitor. A transaction processor is an example of a 3-tier client/server architecture, where the transaction processor supports the application logic (represented by "services" between the GUI front-end and the back-end resource managers. Examples of resource managers are SQL databases, message queues, legacy applications, and other back-end services.

This document explains how to build support binaries for Tuxedo and Virtuoso and how to write services which use the Virtuoso as resource manager.

22.1. Building the Transaction Manager Server

First of all, the administrator needs to build Virtuoso Transaction Manager Server application in order to use Virtuoso server as Resource Manager of global transactions operated by Tuxedo.

Then, the following line needs to be put in the $TUXDIR/udataobj/RM file:

```
Virtuoso:virt_xa_switch: libvirtxa.a -L${HOME}/lib  -lwic -ldk1t -lthrp -lutil2 -
```

The libvirtxa.a library could be found in tuxedo/lib directory.

This allows to build TP monitor with Virtuoso support (VirtTMS) and services which could use Virtuoso as resource manager. libvirtxa.a and other libraries must be accessible by the compiler. The following command builds the VirtTMS:

```
buildtms -o ${TUXDIR}/bin/VirtTMS -r Virtuoso
```

22.2. Configuration

In order to use Virtuoso as resource manager the UBB config file (Tuxedo main configuration file) must contain reference to VirtTMS. The following example of UBB config file configures two services and two virtuoso servers (resource managers) behind them.

```
*RESOURCES
IPCKEY          52617
DOMAINID        qsample
MASTER          NODE
```

```
MODEL           SHM
#
*MACHINES
#
NODE
        LMID = NODE
        TUXDIR = "TUXEDODIR"
        TUXCONFIG = "SERVDIR/tuxconfig"
        TLOGDEVICE ="SERVDIR/tuxconfig"
        TLOGSIZE=10
        APPDIR = "SERVDIR"
        ULOGPFX = "SERVDIR/ULOG"
*GROUPS

DEFAULT:        TMSNAME=VirtTMS TMSCOUNT=2 LMID=NODE
GROUP1  GRPNO=1 OPENINFO = "Virtuoso:dba:dba@NODE:1111"
GROUP2  GRPNO=2 OPENINFO = "Virtuoso:dba@NODE:1112"

*SERVERS
#
DEFAULT:        CLOPT="-A"
VirtRMtest      SRVGRP=GROUP1 SRVID=1
# VirtRMtest    SRVGRP=GROUP2 SRVID=1

# VirtRMtest2   SRVGRP=GROUP1 SRVID=2
VirtRMtest2     SRVGRP=GROUP2 SRVID=2

#server         SRVGRP=GROUP1 SRVID=1
*SERVICES
DEFAULT:        LOAD=50
VRMTEST         PRIO=50
VRMTEST2        PRIO=50
```

22.3. Services

22.3.1. Introduction

The services (in the Tuxedo's term) are special programs which implement business logic. The services could be in the context of a global XA transaction, in this case 2PC control will be set in motion. Each service which uses Virtuoso as resource manager has hdbc connection to the Virtuoso server. This connection is automatically opened when service activated. The connection string (OPENINFO) to the Virtuoso server is the connection string of the group of the service (see GROUPS section in the sample config file). The OPENINFO has the following format: "Virtuoso:user:password@NODENAME:port". The user,password and port are optional.

22.3.2. VQL functions

VQL functions are used to receive access to hdbc for further work with the Virtuoso server. Only HDBCs received by the VQL functions are operated in the context of the distributed transactions.

```
int vql_get_connection (HDBC * hdbc, int type);
```

returns result of setting hdbc to current hdbc connection. "type" argument indicates which hdbc is to select, currently only VQL_CTX_TYPE is supported, other values are reserved.

```
int vql_get_env (HENV * env);
```

returns result of setting current ODBC environment.

Header: vql_client.h

Library: libvirtxa.a

22.3.3. Services concept

Each service has an entry point (some function), which is supposed to perform the application task. The result of whole transaction depends on result of the service's entry function. The scenario of typical workflow is as follows:

- client begins global transaction by ATMI tpbegin() call,
- client calls the service N1 to update some tables on the first Virtuoso server (resource manager),
- client calls the service N2 to update some tables on the second Virtuoso server (resource manager),
- client finishes global transaction by either tpcommit() or tpabort() call of ATMI.

The tx_* functions also could be used, See TUXEDO TxRPC related or ORACLE XA documentation.

- tx_begin()
- tx_commit()
- tx_open()

Services can be built with the following command:

```
buildserver -v -f virt_service1.o -o VirtService1 -r Virtuoso -s VService1
```

where "virt_service1.o" is the service object file which contains VService1 entry function. "-r Virtuoso" indicates that service must be assembled with Virtuoso XA support library.

22.3.4. OPENINFO

OPENINFO is necessary for services to connect to the certain Virtuoso server. OPENINFO for the services (see GROUPS section in the example) consists of 2 parts: "Virtuoso" term and connection string. Connection string provides the service name, password, server and port of Virtuoso server. Common format of connection string is [USER[:PASSWORD]]@SERVER[:PORT]. Only the SERVER name is required, the others are optional.

22.4. Clients

There are no special requirements for the clients of services which use Virtuoso as resource manager. See Tuxedo 8.x documentation

22.5. Service example

```
#include <stdio.h>
#include <time.h>

#include <xa.h>
#include <atmi.h>
#include <userlog.h>

#include "vql_client.h"

#include <libudbc.h>
#ifndef SQL_SUCCEEDED
#define SQL_SUCCEEDED(rc) (((rc)&(~1))==0)
#endif

#define MAXNAME SQL_MAX_DSN_LENGTH

int sql_exec (HDBC hdbc, char* text);

/* VirtRM test service */
int
tpsvrinit(int argc, char *argv[])
{
  if (tpopen () == -1)
    {
      userlog ("tpsrvinit: could not open RM, %s\n", tpstrerror (tperrno));
      return -1;
    }
  /* userlog writes to the central TUXEDO message log */
```

```
  userlog("Welcome to the VirtRMtest server");
  return 0;
}

int
tpsrvdone()
{
  if (tpclose () == -1)
    {
      userlog ("tpsrvinit: could not close RM, %s\n", tpstrerror (tperrno));
      return -1;
    }
  /* userlog writes to the central TUXEDO message log */
  userlog("By!! the VirtRMtest server");
  return 0;
}

static void
DoSQLError (HDBC hdbc, HSTMT hstmt);

#define CHECK_RC(rc) \
        if (!SQL_SUCCEEDED (rc)) { DoSQLError (hdbc, stmt); return -1; };

int sql_exec (HDBC hdbc, char* text)
{
  RETCODE rc;
  HSTMT stmt;

  rc = SQLAllocStmt (hdbc, &stmt);
  CHECK_RC(rc);

  rc = SQLExecDirect (stmt, text, SQL_NTS);
  CHECK_RC(rc);

  SQLFreeStmt (stmt, SQL_DROP);
  return 0;
}

void
VRMTEST(TPSVCINFO *rqst)
{
  HDBC hdbc;
  int rc;

  rc = vql_get_connection (&hdbc, VQL_CTX_TYPE);
  if (rc != VQL_SUCCESS)
    {
#ifdef TMS_TRACE
      userlog ("AP[%d]: vql_get_connection error %d\n", getpid(), rc);
#endif
      goto _fail;
    }
  rc = sql_exec (hdbc, "UPDATE sal set amount = amount + 10 where name = 'jfk'");
  if (rc == -1)
    {
#ifdef TMS_TRACE
      userlog ("AP[%d]: vql_get_connection error %d\n", getpid(), rc);
#endif
      goto _fail;
    }

  userlog ("AP[%d]: exec sql succ.\n", getpid());

  tpreturn(TPSUCCESS, 0, rqst->data, 0L, 0);
  return;

 _fail:
  userlog ("FAILED: exec sql.\n");
  tpreturn(TPFAIL, 0, rqst->data, 0L, 0);
  return;
}

void
```

```
VRMTEST2(TPSVCINFO *rqst)
{
  HDBC hdbc;
  int rc;

  rc = vql_get_connection (&hdbc, VQL_CTX_TYPE);
  if (rc != VQL_SUCCESS)
    goto _fail;

  rc = sql_exec (hdbc, "UPDATE sal set amount = amount - 10 where name = 'jfk'");
  if (rc == -1)
    goto _fail;

  userlog ("AP[%d]: exec sql succ.\n", getpid());

  tpreturn(TPSUCCESS, 0, rqst->data, 0L, 0);
  return;

 _fail:
  userlog ("AP[%d]: service #2 error %d\n", getpid(), rc);
  tpreturn(TPFAIL, 0, rqst->data, 0L, 0);
  return;

  tpreturn(TPSUCCESS, 0, rqst->data, 0L, 0);
}

#define MSG_BUF_SIZE 300

static void
DoSQLError (HDBC hdbc, HSTMT hstmt)
{
  UCHAR szSqlState[MSG_BUF_SIZE];
  UCHAR szErrorMsg[MSG_BUF_SIZE];

  SQLINTEGER fNativeError = 0;
  SWORD cbErrorMsg = MSG_BUF_SIZE;
  RETCODE rc;
  HENV env;

#ifndef TMS_TRACE
  return;
#else
  vql_get_env (&env);

  rc = SQLError (env,
      hdbc,
      hstmt,
      szSqlState, &fNativeError, szErrorMsg, MSG_BUF_SIZE, &cbErrorMsg);

  if (rc != SQL_NO_DATA_FOUND || rc != SQL_ERROR)
    {
      if (fNativeError != 0x1645)       // ignore change database to master context message
        {
          userlog ("SqlState: %s, fNativeError: %x m=%s\n", szSqlState,
                   fNativeError, szErrorMsg);
        }
    }
  else
    {
      userlog ("SQLError() failed: %x, NO_DATA_FOUND OR SQL_ERROR\n", rc);
    }
  return;
#endif
}
```

Chapter 23. Appendix A

Abstract

This reference chapter contains information that needs to be found quickly and accurately without specific guide lines to the topics in question. This chapter contains the SQL Grammar, listings of all system errors messages, release notes for the product, how to get support on the product and a list of the system tables and there schema definitions.

23.1. YACC SQL Grammar Reference

```
/*
 *
 *  SQL Parser
 *
 *  Copyright (C) 1998-2024 OpenLink Software.
 *  All Rights Reserved.
 *
 *  The copyright above and this notice must be preserved in all
 *  copies of this source code.  The copyright above does not
 *  evidence any actual or intended publication of this source code.
 *
 *  This is unpublished proprietary trade secret of OpenLink Software.
 *  This source code may not be copied, disclosed, distributed, demonstrated
 *  or licensed except as authorized by OpenLink Software.
 */

Grammar

    0 $accept: sql_list $end

    1 sql_list: sql_list1 ';'
    2         | sql_list1

    3 sql_list1: sql
```

```
sql: schema_element_list
   | view_def
   | xml_view
   | create_xml_schema
   | alter_constraint
   | create_library
   | create_assembly
   | drop_library
   | drop_assembly

schema_element_list: schema_element
                   | add_column
                   | schema_element_list schema_element
                   | schema_element_list add_column

schema_element: base_table_def
              | create_index_def
              | drop_table
              | drop_index
              | table_rename
              | privilege_def
              | privilege_revoke
              | create_user_statement
              | delete_user_statement
              | set_pass
              | set_group_stmt
              | add_group_stmt
              | delete_group_stmt
              | user_defined_type
              | user_defined_type_drop
              | user_defined_type_alter

identifier: NAME
          | TYPE
          | FINAL_L
          | METHOD
          | CHECKED
          | SYSTEM
          | GENERATED
          | SOURCE
          | RESULT
          | LOCATOR
          | INSTANCE_L
          | CONSTRUCTOR
          | SELF_L
          | OVERRIDING
          | STYLE
          | SQL_L
          | GENERAL
          | DETERMINISTIC
          | NO
          | CONTAINS
          | READS
          | DATA
          | MODIFIES
          | INPUT
          | CALLED
          | ADA
          | C
          | COBOL
          | FORTRAN
          | MUMPS
          | PASCAL_L
          | PLI
          | NAME_L
          | TEXT_L
          | JAVA
          | INOUT_L
          | REMOTE
          | KEYSET
          | VALUE
          | PARAMETER
```

```
          | VARIABLE
          | CLR
          | TEMPORARY
          | ADMIN_L
          | __SOAP_DOC
          | __SOAP_DOCW
          | __SOAP_HEADER
          | __SOAP_HTTP
          | __SOAP_NAME
          | __SOAP_TYPE
          | __SOAP_XML_TYPE
          | __SOAP_FAULT
          | __SOAP_DIME_ENC
          | __SOAP_ENC_MIME
          | __SOAP_OPTIONS
          | START_L
          | ATTRIBUTE
          | REXECUTE
          | PERMISSION_SET
          | AUTOREGISTER_L
          | LIBRARY_L
          | ASSEMBLY_L
          | SAFE_L
          | UNRESTRICTED
          | INCREMENT_L
          | FOREACH

base_table_def: CREATE TABLE new_table_name '(' base_table_element_commalist ')'

base_table_element_commalist: base_table_element
                            | base_table_element_commalist ',' base_table_element

base_table_element: column_def
                  | table_constraint_def

column_def: column column_data_type column_def_opt_list

opt_referential_triggered_action: /* empty */
                                | referential_rule
                                | referential_rule referential_rule

referential_rule: ON UPDATE referential_action
                | delete_referential_rule

delete_referential_rule: ON DELETE_L referential_action

opt_on_delete_referential_rule: /* empty */
                              | delete_referential_rule

referential_action: CASCADE
                  | SET NULLX
                  | SET DEFAULT

references: REFERENCES q_table_name opt_column_commalist opt_referential_triggered_action

column_def_opt_list: /* empty */
                   | column_def_opt_list column_def_opt

identity_opt: START_L WITH signed_literal
            | INCREMENT_L BY INTNUM

identity_opt_list: identity_opt
                 | identity_opt_list ',' identity_opt

column_def_opt: NOT NULLX
              | NULLX
              | IDENTITY
              | IDENTITY '(' identity_opt_list ')'
              | PRIMARY KEY
              | DEFAULT signed_literal
              | COLLATE q_table_name
              | references
              | IDENTIFIED BY column
```

```
132               | CHECK '(' search_condition ')'
133               | WITH SCHEMA column_xml_schema_def
134               | UNIQUE

135 column_xml_schema_def: '(' STRING ',' STRING ')'
136                      | '(' STRING ',' STRING ',' STRING ')'

137 table_constraint_def: UNDER q_table_name
138                     | opt_constraint_name PRIMARY KEY '(' index_column_commalist ')' opt_index_opti
139                     | opt_constraint_name FOREIGN KEY '(' column_commalist ')' references
140                     | opt_constraint_name CHECK '(' search_condition ')'
141                     | opt_constraint_name UNIQUE '(' column_commalist ')'

142 opt_constraint_name: /* empty */
143                    | CONSTRAINT identifier

144 column_commalist: column
145                | column_commalist ',' column

146 index_column_commalist: column opt_asc_desc
147                       | index_column_commalist ',' column opt_asc_desc

148 index_option: CLUSTERED
149             | UNIQUE
150             | OBJECT_ID

151 index_option_list: index_option
152                  | index_option_list index_option

153 opt_index_option_list: /* empty */
154                      | index_option_list

155 create_index_def: CREATE opt_index_option_list INDEX index ON new_table_name '(' index_column_comma

156 drop_index: DROP INDEX identifier opt_table

157 opt_table: /* empty */
158          | q_table_name

159 drop_table: DROP TABLE q_table_name
160           | DROP VIEW q_table_name

161 opt_col_add_column: /* empty */
162                  | COLUMN

163 add_col_column_def_list: column_def
164                        | add_col_column_def_list ',' column_def

165 add_col_column_list: column
166                   | add_col_column_list ',' column

167 add_column: ALTER TABLE q_table_name ADD opt_col_add_column add_col_column_def_list
168           | ALTER TABLE q_table_name DROP opt_col_add_column add_col_column_list
169           | ALTER TABLE q_table_name MODIFY opt_col_add_column column_def

170 table_rename: ALTER TABLE q_table_name RENAME new_table_name

171 constraint_op: ADD
172              | DROP
173              | MODIFY

174 opt_drop_behavior: /* empty */
175                  | CASCADE
176                  | RESTRICT

177 opt_table_constraint_def: CONSTRAINT identifier opt_drop_behavior
178                         | table_constraint_def

179 alter_constraint: ALTER TABLE q_table_name constraint_op opt_table_constraint_def

180 create_xml_schema: CREATE XML SCHEMA STRING

181 view_query_spec: query_exp
```

```
182                | query_no_from_spec

183 @1: /* empty */

184 view_def: CREATE VIEW new_table_name @1 opt_column_commalist AS view_query_spec opt_with_check_opti
185         | CREATE PROCEDURE VIEW new_table_name AS q_table_name '(' column_commalist_or_empty ')' '(

186 opt_with_check_option: /* empty */
187                      | WITH CHECK OPTION

188 opt_column_commalist: /* empty */
189                      | '(' column_commalist ')'

190 priv_opt_column_commalist: /* empty */
191                          | '(' column_commalist ')'

192 privilege_def: GRANT ALL PRIVILEGES TO grantee
193              | GRANT privileges ON table TO grantee_commalist opt_with_grant_option
194              | GRANT EXECUTE ON function_name TO grantee_commalist opt_with_grant_option
195              | GRANT REXECUTE ON STRING TO grantee_commalist
196              | GRANT UNDER ON q_old_type_name TO grantee_commalist opt_with_grant_option
197              | GRANT grantee_commalist TO grantee_commalist opt_with_admin_option

198 opt_with_admin_option: /* empty */
199                      | WITH ADMIN_L OPTION

200 privilege_revoke: REVOKE privileges ON table FROM grantee_commalist
201                 | REVOKE EXECUTE ON function_name FROM grantee_commalist
202                 | REVOKE UNDER ON q_old_type_name FROM grantee_commalist
203                 | REVOKE REXECUTE ON STRING FROM grantee_commalist
204                 | REVOKE grantee_commalist FROM grantee_commalist

205 opt_with_grant_option: /* empty */
206                      | WITH GRANT OPTION

207 privileges: ALL PRIVILEGES
208           | ALL
209           | operation_commalist

210 operation_commalist: operation
211                     | operation_commalist ',' operation

212 operation: SELECT priv_opt_column_commalist
213          | INSERT
214          | DELETE_L
215          | UPDATE priv_opt_column_commalist
216          | REFERENCES priv_opt_column_commalist

217 grantee_commalist: grantee
218                  | grantee_commalist ',' grantee

219 grantee: PUBLIC
220        | user

221 set_pass: SET PASSWORD identifier identifier

222 create_user_statement: CREATE USER user
223                      | CREATE ROLE_L user

224 delete_user_statement: DELETE_L USER user
225                      | DELETE_L USER user CASCADE
226                      | DROP USER user
227                      | DROP USER user CASCADE
228                      | DROP ROLE_L user

229 set_group_stmt: SET USER GROUP user user

230 add_group_stmt: ADD USER GROUP user user

231 delete_group_stmt: DELETE_L USER GROUP user user

232 opt_attach_primary_key: /* empty */
233                       | PRIMARY KEY '(' column_commalist ')'
```

```
234 attach_table: ATTACH TABLE attach_q_table_name opt_attach_primary_key opt_as FROM literal opt_login

235 opt_as: /* empty */
236       | AS new_table_name

237 opt_login: /* empty */
238          | USER scalar_exp PASSWORD scalar_exp

239 opt_not_select: /* empty */
240               | NOT SELECT

241 opt_remote_name: /* empty */
242                | REMOTE AS scalar_exp

243 cursor_type: STATIC_L
244            | DYNAMIC
245            | KEYSET

246 cursor_def: DECLARE identifier CURSOR FOR query_exp
247           | DECLARE identifier cursor_type CURSOR FOR query_exp

248 opt_order_by_clause: /* empty */
249                    | ORDER BY ordering_spec_commalist

250 ordering_spec_commalist: ordering_spec
251                        | ordering_spec_commalist ',' ordering_spec

252 ordering_spec: scalar_exp opt_asc_desc
253              | mssql_xml_col opt_asc_desc

254 opt_asc_desc: /* empty */
255             | ASC
256             | DESC

257 create_snapshot_log: CREATE SNAPSHOT LOGX FOR q_table_name

258 drop_snapshot_log: DROP SNAPSHOT LOGX FOR q_table_name

259 purge_snapshot_log: PURGE SNAPSHOT LOGX FOR q_table_name

260 opt_snapshot_string_literal: /* empty */
261                            | STRING

262 opt_snapshot_where_clause: /* empty */
263                          | WHERE STRING

264 create_snapshot: CREATE SNAPSHOT q_table_name FROM q_table_name opt_snapshot_string_literal opt_sna
265                | CREATE NONINCREMENTAL SNAPSHOT q_table_name AS STRING

266 opt_with_delete: /* empty */
267                | WITH DELETE_L

268 drop_snapshot: DROP SNAPSHOT q_table_name opt_with_delete

269 opt_nonincremental: /* empty */
270                   | AS NONINCREMENTAL

271 refresh_snapshot: UPDATE SNAPSHOT q_table_name opt_nonincremental

272 create_freetext_index: CREATE TEXT_L opt_xml INDEX ON q_table_name '(' column ')' opt_with_key opt_

273 opt_data_modification_action: /* empty */
274                            | USING FUNCTION

275 opt_column: /* empty */
276           | '(' column ')'

277 create_freetext_trigger: CREATE TEXT_L TRIGGER ON q_table_name opt_column

278 drop_freetext_trigger: DROP TEXT_L TRIGGER ON q_table_name opt_column

279 opt_xml: /* empty */
```

```
280         | XML

281 opt_with_key: /* empty */
282             | WITH KEY column

283 opt_with: /* empty */
284         | CLUSTERED WITH '(' column_commalist ')'

285 opt_lang: /* empty */
286         | LANGUAGE STRING

287 opt_enc: /* empty */
288         | ENCODING STRING

289 opt_deffer_generation: /* empty */
290                       | NOT INSERT

291 sql: manipulative_statement

292 manipulative_statement: query_exp
293                       | query_no_from_spec
294                       | update_statement_positioned
295                       | update_statement_searched
296                       | insert_statement
297                       | delete_statement_positioned
298                       | delete_statement_searched
299                       | call_statement
300                       | static_method_invocation
301                       | METHOD CALL static_method_invocation
302                       | top_level_method_invocation
303                       | set_statement
304                       | drop_xml_view
305                       | commit_statement
306                       | rollback_statement
307                       | admin_statement
308                       | use_statement
309                       | attach_table
310                       | create_snapshot_log
311                       | drop_snapshot_log
312                       | purge_snapshot_log
313                       | create_snapshot
314                       | drop_snapshot
315                       | refresh_snapshot
316                       | create_freetext_index
317                       | create_freetext_trigger
318                       | drop_freetext_trigger

319 use_statement: USE identifier

320 close_statement: CLOSE cursor

321 commit_statement: COMMIT WORK

322 delete_statement_positioned: DELETE_L FROM table WHERE CURRENT OF cursor

323 delete_statement_searched: DELETE_L FROM table opt_where_clause

324 fetch_statement: FETCH cursor INTO target_commalist
325                | FETCH cursor NAME INTO target_commalist
326                | FETCH cursor NAME scalar_exp INTO target_commalist

327 insert_mode: INTO
328            | REPLACING
329            | SOFT

330 insert_statement: INSERT insert_mode table priv_opt_column_commalist values_or_query_spec

331 values_or_query_spec: VALUES '(' insert_atom_commalist ')'
332                     | query_spec

333 insert_atom_commalist: insert_atom
334                      | insert_atom_commalist ',' insert_atom
```

```
335 insert_atom: scalar_exp

336 sql_option: ORDER
337           | HASH
338           | LOOP
339           | INDEX identifier
340           | INDEX PRIMARY KEY
341           | INDEX TEXT_L KEY
342           | NAME INTNUM

343 sql_opt_commalist: sql_option
344                  | sql_opt_commalist ',' sql_option

345 opt_sql_opt: /* empty */
346            | OPTION '(' sql_opt_commalist ')'

347 opt_table_opt: /* empty */
348              | TABLE OPTION '(' sql_opt_commalist ')'

349 cursor_option: EXCLUSIVE
350              | PREFETCH INTNUM

351 cursor_options_commalist: cursor_option
352                        | cursor_options_commalist ',' cursor_option

353 opt_cursor_options_list: /* empty */
354                       | '(' cursor_options_commalist ')'

355 open_statement: OPEN cursor opt_cursor_options_list

356 rollback_statement: ROLLBACK WORK

357 with_opt_cursor_options_list: /* empty */
358                             | WITH opt_cursor_options_list

359 select_statement: SELECT opt_top selection INTO target_commalist table_exp with_opt_cursor_options_

360 opt_all_distinct: /* empty */
361                | ALL
362                | DISTINCT

363 opt_ties: /* empty */
364         | WITH TIES

365 opt_top: opt_all_distinct
366        | opt_all_distinct TOP INTNUM opt_ties
367        | opt_all_distinct TOP '(' scalar_exp ')' opt_ties
368        | opt_all_distinct TOP INTNUM ',' INTNUM opt_ties
369        | opt_all_distinct TOP '(' scalar_exp ',' scalar_exp ')' opt_ties

370 update_statement_positioned: UPDATE table SET assignment_commalist WHERE CURRENT OF cursor

371 assignment_commalist: /* empty */
372                     | assignment
373                     | assignment_commalist ',' assignment

374 assignment: column COMPARISON scalar_exp

375 update_statement_searched: UPDATE table SET assignment_commalist opt_where_clause

376 target_commalist: target
377                | target_commalist ',' target

378 target: column_ref
379       | member_observer
380       | lvalue_array_ref

381 opt_where_clause: /* empty */
382                | where_clause

383 opt_best: /* empty */
384         | BEST
```

```
query_exp: query_term
         | non_final_union_exp opt_best UNION opt_corresponding query_term
         | non_final_union_exp opt_best UNION ALL opt_corresponding query_term
         | non_final_union_exp INTERSECT opt_corresponding query_term
         | non_final_union_exp INTERSECT ALL opt_corresponding query_term
         | non_final_union_exp EXCEPT opt_corresponding query_term
         | non_final_union_exp EXCEPT ALL opt_corresponding query_term

non_final_union_exp: non_final_query_term
                   | non_final_union_exp opt_best UNION opt_corresponding non_final_query_term
                   | non_final_union_exp opt_best UNION ALL opt_corresponding non_final_query_term
                   | non_final_union_exp INTERSECT opt_corresponding non_final_query_term
                   | non_final_union_exp INTERSECT ALL opt_corresponding non_final_query_term
                   | non_final_union_exp EXCEPT opt_corresponding non_final_query_term
                   | non_final_union_exp EXCEPT ALL opt_corresponding non_final_query_term

non_final_query_term: non_final_query_spec
                    | XPATH STRING

query_term: query_spec
          | '(' query_exp ')' opt_order_by_clause
          | XPATH STRING

opt_corresponding: /* empty */
                 | CORRESPONDING BY '(' column_commalist ')'

non_final_query_spec: SELECT opt_top selection non_final_table_exp

query_spec: SELECT opt_top selection table_exp

query_no_from_spec: SELECT opt_top selection

selection: select_scalar_exp_commalist

non_final_table_exp: from_clause opt_where_clause opt_group_by_clause opt_having_clause

table_exp: from_clause opt_where_clause opt_group_by_clause opt_having_clause opt_order_by_clause o

from_clause: FROM table_ref_commalist

table_ref_commalist: table_ref
                   | table_ref_commalist ',' table_ref

proc_col_list: column_def
             | proc_col_list ',' column_def

opt_proc_col_list: '(' proc_col_list ')'

column_commalist_or_empty: /* empty */
                         | column_commalist

table_ref: table
         | '(' query_exp ')' identifier
         | '(' query_exp ')' AS identifier
         | joined_table
         | q_table_name '(' column_commalist_or_empty ')' opt_proc_col_list identifier

table_ref_nj: table
            | subquery identifier
            | subquery AS identifier
            | '(' joined_table ')'

jtype: /* empty */
     | LEFT opt_outer
     | RIGHT opt_outer
     | FULL opt_outer
     | INNER
     | CROSS

opt_outer: /* empty */
         | OUTER

join: NATURAL jtype
```

```
438     | jtype

439 joined_table: joined_table_1
440             | BEGIN_OJ_X joined_table_1 ENDX
441             | '(' joined_table_1 ')'

442 joined_table_1: table_ref join JOIN table_ref_nj join_condition

443 join_condition: /* empty */
444               | ON search_condition
445               | USING '(' column_commalist ')'

446 where_clause: WHERE search_condition

447 opt_group_by_clause: /* empty */
448                     | GROUP BY ordering_spec_commalist
449                     | GROUP BY ROLLUP '(' ordering_spec_commalist ')'
450                     | GROUP BY CUBE '(' ordering_spec_commalist ')'

451 opt_having_clause: /* empty */
452                   | HAVING search_condition

453 opt_lock_mode: /* empty */
454              | FOR UPDATE
455              | FOR XML NAME
456              | FOR XML NAME NAME

457 search_condition: search_condition OR search_condition
458                 | search_condition AND search_condition
459                 | NOT search_condition
460                 | '(' search_condition ')'
461                 | predicate

462 predicate: comparison_predicate
463          | between_predicate
464          | like_predicate
465          | test_for_null
466          | in_predicate
467          | all_or_any_predicate
468          | existence_test
469          | scalar_exp_predicate

470 scalar_exp_predicate: scalar_exp

471 comparison_predicate: scalar_exp COMPARISON scalar_exp

472 between_predicate: scalar_exp NOT BETWEEN scalar_exp AND scalar_exp
473                 | scalar_exp BETWEEN scalar_exp AND scalar_exp

474 like_predicate: scalar_exp NOT LIKE scalar_exp opt_escape
475               | scalar_exp LIKE scalar_exp opt_escape

476 opt_escape: /* empty */
477           | ESCAPE atom
478           | BEGINX ESCAPE atom ENDX

479 test_for_null: scalar_exp IS NOT NULLX
480              | scalar_exp IS NULLX

481 in_predicate: scalar_exp NOT IN_L subquery
482             | scalar_exp IN_L subquery
483             | scalar_exp NOT IN_L '(' scalar_exp_commalist ')'
484             | scalar_exp IN_L '(' scalar_exp_commalist ')'

485 all_or_any_predicate: scalar_exp COMPARISON any_all_some subquery

486 any_all_some: ANY
487             | ALL
488             | SOME

489 existence_test: EXISTS subquery

490 scalar_subquery: subquery
```

```
subquery: '(' SELECT opt_top selection table_exp ')'

scalar_exp: scalar_exp '-' scalar_exp
          | scalar_exp '+' scalar_exp
          | scalar_exp '*' scalar_exp
          | scalar_exp '/' scalar_exp
          | '+' scalar_exp
          | '-' scalar_exp
          | assignment_statement
          | string_concatenation_operator
          | column_ref
          | scalar_exp_no_col_ref
          | obe_literal

scalar_exp_no_col_ref: atom_no_obe
                     | aggregate_ref
                     | scalar_subquery
                     | '(' scalar_exp ')'
                     | '(' scalar_exp ',' scalar_exp_commalist ')'
                     | function_call
                     | new_invocation
                     | cvt_exp
                     | cast_exp
                     | simple_case
                     | searched_case
                     | coalesce_exp
                     | nullif_exp
                     | array_ref
                     | static_method_invocation
                     | method_invocation
                     | member_observer

scalar_exp_no_col_ref_no_mem_obs_chain: atom_no_obe
                                      | aggregate_ref
                                      | scalar_subquery
                                      | '(' scalar_exp ')'
                                      | '(' scalar_exp ',' scalar_exp_commalist ')'
                                      | function_call
                                      | new_invocation
                                      | cvt_exp
                                      | cast_exp
                                      | simple_case
                                      | searched_case
                                      | coalesce_exp
                                      | nullif_exp
                                      | array_ref
                                      | static_method_invocation
                                      | method_invocation
                                      | member_observer_no_id_chain

cvt_exp: CONVERT '(' data_type ',' scalar_exp ')'

opt_collate_exp: /* empty */
               | COLLATE q_table_name

cast_exp: CAST '(' scalar_exp AS data_type opt_collate_exp ')'

mssql_xml_col: MSSQL_XMLCOL_NAME1 MSSQL_XMLCOL_INTNUM MSSQL_XMLCOL_NAMEZ
             | MSSQL_XMLCOL_NAME1 MSSQL_XMLCOL_INTNUM MSSQL_XMLCOL_NAMEYZ
             | MSSQL_XMLCOL_NAME1 MSSQL_XMLCOL_INTNUM MSSQL_XMLCOL_NAME MSSQL_XMLCOL_NAMEZ

as_expression: scalar_exp AS identifier data_type
             | scalar_exp AS identifier
             | scalar_exp identifier
             | scalar_exp AS mssql_xml_col

array_ref: scalar_exp_no_col_ref '[' scalar_exp ']'
         | lvalue_array_ref

lvalue_array_ref: column_ref '[' scalar_exp ']'

opt_scalar_exp_commalist: /* empty */
```

```
552                       | scalar_exp_commalist

553 function_name: identifier
554              | identifier '.' method_identifier
555              | identifier '.' identifier '.' method_identifier
556              | identifier '.' identifier '.' identifier '.' method_identifier
557              | identifier '.' '.' method_identifier
558              | identifier '.' '.' identifier '.' method_identifier
559              | LEFT
560              | RIGHT
561              | LOGX

562 kwd_commalist: identifier KWD_TAG scalar_exp
563              | kwd_commalist ',' identifier KWD_TAG scalar_exp

564 as_commalist: as_expression
565             | as_commalist ',' as_expression
566             | as_commalist ',' scalar_exp

567 opt_arg_commalist: /* empty */
568                  | kwd_commalist
569                  | scalar_exp_commalist
570                  | scalar_exp_commalist ',' kwd_commalist
571                  | scalar_exp_commalist ',' as_commalist
572                  | as_commalist

573 function_call: function_name '(' opt_arg_commalist ')'
574              | TIMESTAMP_FUNC '(' SQL_TSI ',' scalar_exp ',' scalar_exp ')'
575              | EXTRACT '(' NAME FROM scalar_exp ')'
576              | BEGIN_FN_X identifier '(' opt_scalar_exp_commalist ')' ENDX
577              | BEGIN_FN_X LEFT '(' opt_scalar_exp_commalist ')' ENDX
578              | BEGIN_FN_X RIGHT '(' opt_scalar_exp_commalist ')' ENDX
579              | BEGIN_FN_X LOGX '(' opt_scalar_exp_commalist ')' ENDX
580              | BEGIN_FN_X identifier '(' scalar_exp IN_L scalar_exp ')' ENDX
581              | BEGIN_CALL_X function_name '(' opt_scalar_exp_commalist ')' ENDX
582              | BEGIN_CALL_X function_name ENDX
583              | BEGIN_FN_X USER '(' opt_scalar_exp_commalist ')' ENDX
584              | BEGIN_FN_X CHARACTER '(' opt_scalar_exp_commalist ')' ENDX
585              | BEGIN_FN_X TIMESTAMP_FUNC '(' SQL_TSI ',' scalar_exp ',' scalar_exp ')' ENDX
586              | BEGIN_FN_X CONVERT '(' scalar_exp ',' NAME ')' ENDX
587              | BEGIN_FN_X EXTRACT '(' NAME FROM scalar_exp ')' ENDX
588              | CALL '(' scalar_exp ')' '(' opt_arg_commalist ')'
589              | CURRENT_DATE
590              | CURRENT_TIME
591              | CURRENT_TIME '(' scalar_exp ')'
592              | CURRENT_TIMESTAMP
593              | CURRENT_TIMESTAMP '(' scalar_exp ')'
594              | GROUPING '(' column_ref ')'

595 sql: BEGIN_EQCALL_X q_table_name ENDX
596    | BEGIN_EQCALL_X q_table_name '(' opt_scalar_exp_commalist ')' ENDX

597 obe_literal: BEGINX identifier atom ENDX
598            | BEGIN_U_X STRING ENDX

599 scalar_exp_commalist: scalar_exp
600                     | scalar_exp_commalist ',' scalar_exp

601 select_scalar_exp_commalist: scalar_exp
602                           | as_expression
603                           | select_scalar_exp_commalist ',' scalar_exp
604                           | select_scalar_exp_commalist ',' as_expression

605 atom_no_obe: parameter_ref
606            | literal
607            | USER

608 atom: atom_no_obe
609     | obe_literal

610 simple_case: CASE scalar_exp simple_when_list ENDX

611 searched_case: CASE searched_when_list ENDX
```

```
612 searched_when_list: searched_when
613                   | searched_when_list searched_when

614 simple_when_list: simple_when
615                 | simple_when_list simple_when

616 simple_when: WHEN scalar_exp THEN scalar_exp
617            | ELSE scalar_exp

618 searched_when: WHEN search_condition THEN scalar_exp
619              | ELSE scalar_exp

620 coalesce_exp: COALESCE '(' scalar_exp_commalist ')'

621 nullif_exp: NULLIF '(' scalar_exp ',' scalar_exp ')'

622 parameter_ref: parameter
623              | parameter parameter
624              | parameter INDICATOR parameter

625 aggregate_ref: AGGREGATE function_name '(' opt_arg_commalist ')'
626              | AMMSC '(' DISTINCT scalar_exp ')'
627              | AMMSC '(' ALL scalar_exp ')'
628              | AMMSC '(' scalar_exp ')'

629 literal: STRING
630        | WSTRING
631        | INTNUM
632        | APPROXNUM
633        | BINARYNUM
634        | NULLX

635 signed_literal: STRING
636               | WSTRING
637               | INTNUM
638               | '-' INTNUM
639               | '+' INTNUM
640               | APPROXNUM
641               | '-' APPROXNUM
642               | '+' APPROXNUM
643               | BINARYNUM
644               | NULLX

645 q_table_name: identifier
646             | identifier '.' identifier
647             | identifier '.' identifier '.' identifier
648             | identifier '.' '.' identifier

649 attach_q_table_name: identifier
650                   | identifier '.' identifier
651                   | identifier '.' identifier '.' identifier
652                   | identifier '.' '.' identifier

653 new_proc_or_bif_name: identifier
654                     | identifier '.' identifier
655                     | identifier '.' identifier '.' identifier
656                     | identifier '.' '.' identifier

657 new_table_name: identifier
658               | identifier '.' identifier
659               | identifier '.' identifier '.' identifier
660               | identifier '.' '.' identifier

661 table: q_table_name opt_table_opt
662      | q_table_name AS identifier opt_table_opt
663      | q_table_name identifier opt_table_opt

664 column_ref: identifier
665           | identifier '.' identifier
666           | identifier '.' identifier '.' identifier
667           | identifier '.' identifier '.' identifier '.' identifier
668           | identifier '.' '.' identifier '.' identifier
```

```
669           | '*'
670           | identifier '.' '*'
671           | identifier '.' identifier '.' '*'
672           | identifier '.' identifier '.' identifier '.' '*'
673           | identifier '.' '.' identifier '.' '*'

674 base_data_type: NUMERIC
675               | NUMERIC '(' INTNUM ')'
676               | NUMERIC '(' INTNUM ',' INTNUM ')'
677               | DECIMAL
678               | DECIMAL '(' INTNUM ')'
679               | DECIMAL '(' INTNUM ',' INTNUM ')'
680               | INTEGER
681               | SMALLINT
682               | FLOAT
683               | FLOAT '(' INTNUM ')'
684               | REAL
685               | DOUBLE PRECISION
686               | LONG VARCHAR
687               | LONG VARBINARY
688               | VARBINARY
689               | VARBINARY '(' INTNUM ')'
690               | BINARY '(' INTNUM ')'
691               | TIMESTAMP
692               | DATETIME
693               | TIME
694               | DATE
695               | NCHAR
696               | NCHAR '(' INTNUM ')'
697               | NVARCHAR
698               | NVARCHAR '(' INTNUM ')'
699               | LONG NVARCHAR
700               | ANY
701               | ANY '(' INTNUM ')'

702 data_type: base_data_type
703          | CHARACTER
704          | VARCHAR
705          | VARCHAR '(' INTNUM ')'
706          | CHARACTER '(' INTNUM ')'

707 array_modifier: ARRAY
708               | ARRAY '[' INTNUM ']'

709 data_type_ref: data_type_ref array_modifier
710              | data_type
711              | q_type_name

712 column_data_type: base_data_type
713                 | CHARACTER
714                 | VARCHAR
715                 | VARCHAR '(' INTNUM ')'
716                 | CHARACTER '(' INTNUM ')'
717                 | q_type_name
718                 | LONG q_type_name
719                 | LONG XML

720 column: identifier
721       | identifier '.' identifier '.' identifier '.' identifier

722 index: identifier

723 cursor: identifier

724 parameter: PARAMETER_L
725          | NAMED_PARAMETER

726 user: identifier

727 opt_log: /* empty */
728        | STRING

729 comma_opt_log: /* empty */
```

```
730             | ',' STRING

731 admin_statement: SHUTDOWN opt_log
732                | CHECKPOINT opt_log
733                | CHECKPOINT STRING STRING
734                | BACKUP STRING
735                | CHECK
736                | SYNC REPLICATION opt_log comma_opt_log
737                | DISCONNECT REPLICATION opt_log
738                | LOGX ON
739                | LOGX OFF

740 sql: user_aggregate_declaration
741    | routine_declaration
742    | module_declaration
743    | method_declaration
744    | trigger_def
745    | drop_trigger
746    | drop_proc

747 user_aggregate_declaration: CREATE AGGREGATE new_table_name rout_parameter_list opt_return FROM new

748 user_aggregate_merge_opt: /* empty */
749                          | ',' new_proc_or_bif_name

750 routine_declaration: CREATE routine_head new_table_name rout_parameter_list opt_return rout_alt_typ
751                    | ATTACH routine_head attach_q_table_name rout_parameter_list opt_return rout_al
752                    | CREATE routine_head new_table_name rout_parameter_list opt_return rout_alt_typ

753 module_body_part: routine_head identifier rout_parameter_list opt_return rout_alt_type compound_sta

754 module_body: module_body_part ';'
755            | module_body module_body_part ';'

756 module_declaration: CREATE MODULE new_table_name BEGINX module_body ENDX

757 routine_head: FUNCTION
758             | PROCEDURE

759 opt_return: /* empty */
760           | RETURNS data_type_ref

761 rout_parameter_list: '(' ')'
762                    | '(' parameter_commalist ')'

763 parameter_commalist: rout_parameter
764                    | parameter_commalist ',' rout_parameter

765 rout_parameter: parameter_mode column_ref data_type_ref rout_alt_type
766               | parameter_mode column_ref data_type_ref DEFAULT signed_literal rout_alt_type
767               | parameter_mode column_ref data_type_ref EQUALS signed_literal rout_alt_type

768 parameter_mode: IN_L
769               | OUT_L
770               | INOUT_L

771 opt_parameter_mode: /* empty */
772                    | parameter_mode

773 opt_soap_enc_mode: /* empty */
774                   | __SOAP_DIME_ENC IN_L
775                   | __SOAP_DIME_ENC OUT_L
776                   | __SOAP_DIME_ENC INOUT_L
777                   | __SOAP_ENC_MIME IN_L
778                   | __SOAP_ENC_MIME OUT_L
779                   | __SOAP_ENC_MIME INOUT_L

780 soap_proc_opt_list: soap_proc_opt
781                    | soap_proc_opt_list ',' soap_proc_opt

782 soap_proc_opt: NAME EQUALS signed_literal

783 soap_kwd: __SOAP_TYPE
```

```
784         | __SOAP_HEADER
785         | __SOAP_FAULT
786         | __SOAP_DOC
787         | __SOAP_XML_TYPE
788         | __SOAP_DOCW
789         | __SOAP_HTTP

790 rout_alt_type: /* empty */
791              | __SOAP_OPTIONS '(' soap_kwd EQUALS STRING opt_soap_enc_mode ',' soap_proc_opt_list '
792              | soap_kwd STRING opt_soap_enc_mode

793 routine_statement: select_statement
794                   | update_statement_positioned
795                   | update_statement_searched
796                   | insert_statement
797                   | delete_statement_positioned
798                   | delete_statement_searched
799                   | close_statement
800                   | fetch_statement
801                   | open_statement
802                   | rollback_statement
803                   | commit_statement
804                   | /* empty */

805 @2: /* empty */

806 compound_statement: BEGINX @2 statement_list ENDX

807 statement_list: statement_in_cs
808               | statement_list statement_in_cs

809 statement_in_cs: local_declaration ';'
810                | compound_statement

811 @3: /* empty */

812 statement_in_cs: @3 statement_in_cs_oper

813 statement_in_cs_oper: routine_statement ';'
814                      | control_statement
815                      | identifier COLON statement_in_cs
816                      | HTMLSTR
817                      | COMPARISON scalar_exp HTMLSTR
818                      | '/' scalar_exp HTMLSTR

819 statement: compound_statement

820 @4: /* empty */

821 statement: @4 routine_statement ';'

822 @5: /* empty */

823 statement: @5 control_statement

824 local_declaration: cursor_def
825                   | variable_declaration
826                   | handler_declaration

827 variable_declaration: DECLARE variable_list data_type_ref

828 variable_list: identifier
829              | variable_list ',' identifier

830 condition: NOT FOUND
831          | SQLSTATE STRING
832          | SQLSTATE VALUE STRING
833          | SQLEXCEPTION
834          | SQLWARNING

835 handler_statement: compound_statement
836                   | routine_statement
837                   | call_statement
```

```
838                   | method_invocation
839                   | static_method_invocation
840                   | set_statement
841                   | RESIGNAL
842                   | RESIGNAL scalar_exp
843                   | return_statement
844                   | assignment_statement
845                   | if_statement
846                   | goto_statement
847                   | for_statement
848                   | while_statement

849 handler_declaration: WHENEVER condition GOTO identifier
850                     | WHENEVER condition GO TO identifier
851                     | WHENEVER condition DEFAULT
852                     | DECLARE handler_type HANDLER FOR cond_value_list handler_statement

853 handler_type: CONTINUE
854              | EXIT

855 cond_value_list: condition
856                 | cond_value_list ',' condition

857 control_statement: call_statement ';'
858                   | method_invocation ';'
859                   | static_method_invocation ';'
860                   | set_statement ';'
861                   | RESIGNAL ';'
862                   | RESIGNAL scalar_exp ';'
863                   | return_statement ';'
864                   | assignment_statement ';'
865                   | if_statement
866                   | goto_statement ';'
867                   | for_statement
868                   | while_statement

869 assignment_statement: lvalue EQUALS scalar_exp
870                       | column_ref '[' scalar_exp ']' EQUALS scalar_exp

871 lvalue: column_ref
872       | member_observer

873 if_statement: IF '(' search_condition ')' statement opt_else

874 opt_else: /* empty */
875         | ELSE statement

876 call_statement: CALL function_name '(' opt_arg_commalist ')'
877                | function_call

878 set_statement: SET identifier COMPARISON scalar_exp
879               | SET identifier ON
880               | SET identifier OFF

881 goto_statement: GOTO identifier
882                | GO TO identifier

883 return_statement: RETURN scalar_exp
884                  | RETURN

885 while_statement: WHILE '(' search_condition ')' statement

886 for_init_statement: assignment_statement
887                     | variable_declaration
888                     | call_statement
889                     | static_method_invocation

890 for_init_statement_list: /* empty */
891                          | for_init_statement
892                          | for_init_statement_list ',' for_init_statement

893 for_inc_statement: assignment_statement
894                   | call_statement
```

```
895                 | static_method_invocation

896 for_inc_statement_list: /* empty */
897                       | for_inc_statement
898                       | for_inc_statement_list ',' for_inc_statement

899 for_opt_search_cond: /* empty */
900                     | search_condition

901 for_statement: FOR query_exp DO statement
902              | FOR '(' for_init_statement_list ';' for_opt_search_cond ';' for_inc_statement_list '
903              | FOREACH '(' data_type_ref identifier IN_L scalar_exp ')' DO statement

904 trigger_def: CREATE TRIGGER identifier action_time event ON q_table_name opt_order opt_old_ref trig

905 opt_order: /* empty */
906          | ORDER INTNUM

907 trig_action: compound_statement

908 action_time: BEFORE
909            | AFTER
910            | INSTEAD OF

911 event: INSERT
912      | UPDATE opt_column_commalist
913      | DELETE_L

914 opt_old_ref: /* empty */
915            | REFERENCING old_commalist

916 old_commalist: old_alias
917              | old_commalist ',' old_alias

918 old_alias: OLD AS identifier
919          | NEW AS identifier

920 drop_trigger: DROP TRIGGER q_table_name

921 drop_proc: DROP AGGREGATE q_table_name
922          | DROP routine_head q_table_name
923          | DROP MODULE q_table_name

924 opt_element: /* empty */
925            | AS identifier

926 xml_col: column_ref
927        | scalar_exp AS identifier
928        | scalar_exp IN_L identifier

929 xml_col_list: xml_col
930             | xml_col_list ',' xml_col

931 opt_xml_col_list: '(' xml_col_list ')'

932 opt_pk: /* empty */
933       | PRIMARY KEY '(' column_commalist ')'

934 opt_join: /* empty */
935         | ON '(' search_condition ')'

936 opt_elt: /* empty */
937        | NAME

938 xml_join_elt: q_table_name identifier opt_element opt_xml_col_list opt_join opt_pk opt_elt opt_xml_

939 opt_xml_child: /* empty */
940              | BEGINX xml_join_list ENDX

941 top_xml_child: query_spec
942              | BEGINX xml_join_list ENDX

943 xml_join_list: xml_join_elt
```

```
944              | xml_join_list ',' xml_join_elt

945 opt_persist: /* empty */
946            | PERSISTENT

947 opt_interval: /* empty */
948              | INTERVAL INTNUM

949 opt_metas: /* empty */
950          | DTD INTERNAL
951          | DTD EXTERNAL
952          | DTD STRING
953          | SCHEMA EXTERNAL
954          | SCHEMA STRING

955 opt_publish: /* empty */
956            | PUBLIC STRING identifier STRING opt_persist opt_interval opt_metas

957 xmlview_param_value: NAME
958                     | STRING

959 xmlview_param: NAME COMPARISON xmlview_param_value

960 xmlview_params: xmlview_param
961                | xmlview_params xmlview_param

962 opt_xmlview_params: /* empty */
963                    | '[' xmlview_params ']'

964 xml_view: CREATE XML VIEW new_table_name AS opt_xmlview_params top_xml_child opt_elt opt_publish

965 drop_xml_view: DROP XML VIEW q_table_name

966 string_concatenation_operator: scalar_exp STRING_CONCAT_OPERATOR scalar_exp

967 q_type_name: identifier
968            | identifier '.' identifier
969            | identifier '.' identifier '.' identifier
970            | identifier '.' '.' identifier

971 q_old_type_name: identifier
972                 | identifier '.' identifier
973                 | identifier '.' identifier '.' identifier
974                 | identifier '.' '.' identifier

975 new_type_name: identifier
976              | identifier '.' identifier
977              | identifier '.' identifier '.' identifier
978              | identifier '.' '.' identifier

979 @6: /* empty */

980 user_defined_type: CREATE TYPE new_type_name opt_subtype_clause opt_external_and_language_clause @6

981 user_defined_type_drop: DROP TYPE q_old_type_name opt_drop_behavior

982 opt_external_and_language_clause: /* empty */
983                                  | LANGUAGE language_name EXTERNAL NAME_L STRING
984                                  | EXTERNAL NAME_L STRING LANGUAGE language_name
985                                  | LANGUAGE language_name

986 opt_subtype_clause: /* empty */
987                    | UNDER q_type_name

988 opt_as_type_representation: /* empty */
989                            | AS type_representation

990 type_representation: '(' type_member_list ')'

991 type_member_list: type_member
992                  | type_member_list ',' type_member

993 opt_external_clause: /* empty */
```

```
994                     | EXTERNAL NAME_L STRING
995                     | EXTERNAL NAME_L STRING EXTERNAL TYPE STRING
996                     | EXTERNAL TYPE STRING

997 opt_soap_clause: /* empty */
998                | __SOAP_NAME STRING
999                | __SOAP_TYPE STRING
1000                | __SOAP_TYPE STRING __SOAP_NAME STRING
1001                | __SOAP_NAME STRING __SOAP_TYPE STRING

1002 opt_external_type: /* empty */
1003                   | EXTERNAL TYPE STRING

1004 type_member: identifier data_type_ref opt_reference_scope_check opt_default_clause opt_collate_exp

1005 opt_reference_scope_check: /* empty */
1006                          | REFERENCES ARE CHECKED opt_on_delete_referential_rule
1007                          | REFERENCES ARE NOT CHECKED

1008 opt_default_clause: /* empty */
1009                   | DEFAULT signed_literal

1010 opt_type_option_list: /* empty */
1011                     | type_option_list

1012 type_option_list: type_option
1013                 | type_option_list type_option

1014 type_option: FINAL_L
1015            | NOT FINAL_L
1016            | REF USING data_type_ref
1017            | REF FROM '(' column_commalist ')'
1018            | REF IS SYSTEM GENERATED
1019            | CAST '(' SOURCE AS REF ')' WITH identifier
1020            | CAST '(' REF AS SOURCE ')' WITH identifier
1021            | SELF_L AS REF
1022            | TEMPORARY
1023            | UNRESTRICTED
1024            | __SOAP_TYPE STRING

1025 opt_method_specification_list: /* empty */
1026                              | method_specification_list

1027 method_specification_list: method_specification
1028                          | method_specification_list ',' method_specification

1029 method_type: /* empty */
1030            | STATIC_L
1031            | INSTANCE_L

1032 decl_parameter_list: '(' ')'
1033                     | '(' decl_parameter_commalist ')'

1034 decl_parameter_commalist: decl_parameter
1035                         | decl_parameter_commalist ',' decl_parameter

1036 decl_parameter: opt_parameter_mode column_ref data_type_ref opt_external_type

1037 partial_method_specification: method_type METHOD method_identifier decl_parameter_list RETURNS dat
1038                             | CONSTRUCTOR METHOD method_identifier decl_parameter_list opt_specifi

1039 method_specification: partial_method_specification opt_self_result opt_method_characteristics
1040                     | OVERRIDING partial_method_specification

1041 opt_self_result: /* empty */
1042                | SELF_L AS RESULT
1043                | SELF_L AS LOCATOR
1044                | SELF_L AS RESULT SELF_L AS LOCATOR

1045 opt_specific_method_name: /* empty */
1046                          | SPECIFIC new_table_name

1047 opt_method_characteristics: /* empty */
```

```
1048                          | method_characteristics

1049 method_characteristics: method_characteristic
1050                       | method_characteristics method_characteristic

1051 method_characteristic: LANGUAGE language_name
1052                      | PARAMETER STYLE SQL_L
1053                      | PARAMETER STYLE GENERAL
1054                      | DETERMINISTIC
1055                      | NOT DETERMINISTIC
1056                      | NO SQL_L
1057                      | CONTAINS SQL_L
1058                      | READS SQL_L DATA
1059                      | MODIFIES SQL_L DATA
1060                      | RETURNS NULLX ON NULLX INPUT
1061                      | CALLED ON NULLX INPUT
1062                      | EXTERNAL NAME_L STRING
1063                      | EXTERNAL VARIABLE NAME_L STRING
1064                      | EXTERNAL TYPE STRING

1065 external_language_name: ADA
1066                        | C
1067                        | COBOL
1068                        | FORTRAN
1069                        | MUMPS
1070                        | PASCAL_L
1071                        | PLI
1072                        | JAVA
1073                        | CLR

1074 language_name: external_language_name
1075              | SQL_L

1076 opt_constructor_return: /* empty */
1077                        | RETURNS new_type_name

1078 method_declaration: CREATE method_type METHOD method_identifier rout_parameter_list opt_return rou
1079                    | CREATE CONSTRUCTOR METHOD q_table_name rout_parameter_list opt_constructor_ret

1080 static_method_invocation: q_type_name DOUBLE_COLON method_identifier '(' opt_arg_commalist ')'

1081 identifier_chain: identifier '.' identifier '.' identifier '.' method_identifier
1082                 | identifier '.' '.' identifier '.' method_identifier
1083                 | identifier '.' identifier_chain

1084 identifier_chain_method: identifier '.' identifier '.' identifier '.' identifier '.' method_identi
1085                         | identifier '.' '.' identifier '.' identifier '.' method_identifier
1086                         | identifier '.' identifier_chain_method

1087 method_invocation: scalar_exp_no_col_ref_no_mem_obs_chain '.' method_identifier '(' opt_arg_commal
1088                  | identifier_chain_method '(' opt_arg_commalist ')'
1089                  | '(' scalar_exp_no_col_ref AS q_type_name ')' '.' method_identifier '(' opt_arg_
1090                  | '(' column_ref AS q_type_name ')' '.' method_identifier '(' opt_arg_commalist '

1091 top_level_method_invocation: METHOD CALL scalar_exp_no_col_ref_no_mem_obs_chain '.' method_identif
1092                             | METHOD CALL identifier_chain_method '(' opt_arg_commalist ')'
1093                             | METHOD CALL '(' scalar_exp_no_col_ref AS q_type_name ')' '.' method_i
1094                             | METHOD CALL '(' column_ref AS q_type_name ')' '.' method_identifier '

1095 member_observer: member_observer_no_id_chain
1096                | identifier '.' identifier_chain

1097 member_observer_no_id_chain: scalar_exp_no_col_ref_no_mem_obs_chain '.' method_identifier
1098                             | '(' scalar_exp_no_col_ref AS q_type_name ')' '.' method_identifier
1099                             | '(' column_ref AS q_type_name ')' '.' method_identifier

1100 method_identifier: identifier
1101                  | EXTRACT

1102 new_invocation: NEW q_type_name '(' opt_arg_commalist ')'

1103 user_defined_type_alter: ALTER TYPE q_type_name alter_type_action
```

```
  1104 alter_type_action: ADD ATTRIBUTE type_member
  1105                  | DROP ATTRIBUTE identifier opt_drop_behavior
  1106                  | ADD method_specification
  1107                  | DROP partial_method_specification opt_drop_behavior

  1108 opt_with_permission_set: /* empty */
  1109                        | WITH PERMISSION_SET COMPARISON SAFE_L
  1110                        | WITH PERMISSION_SET COMPARISON UNRESTRICTED

  1111 opt_with_autoregister: /* empty */
  1112                       | WITH AUTOREGISTER_L

  1113 create_library: CREATE LIBRARY_L q_table_name AS scalar_exp opt_with_permission_set opt_with_autor

  1114 create_assembly: CREATE ASSEMBLY_L q_table_name FROM scalar_exp opt_with_permission_set opt_with_a

  1115 drop_library: DROP LIBRARY_L q_table_name

  1116 drop_assembly: DROP ASSEMBLY_L q_table_name

Terminals, with rules where they appear

$end (0) 0
'(' (40) 99 126 132 135 136 138 139 140 141 155 185 189 191 233 272
    276 284 331 346 348 354 367 369 402 405 417 421 422 424 428 441
    445 449 450 460 483 484 491 506 507 523 524 537 540 573 574 575
    576 577 578 579 580 581 583 584 585 586 587 588 591 593 594 596
    620 621 625 626 627 628 675 676 678 679 683 689 690 696 698 701
    705 706 715 716 761 762 791 873 876 885 902 903 931 933 935 990
    1017 1019 1020 1032 1033 1080 1087 1088 1089 1090 1091 1092 1093
    1094 1098 1099 1102
')' (41) 99 126 132 135 136 138 139 140 141 155 185 189 191 233 272
    276 284 331 346 348 354 367 369 402 405 417 421 422 424 428 441
    445 449 450 460 483 484 491 506 507 523 524 537 540 573 574 575
    576 577 578 579 580 581 583 584 585 586 587 588 591 593 594 596
    620 621 625 626 627 628 675 676 678 679 683 689 690 696 698 701
    705 706 715 716 761 762 791 873 876 885 902 903 931 933 935 990
    1017 1019 1020 1032 1033 1080 1087 1088 1089 1090 1091 1092 1093
    1094 1098 1099 1102
'*' (42) 494 669 670 671 672 673
'+' (43) 493 496 639 642
',' (44) 101 122 135 136 145 147 164 166 211 218 251 334 344 352 368
    369 373 377 414 416 507 524 537 563 565 566 570 571 574 585 586
    600 603 604 621 676 679 730 747 749 764 781 791 829 856 892 898
    917 930 944 992 1028 1035
'-' (45) 492 497 638 641
'.' (46) 554 555 556 557 558 646 647 648 650 651 652 654 655 656 658
    659 660 665 666 667 668 670 671 672 673 721 968 969 970 972 973
    974 976 977 978 1081 1082 1083 1084 1085 1086 1087 1089 1090 1091
    1093 1094 1096 1097 1098 1099
'/' (47) 495 818
';' (59) 1 754 755 809 813 821 857 858 859 860 861 862 863 864 866
    902
'[' (91) 548 550 708 870 963
']' (93) 548 550 708 870 963
error (256)
NAME (258) 33 325 326 342 455 456 575 586 587 782 937 957 959
STRING (259) 135 136 180 195 203 261 263 265 286 288 400 403 598 629
    635 728 730 733 734 752 791 792 831 832 952 954 956 958 983 984
    994 995 996 998 999 1000 1001 1003 1024 1062 1063 1064
WSTRING (260) 630 636
INTNUM (261) 120 342 350 366 368 631 637 638 639 675 676 678 679 683
    689 690 696 698 701 705 706 708 715 716 906 948
APPROXNUM (262) 632 640 641 642
NUM_ERROR (263)
AMMSC (264) 626 627 628
PARAMETER_L (265) 724
NAMED_PARAMETER (266) 725
BEGIN_EQCALL_X (267) 595 596
HTMLSTR (268) 816 817 818
SQL_TSI (269) 574 585
TIMESTAMP_FUNC (270) 574 585
BINARYNUM (271) 633 643
```

```
MSSQL_XMLCOL_NAME (272) 543
MSSQL_XMLCOL_NAME1 (273) 541 542 543
MSSQL_XMLCOL_NAMEYZ (274) 542
MSSQL_XMLCOL_NAMEZ (275) 541 543
MSSQL_XMLCOL_INTNUM (276) 541 542 543
TYPE (277) 34 980 981 995 996 1003 1064 1103
FINAL_L (278) 35 1014 1015
METHOD (279) 36 301 1037 1038 1078 1079 1091 1092 1093 1094
CHECKED (280) 37 1006 1007
SYSTEM (281) 38 1018
GENERATED (282) 39 1018
SOURCE (283) 40 1019 1020
RESULT (284) 41 1042 1044
LOCATOR (285) 42 1043 1044
INSTANCE_L (286) 43 1031
CONSTRUCTOR (287) 44 1038 1079
SELF_L (288) 45 1021 1042 1043 1044
OVERRIDING (289) 46 1040
STYLE (290) 47 1052 1053
SQL_L (291) 48 1052 1056 1057 1058 1059 1075
GENERAL (292) 49 1053
DETERMINISTIC (293) 50 1054 1055
NO (294) 51 1056
CONTAINS (295) 52 1057
READS (296) 53 1058
DATA (297) 54 1058 1059
MODIFIES (298) 55 1059
INPUT (299) 56 1060 1061
CALLED (300) 57 1061
ADA (301) 58 1065
C (302) 59 1066
COBOL (303) 60 1067
FORTRAN (304) 61 1068
MUMPS (305) 62 1069
PASCAL_L (306) 63 1070
PLI (307) 64 1071
NAME_L (308) 65 752 983 984 994 995 1062 1063
TEXT_L (309) 66 272 277 278 341
JAVA (310) 67 1072
INOUT_L (311) 68 770 776 779
REMOTE (312) 69 242
KEYSET (313) 70 245
VALUE (314) 71 832
PARAMETER (315) 72 1052 1053
VARIABLE (316) 73 1063
ADMIN_L (317) 76 199
ROLE_L (318) 223 228
TEMPORARY (319) 75 1022
CLR (320) 74 1073
ATTRIBUTE (321) 89 1104 1105
__SOAP_DOC (322) 77 786
__SOAP_DOCW (323) 78 788
__SOAP_HEADER (324) 79 784
__SOAP_HTTP (325) 80 789
__SOAP_NAME (326) 81 998 1000 1001
__SOAP_TYPE (327) 82 783 999 1000 1001 1024
__SOAP_XML_TYPE (328) 83 787
__SOAP_FAULT (329) 84 785
__SOAP_DIME_ENC (330) 85 774 775 776
__SOAP_ENC_MIME (331) 86 777 778 779
__SOAP_OPTIONS (332) 87 791
FOREACH (333) 98 903
ARE (334) 1006 1007
REF (335) 1016 1017 1018 1019 1020 1021
STATIC_L (336) 243 1030
SPECIFIC (337) 1046
DYNAMIC (338) 244
COLUMN (339) 162
START_L (340) 88 119
AS (341) 184 185 236 242 265 270 422 427 540 544 545 547 662 918 919
    925 927 964 989 1019 1020 1021 1042 1043 1044 1089 1090 1093 1094
    1098 1099 1113
DOUBLE_COLON (342) 1080
```

```
COLON (343) 815
OR (344) 457
AND (345) 458 472 473
NOT (346) 123 240 290 459 472 474 479 481 483 830 1007 1015 1055
COMPARISON (347) 374 471 485 817 878 959 1109 1110
EQUALS (348) 767 782 791 869 870
STRING_CONCAT_OPERATOR (349) 966
UMINUS (350)
ALL (351) 192 207 208 361 387 389 391 394 396 398 487 627
ANY (352) 486 700 701
ATTACH (353) 234 751
ASC (354) 255
AUTHORIZATION (355)
BETWEEN (356) 472 473
BY (357) 120 131 249 405 448 449 450
CASCADE (358) 113 175 225 227
CHARACTER (359) 584 703 706 713 716
CHECK (360) 132 140 187 735
CLOSE (361) 320
COMMIT (362) 321
CONSTRAINT (363) 143 177
CONTINUE (364) 853
CREATE (365) 99 155 180 184 185 222 223 257 264 265 272 277 747 750
    752 756 904 964 980 1078 1079 1113 1114
CUBE (366) 450
CURRENT (367) 322 370
CURSOR (368) 246 247
DECIMAL (369) 677 678 679
DECLARE (370) 246 247 827 852
DEFAULT (371) 115 128 766 851 1009
DELETE_L (372) 110 214 224 225 231 267 322 323 913
DESC (373) 256
DISTINCT (374) 362 626
DOUBLE (375) 685
DROP (376) 156 159 160 168 172 226 227 228 258 268 278 920 921 922
    923 965 981 1105 1107 1115 1116
ESCAPE (377) 477 478
EXISTS (378) 489
FETCH (379) 324 325 326
FLOAT (380) 682 683
FOR (381) 246 247 257 258 259 454 455 456 852 901 902 1078 1079
FOREIGN (382) 139
FOUND (383) 830
FROM (384) 200 201 202 203 204 234 264 322 323 412 575 587 747 751
    1017 1114
GOTO (385) 849 881
GO (386) 850 882
GRANT (387) 192 193 194 195 196 197 206
GROUP (388) 229 230 231 448 449 450
GROUPING (389) 594
HAVING (390) 452
IN_L (391) 481 482 483 484 580 768 774 777 903 928
INDEX (392) 155 156 272 339 340 341
INDICATOR (393) 624
INSERT (394) 213 290 330 911
INTEGER (395) 680
INTO (396) 324 325 326 327 359
IS (397) 479 480 1018
KEY (398) 127 138 139 233 282 340 341 933
LANGUAGE (399) 286 752 983 984 985 1051
ENCODING (400) 288
LIKE (401) 474 475
NULLX (402) 114 123 124 479 480 634 644 1060 1061
NUMERIC (403) 674 675 676
OF (404) 322 370 910
ON (405) 108 110 155 193 194 195 196 200 201 202 203 272 277 278 444
    738 879 904 935 1060 1061
OPEN (406) 355
OPTION (407) 187 199 206 346 348
ORDER (408) 249 336 906
PRECISION (409) 685
PRIMARY (410) 127 138 233 340 933
PRIVILEGES (411) 192 207
```

```
PROCEDURE (412) 185 758
PUBLIC (413) 219 956
REAL (414) 684
REFERENCES (415) 116 216 1006 1007
RESTRICT (416) 176
ROLLBACK (417) 356
ROLLUP (418) 449
SCHEMA (419) 133 180 953 954
SELECT (420) 212 240 359 406 407 408 491
SET (421) 114 115 221 229 370 375 878 879 880
SMALLINT (422) 681
SOME (423) 488
SQLCODE (424)
SQLERROR (425)
TABLE (426) 99 159 167 168 169 170 179 234 348
TO (427) 192 193 194 195 196 197 850 882
UNION (428) 386 387 393 394
UNIQUE (429) 134 141 149
UPDATE (430) 108 215 271 370 375 454 912
USER (431) 222 224 225 226 227 229 230 231 238 583 607
VALUES (432) 331
VIEW (433) 160 184 185 964 965
WHENEVER (434) 849 850 851
WHERE (435) 263 322 370 446
WITH (436) 119 133 187 199 206 267 282 284 358 364 1019 1020 1109 1110
    1112
WORK (437) 321 356
ARRAY (438) 707 708
CONTIGUOUS (439)
OBJECT_ID (440) 150
UNDER (441) 137 196 202 987
CLUSTERED (442) 148 284
VARCHAR (443) 686 704 705 714 715
VARBINARY (444) 687 688 689
BINARY (445) 690
LONG (446) 686 687 699 718 719
REPLACING (447) 328
SOFT (448) 329
HASH (449) 337
LOOP (450) 338
SHUTDOWN (451) 731
CHECKPOINT (452) 732 733
BACKUP (453) 734
REPLICATION (454) 736 737
SYNC (455) 736
ALTER (456) 167 168 169 170 179 1103
ADD (457) 167 171 230 1104 1106
RENAME (458) 170
DISCONNECT (459) 737
MODIFY (460) 169 173
BEFORE (461) 908
AFTER (462) 909
INSTEAD (463) 910
TRIGGER (464) 277 278 904 920
REFERENCING (465) 915
OLD (466) 918
AGGREGATE (467) 625 747 921
FUNCTION (468) 274 757
OUT_L (469) 769 775 778
HANDLER (470) 852
IF (471) 873
THEN (472) 616 618
ELSE (473) 617 619 875
ELSEIF (474)
WHILE (475) 885
BEGINX (476) 478 597 756 806 940 942
ENDX (477) 440 478 576 577 578 579 580 581 582 583 584 585 586 587
    595 596 597 598 610 611 756 806 940 942
RETURN (478) 883 884
CALL (479) 301 588 876 1091 1092 1093 1094
RETURNS (480) 760 1037 1060 1077
DO (481) 901 903
EXCLUSIVE (482) 349
```

```
PREFETCH (483) 350
SQLSTATE (484) 831 832
SQLWARNING (485) 834
SQLEXCEPTION (486) 833
EXIT (487) 854
RESIGNAL (488) 841 842 861 862
REVOKE (489) 200 201 202 203 204
PASSWORD (490) 221 238
OFF (491) 739 880
LOGX (492) 257 258 259 561 579 738 739
TIMESTAMP (493) 691
DATE (494) 694
DATETIME (495) 692
TIME (496) 693
EXECUTE (497) 194 201
REXECUTE (498) 90 195 203
MODULE (499) 756 923
BEGIN_FN_X (500) 576 577 578 579 580 583 584 585 586 587
BEGIN_CALL_X (501) 581 582
BEGIN_OJ_X (502) 440
BEGIN_U_X (503) 598
CONVERT (504) 537 586
CASE (505) 610 611
WHEN (506) 616 618
IDENTITY (507) 125 126
LEFT (508) 430 559 577
RIGHT (509) 431 560 578
FULL (510) 432
OUTER (511) 436
INNER (512) 433
CROSS (513) 434
NATURAL (514) 437
USING (515) 274 445 1016
JOIN (516) 442
USE (517) 319
COALESCE (518) 620
CAST (519) 540 1019 1020
NULLIF (520) 621
NEW (521) 919 1102
CORRESPONDING (522) 405
EXCEPT (523) 390 391 397 398
INTERSECT (524) 388 389 395 396
BEST (525) 384
TOP (526) 366 367 368 369
PERCENT (527)
TIES (528) 364
XML (529) 180 280 455 456 719 964 965
XPATH (530) 400 403
PERSISTENT (531) 946
INTERVAL (532) 948
INCREMENT_L (533) 97 120
DTD (534) 950 951 952
INTERNAL (535) 950
EXTERNAL (536) 752 951 953 983 984 994 995 996 1003 1062 1063 1064
COLLATE (537) 129 539
NCHAR (538) 695 696
NVARCHAR (539) 697 698 699
INCREMENTAL (540)
NONINCREMENTAL (541) 265 270
PURGE (542) 259
SNAPSHOT (543) 257 258 259 264 265 268 271
IDENTIFIED (544) 131
EXTRACT (545) 575 587 1101
KWD_TAG (546) 562 563
LEXICAL_ERROR (547)
CURRENT_DATE (548) 589
CURRENT_TIME (549) 590 591
CURRENT_TIMESTAMP (550) 592 593
PERMISSION_SET (551) 91 1109 1110
AUTOREGISTER_L (552) 92 1112
LIBRARY_L (553) 93 1113 1115
ASSEMBLY_L (554) 94 1114 1116
SAFE_L (555) 95 1109
```

```
UNRESTRICTED (556) 96 1023 1110

Nonterminals, with rules where they appear

$accept (313)
    on left: 0
sql_list (314)
    on left: 1 2, on right: 0
sql_list1 (315)
    on left: 3, on right: 1 2
sql (316)
    on left: 4 5 6 7 8 9 10 11 12 291 595 596 740 741 742 743 744 745
    746, on right: 3
schema_element_list (317)
    on left: 13 14 15 16, on right: 4 15 16
schema_element (318)
    on left: 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32,
    on right: 13 15
identifier (319)
    on left: 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
    51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
    72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92
    93 94 95 96 97 98, on right: 143 156 177 221 246 247 319 339 421
    422 424 426 427 544 545 546 553 554 555 556 557 558 562 563 576
    580 597 645 646 647 648 649 650 651 652 653 654 655 656 657 658
    659 660 662 663 664 665 666 667 668 670 671 672 673 720 721 722
    723 726 753 815 828 829 849 850 878 879 880 881 882 903 904 918
    919 925 927 928 938 956 967 968 969 970 971 972 973 974 975 976
    977 978 1004 1019 1020 1081 1082 1083 1084 1085 1086 1096 1100
    1105
base_table_def (320)
    on left: 99, on right: 17
base_table_element_commalist (321)
    on left: 100 101, on right: 99 101
base_table_element (322)
    on left: 102 103, on right: 100 101
column_def (323)
    on left: 104, on right: 102 163 164 169 415 416
opt_referential_triggered_action (324)
    on left: 105 106 107, on right: 116
referential_rule (325)
    on left: 108 109, on right: 106 107
delete_referential_rule (326)
    on left: 110, on right: 109 112
opt_on_delete_referential_rule (327)
    on left: 111 112, on right: 1006
referential_action (328)
    on left: 113 114 115, on right: 108 110
references (329)
    on left: 116, on right: 130 139
column_def_opt_list (330)
    on left: 117 118, on right: 104 118
identity_opt (331)
    on left: 119 120, on right: 121 122
identity_opt_list (332)
    on left: 121 122, on right: 122 126
column_def_opt (333)
    on left: 123 124 125 126 127 128 129 130 131 132 133 134,
    on right: 118
column_xml_schema_def (334)
    on left: 135 136, on right: 133
table_constraint_def (335)
    on left: 137 138 139 140 141, on right: 103 178
opt_constraint_name (336)
    on left: 142 143, on right: 138 139 140 141
column_commalist (337)
    on left: 144 145, on right: 139 141 145 189 191 233 284 405 419
    445 933 1017
index_column_commalist (338)
    on left: 146 147, on right: 138 147 155
index_option (339)
    on left: 148 149 150, on right: 151 152
index_option_list (340)
```

```
    on left: 151 152, on right: 152 154
opt_index_option_list (341)
    on left: 153 154, on right: 138 155
create_index_def (342)
    on left: 155, on right: 18
drop_index (343)
    on left: 156, on right: 20
opt_table (344)
    on left: 157 158, on right: 156
drop_table (345)
    on left: 159 160, on right: 19
opt_col_add_column (346)
    on left: 161 162, on right: 167 168 169
add_col_column_def_list (347)
    on left: 163 164, on right: 164 167
add_col_column_list (348)
    on left: 165 166, on right: 166 168
add_column (349)
    on left: 167 168 169, on right: 14 16
table_rename (350)
    on left: 170, on right: 21
constraint_op (351)
    on left: 171 172 173, on right: 179
opt_drop_behavior (352)
    on left: 174 175 176, on right: 177 981 1105 1107
opt_table_constraint_def (353)
    on left: 177 178, on right: 179
alter_constraint (354)
    on left: 179, on right: 8
create_xml_schema (355)
    on left: 180, on right: 7
view_query_spec (356)
    on left: 181 182, on right: 184
view_def (357)
    on left: 184 185, on right: 5
@1 (358)
    on left: 183, on right: 184
opt_with_check_option (359)
    on left: 186 187, on right: 184
opt_column_commalist (360)
    on left: 188 189, on right: 116 184 912
priv_opt_column_commalist (361)
    on left: 190 191, on right: 212 215 216 330
privilege_def (362)
    on left: 192 193 194 195 196 197, on right: 22
opt_with_admin_option (363)
    on left: 198 199, on right: 197
privilege_revoke (364)
    on left: 200 201 202 203 204, on right: 23
opt_with_grant_option (365)
    on left: 205 206, on right: 193 194 196
privileges (366)
    on left: 207 208 209, on right: 193 200
operation_commalist (367)
    on left: 210 211, on right: 209 211
operation (368)
    on left: 212 213 214 215 216, on right: 210 211
grantee_commalist (369)
    on left: 217 218, on right: 193 194 195 196 197 200 201 202 203
    204 218
grantee (370)
    on left: 219 220, on right: 192 217 218
set_pass (371)
    on left: 221, on right: 26
create_user_statement (372)
    on left: 222 223, on right: 24
delete_user_statement (373)
    on left: 224 225 226 227 228, on right: 25
set_group_stmt (374)
    on left: 229, on right: 27
add_group_stmt (375)
    on left: 230, on right: 28
delete_group_stmt (376)
```

```
    on left: 231, on right: 29
opt_attach_primary_key (377)
    on left: 232 233, on right: 234
attach_table (378)
    on left: 234, on right: 309
opt_as (379)
    on left: 235 236, on right: 234 751
opt_login (380)
    on left: 237 238, on right: 234
opt_not_select (381)
    on left: 239 240, on right: 234
opt_remote_name (382)
    on left: 241 242, on right: 234
cursor_type (383)
    on left: 243 244 245, on right: 247
cursor_def (384)
    on left: 246 247, on right: 824
opt_order_by_clause (385)
    on left: 248 249, on right: 402 411
ordering_spec_commalist (386)
    on left: 250 251, on right: 249 251 448 449 450
ordering_spec (387)
    on left: 252 253, on right: 250 251
opt_asc_desc (388)
    on left: 254 255 256, on right: 146 147 252 253
create_snapshot_log (389)
    on left: 257, on right: 310
drop_snapshot_log (390)
    on left: 258, on right: 311
purge_snapshot_log (391)
    on left: 259, on right: 312
opt_snapshot_string_literal (392)
    on left: 260 261, on right: 264
opt_snapshot_where_clause (393)
    on left: 262 263, on right: 264
create_snapshot (394)
    on left: 264 265, on right: 313
opt_with_delete (395)
    on left: 266 267, on right: 268
drop_snapshot (396)
    on left: 268, on right: 314
opt_nonincremental (397)
    on left: 269 270, on right: 271
refresh_snapshot (398)
    on left: 271, on right: 315
create_freetext_index (399)
    on left: 272, on right: 316
opt_data_modification_action (400)
    on left: 273 274, on right: 272
opt_column (401)
    on left: 275 276, on right: 277 278
create_freetext_trigger (402)
    on left: 277, on right: 317
drop_freetext_trigger (403)
    on left: 278, on right: 318
opt_xml (404)
    on left: 279 280, on right: 272
opt_with_key (405)
    on left: 281 282, on right: 272
opt_with (406)
    on left: 283 284, on right: 272
opt_lang (407)
    on left: 285 286, on right: 272
opt_enc (408)
    on left: 287 288, on right: 272
opt_deffer_generation (409)
    on left: 289 290, on right: 272
manipulative_statement (410)
    on left: 292 293 294 295 296 297 298 299 300 301 302 303 304 305
    306 307 308 309 310 311 312 313 314 315 316 317 318,
    on right: 291
use_statement (411)
    on left: 319, on right: 308
```

```
close_statement (412)
    on left: 320, on right: 799
commit_statement (413)
    on left: 321, on right: 305 803
delete_statement_positioned (414)
    on left: 322, on right: 297 797
delete_statement_searched (415)
    on left: 323, on right: 298 798
fetch_statement (416)
    on left: 324 325 326, on right: 800
insert_mode (417)
    on left: 327 328 329, on right: 330
insert_statement (418)
    on left: 330, on right: 296 796
values_or_query_spec (419)
    on left: 331 332, on right: 330
insert_atom_commalist (420)
    on left: 333 334, on right: 331 334
insert_atom (421)
    on left: 335, on right: 333 334
sql_option (422)
    on left: 336 337 338 339 340 341 342, on right: 343 344
sql_opt_commalist (423)
    on left: 343 344, on right: 344 346 348
opt_sql_opt (424)
    on left: 345 346, on right: 411
opt_table_opt (425)
    on left: 347 348, on right: 661 662 663
cursor_option (426)
    on left: 349 350, on right: 351 352
cursor_options_commalist (427)
    on left: 351 352, on right: 352 354
opt_cursor_options_list (428)
    on left: 353 354, on right: 355 358
open_statement (429)
    on left: 355, on right: 801
rollback_statement (430)
    on left: 356, on right: 306 802
with_opt_cursor_options_list (431)
    on left: 357 358, on right: 359
select_statement (432)
    on left: 359, on right: 793
opt_all_distinct (433)
    on left: 360 361 362, on right: 365 366 367 368 369
opt_ties (434)
    on left: 363 364, on right: 366 367 368 369
opt_top (435)
    on left: 365 366 367 368 369, on right: 359 406 407 408 491
update_statement_positioned (436)
    on left: 370, on right: 294 794
assignment_commalist (437)
    on left: 371 372 373, on right: 370 373 375
assignment (438)
    on left: 374, on right: 372 373
update_statement_searched (439)
    on left: 375, on right: 295 795
target_commalist (440)
    on left: 376 377, on right: 324 325 326 359 377
target (441)
    on left: 378 379 380, on right: 376 377
opt_where_clause (442)
    on left: 381 382, on right: 323 375 410 411
opt_best (443)
    on left: 383 384, on right: 386 387 393 394
query_exp (444)
    on left: 385 386 387 388 389 390 391, on right: 181 246 247 292
    402 421 422 901
non_final_union_exp (445)
    on left: 392 393 394 395 396 397 398, on right: 386 387 388 389
    390 391 393 394 395 396 397 398
non_final_query_term (446)
    on left: 399 400, on right: 392 393 394 395 396 397 398
query_term (447)
```

```
    on left: 401 402 403, on right: 385 386 387 388 389 390 391
opt_corresponding (448)
    on left: 404 405, on right: 386 387 388 389 390 391 393 394 395
    396 397 398
non_final_query_spec (449)
    on left: 406, on right: 399
query_spec (450)
    on left: 407, on right: 332 401 941
query_no_from_spec (451)
    on left: 408, on right: 182 293
selection (452)
    on left: 409, on right: 359 406 407 408 491
non_final_table_exp (453)
    on left: 410, on right: 406
table_exp (454)
    on left: 411, on right: 359 407 491
from_clause (455)
    on left: 412, on right: 410 411
table_ref_commalist (456)
    on left: 413 414, on right: 412 414
proc_col_list (457)
    on left: 415 416, on right: 185 416 417
opt_proc_col_list (458)
    on left: 417, on right: 424
column_commalist_or_empty (459)
    on left: 418 419, on right: 185 424
table_ref (460)
    on left: 420 421 422 423 424, on right: 413 414 442
table_ref_nj (461)
    on left: 425 426 427 428, on right: 442
jtype (462)
    on left: 429 430 431 432 433 434, on right: 437 438
opt_outer (463)
    on left: 435 436, on right: 430 431 432
join (464)
    on left: 437 438, on right: 442
joined_table (465)
    on left: 439 440 441, on right: 423 428
joined_table_1 (466)
    on left: 442, on right: 439 440 441
join_condition (467)
    on left: 443 444 445, on right: 442
where_clause (468)
    on left: 446, on right: 382
opt_group_by_clause (469)
    on left: 447 448 449 450, on right: 410 411
opt_having_clause (470)
    on left: 451 452, on right: 410 411
opt_lock_mode (471)
    on left: 453 454 455 456, on right: 411
search_condition (472)
    on left: 457 458 459 460 461, on right: 132 140 444 446 452 457
    458 459 460 618 873 885 900 935
predicate (473)
    on left: 462 463 464 465 466 467 468 469, on right: 461
scalar_exp_predicate (474)
    on left: 470, on right: 469
comparison_predicate (475)
    on left: 471, on right: 462
between_predicate (476)
    on left: 472 473, on right: 463
like_predicate (477)
    on left: 474 475, on right: 464
opt_escape (478)
    on left: 476 477 478, on right: 474 475
test_for_null (479)
    on left: 479 480, on right: 465
in_predicate (480)
    on left: 481 482 483 484, on right: 466
all_or_any_predicate (481)
    on left: 485, on right: 467
any_all_some (482)
    on left: 486 487 488, on right: 485
```

```
existence_test (483)
    on left: 489, on right: 468
scalar_subquery (484)
    on left: 490, on right: 505 522
subquery (485)
    on left: 491, on right: 426 427 481 482 485 489 490
scalar_exp (486)
    on left: 492 493 494 495 496 497 498 499 500 501 502,
    on right: 238 242 252 326 335 367 369 374 470 471 472 473 474 475
    479 480 481 482 483 484 485 492 493 494 495 496 497 506 507 523
    524 537 540 544 545 546 547 548 550 562 563 566 574 575 580 585
    586 587 588 591 593 599 600 601 603 610 616 617 618 619 621 626
    627 628 817 818 842 862 869 870 878 883 903 927 928 966 1113 1114
scalar_exp_no_col_ref (487)
    on left: 503 504 505 506 507 508 509 510 511 512 513 514 515 516
    517 518 519, on right: 501 548 1089 1093 1098
scalar_exp_no_col_ref_no_mem_obs_chain (488)
    on left: 520 521 522 523 524 525 526 527 528 529 530 531 532 533
    534 535 536, on right: 1087 1091 1097
cvt_exp (489)
    on left: 537, on right: 510 527
opt_collate_exp (490)
    on left: 538 539, on right: 540 1004
cast_exp (491)
    on left: 540, on right: 511 528
mssql_xml_col (492)
    on left: 541 542 543, on right: 253 547
as_expression (493)
    on left: 544 545 546 547, on right: 564 565 602 604
array_ref (494)
    on left: 548 549, on right: 516 533
lvalue_array_ref (495)
    on left: 550, on right: 380 549
opt_scalar_exp_commalist (496)
    on left: 551 552, on right: 576 577 578 579 581 583 584 596
function_name (497)
    on left: 553 554 555 556 557 558 559 560 561, on right: 194 201
    573 581 582 625 876
kwd_commalist (498)
    on left: 562 563, on right: 563 568 570
as_commalist (499)
    on left: 564 565 566, on right: 565 566 571 572
opt_arg_commalist (500)
    on left: 567 568 569 570 571 572, on right: 573 588 625 876 1080
    1087 1088 1089 1090 1091 1092 1093 1094 1102
function_call (501)
    on left: 573 574 575 576 577 578 579 580 581 582 583 584 585 586
    587 588 589 590 591 592 593 594, on right: 508 525 877
obe_literal (502)
    on left: 597 598, on right: 502 609
scalar_exp_commalist (503)
    on left: 599 600, on right: 483 484 507 524 552 569 570 571 600
    620
select_scalar_exp_commalist (504)
    on left: 601 602 603 604, on right: 409 603 604
atom_no_obe (505)
    on left: 605 606 607, on right: 503 520 608
atom (506)
    on left: 608 609, on right: 477 478 597
simple_case (507)
    on left: 610, on right: 512 529
searched_case (508)
    on left: 611, on right: 513 530
searched_when_list (509)
    on left: 612 613, on right: 611 613
simple_when_list (510)
    on left: 614 615, on right: 610 615
simple_when (511)
    on left: 616 617, on right: 614 615
searched_when (512)
    on left: 618 619, on right: 612 613
coalesce_exp (513)
    on left: 620, on right: 514 531
```

```
nullif_exp (514)
    on left: 621, on right: 515 532
parameter_ref (515)
    on left: 622 623 624, on right: 605
aggregate_ref (516)
    on left: 625 626 627 628, on right: 504 521
literal (517)
    on left: 629 630 631 632 633 634, on right: 234 606 751
signed_literal (518)
    on left: 635 636 637 638 639 640 641 642 643 644,
    on right: 119 128 766 767 782 1009
q_table_name (519)
    on left: 645 646 647 648, on right: 116 129 137 158 159 160 167
    168 169 170 179 185 257 258 259 264 265 268 271 272 277 278 424
    539 595 596 661 662 663 904 920 921 922 923 938 965 1079 1113 1114
    1115 1116
attach_q_table_name (520)
    on left: 649 650 651 652, on right: 234 751
new_proc_or_bif_name (521)
    on left: 653 654 655 656, on right: 747 749
new_table_name (522)
    on left: 657 658 659 660, on right: 99 155 170 184 185 236 747
    750 752 756 964 1046
table (523)
    on left: 661 662 663, on right: 193 200 322 323 330 370 375 420
    425
column_ref (524)
    on left: 664 665 666 667 668 669 670 671 672 673,
    on right: 378 500 550 594 765 766 767 870 871 926 1036 1090 1094
    1099
base_data_type (525)
    on left: 674 675 676 677 678 679 680 681 682 683 684 685 686 687
    688 689 690 691 692 693 694 695 696 697 698 699 700 701,
    on right: 702 712
data_type (526)
    on left: 702 703 704 705 706, on right: 537 540 544 710
array_modifier (527)
    on left: 707 708, on right: 709
data_type_ref (528)
    on left: 709 710 711, on right: 709 760 765 766 767 827 903 1004
    1016 1036 1037
column_data_type (529)
    on left: 712 713 714 715 716 717 718 719, on right: 104
column (530)
    on left: 720 721, on right: 104 131 144 145 146 147 165 166 272
    276 282 374
index (531)
    on left: 722, on right: 155
cursor (532)
    on left: 723, on right: 320 322 324 325 326 355 370
parameter (533)
    on left: 724 725, on right: 622 623 624
user (534)
    on left: 726, on right: 220 222 223 224 225 226 227 228 229 230
    231
opt_log (535)
    on left: 727 728, on right: 731 732 736 737
comma_opt_log (536)
    on left: 729 730, on right: 736
admin_statement (537)
    on left: 731 732 733 734 735 736 737 738 739, on right: 307
user_aggregate_declaration (538)
    on left: 747, on right: 740
user_aggregate_merge_opt (539)
    on left: 748 749, on right: 747
routine_declaration (540)
    on left: 750 751 752, on right: 741
module_body_part (541)
    on left: 753, on right: 754 755
module_body (542)
    on left: 754 755, on right: 755 756
module_declaration (543)
    on left: 756, on right: 742
```

```
routine_head (544)
    on left: 757 758, on right: 750 751 752 753 922
opt_return (545)
    on left: 759 760, on right: 747 750 751 752 753 1078
rout_parameter_list (546)
    on left: 761 762, on right: 747 750 751 752 753 1078 1079
parameter_commalist (547)
    on left: 763 764, on right: 762 764
rout_parameter (548)
    on left: 765 766 767, on right: 763 764
parameter_mode (549)
    on left: 768 769 770, on right: 765 766 767 772
opt_parameter_mode (550)
    on left: 771 772, on right: 1036
opt_soap_enc_mode (551)
    on left: 773 774 775 776 777 778 779, on right: 791 792
soap_proc_opt_list (552)
    on left: 780 781, on right: 781 791
soap_proc_opt (553)
    on left: 782, on right: 780 781
soap_kwd (554)
    on left: 783 784 785 786 787 788 789, on right: 791 792
rout_alt_type (555)
    on left: 790 791 792, on right: 750 751 752 753 765 766 767 1078
routine_statement (556)
    on left: 793 794 795 796 797 798 799 800 801 802 803 804,
    on right: 813 821 836
compound_statement (557)
    on left: 806, on right: 750 753 810 819 835 907 1078 1079
@2 (558)
    on left: 805, on right: 806
statement_list (559)
    on left: 807 808, on right: 806 808
statement_in_cs (560)
    on left: 809 810 812, on right: 807 808 815
@3 (561)
    on left: 811, on right: 812
statement_in_cs_oper (562)
    on left: 813 814 815 816 817 818, on right: 812
statement (563)
    on left: 819 821 823, on right: 873 875 885 901 902 903
@4 (564)
    on left: 820, on right: 821
@5 (565)
    on left: 822, on right: 823
local_declaration (566)
    on left: 824 825 826, on right: 809
variable_declaration (567)
    on left: 827, on right: 825 887
variable_list (568)
    on left: 828 829, on right: 827 829
condition (569)
    on left: 830 831 832 833 834, on right: 849 850 851 855 856
handler_statement (570)
    on left: 835 836 837 838 839 840 841 842 843 844 845 846 847 848
   , on right: 852
handler_declaration (571)
    on left: 849 850 851 852, on right: 826
handler_type (572)
    on left: 853 854, on right: 852
cond_value_list (573)
    on left: 855 856, on right: 852 856
control_statement (574)
    on left: 857 858 859 860 861 862 863 864 865 866 867 868,
    on right: 814 823
assignment_statement (575)
    on left: 869 870, on right: 498 844 864 886 893
lvalue (576)
    on left: 871 872, on right: 869
if_statement (577)
    on left: 873, on right: 845 865
opt_else (578)
    on left: 874 875, on right: 873
```

```
call_statement (579)
    on left: 876 877, on right: 299 837 857 888 894
set_statement (580)
    on left: 878 879 880, on right: 303 840 860
goto_statement (581)
    on left: 881 882, on right: 846 866
return_statement (582)
    on left: 883 884, on right: 843 863
while_statement (583)
    on left: 885, on right: 848 868
for_init_statement (584)
    on left: 886 887 888 889, on right: 891 892
for_init_statement_list (585)
    on left: 890 891 892, on right: 892 902
for_inc_statement (586)
    on left: 893 894 895, on right: 897 898
for_inc_statement_list (587)
    on left: 896 897 898, on right: 898 902
for_opt_search_cond (588)
    on left: 899 900, on right: 902
for_statement (589)
    on left: 901 902 903, on right: 847 867
trigger_def (590)
    on left: 904, on right: 744
opt_order (591)
    on left: 905 906, on right: 904
trig_action (592)
    on left: 907, on right: 904
action_time (593)
    on left: 908 909 910, on right: 904
event (594)
    on left: 911 912 913, on right: 904
opt_old_ref (595)
    on left: 914 915, on right: 904
old_commalist (596)
    on left: 916 917, on right: 915 917
old_alias (597)
    on left: 918 919, on right: 916 917
drop_trigger (598)
    on left: 920, on right: 745
drop_proc (599)
    on left: 921 922 923, on right: 746
opt_element (600)
    on left: 924 925, on right: 938
xml_col (601)
    on left: 926 927 928, on right: 929 930
xml_col_list (602)
    on left: 929 930, on right: 930 931
opt_xml_col_list (603)
    on left: 931, on right: 938
opt_pk (604)
    on left: 932 933, on right: 938
opt_join (605)
    on left: 934 935, on right: 938
opt_elt (606)
    on left: 936 937, on right: 938 964
xml_join_elt (607)
    on left: 938, on right: 943 944
opt_xml_child (608)
    on left: 939 940, on right: 938
top_xml_child (609)
    on left: 941 942, on right: 964
xml_join_list (610)
    on left: 943 944, on right: 940 942 944
opt_persist (611)
    on left: 945 946, on right: 956
opt_interval (612)
    on left: 947 948, on right: 956
opt_metas (613)
    on left: 949 950 951 952 953 954, on right: 956
opt_publish (614)
    on left: 955 956, on right: 964
xmlview_param_value (615)
```

```
    on left: 957 958, on right: 959
xmlview_param (616)
    on left: 959, on right: 960 961
xmlview_params (617)
    on left: 960 961, on right: 961 963
opt_xmlview_params (618)
    on left: 962 963, on right: 964
xml_view (619)
    on left: 964, on right: 6
drop_xml_view (620)
    on left: 965, on right: 304
string_concatenation_operator (621)
    on left: 966, on right: 499
q_type_name (622)
    on left: 967 968 969 970, on right: 711 717 718 987 1078 1079 1080
    1089 1090 1093 1094 1098 1099 1102 1103
q_old_type_name (623)
    on left: 971 972 973 974, on right: 196 202 981
new_type_name (624)
    on left: 975 976 977 978, on right: 980 1077
user_defined_type (625)
    on left: 980, on right: 30
@6 (626)
    on left: 979, on right: 980
user_defined_type_drop (627)
    on left: 981, on right: 31
opt_external_and_language_clause (628)
    on left: 982 983 984 985, on right: 980
opt_subtype_clause (629)
    on left: 986 987, on right: 980
opt_as_type_representation (630)
    on left: 988 989, on right: 980
type_representation (631)
    on left: 990, on right: 989
type_member_list (632)
    on left: 991 992, on right: 990 992
opt_external_clause (633)
    on left: 993 994 995 996, on right: 1004
opt_soap_clause (634)
    on left: 997 998 999 1000 1001, on right: 1004
opt_external_type (635)
    on left: 1002 1003, on right: 1036
type_member (636)
    on left: 1004, on right: 991 992 1104
opt_reference_scope_check (637)
    on left: 1005 1006 1007, on right: 1004
opt_default_clause (638)
    on left: 1008 1009, on right: 1004
opt_type_option_list (639)
    on left: 1010 1011, on right: 752 980
type_option_list (640)
    on left: 1012 1013, on right: 1011 1013
type_option (641)
    on left: 1014 1015 1016 1017 1018 1019 1020 1021 1022 1023 1024
   , on right: 1012 1013
opt_method_specification_list (642)
    on left: 1025 1026, on right: 980
method_specification_list (643)
    on left: 1027 1028, on right: 1026 1028
method_type (644)
    on left: 1029 1030 1031, on right: 1037 1078
decl_parameter_list (645)
    on left: 1032 1033, on right: 1037 1038
decl_parameter_commalist (646)
    on left: 1034 1035, on right: 1033 1035
decl_parameter (647)
    on left: 1036, on right: 1034 1035
partial_method_specification (648)
    on left: 1037 1038, on right: 1039 1040 1107
method_specification (649)
    on left: 1039 1040, on right: 1027 1028 1106
opt_self_result (650)
    on left: 1041 1042 1043 1044, on right: 1039
```

```
opt_specific_method_name (651)
    on left: 1045 1046, on right: 1037 1038
opt_method_characteristics (652)
    on left: 1047 1048, on right: 1039
method_characteristics (653)
    on left: 1049 1050, on right: 1048 1050
method_characteristic (654)
    on left: 1051 1052 1053 1054 1055 1056 1057 1058 1059 1060 1061
    1062 1063 1064, on right: 1049 1050
external_language_name (655)
    on left: 1065 1066 1067 1068 1069 1070 1071 1072 1073,
    on right: 752 1074
language_name (656)
    on left: 1074 1075, on right: 983 984 985 1051
opt_constructor_return (657)
    on left: 1076 1077, on right: 1079
method_declaration (658)
    on left: 1078 1079, on right: 743
static_method_invocation (659)
    on left: 1080, on right: 300 301 517 534 839 859 889 895
identifier_chain (660)
    on left: 1081 1082 1083, on right: 1083 1096
identifier_chain_method (661)
    on left: 1084 1085 1086, on right: 1086 1088 1092
method_invocation (662)
    on left: 1087 1088 1089 1090, on right: 518 535 838 858
top_level_method_invocation (663)
    on left: 1091 1092 1093 1094, on right: 302
member_observer (664)
    on left: 1095 1096, on right: 379 519 872
member_observer_no_id_chain (665)
    on left: 1097 1098 1099, on right: 536 1095
method_identifier (666)
    on left: 1100 1101, on right: 554 555 556 557 558 1037 1038 1078
    1080 1081 1082 1084 1085 1087 1089 1090 1091 1093 1094 1097 1098
    1099
new_invocation (667)
    on left: 1102, on right: 509 526
user_defined_type_alter (668)
    on left: 1103, on right: 32
alter_type_action (669)
    on left: 1104 1105 1106 1107, on right: 1103
opt_with_permission_set (670)
    on left: 1108 1109 1110, on right: 1113 1114
opt_with_autoregister (671)
    on left: 1111 1112, on right: 1113 1114
create_library (672)
    on left: 1113, on right: 9
create_assembly (673)
    on left: 1114, on right: 10
drop_library (674)
    on left: 1115, on right: 11
drop_assembly (675)
    on left: 1116, on right: 12
```

23.2. Error Codes Reference

23.2.1. Virtuoso Error Codes

Table 23.1. Virtuoso Errors Sub-Codes List

Virtuoso Sub-Code	Description
SR	SQL Runtime
FA	File access
SQ	SQL Compile
CL	client-side
HT	HTTP Server

Virtuoso Sub-Code	Description
FT	Free-text
DC	2 phase commit
VD	VDB
DT	date/time routines
IN	Internationalization routines
RE	Replication
TR	Transactional replication
DA	WebDAV domain
SM	SMTP functions, Mail functions
PO	POP3 functions
NN	NNTP server
GN	GNW
XS	XSLT
CR	Crypto API
UD	User Defined Types
KB	Kerberos Library
HO	Hosting

<title>SQL Runtime Error</title>
SR000 22023 <function_name> expects a vector, not an arg of type <dv_type>

SR001 37000 More than one resultset not supported in a procedure called from exec

SR002 22023 Function <function_name> needs a string output as argument <index>, not an arg of type <type_name> (<dv_type>)

SR003 22023 Function <function_name> needs an XML entity as argument <index>, not an arg of type <type_name> (<dv_type>)

SR004 22023 Function <function_name> needs a binary as argument <index>, not an arg of type <type_name> (<dv_type>)

SR005 22023 Function <function_name> needs a string or NULL as argument <index>, not an arg of type <type_name> (<dv_type>)

SR006 22023 Function <function_name> needs a string or NULL as argument <index>, not an arg of type <type_name> (<dv_type>)

SR007 22023 Function <function_name> needs a string or NULL as argument <index>, not an arg of type <type_name> (<dv_type>)

SR008 22023 Function <function_name> needs an integer as argument <index>, not an arg of type <type_name> (<dv_type>)

SR009 22023 Function <function_name> needs a float as argument <index>, not an arg of type <type_name> (<dv_type>)

SR010 22023 Function <function_name> needs a double as argument <index>, not an arg of type <type_name> (<dv_type>)

SR011 22023 Function <function_name> needs an int or a string as argument <index>, not an arg of type <type_name> (<dv_type>)

SR012 22023 Function <function_name> needs a string or an array as argument <index>, not an arg of type <type_name> (<dv_type>)

SR013 22023 Function <function_name> needs a string or an array as argument <index>, not an arg of type <type_name> (<dv_type>)

SR014 22023 Function <function_name> needs a string as argument <index>, not an arg of type <type_name> (<dv_type>)

SR015 22023 Function length is not applicable to XML tree entity

SR016 22023 Function length needs a string or array as its argument, not an argument of type <type_name> (<dv_type>)

SR017 22003 aref: Bad array subscript (zero-based) <index> for an arg of type <type_name> and length <length>.

SR018 22023 non-generic vector for aref_set_0

SR019 22003 Bad subscript for aref_set_0

SR020 22003 Bad array subscript <index> in aset.

SR021 22003 make_array called with an invalid count <count>

SR022 22023 Type for make_array must be float, double, long or any

SR023 22023 Function subseq needs a string or object id as its first argument, not an arg of type <type_name> (<dv_type>)

SR024 22023 Function subseq needs integers or NULLs as its arguments 2 and 3, not an argument of type <type_name> (<dv_type>)

SR025 22011 subseq: Bad string subrange: from=<from>, to=<to>, len=<length>.

SR026 22011 substr: Bad string subrange: from=<from>, to=<to>, len=<length>.

SR027 22023 Source string arg (#1) to replace was not a wide string

SR028 22023 From arg (#2) to replace was not a wide string

SR029 22023 To arg (#3) to replace was not a wide string

SR030 22003 Too few arguments for <function_name>

SR031 22023 Invalid format string for sprintf at escape <index>

SR032 22026 sprintf escape <index> (<escape>) exceeds the internal buffer of 2000

SR033 22026 The length of the data for sprintf argument <index> exceed the maximum of 2000

SR034 22026 The length of the data for sprintf argument <index> exceed the maximum of 2000

SR035 22026 The length of the data for sprintf argument <index> exceed the maximum of 2000

SR036 22025 The URL escaping sprintf escape <index> doesn't support modifiers

SR037 22025 The HTTP escaping sprintf escape <index> doesn't support modifiers

SR038 22023 Invalid format string for sprintf

SR039 22023 The first argument to strstr is not a wide string

SR040 22023 The second argument to strstr is not a wide string

SR041 22023 Argument 1 of locate is not a wide string

SR042 22023 Argument 2 of locate is not a wide string

SR043 22019 the escape should be non-empty string in matches_like

SR044 22023 Function ascii needs a string or similar thing as its argument, not an argument of type <type_name> (<dv_type>)

SR045 22023 Function mod needs two integers as its arguments, not an argument of type <type_name> (<dv_type>)

SR046 22012 Division by zero in mod(<n1>,<n2>)

SR047 22023 Function sign needs a numeric type as its argument, not an argument of type <type_name> (<dv_type>)

SR048 22023 Function abs needs a numeric type as its argument, not an argument of type <type_name> (<dv_type>)

SR049 22023 Data type is not suitable for storage into a global variable (connection_set)

SR050 22023 connenction_vars_set expects a vector or null as argument not of type <type_name> (<dv_type>)

SR051 22023 connenction_vars_set expects a vector of even length, not of length <length> (of type <type_name> (<dv_type>))

SR052 22023 connenction_vars_set expects a string as name of connection variable not of type <type_name> (<dv_type>)

SR053 22023 Data type is not suitable for storage into a global variable (connection_set)

SR054 22023 Function oid_class_spec needs an object id as its argument, not an argument of type <type_name> (<dv_type>)

SR055 22023 Function oid_class_spec detected an object id whose length <length> < 4. oid[0]=<u1>. oid[1]=<u2>. oid[2]=<u3>

SR056 22023 <function_name> expects the type of item searched for <type_name> (<dv_type>) and the type of the vector searched from <type_name> (<dv_type>) to match. Veclen=<length>

SR057 22023 get_keyword expects a vector of even length, not of length <length> (of type <type_name> (<dv_type>))

SR058 22023 get_keyword expects a vector

SR059 22024 get_keyword_ucase expects a vector of even length, not of length <length> (of type <type_name> (<dv_type>))

SR060 22003 position: cannot check every 0th element of vector of type <type_name>

SR061 22023 position: expects a vector whose length is divisible by <d>, not of length <d2> (of type <type_name> (<dv_type>))

SR062 22023 Row in a row function is not a valid row string

SR063 22023 row string must begin with container header

SR064 22015 Conversion overflow from numeric

SR065 22023 Can't convert SQL_DATA_AT_EXEC blob to varchar. Parameter may only be used in insert or update

SR066 22023 Unsupported case in CONVERT

SR067 22023 Collation specified in cast for non-string datatype <type_name>

SR068 22023 XML tree cannot be used as argument of blob_to_string[_output]

SR069 22001 Attempt to convert a persistent XML document longer than VARCHAR maximum in blob_to_string

SR070 22023 blob_to_string[_output] requires a blob or string argument

SR071 22023 Blob argument to blob_to_string[_output] must be a non-interactive blob

SR072 22001 Blob longer than maximum string length not allowed in blob_to_string

SR073 22023 Function log_text needs an array as argument <index> not BLOB HANDLE

SR074 25000 replay must be run in a fresh transaction.

SR075 22023 Bad isolation. Must be uncommitted / committed / repeatable / serializable

SR076 22023 ISOLATION option needs a string as value (uncommitted / committed / repeatable / serializable)

SR077 42S22 Bad option for SET

SR078 22005 The cursor parameter is not settable

SR079 22023 Too few arguments to exec_next(cursor, state, message, row)

SR080 22023 Parameter 4 is not a valid local exec handle

SR081 22023 Parameter 1 is not a valid local exec handle

SR082 22003 Non numeric comparison

SR083 Numeric error (22003 Numeric value out of range | 22012 Division by zero | 37000 Syntax error | S1000 General error | S1107 Row value out of range | 22012 Division by 0)

SR084 42S22 The column <column_name> is not defined in the given table

SR085 22012 Division by 0

SR086 22012 Division by 0

SR087 22003 Non numeric arguments to arithmetic operation

SR088 22012 Division by 0

SR089 22012 Division by 0

SR090 22012 Division by 0

SR091 22003 Non numeric arguments to arithmetic operation modulo

SR092 42000 system call not allowed on this server

SR093 22000 Error in compressing

SR094 22023 string_output_gz_compress needs a string_output as a first argument, not an argument of type <type> (<DV_type>)

SR095 22023 gz_uncompress needs a string_output as a second argument, not an argument of type <type> (<DV_type>)

SR096 21S01 Function crypt needs a string or integer as its second argument. Not an arg of type <type> (<DV_type>)

SR097 2201B regexp error : could not obtain substring (<offset> of <length>)

SR098 2201B regexp error after: <pattern> (<error>)

SR099 22023 BLOB submitted by client as SQL_DATA_AT_EXEC cannot be converted into anything. It must first be stored into a BLOB column

SR100 22003 The requested subsequence of BLOB is longer than 10Mb, thus it cannot be stored as a string

SR101 22023 BLOB submitted by client as SQL_DATA_AT_EXEC cannot be converted into anything. It must first be stored into a BLOB column

SR102 22026 Error in compressing (invalid input)

SR103 Zlib error (22000 Error in compressing | 22003 Error in compressing (compression level parameter is invalid) | 22005 Error in compressing (not enough memory) | 22026 Error in compressing (not enough room in the output buffer))

SR104 22025 Error in decompressing

SR105 42000 Unclassified SQL error

SR106 40009 Out of disk on database

SR107 42000 Read only transaction for modify operation.

SR108 4000X Transaction could not commit after DDL statement. Last DDL statement rolled back.

SR109 42000 Cannot free global server lock if one does not hold it

SR110 42000 backup () must be the first operation in its transaction if the transaction is not read only

SR111 40009 Backup transaction failed

SR112 42000 Transaction not in backup mode

SR113 42000 backup_row needs a _ROW as argument

SR114 42000 Transaction not in backup mode

SR115 42000 Error writing backup_flush

SR116 42000 Transaction not in backup mode

SR117 42000 db_check () must be in read only, non-autocommit transaction mode. e.g. do it from isql in RO mode.

SR118 40009 Database check transaction failed

SR119 42S11 Key <key> has 0 parts. Create index probably failed

SR120 23000 Convert error or not possible to insert a value into a blob column

SR121 42000 Ruling part too long on <key>

SR122 42000 Row too long on <key>

SR123 42S12 Bad key id in row_insert

SR124 42S12 Bad key id in row_insert

SR125 42S12 key_insert: This table does not have this key

SR126 22018 Can't convert <column_name> to numeric

SR127 22003 Can't convert <column_name> to numeric

SR128 22003 Numeric value out of range for <column_name> (<col_scale>, <col_prec>)

SR129 22003 Numeric value out of range

SR130 22005 Bad value for numeric column <column_name>, dtp = <dtp_value>

SR131 22005 Cannot convert <string_value> to number for column <column_name>

SR132 22007 Bad value for date / time column

SR133 23000 Null assigned to non-null column

SR134 42000 A data at exec blob must be assigned to a blob column

SR135 42000 Blob too long to store in a non-blob column

SR136 42S02 Bad table name in GRANT / REVOKE <table_name>

SR137 42000 Bad user name in GRANT/REVOKE

SR138 37000 Operation not allowed for PUBLIC

SR139 42000 No group <group_name>

SR140 42000 The group <group_name> is already a secondary group for the user <user_name>. Delete it first with DELETE USER GROUP.

SR141 42000 No user <user_name>

SR142 37000 Operation not allowed for PUBLIC

SR143 42000 No group <group_name>

SR144 42000 Group <group_name> already assigned as a secondary group of <user_name>

SR145 42000 Group <group_name> already is a primary group of <user_name>

SR146 42000 No user <user_name>

SR147 37000 Operation not allowed for PUBLIC

SR148 42000 No group <group_name>

SR149 42000 No group <group_name> granted to <user_name>

SR150 42000 No Group <group_name> granted to <user_name>

SR151 42000 Group <group_name> is a primary group of <user_name>. Use SET USER GROUP instead

SR152 42000 No user <user_name>

SR153 28000 No user in delete user

SR154 42S02 No table or view in drop table

SR155 42S02 No table in alter table drop column

SR156 42S02 No table in table rename

SR157 42S02 No table alter table add column

SR158 42000 Permission denied. Must be owner of object or member of dba group.

SR159 42000 Function <function_name> restricted to dba group.

SR160 42000 Incorrect old password in set password

SR161 42000 Unsupported security statement.

SR162 07006 BLOB's not allowed in function refs.

SR163 42000 Ruling part too long.

SR164 42000 Error writing the BLOB onto a procedure view temp space

SR165 07006 Parameter BLOB in sorted result.

SR166 42000 Key too long in ORDER BY or DISTINCT temp table.

SR167 07006 Parameter BLOB in sorted result.

SR168 42000 Data too long in ORDER BY or DISTINCT temp table.

SR169 22023 signal state should be an integer 100 (NO DATA FOUND) or a string value, not an <type_name>

SR170 42000 Unsupported arithmetic op.

SR171 S1T00 Transaction timed out.

SR172 40001 Transaction deadlocked

SR173 40003 Transaction out of disk.

SR174 40004 Log out of disk.

SR175 23000 Uniqueness violation. Transaction killed.

SR176 4000X Transaction rolled back due to previous SQL error.

SR177 4000X Misc Transaction Error.

SR178 42000 Stack overflow

SR179 07S01 The function <function_name> does not accept a keyword parameter <param_name>

SR180 07S01 Extra arguments to <function_name>, takes only <number>

SR181 HY105 Cannot pass literal as reference parameter.

SR182 HY502 inout or out parameter <param_name> not supplied in keyword parameter call

SR183 07S01 Required argument <param_name> (no <index>) not supplied to <function_name>

SR184 42000 Built-in function is not allowed as the outermost function in a procedure view. Define an intermediate PL function to call the bif.

SR185 42001 Undefined procedure <procedure name>

SR186 42000 No permission to execute procedure <procedure_name>

SR187 07001 Too few actual parameters for <procedure_name>

SR188 HY105 Cannot pass literal as reference parameter.

SR189 HY008 Async statement killed by SQLCancel.

SR190 24000 Fetch of unopened cursor.

SR191 39000 Unsupported instruction AREF");

SR192 HY105 Like pattern not a string

SR193 HY105 Blob of <length> bytes in like

SR194 HY105 LIKE must be between strings.

SR195 HY001 No starting point for ancestor join

SR196 HY001 No place in from_position clause.

SR197 23000 Non unique primary key on <table_name>

SR198 HY109 Cursor not positioned on delete. <cursor_name>

SR199 HY109 The cursor in positioned DELETE was not on any row.

SR200 42S12 The row being deleted has no valid key.

SR201 42S12 Primary key not found in delete.

SR202 42000 Bad admin op code.

SR203 HY008 Async statement killed by SQLCancel.

SR204 22002 Missing named parameter

SR205 07001 Not enough actual parameters.

SR206 07001 Not enough actual parameters.

SR207 08004 Maximum licensed connections exceeded

SR208 08004 Application access is denied

SR209 S1010 Statement active

SR210 S1010 Async exec busy

SR211 41000 Aborted

SR212 S1010 Statement not prepared.

SR213 S1010 SQLFetch of busy

SR214 40001 Out of server threads. Server temporarily unavailable. Transaction rolled back.

SR215 42000 Too many triggers on <table_name>

SR216 22003 PL Scrollable cursor with a rowset > 1

SR217 01001 Optimistic cursor updated since last read"

SR218 HYC00 Cursor not capable of requested SQLSetPos operation

SR219 HY107 Row in SQLSetPos does not exist in the rowset

SR220 S1010 Unopened cursor referenced by SQLSetPos

SR221 S1010 Not a scrollable cursor in SQLSetPos

SR222 22003 PL cursor with a rowset greater then 1

SR223 HY111 Non static bookmark for a static cursor

SR224 HY111 Incompatible bookmark. Must be identical ordering and primary key columns

SR225 HY111 Static bookmark for a dynamic / keyset cursor

SR226 42000 Misc. cursor error

SR227 3C000 Non unique cursor name

SR228 07001 Too few actual parameters

SR229 42000 Misc. cursor error

SR230 S1010 Statement not executing or not scrollable cursor in SQLExtendedFetch

SR231 42S02 Cursor does not have table <table_name>

SR232 34000 No cursor <cursor_name>

SR233 22023 Wrong type of argument to __scroll_cr_init

SR234 22023 Function <function_name> needs a cursor as argument <index> , not an arg of type <type_name> (<dv_type>)

SR235 24000 Virtuoso/PL Scrollable cursor not opened

SR236 07001 scrollable fetch with different number of output columns

SR237 HY109 Row deleted

SR238 24000 Virtuoso/PL Scrollable cursor not opened

SR239 HY109 Virtuoso/PL Scrollable cursor not positioned on a row

SR240 HY090 Ruling part too long in scrolled current of reference.

SR241 HY109 Row referenced in where current of not present

SR242 42S22 No system status variable <name>

SR243 42S02 No table <table_name> in key_stat

SR244 22023 Allowed stat names are touches, reads, lock_set, lock_waits, deadlocks.

SR245 42S12 Index <key_name> not found in key_stat.

SR246 23000 Error or type mismatch inserting a blob

SR247 42000 Primary key ruling part too long in update

SR248 42000 Row too long in update

SR249 42000 Ruling part too long on <key_name>.

SR250 24000 Cursor not positioned on update. <cursor_name>

SR251 24000 Cursor not on row in positioned UPDATE

SR252 01001 Row deleted while waiting to update

SR253 42S12 Could not find primary key on update.

SR254 42000 The cursor specified is not a SELECT.

SR255 24000 Cursor before first or after end. No current row.

SR256 HY109 Cursor <cursor_name> does not have place.

SR257 34000 No cursor named <cursor_name>.

SR258 42000 The cursor specified is not a SELECT.

SR259 24000 Cursor <cursor_name> before first or after end. No current row.

SR260 HY109 Cursor <cursor_name> does not have place <name>.

SR261 23000 Non unique primary key.

SR262 24000 Cursor not positioned on positioned reference. <cursor_name>

SR263 24000 Cursor not on row in positioned UPDATE

SR264 42S12 Could not find primary key on update.

SR265 37000 Table being dropped is referenced in FOREIGN KEY

SR266 S0002 No table in drop table

SR267 37000 Table being dropped is referenced in FOREIGN KEY

SR268 42S02 No table in drop table

SR269 S0002 No trigger in drop trigger. Make sure the name is qualified with the subject table's qualifier and owner if these are not the default qualifier and owner

SR270 42000 No procedure in DROP PROCEDURE <proc_name>

SR271 42000 Must be in dba group to drop non-owned procedures

SR272 01006 Privilege has not been granted. Use list_grants (0) to see what you can revoke

SR273 42000 No user to delete

SR274 S1001 Column already exists in ALTER TABLE

SR275 42000 Column count too large

SR276 42S02 No table in add column

SR277 42S01 Table already exists in rename table.

SR278 42S02 Bad table in rename table

SR279 22023 parse_collation : invalid table size <size>

SR280 37000 The column is a key or primary key part. Drop the index first

SR281 37000 The column is referenced in foreign key constraint. Drop the foreign key first

SR282 42S22 The column to drop is not defined in the given table

SR283 42S02 No table in add column

SR284 22023 Charset definition is not correct

SR285 42000 Function user_set_password restricted to dba group

SR286 42000 The user <user_name> does not exist

SR287 42000 The new password for <user_name> cannot be empty

SR288 42S22 No columns in the source table <table_name>

SR289 42S22 No columns in the source table <table_name>

SR290 42S22 The column <column_name> in the source table <table_name> doesn't have a match in the destination table <table_name>

SR291 42S11 Primary key modification is prohibited if: Table is a subtable or has subtables, is attached or has a free text index

SR292 42S22 Bad new pk column list

SR293 22023 function fk_check_input_values expect -1, 0 or 1 as first parameter

SR294 42S22 Foreign key references invalid column <column_name> in referenced table <table_name>

SR295 37000 Foreign key references non-unique column <column_name> in table <table_name>

SR296 37000 To add foreign key REFERENCES should be specified

SR297 42S12 Foreign key does not exist in table <table_name> referencing table <table_name>

SR298 42S12 Foreign key <key_name> does not exists

SR299 37000 To drop foreign key should be specified NAME or REFERENCES

SR300 42S11 To modify a foreign key first drop the old and then add the new

SR301 37000 A primary key cannot be added or dropped. It can only be modified. Use alter table .. modify primary key ...

SR302 37000 To modify a unique constraint first drop the old and then add the new

SR303 22023 Function check_col needs string as argument

SR304 S1000 DELETE statement conflicted with COLUMN REFERENCE constraint <name>

SR305 S1000 UPDATE statement conflicted with COLUMN REFERENCE constraint <name>

SR306 S1000 INSERT statement conflicted with FOREIGN KEY constraint referencing table <table_name>

SR307 S1000 UPDATE statement conflicted with FOREIGN KEY constraint referencing table <table_name>

SR308 22023 exec() called with an invalid text to execute

SR309 22023 Non-encrypted session

SR310 22023 Invalid certificate info index

SR311 08004 Shutting down the server permitted only to DBA group

SR312 08004 The server is shutting down

SR313 42000 The procedure <proc_name> is part of a module <module>. Drop the module first.

SR314 42000 Trying to drop a module with drop procedure.

SR315 37000 Invalid module name in soap_sdl.

SR316 37000 No URL specified & soap_sdl called outside HTTP context.

SR317 37000 Invalid module name in soap_wsdl.

SR318 37000 No URL specified & soap_wsdl called outside HTTP context.

SR319 22026 Value too long to insert into VARCHAR column <col_name>.

SR320 42S12 Constraint <constraint name> does not exists for table <table>

SR321 37000 A RETURN statement with a return status can only be used in a stored procedure

SR322 22005 <option> is not valid trace_on option

SR323 22005 <option> is not valid trace_off option

SR324 08U01 Remote server has disconnected making the transaction uncommittable. Transaction has been rolled back.

SR325 S1T00 Transaction aborted due to a database checkpoint or database-wide atomic operation. Please retry transaction

SR326 42S02 No table in alter table modify column

SR327 42S02 ALTER TABLE not supported for views. Drop the view and recreate it instead.

SR328 42S02 ALTER TABLE not supported for views. Drop the view and recreate it instead.

SR329 42S02 ALTER TABLE not supported for views. Drop the view and recreate it instead.

SR330 42S02 ALTER TABLE not supported for views. Drop the view and recreate it instead.

SR331 21000 Scalar subquery returned more than one value.

SR332 42000 The DB.DBA.DBEV_DSN_LOGIN should return strings for UID and PWD.

SR333 42S22 Access denied for foreign key referencing <col> in table <tbl>

SR334 22023 The result names description should be an array in exec_result

SR335 22023 The result names description should be an array in exec_result_names

SR336 22023 Wrong result description in bif_result_string_names

SR337 40006 Transaction aborted because the server is out of memory

SR338 HY111 Malformed bookmark

SR339 22023 Function <function_name> needs an integer not less than <val> as argument <inx>

SR340 22023 Function <function_name> needs an integer not greater than <val> as argument <inx>

SR341 22023 Invalid integer value converting <string_val>

SR342 22023 Procedure view's procedure returned value of type <type>(<dv_val>) instead of <type>(<dv_val>) for column <col_no>

SR343 42000 Access to pwd_magic_calc not permitted. If you are getting this message in the Admin interface and you are a DBA, then you need to enable the function from the INI file in order to use it.

SR344 41000 Malformed RPC

SR345 22023 invalid offset arguments to subseq

SR346 22023 precision (<prec>) overflow in <title>

SR347 22023 The long varchar, long varbinary and long nvarchar data types cannot be used in the WHERE, HAVING, or ON clause, except with the IS NULL predicate for column <name>

SR348 22023 Value type (<type>) not suitable for use in a hash index

SR349 22023 SKIP parameter < 0

SR350 22023 TOP parameter < 0

SR351 22023 SKIP parameter < 0

SR352 22023 TOP parameter < 0

SR353 22023 Sorted TOP clause specifies more then %ld rows to sort. Only %ld are allowed. Either decrease the offset and/or row count or use a scrollable cursor

SR354 23000 Type mismatch inserting user defined type instance as a blob for column <co>

<title>File Access</title>

FA001 42000 Access to <file_name> is denied

FA002 42000 Can't open file <file_name>, error <err_no>

FA003 42000 Access to <file_name> is denied due to access control in ini file

FA004 42000 Access to <file_name> is denied due to access control in ini file

FA005 39000 Can't open file <file_name>, error <error_no>

FA006 39000 Can't open file <file_name>, error : <fs_error>

FA007 39000 Seek error in file <file_name>, error <error_no>

FA008 39000 File <file_name> too long

FA009 39000 Read from <file_name> failed (<error_no>)

FA010 42000 Access to <file_name> is denied due to access control in ini file

FA011 42000 Access to <file_name> is denied due to access control in ini file

FA012 42000 Can't open file <file_name>, error <error_no>

FA013 39000 Read from <file_name> failed (<error_no>)

FA014 42000 Access to <file_name> is denied due to access control in ini file

FA015 42000 Access to <file_name> is denied due to access control in ini file

FA016 42000 Access to <file_name> is denied due to access control in ini file

FA017 39000 Path string is too long.

FA018 42000 Access to <file_name> is denied due to access control in ini file

FA019 39000 Path string is too long.

FA020 39000 <fs_error>

FA021 22003 Third argument of string_to_file function, should be nonnegative offset value, -1 or -2

FA022 22023 Function string_to_file needs a string or blob or string_output as argument 2, not an arg of type <type_name> (<dv_type>)

FA023 42000 Access to <file_name> is denied due to access control in ini file

FA024 42000 Access to <file_name> is denied due to access control in ini file

FA025 39000 Seek error in file <file_name>, error : <fs_error>

FA026 39000 Write to <file_name> failed (<fs_error>)

FA027 39000 Write to <file_name> failed"

FA028 39000 Write to <file_name> failed (<fs_error>)

FA029 39000 Write to <file_name> failed (fs_error)

FA030 39000 Can't open file <file_name>

FA031 39000 Can't open file <file_name>

FA032 39000 Can't open file <file_name>

FA033 39000 Can't open file <file_name>

FA034 39000 Can't open file <file_name>

FA035 39000 Can't open file <file_name>

FA036 42000 Allow & deny file ACL cannot be modified

FA037 39000 Can't update <file_name>

FA038 42000 Cannot open backup file <file_name>

FA039 40003 Out of disk in query temp space.

FA040 42000 Access to <file_name> is denied due to access control in ini file

FA041 42000 Debug session is already open, close it first

FA042 39000 Cannot open debug log file <file_name>, error : <OS error>

<title>SQL Compiler</title>

SQ001 28000 No owner user in __view_changed for <text>

SQ002 42S02 Bad table in drop trigger.

SQ003 42S22 Collation <collation_name> is not defined

SQ004 42S22 Collation defined for a non-string column

SQ005 42000 Column count too large

SQ006 42S22 Table has non unique column names either directly through inheritance

SQ007 37000 Column count too large

SQ008 42S22 No column <name> in <table_name>.

SQ009 07006 Column <name> is a BLOB column and blob columns are not supported as index parts

SQ010 42S22 Column count too large

SQ011 42S22 Column count too large

SQ012 42S12 Cannot inherit table with no keys <name>.

SQ013 42S22 Column count too large

SQ014 07006 Not a DDL type name: <name>

SQ015 07006 Not a DDL type name: <name>

SQ016 42S01 Table <table_name> already exists

SQ017 42S12 No primary key for <table_name>. Specify qualifier and owner if you are not owner of the table.

SQ018 42S02 No table <table_name>

SQ019 42S12 No key <name> in (<table_name>|any table)."

SQ020 23000 Can't drop primary key. Use drop table instead.

SQ021 42S02 bad table or index for build key

SQ022 42S11 Duplicate index name

SQ023 42S02 No table in rename table

SQ024 4000X Transaction could not commit after DDL statement. Last DDL statement rolled back

SQ025 42S02 Bad table in drop table.

SQ026 42000 The subtable <table_name> must have the same owner as the supertable <table_name>

SQ027 42S11 Only one PRIMARY KEY clause allowed

SQ028 42S21 Unsupported table constraint.

SQ029 42S12 A table must either have an UNDER or PRIMARY KEY specification.

SQ030 42S11 A table cannot have both an UNDER and PRIMARY KEY.

SQ031 42000 Unsupported DDL statement.

SQ032 42000 Owner of procedure has been deleted. Cannot recompile procedure

SQ033 42000 Access denied for column <name>.",

SQ034 42S22 Ambiguous column ref <name>

SQ035 37000 Bad function reference in expression, only ones in selection recognized in HAVING / ORDER BY

SQ036 42S22 Bad column/variable ref <name>

SQ037 37000 ancestor_of must be between 2 columns

SQ038 37000 ancestor_of must be between 2 entity subtables

SQ039 37000 Can't have more than 1 ancestor join per table

SQ040 37000 Can't have more than 1 contains/xcontains per table

SQ041 37000 Can't have more than one xpath_contains per table

SQ042 42S12 No index for table <table_name>

SQ043 42000 _ROW requires select permission on the entire table.

SQ044 42S22 Bad column/variable reference <name>

SQ045 42000 Bad compare operator.

SQ046 42S22 Bad column predicate.

SQ047 42S02 No table <table_name>

SQ048 42S12 No key <key_name>

SQ049 42S12 No key named <key_name>

SQ050 42S22 <name> is not the name of a CURRENT OF.

SQ051 42S02 No table <table_name>

SQ052 21S01 Uneven col and value lists in insert.

SQ053 42S22 No such column <name>

SQ054 42000 Bad insert mode.

SQ055 21S01 Odd assignment list for update.

SQ056 42S02 No table <table_name> in update.

SQ057 42S22 No such column <name> in update.

SQ058 42S12 No key in key_insert.

SQ059 42000 No statement <name>

SQ060 42000 Must be in dba group to use EQL.

SQ061 42000 Lisp reader error.

SQ062 42S22 No column <owner>.<name>.

SQ063 42S22 No column <name>

SQ064 42000 Illegal use of '*'.

SQ065 42S22 Col ref ambiguous <name>.

SQ066 37000 Natural join only allowed between tables or views. No derived tables or joins.

SQ067 37000 Explicit join condition not allowed in natural join

SQ068 37000 Empty or USING join condition not allowed with non-natural join

SQ069 42S02 No table <table_name>

SQ070 42000 Must have select privileges on view <view_name>

SQ071 42000 Cannot mix aggregate functions with non-aggregate functions or columns not included in GROUP BY.

SQ072 42000 Goto to undeclared label.

SQ073 37000 Statement not supported.

SQ074 37000 <parser error>

SQ075 42000 Unclassified SQL compilation error.

SQ076 42000 The procedure owner specified is different than the creator.

SQ077 34000 Bad cursor name <cursor_name>

SQ078 42000 Unclassified SQL compilation error.

SQ079 42000 No permission to execute procedure <procedure_name>.

SQ080 37000 ELSE must be last clause in CASE.

SQ081 37000 ELSE must be last clause in CASE.

SQ082 37000 Reference to non-object variable

SQ083 42000 Can't generate scalar exp <type>

SQ084 42000 Subquery predicate not supported.

SQ085 42000 Reference to undefined label.

SQ086 42000 ORDER BY: less than <number> cols in select.

SQ087 09000 Forward only cursor called not with FETCH NEXT

SQ088 39000 Statement not supported in a procedure context.

SQ089 S0022 Bad column <name> in trigger column list

SQ090 42S02 Bad table <table_name> in trigger <name> definition

SQ091 42000 Views not supported as trigger objects

SQ092 42000 Access denied for table <table_name>

SQ093 21S01 Mismatched columns and values in insert.

SQ094 21S01 Too many (<number>) values in insert into <table_name>

SQ095 42S22 No column <name>

SQ096 42S02 No table <table_name>

SQ097 42000 No insert or insert/delete permission for insert / insert replacing

SQ098 42S22 No column <name>

SQ099 21S01 different number of cols and values in insert.

SQ100 42000 A local table of over maximum columns may not be inserted

SQ101 42S02 No table <table_name>

SQ102 42S22 No column <name>

SQ103 42000 Update of <name> not allowed.

SQ104 09000 Cursor with a sorted order by, distinct, grouping etc. is not referenceable in where current of

SQ105 42S02 No table <table_name>

SQ106 42S22 No column <name>.

SQ107 42000 Update of <table_name> not allowed.

SQ108 43000 Permission denied for delete.

SQ109 09000 Cursor with a sorted order by, distinct, grouping etc. is not referenceable in where current of

SQ110 42000 Permission denied for delete.

SQ111 42S02 No table <table_name>

SQ112 42S02 View without table <view_name>

SQ113 37000 Non-view column set in view update

SQ114 37000 View <name> is not updatable in insert.

SQ115 37000 Non-updatable column <name> in view <view_name> (expression or constant)

SQ116 37000 View <name> is not updatable.

SQ117 42000 No column update privilege for <name> in view <view_name>

SQ118 37000 Non-updatable column <name> in view <view_name> (expression or constant)

SQ119 37000 View <name> is not updatable.

SQ120 S0022 Bad CORRESPONDING BY column <name>

SQ121 42S02 No table <table_name>

SQ122 42000 Must have select privileges on view <name>

SQ123 42S02 foreign key references non-existent table

SQ124 42S02 Foreign key reference to non existent table

SQ125 37000 Different number of referencing and referenced columns in foreign key declaration from <table_name> to <table_name>

SQ126 42S22 Foreign key references invalid column <name> in referencing table <table_name>

SQ127 42000 The user has been deleted

SQ128 42000 Unsupported table constraint.

SQ129 42S22 No column <column_name> in <table_name>.

SQ130 42000 The count of supplied parameters to Virtuoso/PL FETCH statement does not match the count of the selected columns.

SQ131 42S22 Column <column_name> from a derived table not found

SQ132 42S01 Table <table_name> already exists

SQ133 37000 Procedure declaration tries to overwrite a module with the same name

SQ134 42000 Table already has an IDENTIFIED BY option

SQ135 42000 Duplicate label name <name>

SQ136 37000 Invalid escape character in LIKE

SQ137 37000 Dots not allowed inside column names

SQ138 37000 Comma expressions not allowed in an IN clause

SQ139 37000 Different number of subquery output columns for a predicate

SQ140 37000 Module declaration tries to overwrite a module with the same name. Drop the module first.

SQ141 37000 Column is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.

SQ142 37000 Different number of expected and generated columns in a select

SQ143 37000 Tables '<tb1>' and '<tb2>' have the same exposed names. Use correlation names to distinguish them.

SQ144 21S01 different number of cols and values in insert.

SQ145 21S01 different number of cols and values in insert.

SQ146 21S01 different number of cols and values in insert.

SQ147 37000 Incorrect ORDER BY clause for query expression.

SQ148 37000 TOP not supported in IN, ANY, SOME, ALL, ONE subqueries

SQ149 42S22 The column '<column name>' was specified multiple times for '<table name>'

SQ150 37000 Column '<col>' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.

SQ151 S0022 Invalid alias declaration: insert trigger cannot reference old values.

SQ152 S0022 Invalid alias declaration: delete trigger cannot reference new values.

SQ153 37000 Other joins in the joined table of another outer join not supported.

SQ154 37000 Error in trying to get the return value of a remote procedure.

SQ155 37000 General internal [Optimized] compiler error in <file>:<line>. Please report the statement compiled.

SQ156 37000 Internal [Optimized] compiler error in <file>:<line>: <message>. Please report the statement compiled.

SQ157 42S02 The supertable <table> in UNDER is not defined.

SQ158 42S12 The supertable <table> in UNDER has no primary key.

SQ159 42S21 Column names in each table must be unique. Column name <col> in table <table> conflicts with a column of the supertable <table>.

SQ160 42000 No Select permission on the table <table>

SQ161 42000 No select permission on the table <table>

SQ162 42000 No insert or insert/delete permission for insert / insert replacing

SQ164 42000 Update of <table> not allowed.

SQ165 37000 Empty namespace name is not valid in create xml view <view>

SQ166 42000 Reference to undefined label.

SQ167 22023 The long varchar, long varbinary and long nvarchar data types cannot be used in the WHERE, HAVING, or ON clause, except with the IS NULL predicate for column <name>

SQ168 22023 The long varchar, long varbinary and long nvarchar data types cannot be used in the WHERE, HAVING, or ON clause, except with the IS NULL predicate for column <name>

SQ169 37000 Invalid fetch direction in FETCH

SQ170 37000 Key <key> is inherited from table <table>. Drop the index from it.

SQ171 42000 an SQL <module|stored procedure|user defined type> with name <name> already exists

SQ172 37000 Invalid external name in CREATE PROCEDURE

SQ173 42000 Maximum number of keys (<num>) already created

<title>Client Side</title>

CL001 IM001 Function not supported : <function_name>

CL002 HY010 Unprepared statement in SQLExtendedFetch

CL003 HY106 Bad fetch type for forward only cursor

CL004 HY107 Specified keyset size must be >= the rowset size

CL005 HY106 Bookmarks not enabled or no bookmark retrieved

CL006 HY111 Bad bookmark for SQLExtendedFetch

CL007 S1010 SQLSetPos only allowed after SQLExtendedFetch

CL008 HY092 SQLSetPos irow out of range

CL009 HY109 Only SQL_POSITION SQLSetPos option supported for forward cursors

CL010 01S02 Option value changed

CL011 01S02 Option value changed

CL012 01S02 Option value changed

CL013 01S02 Option value changed

CL014 01S02 Option value changed

CL015 IM001 Driver does not support this function

CL016 01S02 Option value changed

CL017 IM001 Driver does not support this function

CL018 01S02 Option value changed

CL019 01S02 Option Value Changed

CL020 HY091 Invalid descriptor field identifier

CL021 HY091 Invalid descriptor type

CL022 HY091 Invalid descriptor type

CL023 HY091 Invalid descriptor type

CL024 HY091 Not supported

CL025 HY091 Invalid descriptor type

CL026 IM001 Driver doesn't support this function

CL027 HYC00 Optional feature not supported

CL028 S1010 Statement not prepared.

CL029 07005 Statement does not have output cols.

CL030 07009 Bookmarks not enabled for statement

CL031 S1002 Column index too large.

CL032 S1C00 Information not available.

CL033 S2801 Connect failed to <dsn> = <host_port>

CL034 28000 Bad login

CL035 2C000 Charset <name> not available. Server default <name2> will be used.

CL036 01S02 Switching to the server default charset <name>.

CL037 S1010 Statement not prepared.

CL038 07005 Statement does not have output cols.

CL039 07009 Bookmarks not enabled for statement

CL040 S1002 Column index too large.

CL041 HY010 Can't mix SQLFetch and SQLExtendedFetch.

CL042 HY010 Statement not prepared.

CL043 08S01 Connection lost to server

CL044 07009 Bad parameter index in SQLDescribeParam

CL045 08S01 Lost connection to server

CL046 40001 could not enlist transaction

CL047 25000 could not enlist resource manager in transaction

CL048 S1009 Information not available.

CL049 HY010 Can't mix SQLFetch and SQLExtendedFetch.

CL050 S1010 Bad call to SQLParamData

CL051 S1010 No param was asked for.

CL052 S1010 Bad place to call SQLPutData

CL053 S1010 No data was asked for.

CL054 S1010 Invalid buffer length (even) in passing character data to binary column in SQLPutData

CL055 S1010 Invalid buffer length (even) in passing character data to binary column in SQLPutData

CL056 07009 Bookmarks not enable for statement

CL057 S1010 Statement not fetched in SQLGetData.

CL058 07009 Column out of range in SQLGetData

CL059 01004 Data truncated.

CL060 07006 Non string data received with SQLGetData.

CL061 S1010 Can't set scroll on open cursor

CL062 S1090 Invalid string or buffer length

CL063 S1010 Async call in progress

CL064 08S01 Lost connection to server

CL065 08S01 Lost connection to server

CL066 S1T00 Virtuoso Communications Link Failure (timeout)

CL067 08S01 Lost connection to server

CL068 01000 Non last proc w/array params returned

CL069 22002 Invalid (odd) length in conversion from SQL_C_CHAR to SQL_BINARY

CL070 S1010 Invalid buffer length (even) in passing character data to binary column in SQLPutData

CL071 01004 Data truncated.

CL072 01004 Data truncated.

CL073 01004 Data truncated.

CL074 01004 Data truncated

CL075 01004 Data truncated

CL076 01004 Data truncated.

CL077 01004 Data truncated.

CL078 01004 Data truncated.

CL079 01004 Data truncated.

CL080 01004 Data truncated.

CL081 01S07 Numeric truncated by client

CL082 01S01 Error in row in SQLSetPos

CL083 IM001 Unable to handle array parameters on a scrollable cursor

CL084 IM001 Unable to handle DATA_AT_EXEC params in SQLSetPos

CL085 HY020 Attempt to concatenate NULL value

CL086 28000 Password to be sent in cleartext with no encryption

CL087 01004 String data, right truncation

CL088 01004 String data, right truncation

CL089 42000 Not connected to the data source

CL090 01004 Data truncated

<title>Client Side - Clustering</title>

08C02 CLDPN luster operation cancelled because of peer disconnect HOST:CLUSTER_PORT

This is an inter-cluster communication error which requires the DBA to restart the node corresponding to the HOST:CLUSTERPORT.

<title>HTTP Server</title>

HT001 22023 An interactive blob can't be passed as argument to ses_write

HT002 22023 ses_write requires string or blob as argument 1

HT003 08003 cannot write to session

HT004 42001 No DB.DBA.__HTTP_XSLT defined

HT005 22023 No VSP parameter for /INLINEFILE

HT006 37000 http output function outside of http context and no stream specified: <function_name>

HT007 22023 An interactive blob can't be passed as argument to http

HT008 22023 http requires string or blob as argument 1

HT009 37000 XML output function allowed only inside HTTP request

HT010 42000 This function is only allowed processing a HTTP request

HT011 42000 XML output function allowed only inside HTTP request

HT012 42000 XML output function allowed only inside HTTP request

HT013 42000 http_path() function allowed only inside HTTP request

HT014 22023 Host name is too long

HT015 2E000 Cannot resolve host in http_get

HT016 08001 Cannot connect in http_get

HT017 08006 Error in reading from HTTP server

HT018 22023 URI too long in http_get <uri>

HT019 22023 Proxy must contain port number

HT020 2E000 Cannot resolve host in http_get <uri>

HT021 08001 Cannot connect in http_get <uri>

HT022 08007 Error in writing to the target HTTP server

HT023 08006 Error in reading from target HTTP server

HT024 42000 http_proxy only allowed inside HTTP request

HT025 42000 HTTP proxy function disabled.

HT026 08006 Error in mid read in http_get <uri>

HT027 08006 Error in mid read in http_get <uri>

HT028 22023 no http protocol identifier in http_get URI <uri>

HT029 08006 Misc. error in connection in http_get <uri>

HT030 08006 Error in mid read in http_get <uri>

HT031 22023 The HTTP output is not an STRING session in http_output_flush

HT032 42000 ses_read_line with no argument defaults it direct to the raw client connection. Allowed only inside HTTP request

HT033 08003 cannot read from session

HT034 42000 The http_flush not effective outside an VSP context

HT035 42000 The http_flush already done

HT036 42000 The http_auth not effective outside an VSP context

HT037 42000 The http_client_ip not effective outside an VSP context

HT038 08003 Server address not known

HT039 42000 Not allowed to call the http_xslt in an non VSP context

HT040 42000 http_map_get function outside of http context

HT041 39001 set_user_id function cannot be called twice in the same context.

HT042 22023 Not valid user id <name>

HT043 37000 bif_ses_write not called inside HTTP context

HT044 22023 The physical path must points to the dav domain.

HT045 22023 The port number of host and listen host must be the same.

HT046 22023 The listen host should be equal to SSLPort

HT047 37000 Max nesting level (20) reached when processing <url>

HT048 37000 Unterminated include tag at offset <ofs> in <name>

HT049 37000 An include tag at offset <ofs> with no name or VSP end tag before an include tag in <name>

HT050 22023 Path contains ..

HT051 42000 Invalid SOAP URL

HT052 42000 Persistent XML not allowed as an argument to soap_server

HT053 42000 Function http_body_read not allowed outside http_context

HT054 08006 Error reading the content in http_body_read

HT055 22023 The authentication function required in the authentication hook

HT056 42000 Searching for POST stream parameters not allowed outside HTTP context

HT057 22023 The STRING session is longer than 10Mb. Use substring to access it in parts.

HT058 22023 Missing leading slash in path parameter

HT059 42000 Proxy access denied to [host]

HT060 22023 The port number of SSL host and SSL listen host must be the same.

<title>Free-Text</title>

FT001 22026 Length limit of composite exceeded.

FT002 S0002 bad table for vt_words_next_d_id

FT003 22003 composite index out of range <n1> for length <n2>

FT004 24000 cursor in vt_words_next_d_id is not open or not on row

FT005 42000 vtb_match only allowed after vt_batch_strings

FT006 22023 Unknown encoding name <name>

FT007 22023 vt_batch_d_id requires a composite or a number as id

FT008 22023 composite document id over 32 characters long.

FT009 22023 vt_batch_d_id id's not in ascending order

FT010 22023 vt_batch_d_id id's cannot be 0, -1 or -2

FT011 22023 Bad XML entity tree in vt_batch_feed

FT012 22023 Bad XML entity tree in vt_batch_feed

FT013 22023 Bad XML entity tree in vt_batch_feed

FT014 42000 The vt_batch can't be used in vt_batch_feed after vt_batch_strings has been called.

FT015 22023 The vt_batch_feed can't index XML tree entity argument, only textual data and XPERs are allowed

FT016 22023 vt_batch_strings needs a string_output as a second argument, not an argument of type <type_name> (<dv_type>)

FT017 22003 word_string_insert max length too high

FT018 42S02 The table is not freetext indexed

FT019 22023 The new mode should be ON or OFF, not <mode>

FT020 42S22 No column

FT021 42S02 No table <table_name> in create text index

FT022 42S01 Only one text index allowed per table

FT023 37000 Use NOT INSERT flag, because function hooks does not generated before text index creation.

FT024 22023 the column <name> is not an integer

FT025 42000 id = id

FT026 42000 Misc. error upon update log creation. The free text index cannot be created.

FT027 22008 Invalid XML supplied for an validating free text index of <table_name>

FT028 22008 Invalid XML supplied for an validating free text index of <table_name>

FT029 22008 Invalid XML supplied for an validating free text index of <table_name>

FT030 22008 Invalid XML supplied for an validating free text index of <table_name>

FT031 37000 Setting initial state of the freetext index for <table_name> failed. The data is intact, but the freetext index is unusable. In order to recover from this state the table <name2> should be dropped. This will clear all these created so far. Then the freetext index creation should be retried after removing the cause of the error which is : SQL State [<sql_state>] Explanation : <sql_message>

FT032 42S02 The table <table_name> is not full-text indexed

FT033 Cannot create a scheduled event : <error message>

FT034 42S02 Text index not defined for <table_name>

FT035 42S02 Text index should be enabled for the table <table_name>

FT036 42S22 The id column <name> does not exist in table <table_name> definition

FT03742S22The data column <name> does not exist in table <table_name> definition x

FT038 22015 wildcard has over 1000 matches

FT039 22023 composite expected for composite_ref

FT040 09000 No cursor for vt_words_next_d_id

FT041 22023 negative offset not allowed in composite_ref

FT220 22023 function <function_name> expects a word batch as argument <inx>

FT370 22023 Wildcard word needs at least 3 leading characters

<title>2 Phase Commit</title>

DC001 22023 Bad option for SET 2PC

DC002 37100 MTS support is not enabled

<title>VDB</title>

VD001 22023 Value of vdb_timeout must be an integer

VD002 HY001 Not enough temp space to pass decimal parameter

VD003 HZ000 Cannot pass dtp <dv_type> as param <inx> to remote execute <text>

VD004 HZ000 Unknown result set column C type <c_type_code>

VD005 37000 Too few arguments for rexecute (dsn, text, state, message, params, max, desc, result[, handle)

VD006 37000 More then the maximum parameters supplied for rexecute

VD007 22023 Supplied DSN <dsn> is not valid

VD008 22023 The handle passed to rclose is not valid

VD009 22023 The handle passed to rnext is not valid

VD010 22023 The handle passed to rmoreresults is not valid

VD011 22023 Supplied DSN <dsn> is not valid

VD012 HZ000 Bad DSN in catalog call

VD013 HZ000 Bad DSN in catalog call

VD014 HZ000 Cannot remove DSN from ini

VD015 42000 Unknown windows version. The ODBCINST functionality disabled

VD016 42000 This feature runs only on Windows NT/Windows 2000. The ODBCINST functionality disabled

VD017 HZ000 Cannot get installed drivers

VD018 22002 Init string cannot be empty

VD019 22011 Init string should finish with ;

VD020 22026 DSN name is too long

VD021 22023 DSN=<DSN_NAME> parameter should be specified in init string

VD022 42000 Cannot add DSN

VD023 42000 Cannot write the DSN setting to INI

VD024 01S08 Cannot write the File DSN

VD025 HZ000 Array parameters not supported by <dsn>. Retry statement

VD026 42000 Inconsistent vdb subquery compilation

VD027 HY090 Remote statement text over 19K

VD028 HY090 Remote statement text over 19K

VD029 42S02 Cannot generate remote ref to col w/ no table <name>

VD030 37000 The _ROW virtual column cannot be referenced for remote tables.

VD031 37000 Cannot reprint node <inx> for remote text

VD032 remote prepare : <sql_error>

VD033 42000 No insert or insert/delete permission for insert / insert replacing

VD034 42S22 No column <name>

VD035 42S02 No table <name>

VD036 42S22 No column <name>

VD037 42000 Update of <name> not allowed.

VD038 S0002 No remote procedure <name>

VD039 42S02 No table <table_name>

VD040 42S02 Ref to wrong table in remote current of

VD041 42S22 No column <name>

VD042 42000 Update of <name> not allowed.

VD043 42S02 Ref to wrong table in remote current of

VD044 43000 Permission denied for delete.

VD045 42S02 Table in positioned delete is not the table of the cursor

VD046 S1000 Remote DSN <dsn> not defined

VD047 40001 Transaction killed during VDB call

VD048 40001 Transaction killed during VDB operation

VD049 HZ000 No remote DSN <dsn>

VD050 HZ000 No remote DSN <dsn>

VD051 HZ000 No remote DSN <dsn>

VD052 Remote DSN <dsn> : <err_message>

VD053 HY090 Blob larger than <len> bytes is too long for VDB buffering.

VD054 HY090 Blob larger than <len> bytes is too long for VDB buffering.

VD055 22003 Can't convert remote numeric <numeric_string> to local numeric

VD056 HY001 Not enough temp space to pass decimal param

VD057 22023 Blob parameters not allowed at this point

VD058 22023 Cannot pass dtp <dt_type> as param <inx> to remote query <text>.

VD059 07001 Too many parameters in the call to the remote procedure <name>

VD060 22023 Blobs can't be passed as non-input parameter <inx> to remote procedure <name>

VD061 HY105 The parameter <inx> of the remote procedure <name> is an output parameter

VD062 HZ000 Cannot attach tables from the same server

VD063 42S01 Table already exists in ATTACH

VD064 42S02 no remote table/view or ambiguous remote table/view <table_name> in DSN <dsn>

VD065 42S22 <name> is not a remote view <table_name> column

VD066 42S12 No unique key

VD067 HZ000 Driver not capable for SQLTables

VD068 HZ000 Driver not capable for SQLTables

VD069 HZ000 Driver not capable for SQLTables

VD070 42S01 Table <name> exists as local table, cannot unlink similarly named remote table

VD071 HY090 Blob larger than <len> bytes is too long for VDB buffering.

VD072 HY090 Blob of <len> bytes is too long for VDB buffering.

VD073 HZ000 Bad DSN in sql_columns

VD074 HZ000 Bad DSN in sql_tables

VD075 HZ000 Bad DSN in sql_primary_keys

VD076 HZ000 Bad DSN in sql_statistics

VD077 HZ000 Bad DSN in sql_procedures

VD078 42000 Invalid type of the parameter <num> to rexec. Should be 'INOUT' or 'OUT'

VD079 42000 Invalid datatype of the parameter <num> to rexec.

VD080 42000 Missing input value of the inout parameter <num> to rexec.

VD081 42000 Output parameters require the use of settable resultset variable in rexec.

VD082 42000 Output parameters don't allow use of an rexec cursor.

VD083 42000 Output parameters Output parameters require the use of settable parameters variable in rexec.

VD084 42000 DDL operation not allowed on a remote table '<table_name>'

VD085 42000 Can't (commit | rollback) dsn <dsn_name> because it is (not in manual commit | not transaction capable)

VD086 22023 The statement in RSTMTEXEC should return resultset

VD087 42000 No permission to rexecute on DSN <dsn>

<title>Date/Time Routines</title>

DT001 22007 Function <name> needs a datetime, date or time as argument <inx>, not an arg of type <type_name> (<dv_type>)

DT002 22023 Bad unit in datediff: <unit>

DT003 22015 Interval not supported in <function>: <interval>

DT004 22015 Bad interval in timestampdiff: <interval>.

DT005 22015 Bad interval in extract.

DT006 22007 Cannot convert <literal> to datetime

DT007 22002 Nulls not allowed as parameters to stringdate

DT008 25000 now/get_timestamp: No current txn for timestamp

DT009 22002 Nulls not allowed as parameters to stringtime

DT010 22007 Can't convert <literal> to time

DT011 22007 Can't convert <literal> to time

<title>Internationalization Routines</title>

IN001 2C000 The UTF-8 is not a redefinable charset

IN002 2C000 charset_define : Charset table not a wide string

IN003 2C000 crahset_define : 0 not allowed as a charset definition

IN004 2C000 charset <name> already defined. Drop it first

IN005 2C000 Alias <inx> not of type STRING

IN006 22023 Collation <name> not defined

IN007 2C000 Charset <name> not defined

IN008 2C000 Charset <name> not defined

IN009 22005 Bad value for wide string column <name>, dtp = <dv_type>.

IN010 22021 Untranslatable character in an output context that doesn't allow escapes.

IN011 22005 Bad value for wide string column <name>, type = <dv_type_title>.

IN012 2C000 Narrow source charset specified, but the supplied string is wide

IN013 2C000 Wide source charset specified, but the supplied string not wide

IN014 22005 Invalid data supplied in NVARCHAR -> VARBINARY conversion

IN015 22005 Invalid data supplied in VARBINARY -> NVARCHAR conversion

<title>Transactional Replication</title>

TR001 37000 The publication <name> doesn't exist

TR002 37000 Replication not enabled on this server

TR003 37000 The publication <name> doesn't exist

TR004 37000 The subscription <name> from <server> doesn't exist

TR005 37000 publication and subscription servers have identical names.

TR006 37000 Publishing server must be declared with REPL_SERVER before subscribing

TR007 37000 The subscription <name> already exist

TR008 22023 User name and password should be supplied when subscribe to new publisher

TR009 37000 The WebDAV collection <name> already exist

TR010 37000 The table <name> already exist

TR011 22023 The item type <type> not applicable

TR012 37000 Can't schedule local publication

TR014 22023 You should specify valid user name and password for replication synchronization

TR015 37000 Replication account <acct> from <server> doesn't exist

TR016 37000 The publication <name> does not exist

TR017 37000 The publication <name> doesn't exists.

TR018 37000 The WebDAV collection <name> doesn't exists.

TR019 37000 The DAV collection <name> must added before checkpoint.

TR020 22023 Function DAV_COL_I needs string or array as path

TR021 22023 The first parameter is not valid path string

TR022 37000 Non-existing collection

TR023 22023 Function DAV_COL_U needs strings as paths

TR024 22023 The second parameter is not valid path string

TR025 22023 The first parameter is not valid path string

TR026 37000 Non-existing collection

TR027 37000 Non-existing parent collection

TR028 22023 Function DAV_COL_D needs string as path

TR029 22023 The first parameter is not valid path string

TR030 37000 Non-existing collection

TR031 22023 Function DAV_RES_I needs string or array as path

TR032 22023 The first parameter is not valid path string

TR033 37000 Non-existing collection

TR034 22023 Function DAV_RES_D needs string as path

TR035 22023 The first parameter is not valid path string

TR036 37000 Non-existing resource

TR037 37000 The publication account <name> does't exist

TR038 37000 The dav publication <name> already exist

TR039 37000 The WebDAV collection <name> doesn't exist

TR040 37000 The table <name> doesn't exist

TR041 37000 The procedure name doesn't exist

TR042 37000 The procedures calls to <name> can not be replicated to more than one publication

TR043 22023 The item type <type> not applicable

TR044 37000 The table <name> doesn't exist

TR045 37000 The replication server <server> doesn't exist

TR046 37000 The target table <target_table> already exist

TR047 37000 The publication <acct> doesn't exist

TR048 22023 The grantee is invalid user name : <grantee>

TR049 37000 The publication <acct> doesn't exist

TR050 22023 The grantee is invalid user name : <grantee>

TR051 37000 The grantee <grantee> doesn't exist

TR052 22023 The path string is mandatory for DB.DBA.MKCOL

TR053 37000 Publication server <server> doesn't exist

TR054 37000 Table <tbl> doesn't exist

TR055 37000 The WebDAV collection <name> already exist

TR057 42000 The new name must be the same as in the ServerName (from INI file)

TR058 42000 General error, publication records can't be updated

TR059 42000 The current name of server cannot be altered.

TR060 42000 The name of new server used from another publishing server.

TR061 42000 The publication on "[new_name]" exists with the same name a on "[old_name]". The rename operation cannot be performed

TR065 22023 Replication account can not be empty

TR066 22023 Replication account can not be empty

TR067 37000 No replication account <server> <account>

TR068 37000 No such account

TR069 37000 Server must have a DBName entry in its ini file for replication

TR070 42000 Replication server not enabled in repl_log_text

TR071 42000 Replication account or trail missing for account name <name> in logging replication.

TR072 08001 Replication connect to <server> failed.

<title>WebDAV Domain</title>

DA001 37000 Includes can be owned only by admin & cannot be writable for others

DA002 37000 Max nesting level (20) reached when processing <path>

DA003 37000 Unterminated include tag at offset <offset> in <path>

DA004 37000 An include tag at offset <inx> with no name or VSP end tag before an include tag in <path>

DA005 22023 Unsupported type of metadata

DA006 22023 Unsupported type of metadata

DA007 22023 Function ISPUBL needs string or array as argument.

DA008 08006 Cannot read from client

DA009 42000 Function dav_user_set_password restricted to dba group

DA010 42000 The user [name] does not exist

DA011 42000 The new password for [name] cannot be empty

<title>SMTP/Mail Functions</title>

SM001 08006 <SMTP server response>

SM002 2E000 Cannot resolve host in smtp_send

SM003 08001 Cannot connect in smtp_send

SM004 22023 Default mail server and/or destination server should be specified

SM005 08006 Misc. error while connecting in smtp_send

SM006 22023 Sender can not be empty

<title>POP3 Functions</title>

PO001 2E000 Cannot resolve host in pop3_get

PO002 08001 Cannot connect in pop3_get

PO003 08006 <POP3 server error response>

PO004 08006 Misc. error in connection in pop3_get

<title>NNTP Server</title>

NN001 22023 Large ID in nntp_id_get

NN002 2E000 Invalid address for News Server at <addr>

NN003 08001 Unable to Contact News Server at <addr>

NN004 08006 <NNTP server error message>

NN005 08006 Misc. error in connection in nntp_get

NN006 22023 the command is not recognized

NN007 08006 Misc. error in connection in nntp_auth_get

NN008 22023 the command is not recognized

NN009 08006 Misc. error in connection in nntp_id_get

NN010 22023 The command is not recognized

NN011 2E000 Invalid address for the News Server at <addr>

NN012 08001 Unable Post to the News Server at <addr>

NN013 08007 Misc. error in connection in nntp_post

NN014 08007 <NNTP server error message>

NN015 2E000 Invalid address for the News Server at <addr>

NN016 08001 Unable Post to the News Server at <addr>

NN017 08007 Misc. error in connection in nntp_auth_post

NN018 08007 <NNTP server error message>

<title>XSLT Processor</title>

XS001 37000 Bad match XPATH expression

XS002 XS370 Only xsl:attribute elements allowed inside xsl:attribute-set name=<name>

XS003 XS370 Only xsl:attribute elements allowed inside xsl:attribute-set name=<name>

XS004 XS370 Only xsl:attribute elements and no value allowed inside xsl:attribute-set name=<name>

XS005 XS370 Max nesting (<num>) of XSLT attribute-sets exceeded

XS006 XS370 xsl:text with disable-output-escaping=on content is not a string

XS007 XS370 Element in set to be sorted must be an XML node

XS008 XS370 XSLT select in apply-templates must return a node set

XS009 XS370 Not an entity in select in apply-templates

XS010 XS370 XSLT template <name> not found.

XS011 XS370 select in for-each must return a node set

XS012 XS370 Element in for-each is not an entity

XS013 XS370 Bad ns prefix in q name <name>

XS014 XS370 Qualified name too long

XS015 XS370 Attribute value for <name> is not a string

XS016 XS370 processing instruction body for <name> is not a string

XS017 XS370 comment body is not a string

XS018 XS370 child element of xsl:choose must be a xsl:when or an xsl:otherwise

XS019 XS370 Unsupported numbering level <level>

XS020 XS370 Bad xsl node <node-name>

XS021 22023 The xslt parameters must be an even length generic array

XS022 22023 XSLT parameters must have strings for even numbered elements

XS023 XS370 Required XSLT attribute <attr-name> missing in <name>

XS024 XS370 Required XSLT attribute <attr-name> missing in <name>

XS025 XS370 top element is not stylesheet

XS026 XS370 top element is not stylesheet

XS027 XS370 no top element is stylesheet

XS028 XS379 cyclic reference in: <name> included from: <name>

XS029 XS370 "yes" or "no" required for "output" tag attribute <attr-name> in <name>

XS030 22023 Bad style sheet in xslt_sheet

XS031 XS379 cyclic reference in: <url1> imported from: <uri2>

XS032 XS379 Standalone required but no SYSTEM or PUBLIC doctype

XS033 22023 An XSLT sheet must be a memory based XML tree

XS034 22023 Undefined style sheet <name>

XS035 XS370 Number format <format> redefined with different attributes

XS036 XS370 Number format <format> not defined

XS037 22023 Unquoted special character in format-number()

XS038 22023 Unquoted special character in format-number()

XS039 22023 Too many percent/permille characters in format-number() pattern

XS040 22023 Too many percent/permille characters in format-number() pattern

XS041 22023 Unexpected '0' in format-number() pattern

XS042 22023 Multiple decimal separators in format-number() pattern

XS043 22023 Malformed format-number() pattern

<title>Cryptographic API</title>

CR001 42000 Cannot allocate temp space. SSL Error : <err>

CR002 42000 Cannot read certificates. SSL Error : <err>

CR003 42000 No CA certificates. SSL Error : <err>

CR004 42000 Cannot read PKCS7 attached signature. SSL Error : <err>

CR005 42000 Cannot allocate output storage. SSL Error : <err>

CR006 42000 No CA certificates

CR007 42000 Error reading the signer certificate. SSL Error : <err>

CR008 42000 Error reading the signer private key. SSL Error : <err>

CR009 42000 Cannot generate PKCS7 signature. SSL Error : <err>

CR010 42000 Cannot allocate output storage. SSL Error : <err>

CR011 42000 Cannot allocate temp space. SSL Error : <err>

CR012 42000 Cannot read certificates. SSL Error : <err>

<title>Backup Functions</title>

IB002 42000 Read of page [PAGE NUM] failed

Internal error of backup process. Possible reason - corrupted database file.

IB001 42000 Could not compress page [PAGE NUM]

Internal error of backup process. Possible reason - corrupted database file.

IB003 42000 Could not store backup context

Possible reason - user has no permissions to perform online backup.

IB004 42000 Could not read backup context

Possible reason - user has no permissions to perform online backup.

IB008 42000 Could not clear online backup context

Possible reason - user has no permissions to perform online backup.

IB008 42000 Could not clear online backup context

Possible reason - user has no permissions to perform online backup.

IB005 42000 Backup file [FILE NAME] writing error

Backup file can not be written. Possible reason - disk is full.

IB006 42000 Number of backup pages is less than 100

Minimal allowed maximal number of pages contained in one backup file is 100. Increase second argument of online_backup.

IB007 42000 Could not create backup file [FILE NAME]

Backup file could not be created in the virtuoso directory. Check file permissions.

IB009 42000 Timeout exceeded

online_backup procedure was aborted due to timeout.

IB010 42000 Timeout cannot be negative

Negative timeout parameter in online_backup procedure is not valid.

<title>User Define Types</title>

UD001 22023 User defined type specified for an element

UD002 42S22 Class <class> is TEMPORARY. It can't be used as a column type

UD003 22023 aset has no effect on values returned by member observers

UD004 22023 Invalid proc_name array supplied

UD005 22023 Non-existent user defined type <type>

UD006 22023 No such method in user defined type <type>

UD007 22023 Method <mtd> in user defined type <type> not defined

UD008 22023 Method <mtd> needs an user defined type instance as argument 1, not an arg of type <type> (<tdp>)

UD009 37000 Field <fld> declared of different language than the class <type>

UD010 37000 Field <fld> declared of different language than the class <type>

UD011 37000 Duplicate external name option

UD012 37000 Duplicate external language option

UD013 37000 Duplicate external language option

UD014 37000 EXTERNAL VARIABLE NAME can be used only with STATIC methods

UD015 37000 Method <mtd> declared of different language than the class <type>

UD016 37000 Class <type> already declared

UD017 37000 Class <type> declared of different language from it's forward declaration

UD018 37000 Class <type> declared of different language from it's superclass <class>

UD019 37000 Can't make a (TEMPORARY | PERSISTENT) subclass <class> of a (TEMPORARY | PERSISTENT) class <class>

UD020 42S01 Type <type> already exists

UD021 42000 No user defined class <type>

UD022 42000 Type <type> is used in one or more compiled queries. Drop them first.

UD023 42000 Duplicate member name <fld> in type <type>

UD024 42000 Duplicate method <method> in type <type>

UD025 22023 Function <function_name> needs an user defined type name as argument <arg_no>

UD026 22023 Invalid instance in user defined type observer

UD027 22023 Invalid instance in user defined type mutator

UD028 22023 Invalid instance in user defined type method call

UD029 22023 <type> is not an instance of <type> in user defined type method call

UD030 22023 <type> instance supplied to a static method call <mtd> of <type>

UD031 22023 No instance supplied to a non-static method call <mtd> of <type>

UD032 42000 Method '<mtd>' of type '<type>' not defined

UD033 22023 Non-valid object instance supplied to member observer/mutator for class <class>

UD034 42000 The object (type <type>) is not an instance of <type>

UD035 42000 invalid instance offset <ofs>

UD036 42000 invalid vtable offset <offset>

UD037 22023 The object supplied is not an instance of <type>

UD038 22023 The object (type <type>) is not an instance of <type>

UD039 42000 invalid vtable offset <ofs>

UD040 37000 User defined type <type> not found in member observer (... AS ...) call

UD041 37000 No user defined type <type>

UD042 37000 No method <mtd> in the user defined type <type>

UD043 37000 Ambiguous method <method> in the user defined type <type>

UD044 37000 Non-existent method <method> in dynamic method call

UD045 37000 No method <method> in the user defined type <type>

UD046 37000 Ambiguous method <method> in the user defined type <type>

UD047 37000 Not an proper constructor call

UD048 37000 No constructor in the user defined type <type>

UD049 37000 Ambiguous constructor call for the user defined type <type>

UD050 37000 No class

UD051 37000 Method definition allowed only for SQL user defined types

UD053 37000 No method of that name declared

UD054 37000 Invalid class for the method definition

UD055 22023 Can't write to an user defined type column

UD056 22023 No field <fld> in the user defined type <type>

UD057 22023 No field <fld> in the user defined type <type>

UD058 22023 No user defined type <type>

UD059 22023 Failed to make instance of the user defined type <type>

UD060 22023 XML attributes not supported with user defined types

UD061 22023 No member <fld> in the user defined type <type>

UD062 42000 Method with bad owner, owner = <owner>

UD063 22023 Type <type> is obsolete.

UD064 37000 Type <type> is DISTINCT. ALTER TYPE ADD ATTRIBUTE for distinct types is not supported

UD065 37000 Field with name <name> is already defined(inherited) for type <type>

UD066 42S22 Field with name <name> defined for type <type>, which is an derived type of <type>

UD067 42S22 The definition of type <type> not found in SYS_USER_TYPES

UD068 37000 Type <type> is DISTINCT. ALTER TYPE DROP ATTRIBUTE for distinct types is not supported

UD069 37000 No field with name <name> for type <type>

UD070 37000 User defined type <type> is not instantiable

UD071 37000 Method <name> already defined in type <type>

UD072 37000 User defined type <type> is not instantiable

UD073 37000 No method <name> found in type <type>

UD074 42S22 No user defined type <type>

UD075 42S22 <type> is declared TEMPORARY. ALTER TYPE not supported for TEMPORARY classes

UD076 42S22 <type> is not instantiable

UD077 42S22 <type> is an external hosted user defined type. ALTER TYPE not supported for non-SQL user defined types.

UD078 42000 ALTER TYPE action not implemented

UD079 42000 Internal error: No user defined type to alter

UD080 42000 User defined type <type> is a super type at least of <type>. Drop it and any other such types first.

UD081 42S22 Field with name <name> defined for type <type>, which is an derived type of <type>

UD082 42S22 The definition of type <name> not found in SYS_USER_TYPES

UD083 37000 Type <name> is DISTINCT. ALTER TYPE DROP ATTRIBUTE for distinct types is not supported

UD084 37000 No field with name <name> for type <type>

UD085 37000 User defined type <name> is not instantiable

UD086 37000 Method <name> already defined in type <name>

UD087 37000 User defined type <name> is not instantiable

UD088 37000 No method <name> found in type <name>

UD089 42S22 No user defined type <name>

UD090 42S22 <name> is declared TEMPORARY. ALTER TYPE not supported for TEMPORARY classes

UD091 42S22 <name> is not instantiable

UD092 42S22 <name> is an external hosted user defined type. ALTER TYPE not supported for non-SQL user defined types.

UD093 42000 ALTER TYPE action not implemented

UD094 42000 Internal error: No user defined type to alter

UD095 42000 user defined type <type> owner specified is different than the creator.

UD096 42000 No permission to use type <type> as a superclass for <type>

UD097 42000 No permission to instantiate user defined type <type>

UD098 42000 No permission to access members of user defined type <type>

UD099 42000 No permission to change members of user defined type <type>

UD100 42000 No permission to call methods of user defined type <type>

UD101 42000 No permission to define methods of class <type>

UD102 42000 Cannot compile the temp method for external procedure

<title>Kerberos Library</title>

KB001 42000 while initializing Kerberos 5 library

KB002 42000 while getting default credential cache (ccache)

KB003 42000 when parsing name <name>

KB004 42000 when unparsing name

KB005 42000 when initializing cache

KB006 42000 wrong password

KB007 42000 while storing credentials

KB008 42000 while initializing Kerberos 5 library

KB009 42000 while getting <name> credential cache (ccache)

KB010 42000 while getting default credential cache (ccache)

KB011 42000 while getting remove credential cache (ccache)

<title>Hosting</title>

HO001 22023 Invalid params type <tag> (<name>)

23.2.2. Data Type Errors

Each function that accepts parameters will check that each parameter is of the correct data type. A function will generate a SQL 22023 error value if a supplied parameter is not of the type expected.

23.3. Signals and Exit codes

This section presents the most important exit codes and signals for Virtuoso on Unix/Linux.

23.3.1. Exit codes

Table 23.2. Exit codes

Exit Code	Category	Comment
-1	FATAL	Unrecoverable condition detected like disk or memory full
0	NORMAL	Normal exit with success
1	WARNING	Some kind of failure at startup/shutdown, typically shown with a reason for this error
>1	UNKNOWN	Not used under normal operations
>128	SIGNAL	Virtuoso was killed by a signal it could not process. (error code = signal + 128)

23.3.2. Signals

Table 23.3. Signals

Signal	Name	Category	Comment
1	SIGHUP	NORMAL	Initiate quick shutdown when shell exits while running virtuoso in foreground mode
2	SIGINT	NORMAL	User pressed CTRL-C on Virtuoso running in foreground mode to initiate normal shutdown
3	SIGQUIT	NORMAL	User pressed CTRL-\ on Virtuoso running in foreground mode to initiate quick shutdown
4	SIGILL	FAULT	Illegal instruction resulting in a program crash
7	SIGBUS	FAULT	Misaligned mem read/write resulting in a program crash
8	SIGFPE	FAULT	Divide by 0 or other arithmetical error
9	SIGKILL	SPECIAL	Terminate Virtuoso without flushing dirty pages etc. This signal should NOT be used under normal operations.
11	SIGSEGV	FAULT	Buffer/stack overrun resulting in a program crash
13	SIGPIPE	IGNORED	Other side of a pipe was killed while this side tried to write data to it
15	SIGTERM	NORMAL	Initiate quick shutdown

Signals that are marked as *NORMAL* indicate that these signals can be sent by root or the owner of the virtuoso process to shutdown Virtuoso. The *SIGINT* and *SIGQUIT* signals also work when Virtuoso is running in background mode and can be used in shell scripts.

Signals that are marked as *FAULT* indicate an abnormal condition which should be checked. If enabled (e.g. with ulimit -c unlimited), the operating system can write a core dump allowing post-mortem debugging if the Virtuoso executable was compiled with debugging symbols.

Signals that are marked as *IGNORED* are handled inside Virtuoso.

The *SIGTERM* signal is mostly used by rc.d type shell scripts on Unix/Linux which are called during shutdown, but can also be used inside shell scripts.

The *SIGKILL* signal should only be used as a last resort when all other efforts to stop Virtuoso have failed.

Quick shutdown

In quick shutdown mode, Virtuoso will only flush dirty disk buffers, close transaction log and database files and exit to the operating system. Since Virtuoso does not perform a checkpoint, the next startup will take extra time as Virtuoso first needs to perform a rollback and a checkpoint before it can resume operation. The *SIGTERM* signal mostly used by rc.d type scripts on Unix/Linux which are called during shutdown as such instances it is better to quickly finish operations and delay the checkpoint until the next boot.

Normal shutdown

In normal shutdown mode, Virtuoso will do first do a full checkpoint, then flush dirty disk buffers, close transaction log and database files before exiting to the operating system. Depending on the size of the transaction log this can take some time. The

SIGINT signal can be used in stop scripts to cleanly shutdown the database by the dba or by root.

Note that in all cases signals can only be sent by the kernel, the root user or the uid of the person who started Virtuoso. Anyone else that wants to shutdown the Virtuoso instance should use the isql program to connect to Virtuoso and run the 'shutdown' command.

23.4. Release Notes

23.4.1. New Features

- *SOAP 1.1 Implementation* - this new release includes a wide range of new SOAP features, bringing it up to date with the most recent developments of this rapidly evolving technology :
  - Flexibility in configuring SOAP access points using a WEB based user interface allowing exposure of selected stored procedures as SOAP services. These procedures may be local or linked from any 3rd party databases. The effective privileges for running SOAP exposed procedures can also be set at the access point level.
  - Improved SOAP to Virtuoso/PL Language Binding allowing the receiving and returning of arbitrarily complex arrays and structures, using XML Schema to describe the mapping of Virtuoso internal types to precisely match specification expressed in XML Schema.
  - Automated generation of WSDL descriptions of SOAP access points - Simple procedure parameter lists map automatically to SOAP types described in XML Schema. For complex types, procedures can be annotated with explicit references to XML Schema types. Thus the developer has complete control over the presentation of web services.
  - Import of WSDL descriptions of outside web services. Virtuoso automatically generates Virtuoso/PL wrappers based on an outside WSDL file, making all its services transparently callable as stored procedures from Virtuoso/PL.
  - Virtuoso's SOAP client and server implementations pass the SOAP Interop tests levels 1, 2 and most of 3.
  - Virtuoso Service Module for XML (VSMX) is a mechanism provided by Virtuoso for testing SOAP messages by automatically generating a test page - a VSMX file - that has the extension .vsmx. The VSMX file is generated at the same time as the WSDL file is generated, giving instant access to your SOAP messages to test they perform as expected. Web Service developers would use VSMX to forego the test bed creation step in light of automatic test page generation. Project managers can easily keep track of their developers progress by checking the test page periodically.
- *XSLT 1.1 Implementation* - support for the XSLT 1.1 Draft Specification has been added.
- *Exposure of Stored Procedures as Web Services* - Stored procedures can be exposed as SOAP services very simply from Virtuoso, whether the procedures are native Virtuoso or on remote data sources. This powerful ability means that any database servers already existing within an organization can easily become a component in an eBusiness solution using Virtuoso.
- *XML Schema* - contains a built-in XML Schema 2001 validator. This is accessible from Virtuoso/PL as a function and is used for SOAP and XQuery where appropriate.
- *XML Templates* - Virtuoso XML templates allow execution of SQL/XML queries over HTTP to obtain an XML document in response and/or perform some operation in the database using updategrams. The XML Template file can be stored either on the file system, WebDAV repository or stored on another HTTP server being referenced by a URL parameter.
- *XQuery* - Virtuoso provides support for the XQuery 1.0 XML Query Language specification. This specification is currently in the working draft stage at the W3C XML Query Working Group working in collaboration with the W3C XSL Working Group.
- *Web DAV* - support for redirection of resources, using the Web DAV Redirect Reference Resource Protocol draft 02 specification.
- *Remote Stored Procedures* - Stored Procedures located in remote databases can now be linked into the Virtuoso Server in the same manner as you would link in tables or views. This enables remote stored procedures to be readily and easily exposed as WEB Services.
- *OLE DB Provider* - native OLE DB Provider is now available allowing direct connection from ADO and .Net applications.
- *JDBC Driver* - enhanced JDBC support with full compliance with JDBC J2EE standard.
- *Free Text* - The full text indexing capability has been extended, giving the developer flexibility in choosing complex, multi-part document id's for application specific sorting of hits, efficient storage of secondary, non-free text data in the free text index for best retrieval performance, options for restarting searches at a specific hit, ascending and descending orders of document id's and more.
- *Free Text Triggers* - The free text triggers system allows persisting free text queries into a table, so that when a free text index is updated, the incoming data gets automatically matched against these persisted criteria, generating hits as the

index is updated. This allows very high performance filtering of fast changing text data, e.g. news feeds without having to run periodic query batches and is ideal for creating personalized information feeds.
- *LDAP* - support for the Lightweight Directory Access Protocol (LDAP) protocol for accessing online directory services.
- *Enhanced Flexibility of Database and VDB security* - New SQL hooks allow the developer to customize password checking at the time of accepting SQL connections into the server. Another hook allows an application to dynamically determine the login/password to use when connecting to a remote database. These hooks make it possible for example to consult an LDAP directory for user information when validating logins.
- *File DSN support* - with the ability to describe a database connection in a simple disk file, Virtuoso becomes more portable, and no longer dependent on per-machine DSNs. If you use file DSNs you can move your database file and your connection information to any computer that has the appropriate drivers installed.
- *Scrollable cursor support in the stored Procedure Language (PL)* - Virtuoso PL now allows you direct access to all ODBC standard features of scrollable cursors through PL routines. ODBC calls are still supported but you have a choice of which level to write your code.

23.4.2. Bugs Fixed

- *Bug 3889* Change label of tab "XML Services/Queues".
- *Bug 3891* Admin UI Home Page Update for 3.0.
- *Bug 3892* Server Startup Information (Branding Info).
- *Bug 3893* When v:data-set @nrows = 1 the 'next' and 'prev' buttons do.
- *Bug 3895* Segmentation fault after running dbpump from Yacutia.
- *Bug 3907* Explain HO-S-4 and HO-S-5 login.
- *Bug 3918* Validators can't work inside Data-Set.
- *Bug 3919* There is no way to format columns data in editing controls.
- *Bug 3920* v:button @style=url or @style=image not working properly.
- *Bug 3921* Logical validators error in Data-Set.
- *Bug 3923* xslt_format_number doesn't work properly with value 0.
- *Bug 3924* "Cancel" failed after Edit mode in Data-Set.
- *Bug 3925* v:column is not more needed in v:data-set.
- *Bug 3926* v:update-field obsoleted.
- *Bug 3931* Initializing JAVA Hosting Tutorials hangs server.
- *Bug 3932* Position Holding in Tutorial Frames.
- *Bug 3936* Virtuoso Unix Installer script.
- *Bug 3943* Data-Set don't work properly inside 'if-login' template.
- *Bug 3950* PDF Version of Documentation is not match HTML.
- *Bug 3956* Tutorial Reorganization.
- *Bug 3969* Install.sh text changes.
- *Bug 3972* Configs, Makefile and installer are not "unpublished propriety.
- *Bug 3973* Virtuoso crashes.
- *Bug 3984* Dynamic Generation of Runtime Tutorial Heading.
- *Bug 3998* bad xpath_eval with //.
- *Bug 3999* Group by clause works wrong.
- *Bug 4000* end_result() has wrong side effect when procedure called from procedure view.
- *Bug 4003* Some catalogs missing in drop down in SOAP expose.
- *Bug 4004* Move Reload button in SOAP expose.
- *Bug 4005* Rename Grant and revoke buttons in SOAP expose.
- *Bug 4006* SQL Error 22023 SR016 in SOAP endpoint hierarchical list.
- *Bug 4013* Install.sh conclusion text needs changing.
- *Bug 4017* Installer is broken - uid and password update.
- *Bug 4021* Lack of pre_render_ method for most of controls.
- *Bug 4025* Service Tray Service Manger About contains a typing mistake.
- *Bug 4026* UDDI; typo in Contact details tab "Address".
- *Bug 4030* DB - TABLE CREATION : Data entered by user is lost.
- *Bug 4034* Absence of the strong KEY column for the UDDI.DESCRIPTION t.
- *Bug 4044* style="url" for "Delete" v:button in data-set caused error.
- *Bug 4049* V:URL unsuitable.
- *Bug 4054* WSDL import uncatched error.
- *Bug 4062* Virgin XP installation of Mono Based Server fails to start.
- *Bug 4063* Installer does not set CLASSPATH to point to Virtuoso JDBC.
- *Bug 4066* Virtuoso Java Based Server fails to start.
- *Bug 4067* Linux installer does not present best default Server for install.

- *Bug 4076* System Tray Icon doe not get removed.
- *Bug 4078* gacutil is missing for RT framework.
- *Bug 4079* VSCM toggles the wrong DB.
- *Bug 4084* Server Crash - ASP.NET hosting.
- *Bug 4090* Retrieve in tables list.
- *Bug 4095* Browse button in column name for v:text inspector does not work.
- *Bug 4096* v:login + page-decor + buttons failed.
- *Bug 4099* Problem with ORDER BY.
- *Bug 4100* Problems with using xpath_contains and xquery_eval in SQL statement.
- *Bug 4102* Kill foreground process misses php java mono servers.
- *Bug 4106* SQL-XML unhandled error.
- *Bug 4109* DB hang when running backup with clear backup context checker.
- *Bug 4110* SQL-XML Error message upon unsuccessful execution of query.
- *Bug 4111* SQL-XML Error attempting to reset the form.
- *Bug 4114* Virtuoso service starts even though no valid license found.
- *Bug 4115* Installer doesn't alert user of non-started instance.
- *Bug 4118* case expression incorrectly works in select statement.
- *Bug 4125* eNews fails to display scrolling items.
- *Bug 4126* Driver does not display SQLSTATE error messages.
- *Bug 4129* Views do not show up via SQLTables call.
- *Bug 4130* entity reference   breaks XSLT processor.
- *Bug 4159* Can't insert date/datetime values as string literals.
- *Bug 4169* Page with list of saved SQL->XML queries missing.
- *Bug 4170* Successful/unsuccessful messages in SQL->XML.
- *Bug 4171* SQL-XML misplaced option.
- *Bug 4173* Missing owner/group params in SQL-XML page.
- *Bug 4174* Missing default root element name in SQL-XML execution.
- *Bug 4175* SQL-XML execution Apply XSLT?.
- *Bug 4177* Button names in Profiling.
- *Bug 4181* Can not "manage" the default "db30" instance.
- *Bug 4188* Open Demo.openlinksw.com in New Window.
- *Bug 4196* Min And Max attributes for v:validator.
- *Bug 4197* v:validator with test="sql".
- *Bug 4207* Visit the online tutorials links are inconsistent.
- *Bug 4217* Declare continue handler for block don't work.
- *Bug 4222* ho-s-7 (DataGrid Demo) problems.
- *Bug 4234* Tutorial and offline documentation creation failure in demo.
- *Bug 4245* .NET client can't work properly with ODBC driver.

23.5. Product Support

23.5.1. OpenLink Discussion Forums

If your browser does not parse the Network News Transport Protocol (NNTP) URL's used in the hyperlinks below, then simply point your news client to one of the following news servers: *news.openlinksw.com* or *news.openlinksw.co.uk* .

OpenLink Virtuoso

Table 23.4. Virtuoso Newsgroups

Newsgroup Description	Newsgroup URL		
OpenLink Virtuoso	*openlink.public.virtuoso*		

OpenLink Data Access (Lite Edition)

Dedicated to Database Connectivity issues that relate to OpenLink Data Access Drivers (PC based) that depend on:

Table 23.5. OpenLink Data Access Newsgroups

Newsgroup Description	Newsgroup URL		

Newsgroup Description	Newsgroup URL
*Oracle SQL*Net* for remote Oracle Database Engine Connectivity.	*openlink.public.lite.oracle*
Sybase Open Client for remote Sybase Database Engine Connectivity.	*openlink.public.lite.sybase*
Progress Client Networking for remote Progress Database Engine Connectivity.	*openlink.public.lite.progress*
Informix I-Net or I-Connect for remote Informix Database Engine Connectivity.	*openlink.public.lite.informix*
Microsoft NETLIB for remote MS SQL Server Database Engine Connectivity	*openlink.public.lite.ms-sqlserver*
IBM DB2 Client Networking for remote DB2 Database Engine Connectivity	*openlink.public.lite.db2*
CA-Ingres Net for remote CA-Ingres & CA-OpenIngres Database Engine Connectivity.	*openlink.public.lite.ingres*
3rd Party ODBC Drivers for remote Database Engine Connectivity.	*openlink.public.lite.odbc-agent*

OpenLink Data Access (Multi Tier Edition)

Dedicated to Database Connectivity issues that relate to OpenLink Data Access Drivers that depend on

Table 23.6. OpenLink Data Access Newsgroups

Newsgroup Description	Newsgroup URL
OpenLink's Database Independent Communications Layer for remote Oracle Database Engine Connectivity.	*openlink.public.multi-tier.oracle*
OpenLink's Database Independent Communications Layer for remote Sybase Database Engine Connectivity.	*openlink.public.multi-tier.sybase*
OpenLink's Database Independent Communications Layer for remote Informix Database Engine Connectivity.	*openlink.public.multi-tier.informix*
OpenLink's Database Independent Communications Layer for remote Progress Database Engine Connectivity.	*openlink.public.multi-tier.progress*
OpenLink's Database Independent Communications Layer for remote CA-Ingres Database Engine Connectivity.	*openlink.public.multi-tier.ingres*
OpenLink's Database Independent Communications Layer for remote MS SQL Server Database Engine Connectivity.	*openlink.public.multi-tier.ms-sqlserver*
OpenLink's Database Independent Communications Layer for remote DB2 Database Engine Connectivity.	*openlink.public.multi-tier.db2*
OpenLink's Database Independent Communications Layer for remote Unify 2000 Database Engine Connectivity.	*openlink.public.multi-tier.unify*
OpenLink's Database Independent Communications Layer for remote PostgreSQL Database Engine Connectivity.	*openlink.public.multi-tier.postgres*
OpenLink's Database Independent Communications Layer for remote Solid Database Engine Connectivity.	*openlink.public.multi-tier.solid*
OpenLink's Database Independent Communications Layer for remote Kubl Database Engine Connectivity.	*openlink.public.multi-tier.kubl*
OpenLink's Database Independent Communications Layer for remote Velocis Database Engine Connectivity.	*openlink.public.multi-tier.velocis*
OpenLink's Database Independent Communications Layer for remote ODBC Data Source Connectivity using 3rd Party ODBC Drivers.	*openlink.public.multi-tier.odbc-agent*

23.6. Virtuoso System Tables

23.6.1. Core System Tables

```
create table SYS_COLS (
        TABLE                   varchar,
        COLUMN                  varchar,
        COL_ID                  integer,
        COL_DTP                 integer,
        primary key (COL_ID))

create index SYS_COLS_BY_NAME on SYS_COLS (TABLE, COLUMN)
```

```
create table SYS_KEYS (
        KEY_TABLE               varchar,
        KEY_NAME                varchar,
        KEY_ID                  integer,
        KEY_N_SIGNIFICANT       integer,
        KEY_CLUSTER_ON_ID       integer,
        KEY_IS_MAIN             integer,
        KEY_IS_OBJECT_ID        integer,
        KEY_IS_UNIQUE           integer,
        KEY_MIGRATE_TO          integer,
        primary key (KEY_TABLE, KEY_NAME))

create index SYS_KEYS_BY_ID on SYS_KEYS (KEY_ID)

create table SYS_KEY_PARTS (
        KP_KEY_ID               integer,
        KP_NTH                  integer,
        KP_COL                  integer
        primary key (KP_KEY_ID, KP_NTH))

create table SYS_KEY_SUBKEY (
        SUPER                   integer,
        SUB                     integer,
        primary key (SUPER, SUB))
```

The SYS_COLS table defines all tables and columns. Each column appears once in this table. No entries are made for inherited columns. The SYS_KEYS table defines all indices. A row in this table corresponds to each key in the database, either originally defined or inherited.

KEY_IS_MAIN is non-zero if this is the primary key of its table. The KEY_MIGRATE_TO is the key ID of a new version of this key if this key is obsolete (e.g. the primary key of a table from before an ALTER TABLE). The KEY_N_SIGNIFICANT indicates how many leading key parts are used in ordering keys.

The KEY_ID references the KP_KEY_ID in the SYS_KEY_PARTS table. This table embodies the actual layout of keys and rows. The KP_NTH is a number positioning the KP_COL to the appropriate place in the row. The KP_COL references the COL_ID in SYS_COLS. The KP_NTH is not necessarily a series of consecutive integers but the order matches the order of columns on the row.

The SYS_KEY_SUBKEY table has a row for each pair of keys where one is the immediate subkey of the other. Note that when a table is altered, the obsolete layout is marked as a subtable of the new layout of the primary key. Thus a select on the new primary key will also get the obsolete rows.

The following SQL procedure exemplifies how the SYS_COLS, SYS_KEYS and SYS_KEY_PARTS tables combine. Given a table name it produces the set of columns and the set of indices and their parts.

```
create procedure dt (in tn varchar)
{
        declare index_name, col, dtp varchar;
        declare t, nth integer;

        dtp := '-';

        result_names (col, t);    --- first result set

        declare cr cursor for select COLUMN, COL_DTP
          from SYS_COLS where TABLE = tn;

        whenever not found goto done;
        open cr;
        while (1=1) {
                fetch cr into col, t;
                result (col, t);
        }
done:
        result_names (index_name, nth, col);     -- 2nd result set
        end_result ();

        declare icr cursor for select KEY_NAME, KP_NTH, COLUMN
```

```
                from SYS_KEYS, SYS_KEY_PARTS, SYS_COLS
                where KEY_TABLE = tn and KP_KEY_ID = KEY_ID
                and KP_NTH < KEY_N_SIGNIFICANT and     COL_ID = KP_COL;

              whenever not found goto k_done;
              open icr;
              while (1=1) {
                      fetch icr into index_name, nth, col;
                      result (index_name, nth, col);
              }
      k_done:
              return 0;
      }
```

23.6.2. System Tables

DB objects related to the User and Role objects are created as follows. The terms group and role are used interchangeably.

```
create table
  SYS_USERS (
    U_ID                integer unique,   -- unique id identifying the security object
    U_NAME              char (128),       -- unique name identifying the security object
    U_IS_ROLE           integer         default 0, -- if true it's a group
    U_FULL_NAME         char (128),     -- for information only, name of that user or group
    U_E_MAIL            char (128)      default '', -- e-mail for contact
    U_PASSWORD          char (128),     -- encrypted password
    U_GROUP             integer,        /* the primary group references SYS_USERS (U_ID), */
    U_LOGIN_TIME        datetime,       -- last login time, can be set in login hooks,
        -- otherwise is set when login in webDAV repository
    U_ACCOUNT_DISABLED integer  default 1, -- if true the account is not functional
    U_DAV_ENABLE        integer         default 0, -- true if DAV login allowed
    U_SQL_ENABLE        integer         default 1, -- true if  SQL/(ODBC/JDBC/OLE/DB etc)  login allowed.
    U_DATA              varchar,        /* reserved */
    U_METHODS           integer,        /* reserved */
    U_DEF_PERMS         char (10)       default '110100000R', -- see PERMISSIONS option value
    U_HOME              varchar (128),  -- see HOME option value
    U_PASSWORD_HOOK     varchar,        -- see PASSWORD_MODE option value
    U_PASSWORD_HOOK_DATA varchar,       -- see PASSWORD_MODE_DATA option value
    U_GET_PASSWORD      varchar,        -- see GET_PASSWORD option data
    U_DEF_QUAL          varchar,        -- default qualifier for SQL/ODBC login
    U_OPTS              long varchar,   -- extensibility options for user&group objects
    primary key (U_NAME)
 )

create table
SYS_ROLE_GRANTS (
   GI_SUPER     integer references SYS_USERS (U_ID), -- user object id
         GI_SUB         integer references SYS_USERS (U_ID), -- granted role object id
         GI_DIRECT      integer default 1,      -- false if indirect, i.e.
         -- true by inheritance, not direct grant.
         GI_GRANT       integer,                            -- who has granted this role
         GI_ADMIN       integer default 0,                   -- true if granted with admin option
         primary key (GI_SUPER, GI_SUB, GI_DIRECT));
```

For Backwards compatibility the system tables SYS_DAV_USER, SYS_DAV_GROUP, SYS_DAV_USER_GROUP and SYS_USER_GROUP are re-defined as views:

```
-- WebDAV Users
create view WS.WS.SYS_DAV_USER (U_ID, U_NAME, U_FULL_NAME, U_E_MAIL, U_PWD,
    U_GROUP, U_LOGIN_TIME, U_ACCOUNT_DISABLED, U_METHODS, U_DEF_PERMS, U_HOME)
  as select U_ID, U_NAME, U_FULL_NAME, U_E_MAIL, U_PASSWORD as U_PWD,
    U_GROUP, U_LOGIN_TIME, U_ACCOUNT_DISABLED, U_METHODS, U_DEF_PERMS, U_HOME
        from DB.DBA.SYS_USERS where U_IS_ROLE = 0 and U_DAV_ENABLE = 1;

-- WebDAV Groups
create view WS.WS.SYS_DAV_GROUP (G_ID, G_NAME)
    as select U_ID as G_ID, U_NAME as G_NAME
        from DB.DBA.SYS_USERS where U_IS_ROLE = 1 and U_DAV_ENABLE = 1;

-- The granted groups to the WebDAV user
create view WS.WS.SYS_DAV_USER_GROUP (UG_UID, UG_GID) as select GI_SUPER, GI_SUB from DB.DBA.SYS_ROLE_GRA
```

```
        where GI_DIRECT = 1;

create view SYS_USER_GROUP (UG_UID, UG_GID) as select GI_SUPER, GI_SUB
   from SYS_ROLE_GRANTS where GI_DIRECT = 1;

         create table SYS_GRANTS  (
                G_USER          varchar,
                G_OP            integer,
                G_OBJECT        varchar,
                G_COL           varchar,
                primary key (G_USER, G_OP, G_OBJECT, G_COL));
```

These tables are visible only to dba members. The procedure list_grants shows a summary of granted privileges:

```
SQL> list_grants (0);
```

These tables should not be modified by applications. Only the SQL statements GRANT, REVOKE, CREATE USER, DELETE USER, SET USER GROUP and SET PASSWORD should be used to maintain user and security information. Security information is cached in RAM during the execution of a Virtuoso process and these statements ensure that the cache stays consistent with the tables.

23.6.3. Row Level Security Table

```
create table SYS_RLS_POLICY (
  RLSP_TABLE varchar, - the FQN of the table
  RLSP_OP varchar, - the operation (one per row : I/U/D/S)
  RLSP_FUNC varchar, - the FQN of the stored procedure to be called
  PRIMARY KEY (RLSP_TABLE, RLSP_OP))
```

That system table is used for persisting the definitions

23.6.4. SYS_CHARSETS

```
CREATE TABLE SYS_CHARSETS (
    CS_NAME varchar,                    -- The "preferred" charset name
    CS_TABLE long nvarchar,             -- the mapping table of length 255 Wide chars
    CS_ALIASES long varchar             -- serialized vector of aliases
);
```

23.6.5. Collations System Table

The SYS_COLLATIONS system table holds the data for all the defined collations. It has the following structure:

```
CREATE TABLE SYS_COLLATIONS (
    COLL_NAME VARCHAR,
    COLL_TABLE LONG VARBINARY,
    COLL_IS_WIDE INTEGER);
```

COLL_NAME is the fully qualified name of the collation (it's identifier)

COLL_TABLE holds the collation table (256 bytes or 65536 wide chars)

COLL_IS_WIDE show the collation's type (0 for CHAR and 1 for NCHAR). Note that a 8bit collation cannot be used by a NCHAR data and vice versa.

Collation can be deleted by deleting its row from SYS_COLLATIONS.

Note

The collation will still be available until the server is restarted, as it's definition is cached into memory.

23.6.6. UDDI Schema

```
---=====================
--- Business Entity Table
---=====================
create table UDDI..BUSINESS_ENTITY (
```

```
        BE_BUSINESS_KEY         varchar,
        BE_AUTHORIZED_NAME      varchar,
        BE_OPERATOR             varchar,
        BE_NAME                 varchar not null,
        BE_CREATED              datetime,
        BE_CHANGED              timestamp,
        BE_OWNER                integer,
 PRIMARY KEY (BE_BUSINESS_KEY)
)
;

---=======================
--- Business service Table
---=======================
create table UDDI..BUSINESS_SERVICE (
        BS_BUSINESS_KEY         varchar,                -- references business entity (optional)
        BS_SERVICE_KEY          varchar not null,       -- Unique Key
        BS_NAME                 varchar not null,       -- Name
        BS_CREATED              datetime,
        BS_CHANGED              timestamp,
        BS_OWNER                integer,
PRIMARY KEY (BS_SERVICE_KEY)
)
;

---=========================
--- Binding Templates Table
---=========================
create table UDDI..BINDING_TEMPLATE (
        BT_BINDING_KEY          varchar,
        BT_SERVICE_KEY          varchar,
        BT_ACCESS_POINT         varchar,
        BT_HOSTING_REDIRECTOR   varchar,
        BT_URL_TYPE             varchar,
        BT_CREATED              datetime,
        BT_CHANGED              timestamp,
        BT_OWNER                integer,
        PRIMARY KEY (BT_BINDING_KEY)
)
;

---======================
--- tModel Table
---======================
create table UDDI..TMODEL (
        TM_TMODEL_KEY           varchar,
        TM_AUTHORIZED_NAME      varchar,
        TM_OPERATOR             varchar,
        TM_NAME                 varchar,
        TM_CREATED              datetime,
        TM_CHANGED              timestamp,
        TM_OWNER                integer,
        primary key (TM_TMODEL_KEY)
)
;

---====================
--- Descriptions Table
---====================
create table UDDI..DESCRIPTION (
        UD_DESC                 varchar,          -- Description Text
        UD_LANG                 varchar,
        UD_PARENT_ID            varchar,          -- Parent ID (references tmodel,
                                    -- businessService etc.)
        UD_TYPE                 varchar           -- ParentType (name of parent table)
)
create index DESC_PARENT on DESCRIPTION (UD_TYPE, UD_PARENT_ID)
;

--=============================================================================================
-- Discovery URL table: contains structure - holds a URL addressable discovery documents
--=============================================================================================
```

```
create table UDDI..DISCOVERY_URL (
        DU_PARENT_ID            varchar not null,       -- parent ID
        DU_PARENT_TYPE          varchar not null,       -- name of parent element
        DU_URL                  varchar,                -- URI
        DU_USE_TYPE             varchar                 -- UseType element
)
create index DISCOVERY_URLS_PARENT on DISCOVERY_URL (DU_PARENT_TYPE, DU_PARENT_ID)
;

--====================
-- Address line table
--====================
create table UDDI..ADDRESS_LINE (
        AL_PARENT_ID            varchar not null,       -- Parent key
        AL_PARENT_TYPE          varchar not null,       -- name of parent element
        AL_USE_TYPE             varchar,                -- UseType element
        AL_SORT_CODE            varchar,                -- SortCode element
        AL_LINE                 varchar                 -- The Line content
)
create index ADDR_LINE_PARENT on ADDRESS_LINE (AL_PARENT_TYPE, AL_PARENT_ID)
;

---=================
--- Contacts Table
---===============
create table UDDI..CONTACTS (
        CO_CONTACT_KEY          varchar,
        CO_BUSINESS_ID          varchar not null,   -- references business entity table by business key
        CO_USE_TYPE             varchar,                -- UseType element
        CO_PERSONAL_NAME        varchar not NULL,       -- name
        PRIMARY KEY (CO_CONTACT_KEY)
)
create index IN_BUSINESS on CONTACTS (CO_BUSINESS_ID)
;

--============
-- email table
--============
create table UDDI..EMAIL (
    EM_CONTACT_KEY     varchar not null,
    EM_ADDR            varchar,
    EM_USE_TYPE        varchar
)
create index IN_EMPARENT on EMAIL (EM_CONTACT_KEY)
;

--============
-- phone table
--============
create table UDDI..PHONE (
    PH_CONTACT_KEY     varchar not null,
    PH_PHONE           varchar,
    PH_USE_TYPE        varchar
)
create index IN_PHPARENT on PHONE (PH_CONTACT_KEY)
;

---======================
--- Identifier Bag Table
---======================
create table UDDI..IDENTIFIER_BAG (
        IB_PARENT_ID            varchar not null,
        IB_PARENT_TYPE          varchar not null,
        IB_TMODEL_KEY_ID        varchar,
        IB_KEY_NAME             varchar,
        IB_KEY_VALUE            varchar
)
create index IB_PARENT on IDENTIFIER_BAG (IB_PARENT_ID,IB_PARENT_TYPE)
;

---====================
--- Category Bag Table
```

```
---====================
create table UDDI..CATEGORY_BAG (
    CB_PARENT_ID                varchar not null,
    CB_PARENT_TYPE              varchar not null,
    CB_TMODEL_KEY_ID            varchar,
    CB_KEY_NAME                 varchar,
    CB_KEY_VALUE                varchar
)
create index CB_PARENT on CATEGORY_BAG (CB_PARENT_ID, CB_PARENT_TYPE)
;

---====================
--- Overview Doc Table
---====================
create table UDDI..OVERVIEW_DOC (
    OV_KEY                              varchar,
    OV_PARENT_ID                        varchar not null,
    OV_PARENT_TYPE                      varchar not null,
    OV_URL                              varchar,
    PRIMARY KEY (OV_KEY)
)
create unique index PARENT_OVERVIEW_DOC on OVERVIEW_DOC (OV_PARENT_ID, OV_PARENT_TYPE)
;

---================================
--- TModel Instance Details Table
---================================
create table UDDI..INSTANCE_DETAIL  (
    ID_KEY              varchar not null,
    ID_BINDING_KEY      varchar,                -- references btemplate(bindingkey)
    ID_TMODEL_KEY       varchar,                -- references tmodel(tmodelkey)
    ID_PARMS            varchar,
    primary key (ID_KEY)
)
create index IN_IDPARENT on INSTANCE_DETAIL (ID_BINDING_KEY, ID_TMODEL_KEY)
;
```

23.6.7. Web Robot System Tables

Target sites table.

```
create table WS.WS.VFS_SITE (
  VS_DESCR   varchar,   -- Human readable description
  VS_HOST    varchar,   -- target hostname (eq. www.foo.com)
  VS_URL     varchar,   -- target path (eq. /cgi-bin/ or /)
  VS_INX     char(5),   -- not used
  VS_OWN     integer,   -- ID of the local WebDAV owner
  VS_ROOT    varchar,   -- target WebDAV collection
  VS_NEWER   datetime,  -- Update the link if newer than
  VS_DEL     char(3),   -- Delete local copy if delete on remote detected flag
  VS_FOLLOW  varchar,   -- follow list (list of masks to allow following the links)
  VS_NFOLLOW varchar,   -- the do not follow list
  VS_SRC     char(3),   -- do get of the images - flag
  VS_OPTIONS varchar,   -- username/password credentials for
                        --  authorization on target site
  VS_METHOD  varchar,   -- use HTTP or HTTP/WebDAV to retrieve the target site
  VS_OTHER   char(10),  -- go to other sites flag
  primary key (VS_HOST, VS_ROOT)
)
;
```

The queues table.

```
create table WS.WS.VFS_QUEUE (
  VQ_HOST    varchar,  -- target host
  VQ_TS      datetime, -- when added to the queue
  VQ_URL     varchar,  -- path on target
  VQ_ROOT    varchar,  -- the WebDAV local collection
  VQ_STAT    char (15),-- status of the entry
  VQ_OTHER   varchar,  -- flag for other site generated link
  primary key (VQ_HOST, VQ_URL, VQ_ROOT)
```

```
)
;
```

The retrieved URLs table.

```
create table WS.WS.VFS_URL (
  VU_HOST     varchar,  -- target host
  VU_URL      varchar,  -- path on target
  VU_ROOT     varchar,  -- the local WebDAV collection containing
                        -- content of the retrieved link
  VU_CHKSUM   varchar,  -- checksum
  VU_ETAG     varchar,  -- Etag from target
  VU_CPTIME   datetime, -- when is copied
  VU_OTHER    varchar,  -- is the link retrieved from foreign site
  primary key (VU_HOST, VU_URL, VU_ROOT)
)
;
```

23.6.8. Web Server & DAV System Tables

All Web Server and DAV related tables and procedures are held in the WS catalogue. This reference section illustrates their structure.

Collections (folders)

```
create table WS.WS.SYS_DAV_COL (
  COL_ID        integer,  -- unique collection id
  COL_NAME      char(256),-- collection name
  COL_OWNER     integer,  -- collection owner id (FK SYS_DAV_USER (U_ID))
  COL_GROUP     integer,  -- collection owner group id (FK SYS_DAV_GROUP (G_ID))
  COL_PARENT    integer,  -- parent collection id (FK SYS_DAV_COL (COL_ID))
  COL_CR_TIME   datetime, -- creation time
  COL_MOD_TIME  datetime, -- modification time
  COL_PERMS     char(9),  -- collection access permissions (like UNIX ugo style)
  primary key (COL_NAME, COL_PARENT)
)
;
```

Groups

Locks

```
create table WS.WS.SYS_DAV_LOCK (
  LOCK_TYPE     char (1),     -- type of the lock (R (read) or W (write))
  LOCK_SCOPE    char (1),     -- lock scope (X (exclusive) or S (shared))
  LOCK_TOKEN    char(256),    -- unique lock token
  LOCK_PARENT_TYPE  char (1), -- lock parent type (R (resource), C (collection))
  LOCK_PARENT_ID  integer,    -- lock parent (resource/collection)
                              -- id (FK SYS_DAV_COL (COL_ID) or SYS_DAV_RES (RES_ID))
  LOCK_TIME     datetime,     -- lock creation time
  LOCK_TIMEOUT  integer,      -- lock time-out
  LOCK_OWNER    integer,      -- lock owner id (FK SYS_DAV_USER (U_ID))
  LOCK_OWNER_INFO varchar,    -- lock owner info (if WebDAV client sent
                              -- additional info eq. e-mail etc.)
  primary key (LOCK_PARENT_TYPE, LOCK_PARENT_ID)
)
;
```

Properties

```
create table WS.WS.SYS_DAV_PROP (
  PROP_ID      integer,   -- unique property id
  PROP_NAME    char(256), -- property name
  PROP_TYPE    char (1),  -- property parent type (R (resource), C (collection))
  PROP_PARENT_ID integer, -- parent resource/collection id (FK SYS_DAV_COL (COL_ID)
                          -- or SYS_DAV_RES (RES_ID))
  PROP_VALUE   varchar,   -- value of the property (plain text or serialized XML entity)
  primary key (PROP_NAME, PROP_TYPE, PROP_PARENT_ID)
)
```

```
;
```

Resources (documents)

```
create table WS.WS.SYS_DAV_RES (
  RES_ID                integer,        -- unique resource id
  RES_NAME      char(256),-- resource name
  RES_OWNER     integer,        -- resource owner id (FK SYS_DAV_USER (U_ID))
  RES_GROUP     integer,        -- resource owner group id (FK SYS_DAV_GROUP (G_ID))
  RES_COL       integer,        -- parent collection id (FK SYS_DAV_COL (COL_ID))
  RES_CONTENT   long varchar IDENTIFIED BY RES_FULL_PATH,
                        -- resource content
  RES_TYPE      varchar,        -- resource MIME type (eq. text/plain etc.)
  RES_CR_TIME   datetime,       -- creation time
  RES_MOD_TIME  datetime,       -- modification time
  RES_PERMS     char (9),       -- resource access permissions (like UNIX ugo style)
  RES_FULL_PATH         varchar,        -- resource full path (eq. /DAV/docs/name.txt)
  primary key (RES_COL, RES_NAME)
)
;
```

Resource Types (MIME types)

```
create table WS.WS.SYS_DAV_RES_TYPES (
  T_EXT         varchar,        -- extension
  T_TYPE                varchar,        -- associated MIME type
  T_DESCRIPTION varchar,        -- optional description
  primary key (T_EXT)
)
;
```

Note:
The Resource Types table is also used by the HTTP server for determining the appropriate 'Content-Type' header for any deliverable, either from WebDAV or the file system.

HTTP Virtual Directory Mappings

```
create table DB.DBA.HTTP_PATH (
  HP_HOST             varchar not null, -- mapping Host in HTTP header note: *ini*
  HP_LISTEN_HOST      varchar not null, -- IP address & port for
                                        -- mapping listening session
  HP_LPATH            varchar not null, -- logical path
  HP_PPATH            varchar not null, -- physical path
  HP_STORE_AS_DAV     integer not null, -- flag for webDAV storage
  HP_DIR_BROWSEABLE   integer not null, -- directory listing allowed
  HP_DEFAULT          varchar,  -- default page
  HP_SECURITY         varchar,  -- which method allowed all/https/digest (NULL/SSL/DIGEST)
  HP_REALM            varchar,  -- authentication realm
  HP_AUTH_FUNC        varchar,  -- which function authenticate this directory
  HP_POSTPROCESS_FUNC varchar,  -- function call after request
  HP_RUN_VSP_AS       varchar,  -- uid for VSPs REFERENCES SYS_USERS (U_NAME)
                                --   ON DELETE SET NULL
  HP_RUN_SOAP_AS      varchar,  -- uid for SOAP REFERENCES SYS_USERS (U_NAME)
                                --   ON DELETE SET NULL
  HP_PERSIST_SES_VARS integer,  -- have a persistent session variables
  HP_SOAP_OPTIONS     varchar,  -- SOAP options
  HP_AUTH_OPTIONS     varchar,  -- options for authentication hook function
  primary key (HP_LISTEN_HOST, HP_HOST, HP_LPATH)
)
;
```

HTTP Access Control List

```
create table HTTP_ACL (
HA_LIST         varchar not null,               -- ACL name
HA_ORDER        integer not null,               -- Position  in the list
HA_OBJECT       integer not NULL default -1,    -- Object ID (applicable to news groups also)
HA_CLIENT_IP    varchar not NULL,               -- *PATTERN*
HA_FLAG         integer not NULL default 1,     -- Allow/Deny flag, 0 - allow, 1 - deny
HA_RW           integer default 0,              -- Read/Write flag, 0 - read,  1 - post
HA_DEST_IP      varchar default '*',            -- Destination IP/Host (applicable to the proxy also)
```

```
PRIMARY KEY (HA_LIST, HA_ORDER, HA_CLIENT_IP, HA_FLAG)
);
```

23.6.9. Mail Table Description

```
- - Message queue
create table MAIL_MESSAGE_QUEUE (
    MQ_ID               varchar,    - - FK MAIL_MESSAGE (MM_ID)
    MQ_RECIEVER         varchar,    - - foreign mail exchanger
    primary key (MQ_ID))
;

create index MAIL_MSG_Q_REC on MAIL_MESSAGE_QUEUE (MQ_RECIEVER)
;

-- Mail Table
create table DB.DBA.MAIL_MESSAGE (
    MM_ID               integer,        -- Unique id of message (per user)
    MM_OWN      varchar (128),  -- Local WebDAV account (recipient, mail box owner) name,
                                        FK references WS.WS.SYS_DAV_USER (U_NAME)
    MM_FLD      varchar (128), -- Message Folder (initial 'Inbox')
    MM_FROM     varchar (512),          -- From: RFC822 header field
    MM_TO               varchar (512),          -- To: RFC822 header field
    MM_CC               varchar (512),          -- Cc: RFC822 header field
    MM_BCC      varchar (512),  -- Bcc: RFC822 header field
    MM_SUBJ     varchar (512),  -- Subject of the message
    MM_REC_DATE varchar (50),   -- Date of arrival
    MM_SND_TIME varchar (50),   -- Date of posting
    MM_IS_READED        integer,                -- Read flag (0/1 not read, 1 - read)
    MM_BODY     long varchar,   -- Message content (including message header)
    primary key (MM_OWN, MM_FLD, MM_ID)
)
;
```

Note:

MM_FROM, MM_TO, MM_CC, MM_BCC, MM_SUBJ, MM_SND_TIME exists only if there exists corresponding RFC822 headers in mail message

```
-- Temporary message MIME parts table (for message composition)
create table DB.DBA.MAIL_PARTS (
    MP_ID       integer,                -- Unique id per user (order of parts)
    MP_PART     long varbinary, -- Message MIME part body
    MP_ATTRS    long varbinary, -- Message MIME part attributes
    MP_OWN      varchar (128),  -- WebDAV user name - FK REFERENCES WS.WS.SYS_DAV_USER (U_NAME)
    primary key (MP_OWN, MP_ID)
)
;

- - Local mail users
(this table will be needed only in case of different DB, DAV and MAIL users)
create table MAIL_USERS (
    MU_NAME     varchar,
    MU_PWD      varchar,
    primary key (MU_NAME))
;

- - Allowed relay domains
(only in case of allowed relaying)
create table MAIL_RELAY (
    MR_ALLOWED  varchar;
    primary key (MR_ALLOWED))
;
```

23.6.10. NNTP Server Tables

The server uses the following tables:

```
create table NEWS_MESSAGES (
  NM_ID         varchar (128) not null, -- Message-ID (unique)
  NM_REF        varchar (128),          -- References
  NM_GROUP      varchar (128) not null, -- Newsgroups ID
```

```
  NM_NUM_GROUP  integer not null,       -- ID unique for group
  NM_READED     integer,                -- How many times this message is read
  NM_OWN        varchar (128),          -- Local sender (if sender is non
                                           local should be null)
  NM_REC_DATE   datetime,               -- Receiving date
  NM_STAT       integer,                -- Post from the local user
  NM_HEAD       long varchar,           -- Message header
  NM_BODY       long varchar,           -- Message content
  primary key (NM_ID))

create table NEWS_GROUPS (
  NG_GROUP      integer identity,  -- Newsgroups ID
  NG_NAME       varchar (128) NOT NULL, -- Local name
  NG_DESC       varchar (128), -- Comment
  NG_SERVER     varchar (128), -- Server
  NG_SERV_PORT  integer,       -- Server port
  NG_OUT_GROUP  varchar (128), -- Out name
  NG_POST       integer,       -- Flag 0/1 posting allowed
  NG_UP_TIME    datetime,      -- Last Update
  NG_CREAT      datetime,      -- When group is created (attached)
  NG_UP_INT     integer,       -- Update interval (min)
  NG_CLEAR_INT  integer,       -- Drop interval for messages
  NG_LIFE_READ  integer,       -- Flag 0/1 read messages exist longer
  NG_STAT       integer,       -- Flag 0/3 Result from last update.
              0 - Unsuccessful
              1 - Successful
              3 - Successful  but does not get all available messages.

  NG_AUTO       integer,       -- Flag 0/1 Auto download.
  NG_PASS       integer,       -- Download messages for one pass.
  NG_UP_MESS    integer,       -- Messages from last update.
  NG_NUM        integer,       -- Estimated number of articles in group
  NG_FIRST      integer,       -- First article number in the group
  NG_LAST       integer,       -- Last article number in the group

  primary key (NG_GROUP))

create table NEWS_ACL (
  NA_GROUP  integer not NULL, -- News Group number, referencing
                                 NEWS_GROUPS (NG_GROUP)
  NA_IP     varchar not NULL, -- Client IP (the mask is %)
  NA_A_D    integer not NULL, -- flag  0 - allow, 1 - deny (action)
  NA_RW     integer not NULL, -- flag  0 - read,  1 - post (reading
                              or posting action is allowed/disallowed)
  PRIMARY KEY (NA_GROUP, NA_IP, NA_A_D, NA_RW));
```

23.6.11. WS Reliable Messaging

Receiver-Side Schema Objects

Incoming messages table used to keep successfully received messages. An application on receiver side can access it in order to process the messages.

```
CREATE TABLE SYS_WSRM_IN_MESSAGE_LOG (
  IML_INDENTIFIER varchar,             -- references SYS_WSRM_IN_SEQUENCES.WIS_IDENTIFIER
  IML_MESSAGE_ID int NOT NULL,         -- unique per sequence
  IML_EXPIRE_DATE datetime,            -- when expires
  IML_RECEIVE_DATE timestamp NOT NULL ,
  IML_MESSAGE long varchar NOT NULL,   -- the message itself
  IML_STATE     int,
  primary key (IML_INDENTIFIER, IML_MESSAGE_ID)
)
;
```

Incoming sequences table; used to keep the successfully established message sequence and policy to be applied to it.

```
CREATE TABLE SYS_WSRM_IN_SEQUENCES
       (
  WIS_IDENTIFIER varchar, -- sequence identifier
  WIS_VERSION varchar,
  WIS_DELIVERY_ASSURANCE varchar,
```

```
  WIS_SEQUENCE_EXPIRATION datetime,
  WIS_INACTIVITY_TIMEOUT integer,
  WIS_RETRANSMISSION_INTERVAL integer,
  WIS_ACKNOWLEDGEMENT_INTERVAL integer,
  primary key (WIS_IDENTIFIER)
)
;
```

Sender's Schema Objects

Outgoing messages log table is used to keep the outgoing messages, to track their state. It also keeps parameter to the soap request in order to re-send if are not already accepted.

```
CREATE TABLE SYS_WSRM_OUT_MESSAGE_LOG (
  OML_INDENTIFIER varchar,          -- references SYS_WSRM_OUT_SEQUENCES.WOS_IDENTIFIER
  OML_MESSAGE_ID int NOT NULL,      -- unique per sequence
  OML_EXPIRE_DATE datetime,         -- when expires
  OML_SEND_DATE timestamp NOT NULL, -- when is sent
  OML_MESSAGE long varchar NOT NULL,-- soap call parameters
  OML_STATE int,
  primary key (OML_INDENTIFIER, OML_MESSAGE_ID)
)
;
```

Outgoing sequences table is used to keep parameters for outgoing message sequences. It's used to persist their state.

```
CREATE TABLE SYS_WSRM_OUT_SEQUENCES (
  WOS_IDENTIFIER varchar,           -- sequence identifier
  WOS_VERSION varchar,
  WOS_DELIVERY_ASSURANCE varchar,
  WOS_SEQUENCE_EXPIRATION datetime,
  WOS_INACTIVITY_TIMEOUT integer,
  WOS_RETRANSMISSION_INTERVAL integer,
  WOS_ACKNOWLEDGEMENT_INTERVAL integer,
  primary key (WOS_IDENTIFIER)
)
;
```

23.6.12. WS Trust

```
create table WST_SERVER_ISSUER_TOKENS
(
  WSK_TOKEN_TYPE        varchar,
  WSK_REQUEST_TYPE      varchar,
  WSK_APPLIES_TO        varchar,
  WSK_FROM              varchar,
  WSK_SERVICE_NAME      varchar,
  WSK_PORT_TYPE         varchar,
  WSK_TOKEN             varchar,
  PRIMARY KEY (WSK_TOKEN_TYPE, WSK_FROM, WSK_TOKEN)
)
;
```

23.6.13. SyncML Schema Objects

SyncML Devices

```
create table SYNC_DEVICES (
  DEV_ID integer identity,
  DEV_USER_ID integer,   -- references WS.WS.SYS_DAV_USER(U_ID)
  DEV_URI varchar,
  DEV_MAN varchar,       -- manufacturer
  DEV_MOD varchar,       -- model
  DEV_OEM varchar,       -- OEM
  DEV_FWV varchar,       -- firmware version
  DEV_SWV varchar,       -- software version
  DEV_HWV varchar,       -- hardware version
  DEV_DEVID varchar,     -- device id ; must be unique
  DEV_DEVTYP varchar,    -- device type
  DEV_UTC integer,       -- requires datetime in UTC
```

```
  DEV_SUPP_LOB integer,  -- supports large objects
  DEV_SUPP_NOC integer,  -- supports "number of changes"
  primary key (DEV_URI)  -- constraint foobar unique(DEV_USER_ID, DEV_URI)
)
;
```

SyncML Maps

```
create table SYNC_MAPS (
  MAP_DEV_ID integer, -- references SYNC_DEVICES(DEV_ID)
  MAP_COL_ID int,     -- references WS.WS.SYS_DAV_COL (COL_ID)
  MAP_LUID varchar,   -- local id (from client's view)
  MAP_GUID varchar,   -- references WS.WS.SYS_DAV_RES (RES_ID)
  primary key (MAP_DEV_ID, MAP_LUID, MAP_GUID)
)
;
```

SyncML Sync Anchors

```
create table SYNC_ANCHORS (
  A_COL_ID integer,   -- references WS.WS.SYS_DAV_COL(COL_ID)
  A_DEV_ID integer,   -- references SYNC_DEVICES(DEV_ID)
  A_LAST_LOCAL datetime,    -- last local anchor
  A_LAST_REMOTE varchar,    -- last remote anchor
  A_NEXT_LOCAL datetime,    -- last local anchor
  A_NEXT_REMOTE varchar,    -- last remote anchor
  primary key (A_COL_ID, A_DEV_ID)
)
;
```

SyncML Sync Log

```
create table SYNC_RPLOG (
  RLOG_RES_ID int,      -- references WS.WS.SYS_DAV_RES (RES_ID)
  RLOG_RES_COL int,     -- references WS.WS.SYS_DAV_COL (COL_ID)
  DMLTYPE varchar,      -- IUD - insert/update/delete
  SNAPTIME datetime,    -- timestamp
  primary key (RLOG_RES_ID)
)
;
```

SyncML Device's Sessions

```
create table SYNC_SESSION
        (
        S_ID varchar,   -- session ID (unique per device)
        S_DEV varchar,  -- device URI
        S_DEV_ID int,   -- references SYNC_DEVICES(DEV_ID)
        S_UID int,      -- user ID SYS_USERS.U_ID
        S_LAST_MSG int, -- last massage no
        S_LAST_CMD int, -- last command no
        S_DATA long varbinary,  -- internal use
        S_TS timestamp,         -- last used
        S_AUTH int default 0,   -- authenticated for this session
        S_NONCE varchar default '', -- nonce value
        S_INIT int default 1,       -- init state flag
        primary key (S_ID, S_DEV)
        )
;
```

23.6.14. INFORMATION_SCHEMA views

The INFORMATION_SCHEMA views are described in the SQL200n standard. These views are defined in every qualifier and describe the objects in that qualifier.

TABLES

```
    INFORMATION_SCHEMA.TABLES
        TABLE_CATALOG                   VARCHAR(128),
```

```
    TABLE_SCHEMA                    VARCHAR(128),
    TABLE_NAME                      VARCHAR(128),
    TABLE_TYPE                      VARCHAR(128),
    V_KEY_TABLE                     VARCHAR,
    V_KEY_NAME                      VARCHAR,
    V_KEY_ID                        INTEGER,
    V_KEY_N_SIGNIFICANT             SMALLINT,
    V_KEY_CLUSTER_ON_ID             SMALLINT,
    V_KEY_IS_MAIN                   SMALLINT,
    V_KEY_IS_OBJECT_ID              SMALLINT,
    V_KEY_IS_UNIQUE                 SMALLINT,
    V_KEY_MIGRATE_TO                INTEGER,
    V_KEY_SUPER_ID                  INTEGER,
    V_KEY_DECL_PARTS                SMALLINT,
    V_KEY_STORAGE                   VARCHAR,
    V_KEY_OPTIONS                   ANY
```

COLUMNS

```
INFORMATION_SCHEMA.COLUMNS
    TABLE_CATALOG                   VARCHAR(128),
    TABLE_SCHEMA                    VARCHAR(128),
    TABLE_NAME                      VARCHAR(128),
    COLUMN_NAME                     VARCHAR(128),
    ORDINAL_POSITION                INTEGER,
    COLUMN_DEF                      VARCHAR,
    NULLABLE                        VARCHAR (3),
    DATA_TYPE                       VARCHAR(128),
    CHARACTER_MAXIMUM_LENGTH        INTEGER,
    CHARACTER_OCTET_LENGTH          INTEGER,
    NUMERIC_PRECISION               SMALLINT,
    NUMERIC_PRECISION_RADIX         SMALLINT,
    NUMERIC_SCALE                   SMALLINT,
    DOMAIN_CATALOG                  VARCHAR(128),
    DOMAIN_SCHEMA                   VARCHAR(128),
    DOMAIN_NAME                     VARCHAR(128),
    IS_IDENTITY                     VARCHAR(3),
    IDENTITY_GENERATION             VARCHAR(10),
    IDENTITY_START                  VARCHAR,
    IDENTITY_INCREMENT              VARCHAR,
    V_TABLE                         VARCHAR,
    V_COLUMN                        VARCHAR,
    V_COL_ID                        INTEGER,
    V_COL_DTP                       SMALLINT,
    V_COL_PREC                      INTEGER,
    V_COL_SCALE                     SMALLINT,
    V_COL_DEFAULT                   VARCHAR,
    V_COL_CHECK                     VARCHAR,
    V_COL_NULLABLE                  SMALLINT,
    V_COL_NTH                       SMALLINT,
    V_COL_OPTIONS                   ANY,
    V_KP_NTH                        SMALLINT
```

SCHEMATA

```
INFORMATION_SCHEMA.SCHEMATA
    CATALOG_NAME                    VARCHAR(128),
    SCHEMA_NAME                     VARCHAR(128),
    SCHEMA_OWNER                    VARCHAR(128)
```

CHECK_CONSTRAINTS

```
INFORMATION_SCHEMA.CHECK_CONSTRAINTS
    CONSTRAINT_CATALOG              VARCHAR(128),
    CONSTRAINT_SCHEMA               VARCHAR(128),
    CONSTRAINT_NAME                 VARCHAR(128),
    CHECK_CLAUSE                    VARCHAR,
    V_C_TABLE                       VARCHAR(128),
    V_C_ID                          INTEGER,
```

```
    V_C_TEXT                        VARCHAR(4000),
    V_C_MODE                        LONG VARCHAR
```

CHECK_CONSTRAINTS

```
INFORMATION_SCHEMA.CHECK_CONSTRAINTS
    CONSTRAINT_CATALOG              VARCHAR(128),
    CONSTRAINT_SCHEMA               VARCHAR(128),
    CONSTRAINT_NAME                 VARCHAR(128),
    CHECK_CLAUSE                    VARCHAR,
    V_C_TABLE                       VARCHAR(128),
    V_C_ID                          INTEGER,
    V_C_TEXT                        VARCHAR(4000),
    V_C_MODE                        LONG VARCHAR
```

COLUMN_DOMAIN_USAGE

```
INFORMATION_SCHEMA.COLUMN_DOMAIN_USAGE
    DOMAIN_CATALOG                  VARCHAR(128),
    DOMAIN_SCHEMA                   VARCHAR(128),
    DOMAIN_NAME                     VARCHAR(128),
    TABLE_CATALOG                   VARCHAR(128),
    TABLE_SCHEMA                    VARCHAR(128),
    TABLE_NAME                      VARCHAR(128),
    COLUMN_NAME                     VARCHAR(128)
```

COLUMN_PRIVILEGES

```
INFORMATION_SCHEMA.COLUMN_PRIVILEGES
    GRANTOR                         VARCHAR(128),
    GRANTEE                         VARCHAR(128),
    TABLE_CATALOG                   VARCHAR(128),
    TABLE_SCHEMA                    VARCHAR(128),
    TABLE_NAME                      VARCHAR(128),
    COLUMN_NAME                     VARCHAR(128),
    PRIVILEGE_TYPE                  VARCHAR(10),
    IS_GRANTABLE                    VARCHAR (3),
    V_G_USER                        INTEGER,
    V_G_OP                          INTEGER,
    V_G_OBJECT                      VARCHAR (386),
    V_G_COL                         VARCHAR (386),
    V_G_GRANTOR                     VARCHAR (128),
    V_G_ADMIN_OPT                   VARCHAR (128)
```

KEY_COLUMN_USAGE

```
INFORMATION_SCHEMA.KEY_COLUMN_USAGE
    CONSTRAINT_CATALOG              VARCHAR(128),
    CONSTRAINT_SCHEMA               VARCHAR(128),
    CONSTRAINT_NAME                 VARCHAR(128),
    TABLE_CATALOG                   VARCHAR(128),
    TABLE_SCHEMA                    VARCHAR(128),
    TABLE_NAME                      VARCHAR(128),
    COLUMN_NAME                     VARCHAR(128),
    ORDINAL_POSITION                SMALLINT,
    V_KEY_IS_MAIN                   SMALLINT,
    V_KEY_IS_UNIQUE                 SMALLINT
```

ROUTINES

```
INFORMATION_SCHEMA.ROUTINES
    SPECIFIC_CATALOG                VARCHAR(128),
    SPECIFIC_SCHEMA                 VARCHAR(128),
    SPECIFIC_NAME                   VARCHAR(128),
    ROUTINE_CATALOG                 VARCHAR(128),
```

```
    ROUTINE_SCHEMA                  VARCHAR(128),
    ROUTINE_NAME                    VARCHAR(128),
    MODULE_CATALOG                  VARCHAR(128),
    MODULE_SCHEMA                   VARCHAR(128),
    MODULE_NAME                     VARCHAR(128),
    UDT_CATALOG                     VARCHAR(128),
    UDT_SCHEMA                      VARCHAR(128),
    UDT_NAME                        VARCHAR(128),
    DATA_TYPE                       VARCHAR(128),
    CHARACTER_MAXIMUM_LENGTH        INTEGER,
    CHARACTER_OCTET_LENGTH          INTEGER,
    COLLATION_CATALOG               VARCHAR(128),
    COLLATION_SCHEMA                VARCHAR(128),
    COLLATION_NAME                  VARCHAR(128),
    CHARACTER_SET_CATALOG           VARCHAR(128),
    CHARACTER_SET_SCHEMA            VARCHAR(128),
    CHARACTER_SET_NAME              VARCHAR(128),
    NUMERIC_PRECISION               SMALLINT,
    NUMERIC_PRECISION_RADIX         SMALLINT,
    NUMERIC_SCALE                   SMALLINT,
    DATETIME_PRECISION              SMALLINT,
    INTERVAL_TYPE                   VARCHAR(128),
    INTERVAL_PRECISION              SMALLINT,
    TYPE_UDT_CATALOG                VARCHAR(128),
    TYPE_UDT_SCHEMA                 VARCHAR(128),
    TYPE_UDT_NAME                   VARCHAR(128),
    SCOPE_CATALOG                   VARCHAR(128),
    SCOPE_SCHEMA                    VARCHAR(128),
    SCOPE_NAME                      VARCHAR(128),
    MAXIMUM_CARDINALITY             INTEGER,
    DTD_IDENTIFIER                  VARCHAR(128),
    ROUTINE_BODY                    VARCHAR(30),
    ROUTINE_DEFINTION               VARCHAR,
    EXTERNAL_NAME                   VARCHAR(128),
    EXTERNAL_LANGUAGE               VARCHAR(30),
    PARAMETER_STYLE                 VARCHAR(30),
    IS_DETERMINISTIC                VARCHAR(10),
    SQL_DATA_ACCESS                 VARCHAR(30),
    IS_NULL_CALL                    VARCHAR(10),
    SQL_PATH                        VARCHAR(128),
    SCHEMA_LEVEL_ROUTINE            VARCHAR(10),
    MAX_DYNAMIC_RESULT_SETS         SMALLINT,
    IS_USER_DEFINED_CAST            VARCHAR(10),
    IS_IMPLICITLY_INVOCABLE         VARCHAR(10),
    CREATED                         DATETIME,
    LAST_ALTERED                    DATETIME
```

PARAMETERS

```
INFORMATION_SCHEMA.PARAMETERS
    SPECIFIC_CATALOG                VARCHAR(128),
    SPECIFIC_SCHEMA                 VARCHAR(128),
    SPECIFIC_NAME                   VARCHAR(128),
    ORDINAL_POSITION                INTEGER,
    PARAMETER_MODE                  VARCHAR(10),
    IS_RESULT                       VARCHAR(10),
    AS_LOCATOR                      VARCHAR(10),
    PARAMETER_NAME                  VARCHAR(128),
    DATA_TYPE                       VARCHAR(128),
    CHARACTER_MAXIMUM_LENGTH        INTEGER,
    CHARACTER_OCTET_LENGTH          INTEGER,
    COLLATION_CATALOG               VARCHAR(128),
    COLLATION_SCHEMA                VARCHAR(128),
    COLLATION_NAME                  VARCHAR(128),
    CHARACTER_SET_CATALOG           VARCHAR(128),
    CHARACTER_SET_SCHEMA            VARCHAR(128),
    CHARACTER_SET_NAME              VARCHAR(128),
    NUMERIC_PRECISION               SMALLINT,
    NUMERIC_PRECISION_RADIX         SMALLINT,
    NUMERIC_SCALE                   SMALLINT,
    DATETIME_PRECISION              SMALLINT,
    INTERVAL_TYPE                   VARCHAR(128),
```

```
    INTERVAL_PRECISION              SMALLINT,
    USER_DEFINED_TYPE_CATALOG       VARCHAR(128),
    USER_DEFINED_TYPE_SCHEMA        VARCHAR(128),
    USER_DEFINED_TYPE_NAME          VARCHAR(128),
    SCOPE_CATALOG                   VARCHAR(128),
    SCOPE_SCHEMA                    VARCHAR(128),
    SCOPE_NAME                      VARCHAR(128)
```

REFERENTIAL_CONSTRAINTS

```
INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS
    CONSTRAINT_CATALOG              VARCHAR(128),
    CONSTRAINT_SCHEMA               VARCHAR(128),
    CONSTRAINT_NAME                 VARCHAR(128),
    UNIQUE_CONSTRAINT_CATALOG       VARCHAR(128),
    UNIQUE_CONSTRAINT_SCHEMA        VARCHAR(128),
    UNIQUE_CONSTRAINT_NAME          VARCHAR(128),
    MATCH_OPTION                    VARCHAR(7),
    UPDATE_RULE                     VARCHAR(9),
    DELETE_RULE                     VARCHAR(9),
    V_FK_TABLE                      VARCHAR(128)
```

TABLE_CONSTRAINTS

```
INFORMATION_SCHEMA.TABLE_CONSTRAINTS
    CONSTRAINT_CATALOG              VARCHAR(128),
    CONSTRAINT_SCHEMA               VARCHAR(128),
    CONSTRAINT_NAME                 VARCHAR(128),
    TABLE_CATALOG                   VARCHAR(128),
    TABLE_SCHEMA                    VARCHAR(128),
    TABLE_NAME                      VARCHAR(128),
    CONSTRAINT_TYPE                 VARCHAR(11),
    IS_DEFERRABLE                   VARCHAR(2),
    INITIALLY_DEFERRED              VARCHAR(2)
```

TABLE_PRIVILEGES

```
INFORMATION_SCHEMA.TABLE_PRIVILEGES
    GRANTOR                         VARCHAR(128),
    GRANTEE                         VARCHAR(128),
    TABLE_CATALOG                   VARCHAR(128),
    TABLE_SCHEMA                    VARCHAR(128),
    TABLE_NAME                      VARCHAR(128),
    PRIVILEGE_TYPE                  VARCHAR(10),
    IS_GRANTABLE                    VARCHAR (3),
    V_G_USER                        INTEGER,
    V_G_OP                          INTEGER,
    V_G_OBJECT                      VARCHAR (386),
    V_G_GRANTOR                     VARCHAR (386)
```

VIEWS

```
INFORMATION_SCHEMA.VIEWS
    TABLE_CATALOG                   VARCHAR(128),
    TABLE_SCHEMA                    VARCHAR(128),
    TABLE_NAME                      VARCHAR(128),
    VIEW_DEFINITION                 VARCHAR,
    CHECK_OPTION                    VARCHAR(7),
    IS_UPDATABLE                    VARCHAR(3)
```

23.7. Basic Syntax of Regular Expressions

The two special symbols: `'^'` and `'$'` indicate the *start* and the *end* of a string respectively, like so:

`^The` : matches any string that starts with The;

`of despair$` : matches a string that ends in the substring of despair;

`^abc$` : a string that starts and ends with abc -- that could only be abc itself!

`notice` : a string that has the text notice in it.

Without either of the above special character you are allowing the pattern to occur anywhere inside the string.

The symbols `'*'` , `'+'` , and `'?'` denote the number of times a character or a sequence of characters may occur. What they mean is: zero or more, one or more, and zero or one. Here are some examples:

`ab*` : matches a string that has an *a* followed by zero or more *b* 's (a, ab, abbb, etc.);

`ab+` : same, but there is at least one *b* (ab, abbb, etc.);

`ab?` : there might be a *b* or not;

`a?b+$` : a possible *a* followed by one or more *b* 's ending a string.

You can also use *bounds* , which come inside braces and indicate ranges in the number of occurrences:

`ab{2}` : matches a string that has an *a* followed by exactly two *b* 's (abb);

`ab{2,}` : there are at least two *b* 's (abb, abbbb, etc.);

`ab{3,5}` : from three to five *b* 's (abbb, abbbb, or abbbbb).

Note, that you must always specify the first number of a range (i.e, `{0,2}` , not `{,2}`). Also, as you may have noticed, the symbols '*', '+', and '?' have the same effect as using the bounds `{0,}` , `{1,}` , and `{0,1}` , respectively.

Now, to quantify a sequence of characters, put them inside parentheses:

`a(bc)*` : matches a string that has an *a* followed by zero or more copies of the sequence bc;

`a(bc){1,5}` : one through five copies of bc.

There's also the '|' symbol, which works as an OR operator:

`hi|hello` : matches a string that has either hi or hello in it;

`(b|cd)ef` : a string that has either bef or cdef;

`(a|b)*c` : a string that has a sequence of alternating *a* 's and *b* 's ending in a *c* ;

A period ('.') stands for any single character:

`a.[0-9]` : matches a string that has an *a* followed by one character and a digit;

`^.{3}$` : a string with exactly 3 characters.

Bracket expressions specify which characters are allowed in a single position of a string:

`[ab]` : matches a string that has either an *a* or a *b* (that's the same as `a|b`);

`[a-d]` : a string that has lowercase letters 'a' through 'd' (that's equal to `a|b|c|d` and even `[abcd]`);

`^[a-zA-Z]` : a string that starts with a letter;

`[0-9]%` : a string that has a single digit before a percent sign;

`,[a-zA-Z0-9]$` : a string that ends in a comma followed by an alphanumeric character.

You can also list the characters that do NOT want -- just use a '^' as the first symbol in a bracketed expression (i.e., `%[^a-zA-Z]%` matches a string with a character that is not a letter between two percent signs).

Do not forget that bracket expressions are an exception to that rule--inside them, all special characters, including the backslash ('\'), lose their special powers (i.e., `[*\+?{}.]` matches exactly any of the characters inside the brackets). To include a literal ']' in the list, make it the first character (following a possible '^'). To include a literal '-', make it the first or last character, or the second endpoint of a range.

23.8. Server & client versions compatibility

The RPC protocol has changed between 2.7 and 3.0 versions. Thus, not all clients can connect to all servers. For best results, the version should be identical on both ends of ODBC and JDBC connections. However, the following table shows which combinations should generally be successful.

Table 23.7.

Driver/Server	4.0	3.5	3.2	3.1	3.0	2.7	2.5	2.1
4.0	Y	Y	Y	Y	Y	N	N	N
3.5	Y	Y	Y	Y	Y	N	N	N
3.2	Y	Y	Y	Y	Y	N	N	N
3.1	Y	Y	Y	Y	Y	N	N	N
3.0	Y	Y	Y	Y	Y	N	N	N
2.7	N	N	N	N	N	Y	Y	Y
2.5	N	N	N	N	N	Y	Y	Y
2.1	N	N	N	N	N	Y	Y	Y

Chapter 24. Virtuoso Functions Guide & Reference

Abstract

The Virtuoso SQL Functions Guide contains reference manual entries for built-in SQL functions and procedures.

The Virtuoso XPATH & XQUERY Functions Guide contains reference manual entries of the built-in functions of the Virtuoso XPATH Processor. These can be used in XPATH, XSLT, XQuery.

Administration

- cfg_item_count ()
- cfg_item_name ()
- cfg_item_value ()
- cfg_section_count ()
- cfg_section_name ()
- cfg_write ()
- checkpoint_interval ()
- DB.DBA.VACUUM ()
- disconnect_user ()
- fk_check_input_values ()
- log_enable ()
- log_text ()
- read_log ()
- __dbf_set ()
- stat_import ()
- stat_export ()
- prof_enable ()
- prof_sample ()
- status ()
- key_estimate ()
- system ()
- uptime ()
- sys_lockdown ()
- sys_stat ()
- trace_status ()

- USER_CHANGE_PASSWORD ()
- USER_CREATE ()
- USER_DROP ()
- USER_GET_OPTION ()
- USER_GRANT_ROLE ()
- USER_REVOKE_ROLE ()
- USER_ROLE_CREATE ()
- USER_ROLE_DROP ()
- USER_SET_QUALIFIER ()
- USER_SET_OPTION ()
- user_set_password ()
- USER_KEY_STORE ()
- virtuoso_ini_path ()
- server_root ()

Aggregate Functions

- VAR ()
- VAR_SAMP ()
- VAR_POP ()
- STDDEV ()
- STDDEV_SAMP ()
- STDDEV_POP ()
- REGR_SYY ()
- REGR_SXX ()
- REGR_SXY ()
- REGR_AVGX ()
- REGR_AVGY ()
- REGR_R2 ()
- REGR_COUNT ()
- REGR_INTERCEPT ()
- REGR_SLOPE ()
- COVAR_SAMP ()

- COVAR_POP ()
- CORR ()
- DB.DBA.XQ_SEQUENCE_AGG ()
- DB.DBA.VECTOR_AGG ()
- DB.DBA.VECTOR_OF_NONNULLS_AGG ()
- DB.DBA.VECTOR_OR_NULL_AGG ()
- DB.DBA.VECTOR_OF_NONNULLS_OR_NULL_AGG ()
- DB.DBA.VECTOR_CONCAT_AGG ()
- DB.DBA.BAG_AGG ()
- DB.DBA.BAG_OF_NONNULLS_AGG ()
- DB.DBA.BAG_OR_NULL_AGG ()
- DB.DBA.BAG_OF_NONNULLS_OR_NULL_AGG ()
- DB.DBA.BAG_CONCAT_AGG ()

AMI APIs

- DB.DBA.AMAZON_START_INSTANCE ()
- DB.DBA.AMAZON_RUN_INSTANCE ()
- DB.DBA.AMAZON_STOP_INSTANCE ()
- DB.DBA.AMAZON_TERMINATE_INSTANCE ()
- DB.DBA.AMAZON_CREATE_VOLUME ()
- DB.DBA.AMAZON_DEREGISTER_IMAGE ()
- DB.DBA.AMAZON_CREATE_IMAGE ()
- DB.DBA.AMAZON_CREATE_SNAPSHOT ()
- DB.DBA.AMAZON_DELETE_SNAPSHOT ()
- DB.DBA.AMAZON_DELETE_VOLUME ()
- DB.DBA.AMAZON_DESCRIBE_IMAGES ()
- DB.DBA.AMAZON_DESCRIBE_INSTANCES ()
- DB.DBA.AMAZON_IMPORT_KEY_PAIR ()

Array Manipulation

- aref ()
- aset ()

- concat ()
- concatenate ()
- dvector ()
- get_keyword ()
- get_keyword_ucase ()
- gvector_sort ()
- gvector_digit_sort ()
- isarray ()
- make_array ()
- position ()
- rowvector_digit_sort ()
- vector ()
- vector_concat ()

Backup

- backup ()
- backup_online ()
- backup_context_clear ()
- replay ()

BPEL APIs

- BPEL.BPEL.compile_script ()
- BPEL.BPEL.copy_script ()
- BPEL.BPEL.get_partner_links ()
- BPEL.BPEL.instance_delete ()
- BPEL.BPEL.purge_instance ()
- BPEL.BPEL.script_delete ()
- BPEL.BPEL.script_obsolete ()
- BPEL.BPEL.script_source_update ()
- BPEL.BPEL.script_upload ()
- BPEL.BPEL.wsdl_upload ()
- BPEL.BPEL.getVariableData ()

- BPEL.BPEL.setVariableData ()
- BPEL.BPEL.plink_get_option ()
- BPEL.BPEL.plink_set_option ()
- BPEL.BPEL.import_script ()

Virtuoso Server Extension Interface (VSEI)

- Virtuoso Server Extension Interface (VSEI) functions ()

Compression

- gz_compress ()
- gz_uncompress ()
- string_output_gz_compress ()

Cursor

- bookmark ()

Date & Time Manipulation

- curdate ()
- forget_timezone ()
- is_timezoneless ()
- adjust_timezone ()
- rdf_now_impl ()
- current_timestamp ()
- curdatetime ()
- curdatetimeoffset ()
- curutcdatetime ()
- sysutcdatetime ()
- dateadd ()
- datediff ()
- datestring , datestring_gmt , ()
- datestring_GMT ()
- dayname ()
- dt_set_tz ()
- get_timestamp ()

- getdate ()
- msec_time ()
- now ()
- stringdate ()
- stringtime ()

Debug

- cov_load ()
- cov_report ()
- cov_store ()
- dbg_obj_print ()
- dbg_printf ()
- explain ()
- log_message ()
- pldbg_stats ()
- pldbg_stats_load ()
- trace_off ()
- trace_on ()

Dictionary Manipulation

- dict_dec_or_remove ()
- dict_duplicate ()
- dict_get ()
- dict_inc_or_put ()
- dict_iter_next ()
- dict_iter_rewind ()
- dict_list_keys ()
- dict_destructive_list_rnd_keys ()
- dict_new ()
- dict_put ()
- dict_remove ()
- dict_size ()

- dict_to_vector ()
- dict_zap ()

Encoding & Decoding

- encode_base64 ()
- split_and_decode ()
- uudecode ()
- uuencode ()

File Manipulation

- file_delete ()
- file_open ()
- ft_set_file ()
- file_dirlist ()
- file_mkdir ()
- file_mkpath ()
- file_stat ()
- file_to_string ()
- file_to_string_output ()
- file_unlink ()
- gz_file_open ()
- get_csv_row ()
- csv_load_file ()
- attach_from_csv ()
- csv_load ()
- csv_parse ()
- csv_cols_def ()
- csv_table_def ()
- os_chmod ()
- os_chown ()

Free Text

- composite ()

- composite_ref ()
- contains ()
- vt_batch ()
- vt_batch_d_id ()
- vt_batch_feed ()
- vt_batch_feed_offband ()
- VT_BATCH_UPDATE ()
- vt_create_text_index ()
- VT_DROP_FTT ()
- vt_is_noise ()

Hashing / Cryptographic

- md5 ()
- md5_init ()
- md5_update ()
- md5_final ()
- sha1_digest ()
- tree_md5 ()
- dsig_template_ext ()
- x509_certificate_verify ()
- xenc_X509_certificate_serialize ()
- xenc_decrypt_soap ()
- xenc_delete_temp_keys ()
- xenc_encrypt ()
- encrypt ()
- decrypt ()
- TOTP_generate ()
- xenc_get_key_algo ()
- xenc_get_key_identifier ()
- xenc_key_3DES_read ()
- xenc_key_3DES_create ()
- xenc_key_3DES_rand_create ()

- xenc_key_DSA_read ()
- xenc_key_RSA_read ()
- xenc_key_AES_create ()
- xenc_key_AES_rand_create ()
- xenc_key_create_cert ()
- xenc_key_DSA_create ()
- xenc_key_exists ()
- xenc_key_inst_create ()
- xenc_key_remove ()
- xenc_key_serialize ()
- xenc_set_primary_key ()
- xenc_x509_ss_generate ()
- xenc_x509_generate ()
- xenc_pkcs12_export ()
- xenc_pem_export ()
- xenc_SPKI_read ()
- xenc_bn2dec ()
- xenc_key_RSA_create ()
- xenc_x509_csr_generate ()
- xenc_x509_from_csr ()

Virtuoso Java PL API

- java_call_method ()
- java_set_property ()
- java_get_property ()
- java_load_class ()
- java_new_object ()
- java_vm_attach ()
- java_vm_detach ()

LDAP

- ldap_search ()

- ldap_delete ()
- ldap_add ()
- ldap_modify ()

Locale

- charset_define ()
- charset_recode ()
- charsets_list ()
- current_charset ()
- elh_get_handler ()
- elh_load_handler ()
- lh_get_handler ()
- lh_load_handler ()

Mail

- mime_body ()
- mime_part ()
- mime_tree ()
- nntp_auth_get ()
- nntp_auth_post ()
- nntp_get ()
- nntp_post ()
- pem_certificates_to_array ()
- pop3_get ()
- imap_get ()
- smime_sign ()
- smime_verify ()
- smime_encrypt ()
- smime_decrypt ()
- smtp_send ()
- uuvalidate ()

Number

- abs ()
- atof ()
- atoi ()
- bit_and ()
- bit_or ()
- bit_not ()
- bit_shift ()
- bit_xor ()
- ceiling ()
- either ()
- equ ()
- exp ()
- floor ()
- isdouble ()
- __min ()
- __max ()
- __max_notnull ()
- __min_notnull ()
- log ()
- log10 ()
- lower ()
- mod ()
- power ()
- randomize ()
- rnd ()
- sqrt ()
- trigonometric ()

Phrases

- DB.DBA.ANN_PHRASE_CLASS_ADD ()

- DB.DBA.ANN_PHRASE_CLASS_DEL ()
- AP_BUILD_MATCH_LIST ()
- AP_ADD_PHRASES ()

Replication

- repl_disconnect ()
- REPL_GRANT ()
- REPL_INIT_COPY ()
- repl_new_log ()
- REPL_PUBLISH ()
- REPL_PUB_ADD ()
- REPL_PUB_INIT_IMAGE ()
- REPL_PUB_REMOVE ()
- REPL_REVOKE ()
- REPL_SCHED_INIT ()
- REPL_SERVER ()
- repl_server_rename ()
- REPL_STAT ()
- repl_status ()
- REPL_SUBSCRIBE ()
- repl_sync ()
- repl_sync_all ()
- repl_text ()
- repl_this_server ()
- REPL_UNPUBLISH ()
- REPL_UNSUBSCRIBE ()
- repl_purge ()
- REPL_CREATE_SNAPSHOT_PUB ()
- REPL_CREATE_SNAPSHOT_SUB ()
- REPL_DROP_SNAPSHOT_SUB ()
- REPL_DROP_SNAPSHOT_PUB ()
- REPL_INIT_SNAPSHOT ()

- REPL_UPDATE_SNAPSHOT ()
- REPL_SNP_SERVER ()
- REPL_SERVER_NAME ()
- REPL_ADD_CR ()
- REPL_ADD_DAV_CR ()
- REPL_ADD_SNAPSHOT_CR ()
- RDF_REPL_START ()
- RDF_REPL_STOP ()
- RDF_REPL_SYNC ()
- RDF_REPL_GRAPH_INS ()
- RDF_REPL_GRAPH_DEL ()
- sub_schedule ()

Remote SQL Data Source

- att_local_name ()
- quote_dotted ()
- rclose ()
- rexecute ()
- rstmtexec ()
- rmoreresults ()
- rnext ()
- sql_columns ()
- sql_data_sources ()
- sql_gettypeinfo ()
- sql_primary_keys ()
- sql_statistics ()
- sql_tables ()
- sql_special_columns ()
- sql_procedures ()
- sql_write_private_profile_string ()
- sql_get_private_profile_string ()
- sql_config_data_sources ()

- sql_get_installed_drivers ()
- sql_remove_dsn_from_ini ()
- sql_transact ()
- sql_write_file_dsn ()
- sql_driver_connect ()
- vd_remote_data_source ()
- vd_remote_proc_wrapper ()
- vd_remote_table ()
- vd_statistics ()
- vdd_disconnect_data_source ()
- vdd_measure_rpc_time ()

RDF data

- sparql_to_sql_text ()
- DB.DBA.RDF_AUDIT_METADATA ()
- DB.DBA.RDF_GRAPH_SECURITY_AUDIT ()
- DB.DBA.RDF_DEFAULT_USER_PERMS_SET ()
- DB.DBA.RDF_DEFAULT_USER_PERMS_DEL ()
- DB.DBA.RDF_BACKUP_METADATA ()
- DB.DBA.RDF_VOID_STORE ()
- rdf_load_stop ()
- rdf_loader_run ()
- ld_dir_all ()
- DB.DBA.RDF_RESTORE_METADATA ()
- DB.DBA.RDF_LOAD_RDFXML_MT ()
- DB.DBA.RDF_LOAD_RDFXML ()
- DB.DBA.RDF_LOAD_RDFA ()
- ld_dir ()
- DB.DBA.TTLP ()
- DB.DBA.TTLP_MT ()
- DB.DBA.TTLP_MT_LOCAL_FILE ()
- DB.DBA.RDF_DATATYPE_OF_OBJ ()

- DB.DBA.RDF_TRIPLES_TO_RDF_XML_TEXT ()
- DB.DBA.RDF_TRIPLES_TO_TTL ()
- DB.DBA.RDF_64BIT_UPGRADE ()
- RDF_VIEW_SYNC_TO_PHYSICAL ()
- DB.DBA.RDF_CONVERT_RDFXML_TO_TTL ()
- DB.DBA.RDF_GRAPH_GROUP_CREATE ()
- DB.DBA.RDF_GRAPH_GROUP_INS ()
- DB.DBA.RDF_GRAPH_USER_PERMS_SET ()
- DB.DBA.RDF_GRAPH_USER_PERMS_GET ()
- DB.DBA.RDF_GRAPH_USER_PERMS_DEL ()
- DB.DBA.RDF_ALL_USER_PERMS_DEL ()
- rdfs_rule_set ()
- DB.DBA.RDF_GEO_FILL ()
- DB.DBA.RDF_GEO_ADD ()
- DB.DBA.RDF_GRAPH_COLLECT_FP_LIST ()
- DB.DBA.RDF_GRAPH_DIFF ()
- DB.DBA.RDF_SUO_APPLY_PATCH ()
- DB.DBA.RDF_SUO_DIFF_TTL ()
- DB.DBA.SPARQL_RDB2RDF_CODEGEN ()
- DB.DBA.SPARQL_RDB2RDF_LIST_TABLES ()
- DB.DBA.SPARQL_SELECT_KNOWN_GRAPHS ()
- IsRef ()
- DB.DBA.SAMPLE ()
- DB.DBA.GROUP_CONCAT ()
- DB.DBA.GROUP_CONCAT_DISTINCT ()
- DB.DBA.GROUP_DIGEST ()
- http_nt_triple ()
- http_ttl_triple ()
- iri_split ()
- __xml_get_ns_prefix ()
- __xml_get_ns_uri ()

- __xml_ns_uname ()
- __xml_ns_iristr ()
- __xml_nsexpand_iristr ()
- DB.DBA.SPARQL_EVAL ()
- DB.DBA.SPARQL_EVAL_TO_ARRAY ()
- DB.DBA.SPARQL_REXEC ()
- DB.DBA.SPARQL_REXEC_TO_ARRAY ()
- DB.DBA.SPARQL_REXEC_WITH_META ()
- DB.DBA.RDF_REGEX ()
- DB.DBA.RDF_LANGMATCHES ()
- DB.DBA.RDF_TTL2HASH ()
- DB.DBA.RDF_QUAD_URI ()
- DB.DBA.RDF_QUAD_URI_L ()
- DB.DBA.RDF_QUAD_URI_L_TYPED ()

SOAP

- http_body_read ()
- soap_box_xml_entity ()
- soap_dt_define ()
- soap_call ()
- soap_client ()
- soap_make_error ()
- soap_print_box ()
- soap_sdl ()
- soap_server ()
- soap_wsdl ()
- soap_wsdl_import ()
- soap_box_structure ()
- soap_current_url ()
- wst_cli ()

SQL

- __any_grants ()
- collation_define ()
- complete_table_name ()
- delay ()
- cl_exec ()
- end_result ()
- exec ()
- close ()
- exec_next ()
- exec_result ()
- exec_result_names ()
- exec_metadata ()
- exec_score ()
- identity_value ()
- name_part ()
- registry_get ()
- registry_get_all ()
- registry_name_is_protected ()
- registry_set ()
- registry_remove ()
- result ()
- result_names ()
- row_count ()
- sequence_get_all ()
- sequence_next ()
- sequence_remove ()
- sequence_set ()
- set_row_count ()
- set_user_id ()
- set_identity_column ()

- signal ()
- sinv_create_key_mapping ()
- sinv_create_inverse ()
- sinv_drop_inverse ()
- sys_stat_analyze ()
- sys_stat_histogram ()
- table_set_policy ()
- table_drop_policy ()
- username ()

String

- ascii ()
- blob_to_string ()
- blob_to_string_output ()
- chr ()
- initcap ()
- isblob ()
- lcase ()
- left ()
- length ()
- locate ()
- ltrim ()
- make_string ()
- regexp_match ()
- regexp_parse ()
- regexp_substr ()
- regexp_replace ()
- regexp_instr ()
- regexp_like ()
- repeat ()
- replace ()
- right ()

- rtrim ()
- search_excerpt ()
- serialize ()
- space ()
- sprintf ()
- sprintf_inverse ()
- sprintf_iri ()
- sprintf_iri_or_null ()
- sprintf_or_null ()
- strcasestr ()
- strchr ()
- string_output ()
- string_output_flush ()
- string_output_string ()
- string_to_file ()
- strrchr ()
- strstr ()
- subseq ()
- substring ()
- strcontains ()
- starts_with ()
- ends_with ()
- tmp_file_name ()
- trim ()
- ucase ()
- upper ()

Transaction

- mts_connect ()
- mts_get_timeout ()
- mts_set_timeout ()
- mts_status ()

- txn_error ()
- txn_killall ()

Type Mapping

- import_jar ()
- internal_to_sql_type ()
- internal_type ()
- internal_type_name ()
- isbinary ()
- isentity ()
- isfloat ()
- isinteger ()
- isnull ()
- isnumeric ()
- isfinitenumeric ()
- isstring ()
- iszero ()
- jvm_ref_import ()
- sign ()
- udt_defines_field ()
- udt_get ()
- udt_implements_method ()
- udt_instance_of ()
- udt_set ()
- unimport_jar ()

UDDI

- uddi_delete_binding ()
- uddi_delete_business ()
- uddi_delete_service ()
- uddi_delete_tModel ()
- uddi_discard_authToken ()

- uddi_find_binding ()
- uddi_find_business ()
- uddi_find_service ()
- uddi_find_tModel ()
- uddi_get_authToken ()
- uddi_get_bindingDetail ()
- uddi_get_businessDetail ()
- uddi_get_businessDetailExt ()
- uddi_get_registeredInfo ()
- uddi_get_serviceDetail ()
- uddi_get_tModelDetail ()
- uddi_save_binding ()
- uddi_save_business ()
- uddi_save_service ()
- uddi_save_tModel ()

User Defined Types & The CLR

- import_clr ()
- unimport_clr ()

Web & Internet

- client_attr ()
- connection_get ()
- connection_id ()
- connection_is_dirty ()
- connection_set ()
- connection_vars ()
- connection_vars_set ()
- DAV add & update functions ()
- DAV manipulation functions ()
- DAV lock manipulation functions ()
- DAV search functions ()

- WebDAV Users & Groups administration ()
- DAV_EXP ()
- dbname ()
- ftp_get ()
- ftp_ls ()
- ftp_put ()
- get_certificate_info ()
- http ()
- http_lock ()
- http_unlock ()
- http_acl_set ()
- http_acl_get ()
- http_acl_remove ()
- http_client ()
- http_client_ext ()
- http_client_ip ()
- dns_txt_get ()
- http_debug_log ()
- http_enable_gz ()
- http_file ()
- http_flush ()
- http_internal_redirect ()
- http_get ()
- http_header ()
- http_header_get ()
- http_kill ()
- http_listen_host ()
- http_map_table ()
- http_map_get ()
- http_param ()
- http_path ()

- http_pending_req ()
- http_physical_path ()
- http_proxy ()
- http_request_header ()
- http_request_header_full ()
- http_request_status ()
- http_request_get ()
- http_rewrite ()
- http_root ()
- http_value ()
- json_parse ()
- http_url ()
- http_xslt ()
- URLREWRITE_CREATE_REGEX_RULE ()
- LFS_EXP ()
- SERV_QUEUE_TOP ()
- ses_connect ()
- ses_disconnect ()
- ses_read_line ()
- ses_write ()
- tcpip_gethostbyname ()
- tcpip_gethostbyaddr ()
- VHOST_DEFINE ()
- VHOST_REMOVE ()
- vsp_calculate_digest ()
- wsdl_import_udt ()
- USER_KEY_LOAD ()

XML

- createXML ()
- serialize_to_UTF8_xml ()
- tidy_html ()

- tidy_list_errors ()
- updateXML ()
- xte_head ()
- xte_node ()
- xte_node_from_nodebld ()
- xte_nodebld_acc ()
- xte_nodebld_final ()
- xte_nodebld_init ()
- XMLAGG ()
- XMLATTRIBUTES ()
- XMLAddAttribute ()
- XMLAppendChildren ()
- XMLCONCAT ()
- XMLELEMENT ()
- XMLFOREST ()
- XMLInsertAfter ()
- XMLInsertBefore ()
- XMLReplace ()
- xml_auto ()
- xml_auto_dtd ()
- xml_auto_schema ()
- xml_create_tables_from_mapping_schema_decl ()
- xml_cut ()
- xml_doc_output_option ()
- xml_load_schema_decl ()
- xml_load_mapping_schema_decl ()
- xml_namespace_scope ()
- xml_add_system_path ()
- xml_get_system_paths ()
- xml_persistent ()
- xml_template ()

- xml_tree ()
- xml_tree_doc ()
- xml_tree_doc_media_type ()
- xml_uri_get ()
- xml_validate_dtd ()
- xml_validate_schema ()
- xml_view_dtd ()
- xml_set_ns_decl ()
- xml_view_schema ()
- xmlsql_update ()
- XMLUpdate ()
- xpath_eval ()
- xper_cut ()
- xper_doc ()
- xper_locate_words ()
- XPER navigation ()
- xpf_extension ()
- xpf_extension_remove ()
- xquery_eval ()
- xslt ()
- xslt_format_number ()
- xslt_sheet ()
- xslt_stale ()
- xtree_doc ()
- XMLType.XMLType ()
- XMLType.createNonSchemaBasedXML ()
- XMLType.createSchemaBasedXML ()
- XMLType.createXML ()
- XMLType.existsNode ()
- XMLType.extract ()
- XMLType.getClobVal ()

- XMLType.getNamespace ()
- XMLType.getNumVal ()
- XMLType.getRootElement ()
- XMLType.getSchemaURL ()
- XMLType.getStringVal ()
- XMLType.isFragment ()
- XMLType.isSchemaBased ()
- XMLType.isSchemaValid ()
- XMLType.isSchemaValidated ()
- XMLType.schemaValidate ()
- XMLType.setSchemaValidated ()
- XMLType.toObject ()
- XMLType.transform ()

XPATH & XQUERY

Miscellaneous

- HS_Resolve ()

Geo Spatial

- ST_Affine ()
- st_point ()
- st_x ()
- st_y ()
- st_distance ()
- ST_SRID ()
- ST_SetSRID ()
- st_astext ()
- st_geomfromtext ()
- st_intersects ()
- st_contains ()
- st_within ()
- isgeometry ()

- geo_insert ()
- geo_delete ()
- GeometryType ()
- http_st_dxf_entity ()
- http_st_ewkt ()
- st_ewkt_read ()
- postgis_proj_version ()
- dist_from_point_to_line_segment ()
- earth_radius ()
- ST_ExteriorRing ()
- ST_GeometryN ()
- st_get_bounding_box ()
- st_get_bounding_box_n ()
- haversine_deg_km ()
- ST_InteriorRingN ()
- st_linestring ()
- ST_M ()
- st_may_intersect ()
- ST_MMax ()
- ST_MMin ()
- ST_NumGeometries ()
- ST_NumInteriorRings ()
- ST_Transform ()
- st_transform_by_custom_projection ()
- ST_Translate ()
- ST_TransScale ()
- ST_XMax ()
- ST_XMin ()
- ST_YMax ()
- ST_YMin ()
- ST_Z ()

- ST_ZMax ()
- ST_Zmflag ()
- ST_ZMin ()

VAD

- VAD_CHECK ()
- VAD_CHECK_INSTALLABILITY ()
- VAD_CHECK_UNINSTALLABILITY ()
- VAD_FAIL_CHECK ()
- VAD_INSTALL ()
- VAD_LOAD_FILE ()
- VAD_LOAD_SQL_FILE ()
- VAD_PACK ()
- VAD_SAFE_EXEC ()
- VAD_UNINSTALL ()

VAR

VAR — Returns the variance.

Synopsis

```
numeric VAR ( in expr any );
```

Description

VAR (VARIANCE) returns variance of expr. Virtuoso calculates the variance of expr as follows: 0 if the number of rows in expr = 1; VAR_SAMP if the number of rows in expr > 1

Parameters

expr
: Number expression.

Return Types

The function returns a value of type NUMERIC. If the function is applied to an empty set, then it returns null.

See Also

`VAR()`

`VAR_SAMP()`

`VAR_POP()`

`STDDEV()`

STDDEV_SAMP()

STDDEV_POP()

REGR_SYY()

REGR_SXX()

REGR_SXY()

REGR_AVGX()

REGR_AVGY()

REGR_R2()

REGR_COUNT()

REGR_INTERCEPT()

REGR_SLOPE()

COVAR_SAMP()

COVAR_POP()

CORR()

VAR_SAMP

VAR_SAMP — Returns the sample variance.

Synopsis

```
numeric VAR_SAMP ( in expr any );
```

Description

VAR_SAMP returns the sample variance of a set of numbers after discarding the nulls in this set. The expr is a numeric expression, and the function returns a value of type NUMERIC. If the function is applied to an empty set, then it returns null. The function makes the following calculation:

```
(SUM(expr2) - SUM(expr)2 / COUNT(expr)) / (COUNT(expr) - 1)
```

This function is similar to VAR, except that given an input set of one element, VAR returns 0 and VAR_SAMP returns null.

Parameters

expr
: Number expression.

Return Types

The function returns a value of type NUMERIC. If the function is applied to an empty set, then it returns null.

See Also

VAR()

VAR_SAMP()

VAR_POP()

STDDEV()

STDDEV_SAMP()

STDDEV_POP()

REGR_SYY()

REGR_SXX()

REGR_SXY()

REGR_AVGX()

REGR_AVGY()

REGR_R2()

REGR_COUNT()

REGR_INTERCEPT()

REGR_SLOPE()

COVAR_SAMP()

COVAR_POP()

CORR()

VAR_POP

VAR_POP — Returns the population variance.

Synopsis

```
numeric VAR_POP ( in expr any );
```

Description

VAR_POP returns the population variance of a set of numbers after discarding the nulls in this set. The expr is a number expression, and the function returns a value of type NUMERIC. If the function is applied to an empty set, then it returns null. The function makes the following calculation:

```
(SUM(expr2) - SUM(expr)2 / COUNT(expr)) / COUNT(expr)
```

Parameters

expr
: Number expression.

Return Types

The function returns a value of type NUMERIC. If the function is applied to an empty set, then it returns null.

See Also

VAR()

VAR_SAMP()

VAR_POP()

STDDEV()

STDDEV_SAMP()

STDDEV_POP()

REGR_SYY()

REGR_SXX()

REGR_SXY()

REGR_AVGX()

REGR_AVGY()

REGR_R2()

REGR_COUNT()

REGR_INTERCEPT()

REGR_SLOPE()

COVAR_SAMP()

COVAR_POP()

CORR()

STDDEV

STDDEV — Returns the standard deviation.

Synopsis

```
numeric STDDEV ( in expr1 any ,
                 in expr2 any ) ;
```

Description

STDDEV returns standard deviation. It returns STDDEV_SAMP if the number of pairs is more than one, or NULL.

Parameters

expr1
: Number expression.

expr2
: Number expression.

Return Types

The function returns a value of type NUMERIC. If the function is applied to an empty set, then it returns null.

See Also

VAR()

VAR_SAMP()

VAR_POP()

STDDEV()

STDDEV_SAMP()

STDDEV_POP()

REGR_SYY()

REGR_SXX()

REGR_SXY()

REGR_AVGX()

REGR_AVGY()

REGR_R2()

REGR_COUNT()

REGR_INTERCEPT()

REGR_SLOPE()

COVAR_SAMP()

COVAR_POP()

CORR()

STDDEV_SAMP

STDDEV_SAMP — Returns the cumulative sample standard deviation.

Synopsis

```
numeric STDDEV_SAMP ( in expr any );
```

Description

STDDEV_SAMP computes the cumulative sample standard deviation and returns the square root of the sample variance. The expr is a numeric expression, and the function returns a value of type NUMERIC. This function is same as the square root of the VAR_SAMP function. When VAR_SAMP returns null, this function returns null.

Parameters

expr
: Number expression.

Return Types

The function returns a value of type NUMERIC. If the function is applied to an empty set, then it returns null.

See Also

VAR()

VAR_SAMP()

VAR_POP()

STDDEV()

STDDEV_SAMP()

STDDEV_POP()

REGR_SYY()

REGR_SXX()

REGR_SXY()

REGR_AVGX()

REGR_AVGY()

REGR_R2()

REGR_COUNT()

REGR_INTERCEPT()

REGR_SLOPE()

COVAR_SAMP()

COVAR_POP()

CORR ( )

STDDEV_POP

STDDEV_POP — Returns the population standard deviation.

Synopsis

```
numeric STDDEV_POP ( in expr any );
```

Description

STDDEV_POP computes the population standard deviation and returns the square root of the population variance. This function is the same as the square root of the VAR_POP function. When VAR_POP returns null, returns null.

Parameters

expr
: Number expression.

Return Types

The function returns a value of type NUMERIC. If the function is applied to an empty set, then it returns null.

See Also

VAR()

VAR_SAMP()

VAR_POP()

STDDEV()

STDDEV_SAMP()

STDDEV_POP()

REGR_SYY()

REGR_SXX()

REGR_SXY()

REGR_AVGX()

REGR_AVGY()

REGR_R2()

REGR_COUNT()

REGR_INTERCEPT()

REGR_SLOPE()

COVAR_SAMP()

COVAR_POP()

CORR ( )

REGR_SYY

REGR_SYY — Auxiliary function used to compute various diagnostic statistics.

Synopsis

```
numeric REGR_SYY ( in expr1 any ,
                   in expr2 any );
```

Description

REGR_SYY makes the following computation after eliminating NULL (expr1, expr2) pairs:

```
REGR_COUNT(expr1, expr2) * VAR_POP(expr1)
```

Parameters

expr1
: Number expression.

expr2
: Number expression.

Return Types

The function returns a value of type NUMERIC. If the function is applied to an empty set, then it returns null.

See Also

VAR()

VAR_SAMP()

VAR_POP()

STDDEV()

STDDEV_SAMP()

STDDEV_POP()

REGR_SYY()

REGR_SXX()

REGR_SXY()

REGR_AVGX()

REGR_AVGY()

REGR_R2()

REGR_COUNT()

REGR_INTERCEPT()

REGR_SLOPE()

COVAR_SAMP()

COVAR_POP()

CORR()

REGR_SXX

REGR_SXX — Auxiliary function used to compute various diagnostic statistics.

Synopsis

```
numeric REGR_SXX ( in expr1 any ,
                   in expr2 any );
```

Description

REGR_SXX makes the following computation after eliminating NULL (expr1, expr2) pairs:

```
REGR_COUNT(expr1, expr2) * VAR_POP(expr2)
```

Parameters

expr1
: Number expression.

expr2
: Number expression.

Return Types

The function returns a value of type NUMERIC. If the function is applied to an empty set, then it returns null.

See Also

VAR()

VAR_SAMP()

VAR_POP()

STDDEV()

STDDEV_SAMP()

STDDEV_POP()

REGR_SYY()

REGR_SXX()

REGR_SXY()

REGR_AVGX()

REGR_AVGY()

REGR_R2()

REGR_COUNT()

REGR_INTERCEPT()

REGR_SLOPE()

COVAR_SAMP()

COVAR_POP()

CORR()

REGR_SXY

REGR_SXY — Auxiliary function used to compute various diagnostic statistics.

Synopsis

```
numeric REGR_SXY ( in expr1 any ,
                   in expr2 any );
```

Description

REGR_SXY makes the following computation after eliminating NULL (expr1, expr2) pairs:

```
REGR_COUNT(expr1, expr2) * COVAR_POP(expr1, expr2)
```

Parameters

expr1
: Number expression.

expr2
: Number expression.

Return Types

The function returns a value of type NUMERIC. If the function is applied to an empty set, then it returns null.

See Also

VAR()

VAR_SAMP()

VAR_POP()

STDDEV()

STDDEV_SAMP()

STDDEV_POP()

REGR_SYY()

REGR_SXX()

REGR_SXY()

REGR_AVGX()

REGR_AVGY()

REGR_R2()

REGR_COUNT()

REGR_INTERCEPT()

REGR_SLOPE()

COVAR_SAMP()

COVAR_POP()

CORR()

REGR_AVGX

REGR_AVGX — Evaluates the average of the independent variable (expr2) of the regression line.

Synopsis

```
numeric REGR_AVGX ( in expr1 any ,
                    in expr2 any ) ;
```

Description

REGR_AVGX evaluates the average of the independent variable (expr2) of the regression line. It makes the following computation after the elimination of null (expr1, expr2) pairs:

```
AVG(expr2)
```

Parameters

expr1
: Number expression.

expr2
: Number expression.

Return Types

The function returns a value of type NUMERIC. If the function is applied to an empty set, then it returns null.

See Also

VAR()

VAR_SAMP()

VAR_POP()

STDDEV()

STDDEV_SAMP()

STDDEV_POP()

REGR_SYY()

REGR_SXX()

REGR_SXY()

REGR_AVGX()

REGR_AVGY()

REGR_R2()

REGR_COUNT()

REGR_INTERCEPT()

REGR_SLOPE()

COVAR_SAMP()

COVAR_POP()

CORR()

REGR_AVGY

REGR_AVGY — Evaluates the average of the independent variable (expr1) of the regression line.

Synopsis

```
numeric REGR_AVGY ( in expr1 any ,
                    in expr2 any ) ;
```

Description

REGR_AVGY evaluates the average of the independent variable (expr1) of the regression line. It makes the following computation after the elimination of null (expr1, expr2) pairs:

```
AVG(expr1)
```

Parameters

expr1
: Number expression.

expr2
: Number expression.

Return Types

The function returns a value of type NUMERIC. If the function is applied to an empty set, then it returns null.

See Also

VAR()

VAR_SAMP()

VAR_POP()

STDDEV()

STDDEV_SAMP()

STDDEV_POP()

REGR_SYY()

REGR_SXX()

REGR_SXY()

REGR_AVGX()

REGR_AVGY()

REGR_R2()

REGR_COUNT()

REGR_INTERCEPT()

REGR_SLOPE()

COVAR_SAMP()

COVAR_POP()

CORR()

REGR_R2

REGR_R2 — Returns the coefficient of determination (R-squared) of the regression line.

Synopsis

```
numeric REGR_R2 ( in expr1 any ,
                  in expr2 any ) ;
```

Description

REGR_R2 returns the coefficient of determination (also called "R-squared" or "goodness of fit") for the regression. `VAR_POP(expr1)` and `VAR_POP(expr2)` are evaluated after the elimination of null pairs. The return values are:

NULL if VAR_POP(expr2) = 0
1 if VAR_POP(expr1) = 0 and VAR_POP(expr2) != 0
(CORR(expr1,expr2) * CORR(expt1, expr2)) if VAR_POP(expr1) > 0 and VAR_POP(expr2) != 0

Parameters

expr1
: Number expression.

expr2
: Number expression.

Return Types

The function returns a value of type NUMERIC. If the function is applied to an empty set, then it returns null.

See Also

`VAR()`

`VAR_SAMP()`

`VAR_POP()`

`STDDEV()`

`STDDEV_SAMP()`

`STDDEV_POP()`

`REGR_SYY()`

`REGR_SXX()`

`REGR_SXY()`

`REGR_AVGX()`

`REGR_AVGY()`

`REGR_R2()`

`REGR_COUNT()`

REGR_INTERCEPT()

REGR_SLOPE()

COVAR_SAMP()

COVAR_POP()

CORR()

REGR_COUNT

REGR_COUNT — Returns the number of non-null numbers used to fit the regression line.

Synopsis

```
numeric REGR_COUNT ( in expr1 any ,
                     in expr2 any );
```

Description

REGR_COUNT Returns the number of non-null numbers used to fit the regression line.

Parameters

expr1
: Number expression.

expr2
: Number expression.

Return Types

The function returns a value of type NUMERIC.

See Also

VAR()

VAR_SAMP()

VAR_POP()

STDDEV()

STDDEV_SAMP()

STDDEV_POP()

REGR_SYY()

REGR_SXX()

REGR_SXY()

REGR_AVGX()

REGR_AVGY()

REGR_R2()

REGR_COUNT()

REGR_INTERCEPT()

REGR_SLOPE()

COVAR_SAMP()

COVAR_POP()

CORR()

REGR_INTERCEPT

REGR_INTERCEPT — Returns the y-intercept of the regression line.

Synopsis

```
numeric REGR_INTERCEPT ( in expr1 any ,
                         in expr2 any ) ;
```

Description

REGR_INTERCEPT returns the y-intercept of the regression line. After the elimination of null (expr1, expr2) pairs, it makes the following computation:

```
AVG(expr1) - REGR_SLOPE(expr1, expr2) * AVG(expr2)
```

Parameters

expr1
: Number expression.

expr2
: Number expression.

Return Types

The function returns a value of type NUMERIC. If the function is applied to an empty set, then it returns null.

See Also

VAR()

VAR_SAMP()

VAR_POP()

STDDEV()

STDDEV_SAMP()

STDDEV_POP()

REGR_SYY()

REGR_SXX()

REGR_SXY()

REGR_AVGX()

REGR_AVGY()

REGR_R2()

REGR_COUNT()

REGR_INTERCEPT()

REGR_SLOPE()

COVAR_SAMP()

COVAR_POP()

CORR()

REGR_SLOPE

REGR_SLOPE — Returns the slope of the line.

Synopsis

```
numeric REGR_SLOPE ( in expr1 any ,
                     in expr2 any ) ;
```

Description

REGR_SLOPE returns the slope of the line. After the elimination of null (expr1, expr2) pairs, it makes the following computation:

```
COVAR_POP(expr1, expr2) / VAR_POP(expr2)
```

Parameters

expr1
: Number expression.

expr2
: Number expression.

Return Types

The function returns a value of type NUMERIC. If the function is applied to an empty set, then it returns null.

See Also

`VAR()`

`VAR_SAMP()`

`VAR_POP()`

`STDDEV()`

`STDDEV_SAMP()`

`STDDEV_POP()`

`REGR_SYY()`

`REGR_SXX()`

`REGR_SXY()`

`REGR_AVGX()`

`REGR_AVGY()`

`REGR_R2()`

`REGR_COUNT()`

`REGR_INTERCEPT()`

REGR_SLOPE()

COVAR_SAMP()

COVAR_POP()

CORR()

COVAR_SAMP

COVAR_SAMP — Returns the sample covariance of a set of number pairs.

Synopsis

```
numeric COVAR_SAMP ( in expr1 any ,
                     in expr2 any ) ;
```

Description

COVAR_SAMP returns the sample covariance of a set of number pairs. Both expr1 and expr2 are numeric expressions. Virtuoso applies the function to the set of (expr1, expr2) pairs after eliminating all pairs for which either expr1 or expr2 is null. Then Virtuoso makes the following computation:

```
(SUM(expr1 * expr2) - SUM(expr2) * SUM(expr1) / n) / (n-1)
```

where n is the number of (expr1, expr2) pairs where neither expr1 nor expr2 is null.

Parameters

expr1
: Number expression.

expr2
: Number expression.

Return Types

The function returns a value of type NUMERIC. If the function is applied to an empty set, then it returns null.

See Also

VAR()

VAR_SAMP()

VAR_POP()

STDDEV()

STDDEV_SAMP()

STDDEV_POP()

REGR_SYY()

REGR_SXX()

REGR_SXY()

REGR_AVGX()

REGR_AVGY()

REGR_R2()

REGR_COUNT()

REGR_INTERCEPT()

REGR_SLOPE()

COVAR_SAMP()

COVAR_POP()

CORR()

COVAR_POP

COVAR_POP — Returns the population covariance of a set of number pairs.

Synopsis

```
numeric COVAR_POP ( in expr1 any ,
                    in expr2 any );
```

Description

COVAR_POP returns the population covariance of a set of number pairs. Both expr1 and expr2 are numeric expressions. Virtuoso applies the function to the set of (expr1, expr2) pairs after eliminating all pairs for which either expr1 or expr2 is null. Then Virtuoso makes the following computation:

```
(SUM(expr1 * expr2) - SUM(expr2) * SUM(expr1) / n) / n
```

where n is the number of (expr1, expr2) pairs where neither expr1 nor expr2 is null.

Parameters

expr1
: Number expression.

expr2
: Number expression.

Return Types

The function returns a value of type NUMERIC. If the function is applied to an empty set, then it returns null.

See Also

VAR()

VAR_SAMP()

VAR_POP()

STDDEV()

STDDEV_SAMP()

STDDEV_POP()

REGR_SYY()

REGR_SXX()

REGR_SXY()

REGR_AVGX()

REGR_AVGY()

REGR_R2()

REGR_COUNT()

REGR_INTERCEPT()

REGR_SLOPE()

COVAR_SAMP()

COVAR_POP()

CORR()

CORR

CORR — Returns the coefficient of correlation of a set of number pairs.

Synopsis

```
numeric CORR ( in expr1 any ,
               in expr2 any );
```

Description

CORR returns the coefficient of correlation of a set of number pairs. Both expr1 and expr2 are number expressions. Virtuoso applies the function to the set of (expr1, expr2) after eliminating the pairs for which either expr1 or expr2 is null. Then Virtuoso makes the following computation:

```
COVAR_POP(expr1, expr2) / (STDDEV_POP(expr1) * STDDEV_POP(expr2))
```

Parameters

expr1
: Number expression.

expr2
: Number expression.

Return Types

The function returns a value of type NUMERIC. If the function is applied to an empty set, then it returns null.

See Also

VAR()

VAR_SAMP()

VAR_POP()

STDDEV()

STDDEV_SAMP()

STDDEV_POP()

REGR_SYY()

REGR_SXX()

REGR_SXY()

REGR_AVGX()

REGR_AVGY()

REGR_R2()

REGR_COUNT()

REGR_INTERCEPT()

REGR_SLOPE()

COVAR_SAMP()

COVAR_POP()

CORR()

DB.DBA.XQ_SEQUENCE_AGG

DB.DBA.XQ_SEQUENCE_AGG — Creates an XQuery sequence object that consists of all values passed the the aggregate.

Synopsis

```
vector DB.DBA.XQ_SEQUENCE_AGG ( value_expression any );
```

Description

DB.DBA.XQ_SEQUENCE_AGG creates an XQuery sequence object that consists of all values passed the aggregate.

Parameters

value_expression
: Values are "flatten" before being added to the sequence. It means that --- NULLs are not added at all.

Return Types

Any

See Also

xquery_eval()

DB.DBA.VECTOR_AGG()

DB.DBA.VECTOR_OF_NONNULLS_AGG()

DB.DBA.VECTOR_OR_NULL_AGG()

DB.DBA.VECTOR_OF_NONNULLS_OR_NULL_AGG()

DB.DBA.VECTOR_CONCAT_AGG()

DB.DBA.BAG_AGG()

DB.DBA.BAG_OF_NONNULLS_AGG()

DB.DBA.BAG_OR_NULL_AGG()

DB.DBA.BAG_OF_NONNULLS_OR_NULL_AGG()

DB.DBA.BAG_CONCAT_AGG()

DB.DBA.GROUP_CONCAT()

DB.DBA.GROUP_CONCAT_DISTINCT()

DB.DBA.GROUP_DIGEST()

DB.DBA.SAMPLE()

DB.DBA.VECTOR_AGG

DB.DBA.VECTOR_AGG — Returns a vector of passed values.

Synopsis

```
vector DB.DBA.VECTOR_AGG ( value_expression any );
```

Description

DB.DBA.VECTOR_AGG returns a vector of passed values. The order of items in the vector repeats the order of passing individual values to the aggregate.

Parameters

value_expression
: Value of item for aggregation.

Return Types

Any

Examples

Example 24.1.

As the order of items in the vector repeats the order of passing individual values to the aggregate, it is possible to write, for instance:

```
-- Get an ordered vector of X-es
SELECT DB.DBA.VECTOR_AGG (subquery.X)
FROM (SELECT X FROM ... ORDER BY X) as subquery
```

or

```
SELECT DB.DBA.VECTOR_AGG (subquery.Y)
FROM (SELECT Y FROM ... ORDER BY X) as subquery
```

and get vector of Y-s that are ordered by matching values of X-es.

See Also

DB.DBA.XQ_SEQUENCE_AGG()

DB.DBA.VECTOR_OF_NONNULLS_AGG()

DB.DBA.VECTOR_OR_NULL_AGG()

DB.DBA.VECTOR_OF_NONNULLS_OR_NULL_AGG()

DB.DBA.VECTOR_CONCAT_AGG()

DB.DBA.BAG_AGG()

DB.DBA.BAG_OF_NONNULLS_AGG()

DB.DBA.BAG_OR_NULL_AGG()

DB.DBA.BAG_OF_NONNULLS_OR_NULL_AGG()

DB.DBA.BAG_CONCAT_AGG()

DB.DBA.GROUP_CONCAT()

DB.DBA.GROUP_CONCAT_DISTINCT()

DB.DBA.GROUP_DIGEST()

DB.DBA.SAMPLE()

DB.DBA.VECTOR_OF_NONNULLS_AGG

DB.DBA.VECTOR_OF_NONNULLS_AGG — Returns a vector of passed values ingoring NULL arguments.

Synopsis

```
vector DB.DBA.VECTOR_OF_NONNULLS_AGG ( value_expression any );
```

Description

DB.DBA.VECTOR_OF_NONNULLS_AGG returns a vector of passed values. Similar to DB.DBA.VECTOR_AGG() but ignores NULL arguments.

The order of items in the vector repeats the order of passing individual values to the aggregate.

Parameters

value_expression
: Value of item for aggregation.

Return Types

Any

Examples

Example 24.2. Simple Use

As the order of items in the vector repeats the order of passing individual values to the aggregate, it is possible to write, for instance:

```
-- Get an ordered vector of X-es
SELECT DB.DBA.VECTOR_OF_NONNULLS_AGG (subquery.X)
FROM (SELECT X FROM ... ORDER BY X) as subquery
```

or

```
SELECT DB.DBA.VECTOR_OF_NONNULLS_AGG (subquery.Y)
FROM (SELECT Y FROM ... ORDER BY X) as subquery
```

and get vector of Y-s that are ordered by matching values of X-es.

See Also

DB.DBA.VECTOR_AGG()

DB.DBA.VECTOR_OR_NULL_AGG()

DB.DBA.VECTOR_OF_NONNULLS_OR_NULL_AGG()

DB.DBA.VECTOR_CONCAT_AGG()

DB.DBA.BAG_AGG()

DB.DBA.BAG_OF_NONNULLS_AGG()

DB.DBA.BAG_OR_NULL_AGG()

DB.DBA.BAG_OF_NONNULLS_OR_NULL_AGG()

DB.DBA.BAG_CONCAT_AGG()

DB.DBA.GROUP_CONCAT()

DB.DBA.GROUP_CONCAT_DISTINCT()

DB.DBA.GROUP_DIGEST()

DB.DBA.SAMPLE()

DB.DBA.VECTOR_OR_NULL_AGG

DB.DBA.VECTOR_OR_NULL_AGG — Returns a vector of passed values.

Synopsis

```
vector DB.DBA.VECTOR_OR_NULL_AGG ( value_expression any );
```

Description

DB.DBA.VECTOR_OR_NULL_AGG returns a vector of passed values. Similar to DB.DBA.VECTOR_AGG() but returns NULL instead of an empty vector if no arguments were actually passed.

The order of items in the vector repeats the order of passing individual values to the aggregate.

Parameters

value_expression
: Value of item for aggregation.

Return Types

Any. If no arguments were actually passed returns NULL instead of an empty vector.

Examples

Example 24.3. Simple Use

As the order of items in the vector repeats the order of passing individual values to the aggregate, it is possible to write, for instance:

```
-- Get an ordered vector of X-es
SELECT DB.DBA.VECTOR_OR_NULL_AGG (subquery.X)
FROM (SELECT X FROM ... ORDER BY X) as subquery
```

or

```
SELECT DB.DBA.VECTOR_OR_NULL_AGG (subquery.Y)
FROM (SELECT Y FROM ... ORDER BY X) as subquery
```

and get vector of Y-s that are ordered by matching values of X-es.

See Also

DB.DBA.XQ_SEQUENCE_AGG()

DB.DBA.VECTOR_AGG()

DB.DBA.VECTOR_OF_NONNULLS_AGG()

DB.DBA.VECTOR_OF_NONNULLS_OR_NULL_AGG()

DB.DBA.VECTOR_CONCAT_AGG()

DB.DBA.BAG_AGG()

DB.DBA.BAG_OF_NONNULLS_AGG()

DB.DBA.BAG_OR_NULL_AGG()

DB.DBA.BAG_OF_NONNULLS_OR_NULL_AGG()

DB.DBA.BAG_CONCAT_AGG()

DB.DBA.GROUP_CONCAT()

DB.DBA.GROUP_CONCAT_DISTINCT()

DB.DBA.GROUP_DIGEST()

DB.DBA.SAMPLE()

DB.DBA.VECTOR_OF_NONNULLS_OR_NULL_AGG

DB.DBA.VECTOR_OF_NONNULLS_OR_NULL_AGG — Returns a vector of passed values ingoring NULL arguments.

Synopsis

```
vector DB.DBA.VECTOR_OF_NONNULLS_OR_NULL_AGG ( value_expression any );
```

Description

DB.DBA.VECTOR_OF_NONNULLS_OR_NULL_AGG returns a vector of passed values. Similar to DB.DBA.VECTOR_OF_NONNULLS_AGG() but returns NULL instead of an empty vector if no arguments were passed or only NULLs were passed.

Parameters

value_expression
: Value of item for aggregation.

Return Types

Any. If no arguments were passed teturns NULL instead of an empty vector or only NULLs were passed.

Examples

Example 24.4. Simple Use

As the order of items in the vector repeats the order of passing individual values to the aggregate, it is possible to write, for instance:

```
-- Get an ordered vector of X-es
SELECT DB.DBA.VECTOR_OF_NONNULLS_OR_NULL_AGG (subquery.X)
FROM (SELECT X FROM ... ORDER BY X) as subquery
```

or

```
SELECT DB.DBA.VECTOR_OF_NONNULLS_OR_NULL_AGG (subquery.Y)
FROM (SELECT Y FROM ... ORDER BY X) as subquery
```

and get vector of Y-s that are ordered by matching values of X-es.

See Also

DB.DBA.XQ_SEQUENCE_AGG()

DB.DBA.VECTOR_AGG()

DB.DBA.VECTOR_OF_NONNULLS_AGG()

DB.DBA.VECTOR_OR_NULL_AGG()

DB.DBA.VECTOR_CONCAT_AGG()

DB.DBA.BAG_AGG()

DB.DBA.BAG_OF_NONNULLS_AGG()

DB.DBA.BAG_OR_NULL_AGG()

DB.DBA.BAG_OF_NONNULLS_OR_NULL_AGG()

DB.DBA.BAG_CONCAT_AGG()

DB.DBA.GROUP_CONCAT()

DB.DBA.GROUP_CONCAT_DISTINCT()

DB.DBA.GROUP_DIGEST()

DB.DBA.SAMPLE()

DB.DBA.VECTOR_CONCAT_AGG

DB.DBA.VECTOR_CONCAT_AGG — Expects SQL vectors as arguments and the result is a concatenation of all passed vectors.

Synopsis

```
vector DB.DBA.VECTOR_CONCAT_AGG ( _items any );
```

Description

DB.DBA.VECTOR_CONCAT_AGG expects SQL vectors as arguments and the result is a concatenation of all passed vectors.

Parameters

_items
: SQL vectors.

Return Types

Any

Examples

Example 24.5. Simple Use

The most typical usage is the composing of get-keyword style vector, for ex:

```
  SELECT DB.DBA.VECTOR_CONCAT_AGG (vector (U_NAME, U_ID))
    FROM DB.DBA.SYS_USERS
ORDER BY U_NAME;
```

which returns a vector of passed values. The order of items in the vector repeats the order of passing argument vectors to the aggregate.

```
 (dba, 0, dav, 2, administrators, 3, nobody, 5, nogroup, 6, __rdf_repl, 7, SPARQL_SELECT, 100, SPARQL_SPO
No. of rows in result: 1
```

See Also

DB.DBA.XQ_SEQUENCE_AGG()

DB.DBA.VECTOR_AGG()

DB.DBA.VECTOR_OF_NONNULLS_AGG()

DB.DBA.VECTOR_OR_NULL_AGG()

DB.DBA.VECTOR_OF_NONNULLS_OR_NULL_AGG()

DB.DBA.BAG_AGG()

DB.DBA.BAG_OF_NONNULLS_AGG()

DB.DBA.BAG_OR_NULL_AGG()

DB.DBA.BAG_OF_NONNULLS_OR_NULL_AGG()

DB.DBA.BAG_CONCAT_AGG()

DB.DBA.GROUP_CONCAT()

DB.DBA.GROUP_CONCAT_DISTINCT()

DB.DBA.GROUP_DIGEST()

DB.DBA.SAMPLE()

DB.DBA.BAG_AGG

DB.DBA.BAG_AGG — Returns a vector of passed values.

Synopsis

```
vector DB.DBA.BAG_AGG ( value_expression any );
```

Description

`DB.DBA.BAG_AGG()` returns a vector of passed values.

This is similar to DB.DBA.VECTOR_AGG() and returns a vector of passed values but the order of items in the vector is not predefined and may vary from run to run.

A query with `DB.DBA.BAG_AGG()` may work faster than a similar query with DB.DBA.VECTOR_AGG() because `DB.DBA.BAG_AGG()` gives more freedom to the SQL optimizer. If the order is not important for your application, consider using of `DB.DBA.BAG_AGG()` .

Parameters

value_expression
: Value of item for aggregation.

Return Types

Any

Examples

Example 24.6. Simple Use

```
SELECT DB.DBA.BAG_AGG (subquery.X)
FROM (SELECT X FROM ... ORDER BY X) as subquery
```

See Also

DB.DBA.XQ_SEQUENCE_AGG()

DB.DBA.VECTOR_AGG()

DB.DBA.VECTOR_OF_NONNULLS_AGG()

DB.DBA.VECTOR_OR_NULL_AGG()

DB.DBA.VECTOR_OF_NONNULLS_OR_NULL_AGG()

DB.DBA.VECTOR_CONCAT_AGG()

DB.DBA.BAG_OF_NONNULLS_AGG()

DB.DBA.BAG_OR_NULL_AGG()

DB.DBA.BAG_OF_NONNULLS_OR_NULL_AGG()

DB.DBA.BAG_CONCAT_AGG()

DB.DBA.GROUP_CONCAT()

DB.DBA.GROUP_CONCAT_DISTINCT()

DB.DBA.GROUP_DIGEST()

DB.DBA.SAMPLE()

DB.DBA.BAG_OF_NONNULLS_AGG

DB.DBA.BAG_OF_NONNULLS_AGG — Returns a vector of passed values ingoring NULL arguments.

Synopsis

```
vector DB.DBA.BAG_OF_NONNULLS_AGG ( value_expression any );
```

Description

DB.DBA.BAG_OF_NONNULLS_AGG returns a vector of passed values. Similar to DB.DBA.VECTOR_OF_NONNULLS_AGG() but ignores NULL arguments.

The order of items in the vector is not predefined and may vary from run to run.

Parameters

value_expression
: Value of item for aggregation.

Return Types

Any

Examples

Example 24.7. Simple Use

```
SELECT DB.DBA.BAG_OF_NONNULLS_AGG (subquery.X)
FROM (SELECT X FROM ... ORDER BY X) as subquery

SELECT DB.DBA.BAG_OF_NONNULLS_AGG (subquery.Y)
FROM (SELECT Y FROM ... ORDER BY X) as subquery
```

See Also

DB.DBA.XQ_SEQUENCE_AGG()

DB.DBA.VECTOR_AGG()

DB.DBA.VECTOR_OF_NONNULLS_AGG()

DB.DBA.VECTOR_OR_NULL_AGG()

DB.DBA.VECTOR_OF_NONNULLS_OR_NULL_AGG()

DB.DBA.VECTOR_CONCAT_AGG()

DB.DBA.BAG_AGG()

DB.DBA.BAG_OR_NULL_AGG()

DB.DBA.BAG_OF_NONNULLS_OR_NULL_AGG()

DB.DBA.BAG_CONCAT_AGG()

DB.DBA.GROUP_CONCAT()

DB.DBA.GROUP_CONCAT_DISTINCT()

DB.DBA.GROUP_DIGEST()

DB.DBA.SAMPLE()

DB.DBA.BAG_OR_NULL_AGG

DB.DBA.BAG_OR_NULL_AGG — Returns a vector of passed values.

Synopsis

```
vector DB.DBA.BAG_OR_NULL_AGG ( value_expression any );
```

Description

DB.DBA.BAG_OR_NULL_AGG returns a vector of passed values. Similar to DB.DBA.BAG_AGG() but returns NULL instead of an empty vector if no arguments were actually passed.

The order of items in the vector is not predefined and may vary from run to run.

Parameters

value_expression
: Value of item for aggregation.

Return Types

Any. If no arguments were actually passed returns NULL instead of an empty vector.

Examples

Example 24.8. Simple Use

```
SELECT DB.DBA.BAG_OR_NULL_AGG (subquery.X)
FROM (SELECT X FROM ... ORDER BY X) as subquery
```

See Also

DB.DBA.XQ_SEQUENCE_AGG()

DB.DBA.VECTOR_AGG()

DB.DBA.VECTOR_OF_NONNULLS_AGG()

DB.DBA.VECTOR_OR_NULL_AGG()

DB.DBA.VECTOR_OF_NONNULLS_OR_NULL_AGG()

DB.DBA.VECTOR_CONCAT_AGG()

DB.DBA.BAG_AGG()

DB.DBA.BAG_OF_NONNULLS_AGG()

DB.DBA.BAG_OF_NONNULLS_OR_NULL_AGG()

DB.DBA.BAG_CONCAT_AGG()

DB.DBA.GROUP_CONCAT()

DB.DBA.GROUP_CONCAT_DISTINCT()

DB.DBA.GROUP_DIGEST()

DB.DBA.SAMPLE()

DB.DBA.BAG_OF_NONNULLS_OR_NULL_AGG

DB.DBA.BAG_OF_NONNULLS_OR_NULL_AGG — Returns a vector of passed values ingoring NULL arguments.

Synopsis

```
vector DB.DBA.BAG_OF_NONNULLS_OR_NULL_AGG ( value_expression any );
```

Description

DB.DBA.BAG_OF_NONNULLS_OR_NULL_AGG returns a vector of passed values. Similar to DB.DBA.BAG_OF_NONNULLS_AGG() but returns NULL instead of an empty vector if no arguments were passed or only NULLs were passed.

The order of items in the vector is not predefined and may vary from run to run.

Parameters

value_expression
: Value of item for aggregation.

Return Types

Any. If no arguments were passed teturns NULL instead of an empty vector or only NULLs were passed.

Examples

Example 24.9. Simple Use

```
SELECT DB.DBA.BAG_OF_NONNULLS_OR_NULL_AGG (subquery.X)
FROM (SELECT X FROM ... ORDER BY X) as subquery
```

See Also

DB.DBA.XQ_SEQUENCE_AGG()

DB.DBA.VECTOR_AGG()

DB.DBA.VECTOR_OF_NONNULLS_AGG()

DB.DBA.VECTOR_OR_NULL_AGG()

DB.DBA.VECTOR_OF_NONNULLS_OR_NULL_AGG()

DB.DBA.VECTOR_CONCAT_AGG()

DB.DBA.BAG_AGG()

DB.DBA.BAG_OF_NONNULLS_AGG()

DB.DBA.BAG_OR_NULL_AGG()

DB.DBA.BAG_CONCAT_AGG()

DB.DBA.GROUP_CONCAT()

DB.DBA.GROUP_CONCAT_DISTINCT()

DB.DBA.GROUP_DIGEST()

DB.DBA.SAMPLE()

DB.DBA.BAG_CONCAT_AGG

DB.DBA.BAG_CONCAT_AGG — Expects SQL vectors as arguments and the result is a concatenation of all passed vectors.

Synopsis

```
vector DB.DBA.BAG_CONCAT_AGG ( _items any );
```

Description

DB.DBA.BAG_CONCAT_AGG expects SQL vectors as arguments and the result is a concatenation of all passed vectors.

Parameters

_items
: SQL vectors.

Return Types

Any

Examples

Example 24.10. Simple Use

The most typical usage is the composing of get-keyword style vector, for ex:

```
  SELECT DB.DBA.BAG_CONCAT_AGG (vector (U_NAME, U_ID))
    FROM DB.DBA.SYS_USERS
ORDER BY U_NAME;
```

which returns a vector of passed values. The order of items in the vector repeats the order of passing argument vectors to the aggregate.

```
 (dba, 0, dav, 2, administrators, 3, nobody, 5, nogroup, 6, __rdf_repl, 7, SPARQL_SELECT, 100, SPARQL_SPO
No. of rows in result: 1
```

See Also

DB.DBA.XQ_SEQUENCE_AGG()

DB.DBA.VECTOR_AGG()

DB.DBA.VECTOR_OF_NONNULLS_AGG()

DB.DBA.VECTOR_OR_NULL_AGG()

DB.DBA.VECTOR_OF_NONNULLS_OR_NULL_AGG()

DB.DBA.VECTOR_CONCAT_AGG()

DB.DBA.BAG_AGG()

DB.DBA.BAG_OF_NONNULLS_AGG()

DB.DBA.BAG_OR_NULL_AGG()

DB.DBA.BAG_OF_NONNULLS_OR_NULL_AGG()

DB.DBA.GROUP_CONCAT()

DB.DBA.GROUP_CONCAT_DISTINCT()

DB.DBA.GROUP_DIGEST()

DB.DBA.SAMPLE()

abs

abs — Return the absolute value of a number

Synopsis

abs (in *num* any);

Description

`abs` returns the absolute value of its argument.

Parameters

num
: Numeric value whose absolute value is to be returned

Return Types

Same as parameter type.

Examples

Example 24.11. Simple example

Get absolute values of several numeric values

```
SQL> select cast (abs (-2.343) as numeric),
  cast(abs (0) as numeric),
  cast (abs (3.1415) as numeric);

callret          callret         callret
DECIMAL          DECIMAL         DECIMAL
_____________________________________________

2.343            0               3.1415

1 Rows. -- 5 msec.
```

__any_grants

__any_grants — Checks a table for grants.

Synopsis

```
integer __any_grants ( in tablename varchar );
```

Description

The `__any_grants()` can be used to test whether there are any rights granted (for insert/update/delete) to a table for current SQL account.

Parameters

tablename
: The table name to be tested.

Return Types

An integer will be returned to indicate whether the table supplied has any privileges granted (1) or not granted (zero 0) for current SQL user.

Errors

Table 24.1. Errors signalled by

SQLState	Error Code	Error Text	Description
22023	SR014	Function any_grants needs a string as argument 1, not an arg of type [type]	

Examples

Example 24.12. Using the __any_grants() function

This simple example shows testing a grant rights for a table, using the __any_grants() function from ISQL and demo SQL user accout.

```
$isql localhost:1112 demo demo
-- the demo account owns an Orders table, so it's granted
SQL> select __any_grants ('Demo.demo.Orders');
callret
INTEGER
_______________________________________________________________________________

1

1 Rows. -- 1 msec.

-- but it has no permission to access the table SYS_REPL_ACCOUNTS
SQL> select __any_grants ('DB.DBA.SYS_REPL_ACCOUNTS');
callret
INTEGER
_______________________________________________________________________________

0

1 Rows. -- 5 msec.
```

aref

aref — returns specific element of an array or string

Synopsis

aref (*arg* any ,
nth integer) ;

Description

`aref` returns the *`nth`* element of an array , string or string_session , where *`nth`* is a zero-based index. If the first argument is a string or string_session , the integer ASCII value of the *`nth`* character is returned. If the first argument is an array of any , then the corresponding element is returned.

Parameters

arg
: array , vector or string .

nth
: integer zero-based index.

Return Values

An integer character code or *`nth`* element as whatever type the element is if *`arg`* is an array or vector (heterogeneous array.)

Errors

Table 24.2. Errors signalled by aref

SQLState	Error Code	Error Text	Description
22003	SR017	aref: Bad array subscript (zero-based) %d for an arg of type %s (%d) and length %d.	
22023	SR000	aref expects an array or vector, not an arg of type %d.	

Examples

Example 24.13. Simple examples

```
SQL> select aref ('Abacus', 0);
callret
VARCHAR
_______________________________________________________________________________

65

1 Rows. -- 3 msec.
SQL> select aref (vector ('Primero', 2, 3.333), 2);
callret
VARCHAR
_______________________________________________________________________________

3.333

1 Rows. -- 4 msec.
SQL>
```

See Also

`vector aset ascii`

ascii

ascii — Get ASCII value of a character

Synopsis

```
integer ascii ( arg any );
```

Description

`ascii` returns the ASCII value of the first character of a string. If an empty string is given, then zero is returned.

Parameters

arg
: A string

Return Values

The integer ASCII value of the first character of the input string is returned. If the input string is empty, then zero is returned

Errors

Table 24.3. Errors signalled by ascii

SQLState	Error Code	Error Text	Description
22023	SR044	Function ascii needs a string as an argument, not an argument of type %d (= %s)	

Examples

Example 24.14. Simple example

```
SQL> select ascii('Zardoz');
callret
INTEGER
_______________________________________________________________________________

90

1 Rows. -- 14 msec.
```

See Also

`aref`, `chr`

aset

aset — set array element

Synopsis

aset (in *arg* any ,
in *nth* integer ,
in *new_elem* any) ;

Description

aset sets the nth element of a string, array or vector where nth is a zero-based index. If the first argument is a string, the nth character of string is replaced with the ASCII value given in the third argument elem.

Parameters

arg
: A string, array or vector.

nth
: Zero-based element index.

nelem
: The new element. If arg is a string, its nth element will be replaced by the ASCII value given in new_elem .

Return Values

`Aset` returns *`nelem`* . It modifies its first argument.

Errors

Table 24.4. Errors signalled by aset

SQLState	Error Code	Error Text	Description
22003	SR020	Bad array subscript %d in aset.	

Examples

Example 24.15. Using `make_string` and `aref`

Make a string, fill with character sequence from A to Z.

```
SQL> create procedure
alphabet_string ()
{
  declare _inx integer;
  declare _str varchar;
  _str := make_string (26);

  while (_inx < 26)
    {
      aset (_str, _inx, _inx + 65);
      _inx := _inx + 1;
    }

  return (_str);
}
;
```

```
Done. -- 6 msec.
SQL> select alphabet_string ();
callret
VARCHAR NOT NULL
_______________________________________________________________________________

ABCDEFGHIJKLMNOPQRSTUVWXYZ

1 Rows. -- 4 msec.
```

Example 24.16. Reverse a string using aset

Note that str is modified by aset.

```
SQL> create procedure
revstr (in str varchar)
{
    declare len, inx1, inx2, tmp integer;

    if (str is null) return (str);

    len := length (str);
    if (len < 2)
      return (str); -- No need for further processing

    inx1 := 0;       -- Index from the left.
    inx2 := len - 1; -- Index from the right.
    len  := len / 2; -- Upper limit for inx1.

    while (inx1 < len)
      {
        tmp := aref (str, inx1);
        aset (str, inx1, aref (str, inx2));
        aset (str, inx2, tmp);
        inx1 := inx1 + 1;
        inx2 := inx2 - 1;
      }
    return (str);
}
;

Done. -- 7 msec.
SQL> select revstr ('repaid'), revstr ('Alli, tapa pulu papatilla!');
callret   callret
VARCHAR   VARCHAR
_______________________________________________________________________________

diaper    !allitapap ulup apat ,illA

1 Rows. -- 11 msec.
```

See Also

`aref()`,`vector()`,`make_string()`

atof

atof — Convert a string to single precision float

Synopsis

atof (in *arg* string
);

Description

atof returns its argument as a single precision floating point. If the string cannot be parsed and converted to a valid float, a value 0.0 is returned.

Parameters

arg
: A string input parameter

Return Types

Single precision floating point

Examples

Example 24.17. Simple examples

```
SQL> select atof('1.23456789');
callret
REAL
_______________________________________________________________________________

1.234568

1 Rows. -- 5 msec.

SQL> select atof ('Cadena de los patos amarillos');
callret
REAL
_______________________________________________________________________________

0.000000

1 Rows. -- 4 msec.

SQL> select atof (sprintf ('%f', 2.3423));
callret
REAL
_______________________________________________________________________________

2.342300

1 Rows. -- 5 msec.
```

See Also

atoi

atoi

atoi — Convert a string to an integer

Synopsis

atoi (in *arg* string
);

Description

atoi returns its argument as an integer. If the string cannot be parsed and converted to a valid integer, a value 0 is returned.

Parameters

arg
: A string input parameter

Return Types

The string converted to an integer. If the string is a number then the absolute value of the literal will be returned as an integer. Otherwise 0 is returned.

Examples

Example 24.18. Simple examples

```
SQL> select atoi('1.23456789');
callret
INTEGER
_______________________________________________________________________________

1

1 Rows. -- 5 msec.

SQL> select atoi ('Cadena de los patos amarillos');
callret
INTEGER
_______________________________________________________________________________

0

1 Rows. -- 4 msec.
```

See Also

atof

att_local_name

att_local_name — Compose a fully qualified table name based on DSN and remote table name.

Synopsis

```
varchar att_local_name ( in dsn varchar ,
                         in table varchar ) ;
```

Description

The utility function, `att_local_name()` , can be used to make a fully qualified table name from non-qualified or qualified one, i.e. the qualifier and owner will be added if they are missing. The schema name will be replaced with current qualifier on execution, owner will be replaced or added with name of supplied DSN name. All non-alphanumeric characters in the name will be replaced with underscore symbol.

Parameters

dsn
: The name of remote data source.

table
: The name of remote table.

Return Types

A string will be returned containing the fully qualified table name.

Examples

Example 24.19. Using the att_local_name() function

This simple example shows retrieval of fully qualified table name. using the att_local_name() function from ISQL.

```
SQL> select att_local_name ('Oracle', 'DEMO.EMP');
callret
VARCHAR
_______________________________________________________________________________

DB.ORACLE.EMP

1 Rows. -- 9 msec.
```

backup

backup , backup_close , backup_flush , backup_prepare , backup_row — Write data into transaction log format for backup purposes. Deprecated.

Synopsis

backup (in *file* varchar);

backup_close ( );

backup_flush ( );

backup_prepare (in *file* varchar);

backup_row (in *row* any);

Description

Important

These functions are deprecated. The use of backup_online () is preferred for database backups. Performing a backup with the destination file /dev/null is a good way of verifying a database's physical integrity.

All backup files, whether complete (created with `backup`) or partial (created with `backup_prepare` and `backup_row` of selected rows), begin with the complete schema that was effective at the time of the backup.

Backup and log files contain assumptions about the schema and row layout of the database. Hence it is not possible to use these for transferring data between databases. Attempt to do so will result in unpredictable results. Thus a log or backup may only be replayed on the same database, an empty database or a copy of the database which has had no schema changed since it was made. Also, when replaying a backup file onto an empty database, the +replay-crash-dump switch should be given on the executable command line.

This function requires dba privileges.

Full backup
: The backup function takes a file name as argument. The file produced will be in the log format and will recreate the database as it was at the time of the last checkpoint when replayed on an empty database. Such a file cannot be replayed on anything except an empty database. Logs made after the backup can be replayed over the database resulting from the backup file's replay. No schema operations are allowed between replays.

Partial backups
: The backup_prepare , backup_row and backup_close operations allow making specific partial backups.

Parameters

file
: varchar file filename for the generated log.

backup_row
: any row a value of the pseudo column _ROW .

See Also

replay

`backup_online()` , `backup_context_clear()`

backup_online

backup_online — perform online backup of database

Synopsis

```
int backup_online ( in file_prefix varchar ,
                    in pages integer ,
                    in timeout integer ,
                    in dirs any ) ;
```

Description

This procedure will backup all information from the checkpoint space to a series of files named "<file_prefix><n>.bp ", where <n> is the sequence number of the file in the backup series. The first backup will be a full gzip compressed dump of database pages in the checkpoint space. Any subsequent call will only backup pages which have changed since the last backup was made. To start with a fresh full backup, use backup_clear_context to clear the change tracking data. At each checkpoint the checkpoint space will be updated, and the next "backup_online;" procedure will create new files. Once backup_online() has been called for the first time, the arguments supplied will be used for subsequent calls to it. Hence, arguments supplied to this procedure (except the "dirs" argument) will be ignored in subsequent calls.

Before a new backup series can be started, the `backup_context_clear();` procedure must be called first. This procedure will clear the current backup context and mark all pages in the checkpoint space as ready for backup.

A database checkpoint cannot be performed while an online backup is in progress. Attempt to do a checkpoint will wait until the backup is complete.

This is the preferred means of backing up databases and replaces any other prior means. As an alternative, copying database files while the database is running will still work, as long as no checkpoint is made during the copy process.

Parameters

file_prefix
: A string to prefix to the filename of the backup files.

pages
: The pages argument indicates the maximum number of 8K pages that will be backed up into each file. This argument must be larger than 100. If pages < 100 an error will be returned.

timeout
: This parameter has no effect.

dirs
: This optional parameter must be an array of directory names (array of strings). The backup files are first stored in the first directory. When running out of disk any consecutive backup files are stored in the next directory in the list. If there are no more directories then an error is signalled and all the files written by this call to backup_online are deleted so as not to leave half made backups.

Return Types

This function will return the number of 8k pages that were backed-up.

Errors

This function can generate the following errors: IB001 IB002 IB003 IB004 IB005 IB006 IB007 IB008 IB009 IB010

Examples

Example 24.20. Performing an online backup

If there are 2010 new pages in the checkpoint space and user invokes:

```
"backup_online ('dump-20021010_#', 500);"
```

from ISQL, then the following series of backup files will be created in the Virtuoso database directory:

```
dump-20011010_#1.bp
dump-20011010_#2.bp
dump-20011010_#3.bp
dump-20011010_#4.bp
dump-20011010_#5.bp
```

The first 4 files will each contain 500 8K pages. The actual length of the files will vary due to varying compression ratio.

Example 24.21. Restoring an Online Backup

The following command could be used to restore the database from the backup files created:

```
virtuoso-iodbc-t +restore-backup dump-20011010_#
```

or:

```
virtuoso-odbc-t.exe +restore-backup dump-20011010_#
```

See Also

`backup_context_clear();`

backup_context_clear

backup_context_clear — Clears the backup context and marks all pages in checkpoint space as ready for online backup

Synopsis

```
backup_context_clear (  );
```

Description

This procedure must be called before a new backup series can be started, This procedure will clear the current backup context and mark all pages in the checkpoint space as ready for backup.

See Also

`backup_online();`

Virtuoso Server Extension Interface (VSEI) functions

bif_arg , bif_array_arg , bif_array_or_null_arg , bif_bin_arg , bif_double_arg , bif_entity_arg , bif_float_arg , bif_long_arg , bif_long_or_char_arg , bif_string_arg , bif_string_or_null_arg , bif_string_or_wide_or_null_arg , bif_strses_arg — Get parameters for built in function

Synopsis

```
caddr_t bif_arg ( caddr_t * qst ,
                  state_slot_t ** args ,
                  int nth ,
                  char * func ) ;

caddr_t bif_array_arg ( caddr_t * qst ,
                        state_slot_t ** args ,
                        int nth ,
                        char * func ) ;

caddr_t bif_array_or_null_arg ( caddr_t * qst ,
                                state_slot_t ** args ,
                                int nth ,
                                char * func ) ;

caddr_t bif_bin_arg ( caddr_t * qst ,
                      state_slot_t ** args ,
                      int nth ,
                      char * func ) ;

caddr_t bif_double_arg ( caddr_t * qst ,
                         state_slot_t ** args ,
                         int nth ,
                         char * func ) ;

caddr_t bif_entity_arg ( caddr_t * qst ,
                         state_slot_t ** args ,
                         int nth ,
                         char * func ) ;

caddr_t bif_float_arg ( caddr_t * qst ,
                        state_slot_t ** args ,
                        int nth ,
                        char * func ) ;

caddr_t bif_long_arg ( caddr_t * qst ,
                       state_slot_t ** args ,
                       int nth ,
                       char * func ) ;

caddr_t bif_long_or_char_arg ( caddr_t * qst ,
                               state_slot_t ** args ,
```

```
                                int nth ,
                                char * func ) ;

caddr_t bif_string_arg ( state_slot_t ** args ,
                         int nth ,
                         char * func ) ;

caddr_t bif_string_or_null_arg ( caddr_t * qst ,
                                 state_slot_t ** args ,
                                 int nth ,
                                 char * func ) ;

caddr_t bif_string_or_wide_or_null_arg ( caddr_t * qst ,
                                         state_slot_t ** args ,
                                         int nth ,
                                         char * func ) ;

caddr_t bif_strses_arg ( caddr_t * qst ,
                         state_slot_t ** args ,
                         int nth ,
                         char * func ) ;
```

Description

The bif_...._arg functions are used in the Virtuoso shared object. They allow customization of the Virtuoso server. They are written in C code, and linked with the Virtuoso shared object.

The _or_null_ variants of these functions represent the SQL null value as a 00 pointer, even though getting the same argument with bif_arg() would return a box with the DV_DB_NULL tag. All number functions will coerce other types of numbers to the result type. All array argument functions will accept any type of array. It is up to the VSE itself to distinguish.

Virtuoso Server Extensions (VSEs) were formally referred to as Built-In Functions (BIFs).

Parameters

qst
: Query instance.

args
: input arguments.

nth
: which argument to retrieve (zero indexed).

func
: name of calling function. This will be printed in the SQL error message.

Return Values

On success, return a valid pointer to the data retrieved, or NULL. A SQL error is generated if the datatype is not valid. The bif_arg is the only function not to have validation of a data type.

The values returned by any of these belong to the query instance, and the bif is not authorized to modify or free them.

Table 24.5. Return types

Function	Type validated and returned
bif_arg	Any type of data. (no data type check)
bif_array_arg	string or array data
bif_array_or_null_arg	string or array or NULL data
bif_bin_arg	binary data
bif_double_arg	double data
bif_entity_arg	XML entity data
bif_float_arg	float data
bif_long_arg	long data
bif_long_or_char_arg	long or character data
bif_string_arg	string data
bif_string_or_null_arg	string or null data
bif_string_or_wide_or_null_arg	strings, widestrings or null data
bif_strses_arg	string session data

Examples

Example 24.22. Creating a function my_length()

A function called `my_length` written by developers, may process a string argument by making a call to `bif_string_arg` .

bit_and

bit_and — Returns bitwise AND of two 32-bit integers.

Synopsis

bit_and (in *n1* integer ,
in *n2* integer) ;

Description

The function returns bitwise AND of two given integers.

On 64-bit platforms, both arguments are intentionally truncated to 32 bits to maintain compatibility.

Return Types

Integer.

Examples

Example 24.23. Using bitwise-AND

```
select bit_and (18, 6);
2

select bit_and (-18, -6);
-22
```

See Also

`bit_not()`

`bit_or()`

`bit_shift()`

`bit_xor()`

bit_or

bit_or — Returns bitwise OR of two 32-bit integers.

Synopsis

bit_or (in *n1* integer ,
in *n2* integer) ;

Description

The function returns bitwise OR of two given integers.

On 64-bit platforms, both arguments are intentionally truncated to 32 bits to maintain compatibility.

Return Types

Integer.

Examples

Example 24.24. Using bitwise-OR

```
select bit_or (18, 6);
22

select bit_or (-18, -6);
-2
```

See Also

`bit_and()`

`bit_not()`

`bit_shift()`

`bit_xor()`

bit_not

bit_not — Returns bitwise NOT of a 32-bit integer.

Synopsis

bit_not (in *n1* integer);

Description

The function returns bitwise NOT of a given integer.

On 64-bit platforms the argument is intentionally truncated to 32 bits to maintain compatibility.

Return Types

Integer.

Examples

Example 24.25. Using bitwise-NOT

```
select bit_not (18);
-19

select bit_not (-18);
-17
```

See Also

`bit_and()`

`bit_or()`

`bit_shift()`

`bit_xor()`

bit_shift

bit_shift — Returns the result of bitwise shift operation over two 32-bit integers.

Synopsis

bit_shift (in *value* integer ,
in *distance* integer) ;

Description

The function returns bitwise shift of two given integers. Depending on the arguments, the shift may be left or right. For right-shift, leftmost bits of the result are filled by the value of the 31-st bit.

On 64-bit platforms, both arguments are intentionally truncated to 32 bits and the shift is restricted to 32 bits to maintain compatibility.

Parameters

value
: The value to be shifted.

distance
: The sign of the parameter specifies the direction of the shift: positive values indicate shift to the left, negative values indicate shift to the right. The absolute value of the parameter specifies the number of bits to shift. The value of zero means that the result is equal to the value of the first argument.

Return Types

Integer.

Examples

Example 24.26. Bitwise-shifting

```
select bit_shift (18, 6);
1152

select bit_shift (-18, 6);
-1152

select bit_shift (1152, -6);
18

select bit_shift (-1152, -6);
-18
```

See Also

`bit_and`

`bit_or`

`bit_not`

`bit_xor`

bit_xor

bit_xor — Returns bitwise XOR (exclusive "or") of two 32-bit integers.

Synopsis

bit_xor (in *n1* integer ,
in *n2* integer) ;

Description

The function returns bitwise XOR (exclusive "or") of two given integers.

On 64-bit platforms, both arguments are intentionally truncated to 32 bits, to provide compartibility.

Return Types

Integer.

Examples

Example 24.27. Bitwise-XOR

```
select bit_xor (18, 6);
20

select bit_xor (-18, -6);
20

select bit_xor (-18, 6);
-24
```

See Also

`bit_and()`

`bit_or()`

`bit_not()`

`bit_shift()`

blob_to_string

blob_to_string — Convert a blob to string

Synopsis

```
varchar blob_to_string (     in blob any
);
```

Description

Although primarily used for converting blobs (long varbinary , long varchar) to string , `blob_to_string` may also be used to convert from wide string , persistent XML (XPER) and string_output streams . If the data being converted is longer than maximum length of a string , `blob_to_string` will signal an error.

Note

This function is equivalent to

```
cast (x as varchar)
```

. Using `cast` is preferred.

Parameters

blob
: blob handle (long varbinary or long varchar), string_output or XPER (persistent XML)

Return Types

Varchar contents of *`blob`* .

Errors

Table 24.6. Errors signalled by `blob_to_string`

SQLState	Error Code	Error Message	Description
22023	SR068	XML tree cannot be used as argument of blob_to_string	
22001	SR069	Attempt to convert a persistent XML document longer than VARCHAR maximum in blob_to_string	
22023	SR070	blob_to_string requires a blob as argument	
22023	SR071	Blob argument to blob_to_string must be a non-interactive blob	
22001	SR072	Blob too long for the string.	

Examples

Example 24.28. Various conversions

```
SQL> use Demo;

Done. -- 20 msec.
SQL> select blob_to_string (Description) from Categories;
callret
VARCHAR
_______________________________________________________________________________

Soft drinks, coffees, teas, beers, and ales
Sweet and savory sauces, relishes, spreads, and seasonings
Desserts, candies, and sweet breads
```

```
Cheeses
Breads, crackers, pasta, and cereal
Prepared meats
Dried fruit and bean curd
Seaweed and fish

8 Rows. -- 37 msec.
SQL>
```

See Also

Casting , `string_output` .

blob_to_string_output

blob_to_string_output — Convert a blob to string session

Synopsis

`varchar` **`blob_to_string_output`** `(` in *blob* any `);`

Description

Although primarily used for converting blobs (long varbinary , long varchar) to string output object , `blob_to_string_output` may also be used to convert from wide string , persistent XML (XPER) and string_output streams .

Parameters

blob
: blob handle (long varbinary or long varchar), string_output or XPER (persistent XML)

Return types

string output object contents of *`blob`* .

Errors

Table 24.7. Errors signalled by

SQLState	Error Code	Error Text	Description
22023	SR068	XML tree cannot be used as argument of blob_to_string_output	
22023	SR070	blob_to_string_output requires a blob or string argument	
22023	SR071	Blob argument to blob_to_string_output must be a non-interactive blob	

Examples

Example 24.29. Various conversions

```
SQL> use Demo;

Done. -- 20 msec.
SQL> select string_output_string (blob_to_string_output ("Description")) from "Categories";
callret
VARCHAR
_______________________________________________________________________________

Soft drinks, coffees, teas, beers, and ales
Sweet and savory sauces, relishes, spreads, and seasonings
Desserts, candies, and sweet breads
Cheeses
Breads, crackers, pasta, and cereal
Prepared meats
Dried fruit and bean curd
Seaweed and fish

8 Rows. -- 37 msec.
SQL>
```

See Also

`blob_to_string`

bookmark

bookmark — Return the bookmark for current row of a scrollable cursor

Synopsis

```
any bookmark ( in cursor CURSOR );
```

Description

`bookmark` returns a bookmark for the current row of an open scrollable cursor. Given an invalid argument, i.e. no cursor, no current row or non-open cursor, it signals an error. The returned value can be used in subsequent FETCH .. BOOKMARK over the same cursor.

Parameters

bookmark
: Given Bookmark.

cursor
: Open scrollable cursor.

Return Types

`integer` bookmark id.

Errors

Table 24.8. Errors signalled by

SQLState	Error Code	Error Text	Description
24000	SR238	Virtuoso/PL Scrollable cursor not opened	
HY109	SR239	Virtuoso/PL Scrollable cursor not positioned on a row	

See Also

Cursors in Virtuoso/PL guide

ceiling

ceiling — Round a number to positive infinity.

Synopsis

ceiling (in *x* double precision) ;

Description

`ceiling` calculates the smallest integer greater than or equal to *x* .

Parameters

x
: double precision

Return Values

`ceiling` returns a 32-bit integer .

Examples

Example 24.30. Simple Examples

```
SQL> select ceiling (12.3456), ceiling (-12.3456), ceiling (0.513513);
callret   callret   callret
INTEGER   INTEGER   INTEGER
_______________________________________________________________________________

13        -12       1

1 Rows. -- 4 msec.
```

See Also

`floor`

cfg_item_count

cfg_item_count — return number of items in a section in configuration file

Synopsis

cfg_item_count (in *path* varchar ,
in *section* varchar) ;

Description

Return the number of items that exist in the specified section of the INI file.

Parameters

path
: Name of the INI file.

section
: Name of the section in the INI file.

Return Values

An integer containing the number of items that exist in the section.

Examples

Example 24.31. Simple examples

Find number of items in the [Parameters] section of the current virtuoso.ini file.

```
SQL> select cfg_item_count(virtuoso_ini_path(), 'Parameters');
callret
INTEGER
_______________________________________________________________

19
```

See Also

`virtuoso_ini_path cfg_section_count cfg_section_name cfg_item_name cfg_item_value cfg_write`

cfg_item_name

cfg_item_name — get nth item name from ini file

Synopsis

cfg_item_name (in *path* varchar ,
in *section* varchar ,
in *item_index* integer) ;

Description

Returns the name of the item specified by item_index (begins from zero). If the index and section name do not point to a valid item, then zero is returned, otherwise on success the function returns the item name.

Parameters

path
: Name of the INI file.

section
: Name of the section in the INI file.

item_index
: Zero based index to the item within the section to be listed.

Return Values

A varchar containing the name of item referenced by *section* and *item_index*.

Examples

Example 24.32. Simple examples

Display the first keyname in the [Parameters] section of the current virtuoso.ini file.

```
SQL> select cfg_item_name(virtuoso_ini_path(), 'Parameters',0);
callret
VARCHAR
_______________________________________________

ServerPort
```

See Also

virtuoso_ini_path cfg_section_count cfg_section_name cfg_item_count cfg_item_value cfg_write

cfg_item_value

cfg_item_value — returns the value of an item from the ini file

Synopsis

```
cfg_item_value ( in path varchar ,
                 in section varchar ,
                 in item_name varchar ) ;
```

Description

Return the value of an item identified by *item_name* and *section* parameters from the specified INI file.

Parameters

path
: Name of the INI file.

section
: Name of the section in the INI file.

item_name
: Name of the item in the section.

Return Values

A varchar containing the item value. Return zero if section or item_name is invalid.

Examples

Example 24.33. Simple examples

Read the value of the ServerPort item from the [Parameters] section of the current virtuoso.ini file.

```
SQL> select cfg_item_value(virtuoso_ini_path(), 'Parameters','ServerPort');
callret
VARCHAR
_______________________________________________________________________________

1112
```

See Also

virtuoso_ini_path cfg_section_count cfg_section_name cfg_item_count cfg_item_name cfg_write

cfg_section_count

cfg_section_count — get number of sections in an INI file

Synopsis

cfg_section_count (in *path* varchar) ;

Description

Returns the number of sections in an INI file.

Parameters

path
: Name of the INI file.

Return Values

An integer containing the number of sections in the INI file.

Examples

Example 24.34. Simple examples

Find out how many sections are in the current virtuoso.ini file.

```
SQL> select cfg_section_count(virtuoso_ini_path());
callret
INTEGER
_______________________________________________________

7
```

See Also

`virtuoso_ini_path cfg_section_name cfg_item_count cfg_item_name cfg_item_value cfg_write`

cfg_section_name

cfg_section_name — returns INI file section name

Synopsis

```
cfg_section_name ( in path varchar ,
                   in index integer ) ;
```

Description

Returns the name of section specified by the index (begins from zero). If the index can reference a section, the that section name is returned, otherwise returns zero on error.

Parameters

path
: Name of the INI file.

index
: Zero based index that references a section.

Return Values

An varchar containing the name of section.

Examples

Example 24.35. Simple examples

Get the second section name from the current virtuoso.ini file.

```
SQL> select cfg_section_name(virtuoso_ini_path(), 1);
callret
VARCHAR
_______________________________________________________

Parameters
```

See Also

`virtuoso_ini_path cfg_section_count cfg_item_count cfg_item_name cfg_item_value cfg_write`

cfg_write

cfg_write — Writes the item=value to an INI file

Synopsis

cfg_write (in *path* varchar ,
in *section* varchar ,
in *item_name* varchar ,
in *item_value* varchar) ;

Description

This function requires dba privileges.

This function allows modification of existing entries, or update updating existing items in an INI file.

Parameters

path
: Name of the INI file.

section
: Name of the section in the INI file.

item_name
: Name of item that will be assigned the item_value.

item_value
: Value to be assigned to the item_name.

Examples

Example 24.36. Simple examples

Write ServerPort=2222 under the [Parameters] section of the current virtuoso.ini file.

```
SQL> cfg_write(virtuoso_ini_path(), 'Parameters','ServerPort','2222');

-- Verification
SQL> select cfg_item_value(virtuoso_ini_path(), 'Parameters','ServerPort');
callret
VARCHAR
_______________________________________________________________________________

2222
```

See Also

`virtuoso_ini_path cfg_section_count cfg_section_name cfg_item_count cfg_item_name cfg_item_value`

charset_define

charset_define — Define a character set.

Synopsis

charset_define (in *name* varchar ,
in *charset_string* any ,
in *aliases* any) ;

Description

This function creates a new narrow language-specific character set, or redefines an existing one.

Parameters

name
: The name of the character set to define. This becomes the "preferred" name of the character set.

charset_string
: Wide string with the character codes for each given character from 1 to 255. That is, a 255-byte long NCHAR defining the Unicode codes for narrow chars 1-255.

aliases
: Vector of character set names that are to be aliases of the character set being defined. Use NULL if there are to be no aliases.

Return Types

None

Errors

Table 24.9. Errors signalled by

sqlstate	error code	error text
2C000	IN001	The UTF-8 is not a redefinable charset
2C000	IN002	charset_define : Charset table not a wide string
2C000	IN003	charset_define : 0 not allowed as a charset definition
2C000	IN004	charset <name> already defined. Drop it first
2C000	IN005	Alias <position> is not of type STRING

Examples

Example 24.37. New 4th character, and aliases

To setup the 4th character to be Unicode 0xffce, and give the character set 2 aliases, you would make a call like this:

```
charset_define ('NEW-CHARSET',
      N'\x1\x2\x3\xffce\x5....' , vector ('SOME-CHARSET', 'ANOTHER-CHARSET'));
```

The character string should contain 255 wide characters that are arbitrary unicode values.

See Also

`charsets_list()`

charset_recode

charset_recode — Translate a string to another character set

Synopsis

```
any charset_recode ( in src_string varchar/nvarchar ,
                     in src_charset varchar ,
                     in dst_charset varchar ) ;
```

Description

This function translates a string from a given source charset to a destination charset. It provides a generic way of recoding string entities.

The *src_charset* may be a narrow or a wide string . If it's a narrow string (VARCHAR) then the *src_charset* is taken into account and defines the current encoding of the *src_string* . In any other case *src_charset* is ignored.

src_charset and *dst_charset* are names of system-defined 8 bit charset tables. Use charsets_list to obtain a list of currently defined character sets and aliases. If either of these is null, then the charset in effect is used. There are two special character set names - "UTF-8" and "_WIDE_" - that are recognized by this function. These represent UTF-8 encoding of characters and wide string (NVARCHAR).

Parameters

src_string
: The input data to be converted. String or wide string.

src_charset
: Input data character set, string .

dst_charset
: The charset to convert to, string .

Examples

Example 24.38. Recoding a narrow ISO-8859-1 string as UTF-8

```
select cast (charset_recode ('\xA9', 'ISO-8859-1', 'UTF-8') as varbinary)
  -- converts "Copyright sign" to UTF-8 (output 0xC2A9)
select cast (charset_recode ('\xC0', 'WINDOWS-1251', 'ISO-8859-5') as varbinary)
  -- converts "Cyrillic A" from WINDOWS-1251 charset to ISO-8859-5 (output 0xB0).
select cast (charset_recode (N'\x410', '_WIDE_', 'WINDOWS-1251') as varbinary)
  -- converts "Cyrillic A" from Unicode to WINDOWS-1251 charset (result '\xC0').
select charset_recode (N'\x410', '_WIDE_', 'ISO-8859-1')
  -- converts "Cyrillic A" from Unicode to ISO-8859-1 charset (Not available : result '?').
```

See Also

elh_get_handler

elh_load_handler

lh_get_handler

lh_load_handler

charsets_list

charsets_list — List known character set names and aliases.

Synopsis

any **charsets_list** (in *gen_res_set* integer);

Description

This function produces a list of all character set names and aliases known to Virtuoso. The returned value is an array of string s with a character set name as each element. If the *gen_res_set* flag is 1, the function also produces a result set in which each row contains one varchar column with a name of a character set or alias.

Parameters

gen_res_set
: Integer flag to determine whether to produce a result set: 0 means no, 1 means yes.

Return Types

An array of string , optionally generates a result set of single varchar columns.

Errors

This function can generate the following errors: SR008 SR001

Examples

Example 24.39. List character sets as a result set

```
SQL> charsets_list(1);
CS_NAME
VARCHAR
_______________________________________________

437
819
850
855
866
874
999
CP1250
CP1251
CP1252
CP1257
CP437
....
```

There are 132 predefined character sets in total that would be listed.

Example 24.40. Get first 2 character set names/aliases

```
....
x:= charsets_list (0);
y := aref (x, 0); -- will be '437'
y := aref (x, 1); -- will be '819'
....
```

See Also

`charset_define()`

checkpoint_interval

checkpoint_interval — Configure database checkpointing

Synopsis

```
integer checkpoint_interval ( in minutes integer );
```

Description

This function changes the database checkpointing interval to the given value in minutes. It may also be used to disable checkpointing in two ways: By setting checkpoint interval to 0, the checkpoint will only be performed after roll forward upon database startup. A setting of -1 will disable all checkpointing. Main use for this function is to ensure a clean online backup of the database slices. Copying of the database may take long and checkpointing would modify those files in mid-copy, thus rendering the resulting copy unusable. In case the system should, for some reason or another, become unstable, it is sometimes better to disable checkpointing after a database restart to resume backing up from where it was left prior to a system crash. Disabling all checkpointing by giving checkpoint_interval the value of -1 will do just that.

The interval setting will be saved in the server configuration file as value of CheckpointInterval in section [Parameters], thus it will persist over consecutive server shutdown/restart cycles. A long checkpoint_interval setting will produce longer transaction logs, which in turn prolongs the time it takes for the database to perform a roll forward upon restart in case it was shut down without making a checkpoint.

Parameters

minutes
: integer number of minutes between checkpoints.

Return Types

Previous value of CheckpointInterval in the configuration file as an integer.

Errors

Parameter data type checking errors

Examples

Example 24.41. Simple examples

Disable checkpoints:

```
SQL> checkpoint_interval(-1);

Done. -- 25 msec.
```

Re-enable checkpoints (every 2 hrs):

```
SQL> checkpoint_interval(120);
callret
VARCHAR
_______________________________________________________________

-1

Done. -- 4 msec.
```

See Also

`backup`

CheckpointInterval setting in Configuring Server Startup Files section

chr

chr — Convert a long character code to a character or wide character

Synopsis

```
varchar chr ( in chr_code long );
```

Description

`chr` returns a new one character long string containing the character with character code given as parameter.

Parameters

chr_code
: The LONG character code value for the character or wide character to be produced.

Return Values

If the *chr_code* is smaller than or equal to 255, the returned 1 character string will be of type VARCHAR . Otherwise the returned type is NVARCHAR .

Errors

Table 24.10. Errors signalled by chr

Examples

Example 24.42. Simple example

```
SQL> select chr (33);
callret
VARCHAR
_______________________________________________________________________________

!

1 Rows. -- 3 msec.
```

Example 24.43. Stored procedure example

This stored procedure lists the ASCII values and characters in a string given as argument.

```
SQL> create procedure
test_chr (in str varchar)
{
  declare pos integer; pos := 0;
  declare c_code, c_code2 integer; declare c_char varchar;

  result_names (c_code, c_code2, c_char);

  while (pos < length (str))
    {
      result (aref (str, pos), ascii (subseq (str, pos, pos+1, 1)),
        chr (aref (str, pos)));
      pos := pos + 1;
    }
}
```

```
;

Done. -- 7 msec.
SQL> test_chr ('Nebuchadnessar');
c_code             c_code2            c_char
INTEGER NOT NULL   INTEGER NOT NULL   VARCHAR NOT NULL
_____________________________________________________________________________

78                 78                 N
101                101                e
98                 98                 b
117                117                u
99                 99                 c
104                104                h
97                 97                 a
100                100                d
110                110                n
101                101                e
115                115                s
115                115                s
97                 97                 a
114                114                r

14 Rows. -- 7 msec.
```

See Also

aref, ascii

client_attr

client_attr — returns a varchar containing the requested information from the connection

Synopsis

```
varchar client_attr ( in attr varchar );
```

Parameters

attr
: can by one of the following:

Errors

Table 24.11. Errors signalled by `client_attr`

SQLState	Error Code	Error Text
22005	SR401	Server address not known.
22005	SR402	Cannot allocate temp space. SSL error : xxx
22005	SR403	xxx is not valid client_attr option.

collation_define

collation_define — define a new collation

Synopsis

```
collation_define ( in COLLATION_NAME varchar ,
                   in FILE_PATH varchar ,
                   in ADD_TYPE integer ) ;
```

Description

This function lets you define a new collation.

Parameters

COLLATION_NAME
: The name to be assigned to the new collation.

FILE_PATH
: The path and file name of the collation definition file. The path must be specified in the format appropriate to the operating system hosting the Virtuoso server.

ADD_TYPE
: is the type of the new collation: 1 for 8-bit collation (256-byte blob); 2 for UNICODE collation (65536-byte Unicode blob). A value of 0 instructs the function only to check the validity of the definition file and to return a resultset containing the codes of the valid exception definitions.

Return Types

If 0 is supplied to the ADD_TYPE parameter then a result set will be returned containing the codes of the valid definitions.

complete_table_name

complete_table_name — Returns a fully qualified table name.

Synopsis

```
varchar complete_table_name ( in tablename varchar ,
                              in mode integer ) ;
```

Description

The `complete_table_name()` can be used to make a fully qualified table name from non-qualified one, i.e. the qualifier and owner will be added if they are missing.

Parameters

tablename
: The table name to be retrieved.

mode
: If this mode parameter is set to 1 this function will first look-up the supplied tablename for a match in the system tables. If a match is found the full name will be returned, if the table is not found the function will continue as if the mode were set to 0.

Return Types

A string will be returned containing the fully qualified table name.

Errors

Table 24.12. Errors signalled by

SQLState	Error Code	Error Text	Description
22023	SR014	Function complete_table_name needs a string as argument 1, not an arg of type [type]	
22023	SR008	Function complete_table_name needs an integer as argument 2, not an arg of type [type]	

Examples

Example 24.44. Using the complete_table_name() function

This simple example shows retrieval of a fully qualified table name using the complete_table_name() function from ISQL. In the first case the table name supplied is a system table, in the second case the table does not exist and the result is generated based on the users details:

```
SQL> use WS;

Done. -- 4 msec.
SQL> select complete_table_name('SYS_DAV_RES', 1);
callret
VARCHAR
_______________________________________________________________________________

WS.WS.SYS_DAV_RES

1 Rows. -- 1 msec.

SQL> select complete_table_name('SYS_DAV_RES__', 1);
callret
VARCHAR
_______________________________________________________________________________
```

```
WS.DBA.SYS_DAV_RES__

1 Rows. -- 1 msec.
```

composite

composite — create a composite object

Synopsis

```
composite obj composite (  in x any
                           ,
                           ... );
```

Description

Create a composite object

Returns a composite object containing the serialization of each argument. The total serialized length of the arguments may not exceed 255.

Parameters

x ...
: The function takes a variable number of parameters of any type.

Return Values

A composite object

Errors

Table 24.13. Errors signalled by composite

SQLState	Error Code	Error Text	Description
22026	FT001	Length limit of composite exceeded.	The sum of length of the components exceeds 255.

Examples

Example 24.45. Simple example

Get second component of a composite of two integers. See reference page for composite_ref .

```
SQL> select composite_ref (composite (10,20), 1);
callret
VARCHAR
_______________________________________________________________________________

20

1 Rows. -- 6 msec.
```

See Also

`composite_ref` . The chapter Composite Data Type for a definition of properties of composite objects.

composite_ref

composite_ref — get member of a composite object

Synopsis

```
integer composite_ref ( in c any ,
                        in nth integer ) ;
```

Description

`composite_ref` returns the *`nth`* element of the composite. The index is 0 based.

Parameters

nth
: integer

Return types

The type returned is the type of the composite member referred to.

Errors

Table 24.14. Errors signalled by `composite_ref`

22023	FT002	composite expected for composite ref	
22003	FT003	composite index out of range %d for length %d	

Examples

Example 24.46. Simple example

Get first member of a composite consisting of VARCHAR values.

```
SQL> select composite_ref (composite ('Miles','Herbie','Wayne','Ron','Tony'), 0);
callret
VARCHAR
_______________________________________________________________________________

Miles

1 Rows. -- 4 msec.
```

See Also

The Composite Data Type for a definition of the properties of composite objects.

The function `composite` .

concat

concat — Concatenate strings

Synopsis

concat (*str1* varchar ,
str2 varchar ,
... ,
strn varchar) ;

Description

`concat` returns a new string , concatenated from a variable number of strings given as arguments. NULL arguments are handled as empty strings.

```
concat (str)
```

returns a copy of *str* .

```
concat ()
```

returns an empty string .

Parameters

Concat takes a variable number of string arguments.

Return types

The `concat` returns a single string . If there are wide strings amongst the arguments, the string returned will also be wide .

Errors

Table 24.15. Errors signalled by concat()

sqlstate	error code	error text
22023	SR007	Function concat needs a string or NULL as argument <argument number>, not an arg of type <offending type>

Examples

Example 24.47. Simple Example

Cross a mule with an ass.

```
SQL> select concat ('Muuli', 'aasi');
callret
VARCHAR
_______________________________________________________________________________

Muuliaasi

1 Rows. -- 4 msec.
```

concatenate

concatenate — concatenate strings

Synopsis

```
string concatenate ( in arg_1 any ,
                     ... );
```

Description

`Concatenate` is an alias of `concat` .

See Also

`concat`

connection_get

connection_get — Get connection variable

Synopsis

```
any connection_get ( in name varchar );
```

Description

`connection_get` is used to retrieve values stored within the current connection context. See `connection_set` for a more detailed discussion of connection variables.

Parameters

name
: Name of the connection variable

Return Values

A value associated to the connection by connection_set in the datatype of the original value. If the variable does not exist, a NULL is returned.

Examples

Example 24.48. Setting and Retrieving Connection Variables

```
SQL> select connection_get('var_demo');
callret
VARCHAR
_______________________________________________________________________

NULL

SQL> connection_set('var_demo', 'some arb data');

SQL> select connection_get('var_demo');
callret
VARCHAR
_______________________________________________________________________

some arb data
```

In a VSP page...

```
<?vsp
  declare id integer;
  declare saved_req any;
  id := connection_get ('ID');
  saved_req := connection_set ('saved_request');
?>
```

See Also

connection_set

connection_id

connection_id — get connection identifier

Synopsis

```
connection_id (   );
```

Description

This function returns a string uniquely identifying the connection in this server instance. It is usually a combination of server's port number and a serial number of the client.

Note

The value returned is usually not useful in HTTP invocation context (VSP or SOAP), since consecutive requests by the same client will typically not be on the same connection the way the server sees it.

Return Values

A VARCHAR connection identifier.

Examples

Example 24.49. Return The Current Connection ID

```
SQL> select connection_id();
callret
VARCHAR
_______________________________

1111:4
```

connection_is_dirty

connection_is_dirty — check if current session connection variables have been altered

Synopsis

```
integer connection_is_dirty ( );
```

Description

This function is used to determine if the session variables have changed between a call to `connection_vars_set` and current point of execution. A call to connection_vars_set will cause subsequent calls to `connection_is_dirty` to return true.

The function is useful in postprocessing functions for making conditional storage of session variables in a database table.

Return Values

An INTEGER indicating true (1) or false (0).

Examples

Example 24.50. Checking Connection Variables

```
<?vsp
  declare rc integer;
  connection_vars_set ();
  rc := connection_is_dirty (); -- rc is equal to 0
  connection_set ('ID', 14);
  rc := connection_is_dirty (); -- rc is equal to 1
?>
```

See Also

connection_get

connection_set

connection_vars

connection_vars_set

connection_set

connection_set — Associates a value to the name in the context of the present connection

Synopsis

connection_set (in *name* varchar ,
in *value* any) ;

Description

This associates a value to the name in the context of present connection. The name should be a string and the value can be any data type except blob, open cursor or an XML entity. If the value is an array it may not contain the restricted types.

Connection variable setting is not logged and information stored will be lost when the connection is closed. The value can be retrieved by any future statement executing within the same connection. Connection variables can be used as a global variable mechanism for stored procedures, the scope being the client connection.

In the case of VSP or SOAP this mechanism cannot be used to pass information between requests by the same client. It will however, be useful for having 'global variables' between procedures called within the same HTTP request. Note that this mechanism is used to provide persistent HTTP session variables in some cases but this works through special before and after code which stores the values set with this function into an external session structure. In this sense this function itself has nothing to do with web session management although it can be used as a component for such.

Parameters

name
: VARCHAR name to associate the value with.

value
: value to be stored. May be any data type except LOB , open cursor or XML entity. If the value is an array , it may not contain the restricted types.

Return Values

None

Errors

Table 24.16. Errors signalled by connection_set

SQLState	Error Code	Error Text	Description
22023	SR049	Data type is not suitable for storage into a global variable (connection_set)	

Examples

Example 24.51. Setting and Retrieving Connection Variables

From SQL

```
SQL> select connection_get('var_demo');
callret
VARCHAR
_______________________________________________________________________

NULL

SQL> connection_set('var_demo', 'some arb data');
```

```
SQL> select connection_get('var_demo');
callret
VARCHAR
_______________________________________________________________________

some arb data
```

From a VSP page

```
<?vsp
  declare id integer;
  id := 12;
  connection_set ('ID', id);
  connection_set ('saved_request' , lines);
?>
```

See Also

connection_get

connection_is_dirty

connection_vars_set

connection_vars

connection_vars — Retrieve all connection variables

Synopsis

```
any connection_vars ( );
```

Description

This function returns all stored session variables in an array with name/value pairs.

Connection variables do not persist across sessions, one may maintain persistence of variables by storing them in a database table: see the Session Variables Section.

Return Types

Array of name-value pairs of all connection variables.

Examples

Example 24.52. Retrieving Connection Variables

```
<?vsp
  declare ses_vars any;
  ses_vars := connection_vars ();
  update session set s_vars = serialize (ses_vars) where s_id = get_keyword ('sid', params);
?>
```

See Also

connection_get

connection_set

connection_vars_set

connection_vars_set

connection_vars_set — set all connection variables

Synopsis

connection_vars_set (in *var_array* any) ;

Description

This function clears all connection variables for current session and sets new ones defined in the array passed as parameter.

Connection variables do not persist across sessions, one may maintain persistence of variables by storing them in a database table, as discussed in Session Variables -section.

Parameters

var_array
: An array of even number of elements, containing name-value pairs. NULL, will cause all connection variables for current connection to be erased.

Examples

Example 24.53. Usage

```
<?vsp
  connection_vars_set (vector ('ID', 12));
?>
```

Example 24.54. Setting and Retrieving Connection Variables

```
SQL>connection_vars_set(vector('v1', 'a', 'v2', 1));

Done. -- 0 msec.
SQL> select connection_vars();
callret
VARCHAR
_______________________________________________________________________________

vector(0x004e419c,0x004e40c4,0x004dec9c,1)

1 Rows. -- 70 msec.
SQL>select aref(connection_vars(),0),
      aref(connection_vars(), 1),
      aref(connection_vars(), 2),
      aref(connection_vars(), 3);

callret                  callret                  callret                  callret
VARCHAR                  VARCHAR                  VARCHAR                  VARCHAR
_______________________________________________________________________________

v1                       a                        v2                       1

1 Rows. -- 80 msec.
```

See Also

connection_get

connection_set

connection_vars

contains

contains — A text contains predicate

Synopsis

```
boolean contains ( column varchar ,
                   expression varchar ,
                   opt_or_value integer ,
                   ...) ;
```

Description

This is a SQL predicate that specifies a condition on a column on which a free text index exists. The expression is a string matching the grammar of a text search expression. This is computed for each evaluation of the contains predicate and does not have to be a constant. For example a parameter or variable of a containing score (e.g. procedure) is accepted.

The *score_limit* is optional. If specified, it should be a numeric expression determining the minimum score required to produce a hit.

A virtual column named 'SCORE' is available in queries involving a `contains` predicate. This can for example be returned in a result set or used for sorting. Note that the name is in upper case and is case sensitive in all case modes.

Parameters

column
: The table column whose contents are free text indexed

expression
: A string matching the grammar of a text search expression.

opt_or_value
: May be one or more of the following: DESCENDING specifies that the search will produce the hit with the greatest id first, as defined by integer or composite collation.START_ID ',' scalar_exp the first allowed document id to be selected by the expression in its traversal order, e.g. least or equal for ascending and greatest or equal for descending.END_ID ',' scalar_exp the last allowed id in the traversal order. For descending order the START_ID must be >= END_ID for hits to be able to exist. For ascending order the START_ID must be <= END_ID for hits to be able to exist.SCORE_LIMIT ',' scalar_exp Minimum score that hits must have or exceed to be considered matches of the predicate.RANGES ',' scalar_exp specifies that the query variable following the RANGES keyword will be bound to the word position ranges of the hits of the expression inside the document. The variable is in scope inside the enclosing SELECT statement. OFFBAND ',' column Specifies that the following column will be retrieved from the free text index instead of the actual table. For this to be possible the column must have been declared as offband with the CLUSTERED WITH option of the CREATE TEXT INDEX statement.

Return Types

The contains is a predicate, therefore returning a boolean .

Examples

Example 24.55. Querying Free Text Indexed Columns using contains()

- Return the number of documents with one or more occurrences of "virtual" immediately followed by "database".

```
select count (*) from docs
where contains (text, '"virtual database"')
```

- Specify documents with performance and either 'tuning' or optimization' in any respective positions.

```
'performance and (tuning or optimization)'
```

- Match documents with the word graphics more than 100 words away from 'user' or 'interface'.

```
'graphics and not (graphics near user near interface)'
```

- Match documents with SQL followed by a word beginning with 'interfac'.

```
'"sql interfac*"'
```

- Match documents with words beginning with 'dragon' and not containing the phrase 'once upon a time'.

```
'"dragon*" and not "once upon a time"'
```

Note:
An expression may not consist of all negative terms, e.g. (not a) and (not b) is not a valid expression but 'c and not a and not b' is a valid expression.

Note that the NEAR connective may not be used between AND'ed or OR'ed terms. It can be used to combine words or phrases.

See Also

`xpath_contains()`

The Free Text Index Chapter.

Other Free Text Functions.

Free Text Triggers.

cov_load

cov_load — Load test coverage data from file.

Synopsis

cov_load (in *fname* varchar) ;

Description

This function accepts one argument with name of file containing test coverage data. The expected format of the file is described in the Branch Coverage section. The loaded file is merged with the data already collected by Virtuoso. More than one file can be loaded. The ultimate report will be the union of all files.

Parameters

fname
: Name of coverage file to load.

Return Types

None.

See Also

`cov_store()`

`cov_report()`

`pldbg_stats()`

`pldbg_stats_load()`

cov_report

cov_report — Produce a text coverage report.

Synopsis

```
cov_report ( in fname varchar ,
             in outdir varchar ) ;
```

Description

This function is used to produce a coverage report in text format. The format of the file is described in the Branch Coverage section. This function takes a filename 'fname' as a path to the extended coverage report file and 'outdir' as a path to a directory where .cov files will be stored. The profile.prof file in the output directory is a summary of function call counts and execution times, once ranked by self time, once by time spent inside the function and functions called from there. Note that directory must exists before calling this function.

Parameters

fname
: Name of coverage report file.

outdir
: Destination directory for storing .cov coverage files.

Return Types

None.

See Also

`cov_load()`

`cov_store()`

`pldbg_stats()`

`pldbg_stats_load()`

cov_store

cov_store — Writes a test coverage to a file.

Synopsis

cov_store (in *fname* varchar ,
in *add_comments* integer) ;

Description

This function is used to produce a coverage file called `fname` . The expected format of the file is described in the Branch Coverage section. By default a normal concise coverage will be produced. If the `add_comment` parameter is set to 1 then the coverage will include code excerpts contained in line entities.

Parameters

fname
: Name of coverage file to load.

add_comments
: This flag controls whether to use coverage extensions. By default this is 0. If 1 is supplied then extensions are used.

Return Types

None.

See Also

`cov_load()`

`cov_report()`

`pldbg_stats()`

`pldbg_stats_load()`

createXML

createXML

Synopsis

```
XMLType createXML ( in src any ,
                    in schema_uri any ,
                    in validated any ,
                    in wellformed any ) ;
```

Description

This function creates an XMLType instance. It works absolutely identically to the `XMLType()` constructor and is provided solely for compatibility.

Parameters

src
: An XML entity or a value that can be converted to an XML entity.

schema_uri
: An URI of the schema of the document. The default is NULL to make result non-schema based.

validated
: An integer flag that indicates if the document is already validated against the schema of the document (this is to avoid redundant validations). The default is 0.

wellformed
: This parameter is unused and is listed solely for compatibility.

Return Types

The function returns a new instance of XMLType.

Examples

Example 24.56. Sample example

```
create table XMLTYPE_TEST (I integer primary key, XMLVAL long xml)

Done. -- 00000 msec.

insert into XMLTYPE_TEST values (1, createXML('<emp><empno>221</empno><ename>John</ename></emp>'))

Done. -- 00000 msec.

insert into XMLTYPE_TEST values (2, createXML('<po><pono>331</pono><poname>PO_1</poname></po>'))

Done. -- 00000 msec.

select e.XMLVAL.extract ('//empno/text()').getNumVal() as EMPNO
  from XMLTYPE_TEST as e
  where e.XMLVAL.existsNode('/emp/empno') = 1
EMPNO
DECIMAL
_______________________________________________________________________________

221

1 Rows. -- 00000 msec.
```

See Also

`XMLType()`

`xtree_doc()`

`xper_doc()`

curdate

curdate , curtime , curdatetime — get current date and time

Synopsis

```
datetime curdate ( );

datetime curdatetime ( );

time curtime ( );
```

Description

These functions return the current date or time as a date, time or datetime, respectively. Internally they all return the same value but data type reported to client differs.

Parameters

None.

Return Types

In a result set to client, the column types reported are as follows: TIME for `curtime` , DATE for `curdate` and DATETIME for `curdatetime` .

Errors

These functions do not signal errors.

Examples

Example 24.57. Simple example

Get current date in different datatypes

```
SQL> select curdate(), curtime(), curdatetime();
callret     callret   callret
DATE        TIME      BINARY
_______________________________________________________________________________

2001.10.12  11:21.51  2001-10-12 11:21:51
```

See Also

`now`

`stringdate`

`datestring`

forget_timezone

forget_timezone — returns its first argument as a timezoned value

Synopsis

```
datetime forget_timezone ( in dt datetime ,
                           in ignore_timezone integer ) ;
```

Description

The function returns its first argument as a timezoned value. If the first argument is timezoneless then it is returned unchanged. If the first argument is timezoned and second argument is missing or zero then the result is timezoneless value that "looks like" local time notation. If the first argument is timezoned and second argument is nonzero then the value is first made GMT and then it becomes timezoneless.

Parameters

dt
: Datetime value.

ignore_timezone
: Flag

Examples

Example 24.58. Simple example

```
SQL> select cast (forget_timezone (cast ('1999-12-31 11:59+02:00' as datetime)) as varchar);
VARCHAR
'1999-12-31 11:59:00'
No. of rows in result: 1
```

See Also

now

timezone

is_timezoneless

adjust_timezone

rdf_now_impl

current_timestamp

curdatetime

curdatetimeoffset

curutcdatetime

sysutcdatetime

stringdate

`datestring`

is_timezoneless

is_timezoneless — returns 1 for timezoneless arguments, zero for timezoned

Synopsis

```
datetime is_timezoneless ( in dt datetime );
```

Description

The function returns 1 for timezoneless arguments, zero for timezoned.

Parameters

dt
: Datetime value.

Examples

Example 24.59. Simple example

```
SQL> select is_timezoneless (cast ('1999-12-31 11:59' as datetime));
INTEGER
 0
No. of rows in result: 1
```

See Also

now

timezone

forget_timezone

adjust_timezone

rdf_now_impl

current_timestamp

curdatetime

curdatetimeoffset

curutcdatetime

sysutcdatetime

stringdate

datestring

adjust_timezone

adjust_timezone — returns its first argument with unchanged GMT value but new timezone offset, as it is specified by the second argument

Synopsis

```
datetime adjust_timezone ( in dt datetime ,
                           in tz_offset integer ,
                           in ignore_timezone integer ) ;
```

Description

The function returns its first argument with unchanged GMT value but new timezone offset, as it is specified by the second argument. If the first argument is timezoneless and third argument is missing or zero then error 22023 is signaled. If the first argument is timezoneless and third argument is nonzero then no error is signaled and the argument is handled like it is a GMT value.

Parameters

dt
: Datetime value.

tz_offset
: Timezone offset.

ignore_timezone
: Flag.

Examples

Example 24.60. Simple example

```
SQL> SELECT adjust_timezone(cast('2014-12-31 15:00-12:00' as datetime),0,1) ;
DATETIME
2015-01-01 03:00:00
No. of rows in result: 1
```

See Also

now

timezone

forget_timezone

is_timezoneless

rdf_now_impl

current_timestamp

curdatetime

curdatetimeoffset

curutcdatetime

sysutcdatetime

stringdate

datestring

rdf_now_impl

rdf_now_impl — returns the current transaction timestamp

Synopsis

```
rdf_now_impl (  );
```

Description

This function returns the timestamp associated with current transaction as a DATETIME . The function name refer to one function (alias of `now()`) that returns the timestamp of current transaction. It is the datetime of the beginning of current transaction with the fractional part of seconds replaced with serial number of a transaction within the second. If `TimezonelessDatetimes=0` then the time has local timezone offset (as it was set at the time of last server start); otherwise it is timezoneless.

Parameters

None

Examples

Example 24.61. Simple example

Get a timestamp in human-readable form:

```
SQL> select rdf_now_impl();
DATETIME
2015-05-04 08:28:18.228407
No. of rows in result: 1
```

See Also

`now`

`timezone`

`forget_timezone`

`is_timezoneless`

`adjust_timezone`

`current_timestamp`

`curdatetime`

`curdatetimeoffset`

`curutcdatetime`

`sysutcdatetime`

`stringdate`

`datestring`

current_timestamp

current_timestamp — returns the current transaction timestamp

Synopsis

```
current_timestamp (  );
```

Description

Now returns the timestamp associated with current transaction as a DATETIME .

The function name refer to one function (alias of `now()`) that returns the timestamp of current transaction. It is the datetime of the beginning of current transaction with the fractional part of seconds replaced with serial number of a transaction within the second. If `TimezonelessDatetimes=0` then the time has local timezone offset (as it was set at the time of last server start); otherwise it is timezoneless.

Parameters

`current_timestamp` has no parameters.

Return Types

A DATETIME timestamp.

Errors

`current_timestamp` does not return errors.

Examples

Example 24.62. Simple example

```
SQL>select current_timestamp ;
DATETIME
 2015-05-05 03:36:41.225581
```

See Also

`now`

`timezone`

`forget_timezone`

`is_timezoneless`

`adjust_timezone`

`rdf_now_impl`

`curdatetime`

`curdatetimeoffset`

`curutcdatetime`

`sysutcdatetime`

`datestring()`

casting

`curdate()`, `curdatetime()`, `curtime()`

curdatetime

curdatetime — returns current datetime with adjusted fractional part of seconds

Synopsis

curdatetime (in *fraction_microseconds* integer);

Description

The function returns current datetime, like `now()` , but fractional part of seconds can be adjusted by providing the number of "microseconds" as the argument.

Parameters

fraction_microseconds
: Microseconds.

Return Types

A DATETIME timestamp.

Errors

`curdatetime` does not return errors.

Examples

Example 24.63. Get a timestamp

Get a timestamp in human-readable form.

```
SQL> select curdatetime();
DATETIME
2015-05-04 09:02:05.242215
No. of rows in result: 1

SQL>select curdatetime(1);
DATETIME
2015-05-04 09:02:47
No. of rows in result: 1
```

See Also

`now`

`timezone`

`forget_timezone`

`is_timezoneless`

`adjust_timezone`

`rdf_now_impl`

`current_timestamp`

curdatetimeoffset

curutcdatetime

sysutcdatetime

casting

curdate(), curdatetime(), curtime()

curdatetimeoffset

curdatetimeoffset — returns current datetime in GMT timezone with adjusted fractional part of seconds

Synopsis

curdatetimeoffset (in *fraction_microseconds* integer) ;

Description

The function returns current datetime, like `now()` , but fractional part of seconds can be adjusted by providing the number of "microseconds" as the argument. It is like `curdatetime()` but the returned datetime is in GMT timezone.

Parameters

fraction_microseconds
: Microseconds.

Return Types

A DATETIME timestamp.

Errors

`curdatetimeoffset` does not return errors.

Examples

Example 24.64. Get a timestamp

Get a timestamp in human-readable form.

```
SQL> select curdatetimeoffset();
DATETIME
2015-05-04 09:07:57.847185
No. of rows in result: 1

SQL>select curdatetimeoffset(1);
DATETIME
2015-05-04 09:07:34
No. of rows in result: 1
```

See Also

`now`

`timezone`

`forget_timezone`

`is_timezoneless`

`adjust_timezone`

`rdf_now_impl`

`current_timestamp`

`curdatetime`

`curutcdatetime`

`sysutcdatetime`

casting

`curdate()`, `curdatetime()`, `curtime()`

curutcdatetime

curutcdatetime — returns current datetime in GMT timezone with adjusted fractional part of seconds

Synopsis

curutcdatetime (in *fraction_microseconds* integer);

Description

The function returns current datetime, like `now()` , but fractional part of seconds can be adjusted by providing the number of "microseconds" as the argument.

The function is similar to `curdatetime()` but the returned datetime is in GMT timezone.

Parameters

fraction_microseconds
: Microseconds.

Return Types

A DATETIME timestamp.

Errors

`curutcdatetime` does not return errors.

Examples

Example 24.65. Get a timestamp

Get a timestamp in human-readable form.

```
SQL> select curutcdatetime();
DATETIME
2015-05-04 13:11:30.701568
No. of rows in result: 1

SQL>select curutcdatetime(1);
DATETIME
2015-05-04 13:11:41
No. of rows in result: 1
```

See Also

`now`

`timezone`

`forget_timezone`

`is_timezoneless`

`adjust_timezone`

`rdf_now_impl`

`current_timestamp`

`curdatetime`

`curdatetimeoffset`

`sysutcdatetime`

casting

`curdate()`, `curdatetime()`, `curtime()`

sysutcdatetime

sysutcdatetime — returns current datetime in GMT timezone with adjusted fractional part of seconds

Synopsis

sysutcdatetime (in *fraction_microseconds* integer);

Description

The function returns current datetime, like `now()` , but fractional part of seconds can be adjusted by providing the number of "microseconds" as the argument.

The function is similar to `curdatetime()` but the returned datetime is in GMT timezone.

Parameters

fraction_microseconds
: Microseconds.

Return Types

A DATETIME timestamp.

Errors

`sysutcdatetime` does not return errors.

Examples

Example 24.66. Get a timestamp

Get a timestamp in human-readable form.

```
SQL> select sysutcdatetime();
DATETIME
2015-05-04 13:11:30.701568
No. of rows in result: 1

SQL>select sysutcdatetime(1);
DATETIME
2015-05-04 13:11:41
No. of rows in result: 1
```

See Also

`now`

`timezone`

`forget_timezone`

`is_timezoneless`

`adjust_timezone`

`rdf_now_impl`

`current_timestamp`

`curdatetime`

`curdatetimeoffset`

`curutcdatetime`

`sysutcdatetime`

casting

`curdate()`, `curdatetime()`, `curtime()`

current_charset

current_charset — Get name of current charset.

Synopsis

```
current_charset (  );
```

Description

This function returns the "preferred" name of the current charset as a string.

It returns value which is set in the INI file or those which is set with 'SET CHARSET=..' call. This cannot be used to return charset for the current HTTP connection in the VSP or VSPX context.

Parameters

None.

Return Types

A string containing the name.

Errors

None signalled.

Examples

Example 24.67. Get name of current charset

```
SQL> select current_charset();
callret
VARCHAR
_______________________________________________________________________________
ISO-8859-1
```

See Also

`charset_define()`, `charset_recode()`, `charsets_list()`

DB.DBA.VACUUM

DB.DBA.VACUUM — Compact the database

Synopsis

DB.DBA.VACUUM (in *table_name* varchar (default %) ,
in *index_name* varchar (default %)) ;

Description

This function reads through the specified tables and indices and finds groups of adjacent pages holding data that will fit on fewer pages than it currently occupies. If such a compression can be made, the pages are thus compacted.

The pages become part of the committed state and will be written to the checkpoint space on the next checkpoint.

The vacuum operation is non-locking and can be run on a busy database. It will simply skip pages with ongoing activity such as pending cursors or locks. The vacuum procedure returns only after it has read through the indices it affects but it will not prevent other activity on the indices. The vacuum operation may run out of disk space even if it makes net gains because the modified pages will not be final until the next checkpoint and the originals will not be free until this same checkpoint. Thus manually running a checkpoint after vacuum runs out of space will free the space and vacuum may be rerun.

Parameters

table_name
: This is a LIKE pattern for tables to vacuum. The default is all tables. The name is case sensitive and must have all the three parts given, e.g. APP.USER.DATA

index_name
: This allows specifying an individual index to compress. The specified table(s) must have this index. The index name is a LIKE pattern and if given should match the case and spelling of index names as returned by the ODBC call SQLStatistics or equivalent, which is also the KEY_NAME column of SYS_KEYS.

Examples

Example 24.68. Simple example

Compact the entire database:

```
SQL> DB.DBA.vacuum ();
```

sparql_to_sql_text

sparql_to_sql_text — Converts a sparql query directly to SQL.

Synopsis

```
varchar sparql_to_sql_text ( in squery varchar );
```

Description

The `sparql_to_sql_text()` function can be used to convert a SPARQL query directly to the SQL it would be executed as.

Parameters

squery
: The sparql query to be converted.

Return Types

A string will be returned containing the converted sparql query.

Examples

Example 24.69. Using the sparql_to_sql_text() function

This simple example shows how to covert sparql query in SQL:

```
SQL> select sparql_to_sql_text('select * where {?s ?p ?o}');
sparql_to_sql_text
VARCHAR
_______________________________________________________________________________

SELECT __id2i ( /*retval[*/ "s-1-1-t0"."S" /* s */ /*]retval*/ ) AS /*tmpl*/ "s",
  __id2i ( /*retval[*/ "s-1-1-t0"."P" /* p */ /*]retval*/ ) AS /*tmpl*/ "p",
  __ro2sq ( /*retval[*/ "s-1-1-t0"."O" /* o */ /*]retval*/ ) AS /*tmpl*/ "o"
FROM DB.DBA.RDF_QUAD AS "s-1-1-t0"
OPTION (QUIETCAST)

1 Rows. -- 15 msec.
```

See Also

`DB.DBA.SPARQL_EVAL_TO_ARRAY`

`DB.DBA.SPARQL_EVAL`

`DB.DBA.SPARQL_RDB2RDF_CODEGEN`

`DB.DBA.SPARQL_RDB2RDF_LIST_TABLES`

`DB.DBA.SPARQL_REXEC`

`DB.DBA.SPARQL_REXEC_TO_ARRAY`

`DB.DBA.SPARQL_REXEC_WITH_META`

`DB.DBA.SPARQL_SELECT_KNOWN_GRAPHS`

dateadd

dateadd — arithmetic add to a date

Synopsis

dateadd (in *unit* string ,
in *number* integer ,
in *date* datetime) ;

Description

dateadd adds a positive or negative quantity of units to a date (in the internal date time format), and returns a new date so formed. The unit is specified as a string and can be one of the following: 'second', 'minute', 'hour', 'day', 'month', or 'year'. Use datestring to convert the result to a human-readable string.

Parameters

unit
: String value denoting the unit to use in the addition.

number
: Integer number of unit units to be added.

date
: Datetime value to which the number of unit s is to be added

Examples

Example 24.70. Simple date addition

Add 10 days to 1996.10.10.

```
SQL> select dateadd ('day', 10, stringdate ('1996.10.10'));
callret
BINARY
_______________________________________________________________________________

1996-10-20 01:00:00

1 Rows. -- 92 msec.
```

See Also

datediff , datestring , stringdate

datediff

datediff — get difference of 2 dates

Synopsis

datediff (in *unit* string ,
in *date1* datetime ,
in *date2* datetime) ;

Description

datediff subtracts date1 from date2 and returns the difference as an integer in the specified units.

Parameters

unit
: The resulting unit name as a string. May be 'millisecond', 'second', 'minute', 'hour', 'day', 'month', or 'year'

date1
: The datetime value that will be subtracted from date2

date2
: The datetime value date1 is subtracted from

Return Values

Difference of *date1* and *date2* as an integer .

Errors

Table 24.17. Errors signalled by datediff

SQLState	Error Code	Error Text	Description
22023	DT002	Bad unit in datediff: >offending unit string<	The unit given was not one of the units listed above.

Examples

Example 24.71. A simple example

Get hour difference of 1996.10.10 and 1996.10.11.

```
SQL> SELECT datediff ('hour',  stringdate ('1996.10.10'), stringdate ('1996.10.11'));
callret
INTEGER
_______________________________________________________________________________
24

1 Rows. -- 57 msec.
```

Example 24.72. Demo DB example

Get average order processing time in days

```
SQL> use "Demo";
SQL> SELECT avg (datediff ('day', "OrderDate", "ShippedDate"))
```

```
  as "Avg_Processing_Time" from "Orders" where "ShippedDate" is not null;
Avg_Processing_Time
INTEGER
_______________________________________________________________________________
8

1 Rows. -- 11 msec.
```

See Also

dateadd, datestring

datestring , datestring_gmt ,

datestring , datestring_gmt — convert a datetime from internal to external date-time representation

Synopsis

datestring (in *date* datetime) ;

datestring_gmt (in *date* datetime) ;

Description

datestring and datestring_gmt convert timestamps or datetimes from internal to external date-time representation. The external representation is a human-readable ASCII string of up to 30 characters.

The external format is: YYYY-MM-DD hh:mm:ss uuuuuu where uuuuuu represents microseconds.

Note:
Using cast (dt as varchar) is preferred over datestring .

Parameters

date
: A datetime value.

Return values

datestring and datestring_GMT return a varchar .

Examples

Example 24.73. Simple Example

Get current date in human-readable form.

```
SQL> select datestring(now());
callret
VARCHAR
_______________________________________________________________________________

2001-03-01 12:49:59.000000

1 Rows. -- 585 msec.
```

Example 24.74. A table query example

Get orders shipped later than July 3, 1996

```
SQL> select left (datestring(ShippedDate), 10) from Orders
   where ShippedDate > stringdate ('1996.6.3');
callret
VARCHAR
_______________________________________________________________________________

1996-06-04
1996-06-04
1996-06-05
1996-06-05
1996-06-05
```

```
5 Rows. -- 3 msec.
```

See Also

`datestring_gmt` , `stringdate`

The discussion of TIMESTAMP datatype in section Database Concepts of the documentation.

datestring_GMT

datestring_GMT — convert a timestamp to external format string in GMT

Synopsis

datestring_GMT (in *dt* datetime) ;

Description

Converts the local datetime to GMT and returns its external representation as a string.

Parameters

dt
: A datetime value.

Return values

A VARCHAR containing the external representation.

Examples

Example 24.75. Example with datestring_GMT

We are at central european time zone CET(GMT+1).

```
SQL> use Demo;

 Done. -- 3 msec.
 SQL> select (datestring_GMT (stringdate ('2000-01-01 22:00')));
 callret
 VARCHAR
 _______________________________________________________________________________

 2000-01-01 21:00:00.000000

 1 Rows. -- 45 msec.
```

See Also

`datestring`, `stringdate`

The discussion of TIMESTAMP datatype in section Database Concepts of the documentation.

DAV add & update functions

DAV_COL_CREATE , DAV_RES_UPLOAD , DAV_DELETE — functions for adding, updating, deleting of DAV collections or resources

Synopsis

```
integer DAV_COL_CREATE ( in path varchar ,
                         in permissions varchar ,
                         in uname varchar ,
                         in gname varchar ,
                         in auth_uname varchar ,
                         in auth_pwd varchar ) ;

varchar DAV_RES_UPLOAD ( in path varchar ,
                         in content any ,
                         in type varchar ,
                         in permissions varchar ,
                         in uname varchar ,
                         in gname varchar ,
                         in auth_uname varchar ,
                         in auth_pwd varchar ) ;

integer DAV_DELETE ( in path varchar ,
                     in silent integer ,
                     in auth_uname varchar ,
                     in auth_pwd varchar ) ;
```

Description

DAV_COL_CREATE creates a new collection on path, with supplied security permissions, returning a collection id (COL_ID) upon success.

DAV_RES_UPLOAD creates or replaces an existing resource on path with content, mime type and supplied security permissions. Returns a resource id (RES_ID) on success.

DAV_DELETE Removes an existing collection/resource. If *silent* is set to a nonzero value, no errors codes will be returned. returns 1 on success.

Parameters

path
: Collection (directory) path and name of destination of upload.

content
: The resource data to upload.

type
: Mime type of the uploaded resource. Defaults to '' if not supplied.

permissions
: Access permission string of Dav collection or resource. Defaults to '110100000R' if not supplied.

silent

If non-zero, no errors will be returned. Default is 0, meaning errors are returned.

uname
: Owner user name. Default is 'dav'.

group name
: Owner group name. Default is 'dav'.

auth_uname
: Name of administration user capable of performing the operation. default is null.

auth_pwd
: Administrator password. Default is null.

Errors

Table 24.18. Errors signalled by DAV_* functions

Error Code	Description
>=0	success
-1	The path (target of operation) is not valid
-2	The destination (path) is not valid
-3	Overwrite flag is not set and destination exists
-4	The target is resource, but source is collection (in copy move operations)
-5	Permissions are not valid
-6	uid is not valid
-7	gid is not valid
-8	Target is locked
-9	Destination is locked
-10	Property name is reserved (protected or private)
-11	Property does not exists
-12	Authentication failed
-13	Operation is forbidden (the authenticated user do not have a permissions for the action)
-14	the target type is not valid
-15	The umask is not valid
-16	The property already exists
-17	Invalid property value
-18	no such user
-19	no home directory

Examples

Example 24.76. Creating a resource and collection

The following example shows collection creation, resource upload and removal. This sequence of commands would leave a resource A.txt in `http://[host:port]/DAV/user/A/`

```
SQL> select DB.DBA.DAV_COL_CREATE ('/DAV/user/','110100000R', 'dav','dav','dav','dav');
SQL> select DB.DBA.DAV_COL_CREATE ('/DAV/user/A/','110100000R','dav','dav','dav','dav');
SQL> select DB.DBA.DAV_RES_UPLOAD ('/DAV/user/A/A.txt','this is a test','text/plain','110100000R','dav','
SQL> select DB.DBA.DAV_RES_UPLOAD ('/DAV/user/A/B.txt','this is a second test','text/plain','110100000R',
SQL> select DB.DBA.DAV_DELETE ('/DAV/user/A/B.txt', 0, 'dav', 'dav');
```

See Also

DAV content manipulation functions

DAV lock manipulation functions

DAV search functions

DAV user management functions

DAV manipulation functions

DAV_COPY , DAV_MOVE , DAV_PROP_SET , DAV_PROP_REMOVE — Functions for manipulating an existing DAV collection or resource

Synopsis

```
integer DAV_COPY ( in path varchar ,
                   in destination varchar ,
                   in overwrite integer ,
                   in permissions varchar ,
                   in uname varchar ,
                   in gname varchar ,
                   in auth_uname varchar ,
                   in auth_pwd varchar ) ;

varchar DAV_MOVE ( in path varchar ,
                   in destination varchar ,
                   in overwrite integer ,
                   in auth_uname varchar ,
                   in auth_pwd varchar ) ;

integer DAV_PROP_SET ( in path varchar ,
                       in propname varchar ,
                       in propvalue any ,
                       in auth_uname varchar ,
                       in auth_pwd varchar ) ;

varchar DAV_PROP_REMOVE ( in path varchar ,
                          in propname varchar ,
                          in silent integer ,
                          in auth_uname varchar ,
                          in auth_pwd varchar ) ;
```

Description

DAV_COPY copies the resource or collection taken from path to the destination. returns COL_ID or RES_ID on success.

DAV_MOVE moves the collection or resource to the destination path returns 1 on success.

DAV_PROP_SET defines or updates the property with name *propname* with *propvalue* . Returns PROP_ID on success.

DAV_PROP_GET returns the value of previously defined property with name *propname* .

DAV_PROP_REMOVE removal of the existing property on target path. If silent supplied then no error will be returned.

Some attributes of resources and collections are accessible as predefined properties. E.g., owner user ID of the resource can be retrieved or changed by DAV_PROP_GET or DAV_PROP_SET with *propname* equal to ':virtowneruid'. Some of these properties are read-only for all resources, some are writable for some but not for all resources. Names of all predefined properties starts with ':' so application-specific properties should not start with colon. Moreover, it is strongly suggested to use "unqualified XML names" as property names, otherwise PROPGET and PROPPATH DAV requests may return invalid XML responses. *propname* .

Parameters

path
: Directory and name of source to be operated on.

destination
: Directory and name of destination.

overwrite
: If non zero then overwrite is enabled. Default is 0.

permissions
: Access permission of Dav collection or resource. Defaults to '110100000R' if not supplied.

propname
: Property name.

propvalue
: Property value.

silent
: If specified as non zero, then no error will be returned. Default is 0, so errors are returned.

uname
: User identifier. Default is 'dav'.

gname
: Group identifier. Default is 'dav'.

auth_uname
: Administration user capable of performing the operation. Default is null.

auth_pwd
: Password of Administrator. Default is null.

System Properties

Names of 'standard live properties' matches tag names used in PROPFIND response for same purposes.

Names of virtuoso-specific system properties starts with ':virt' substring.

In the table below, 'Read/Write' access type means that the application can try to set the property. This does not mean that the property can be successfully changed for any particular resource or collection.

Table 24.19. System properties supported by DAV_PROP_GET and DAV_PROP_SET functions

propname	Access Type	Data Type	Description
:getlastmodified	Read/Write	datetime	Time of the last modification.
:creationdate	Read/Write	datetime	Time of creation.
:lastaccessed	Read Only	datetime	Time of the last access to the resource, may be inaccurate by a large amount.
:getetag	Read Only	varchar	The value of 'Etag' field of response header as reported by HEAD HTTP request.
:getcontenttype	Read/Write	varchar	MIME type of the resource ('dav/unix-directory' for collections).
:getcontentlength	Read Only	integer	Resource length in bytes. 0 for collections.
:resourcetype	Read Only	XML entity	'<D:collection/>' for collections, NULL for resources.
:virtowneruid	Read/Write	integer	User ID of resource owner user.
:virtownergid	Read/Write	integer	Group ID of resource owner group.
:virtpermissions	Read/Write	char(10)	Access permissions string.
:virtacl	Read/Write	long varbinary	Access Control List.

propname	Access Type	Data Type	Description
:virtdet	Read/Write	varchar	DAV Extension Type of special collection, NULL for plain collections and resources.

Note that the value of 'Etag' field depends on ':getlastmodified' value. Some HTTP clients, such as download managers, web indexing robots and proxy servers, use 'Etag' field to support caching and/or partial document transfer. It is unsafe to decrement ':getlastmodified' because it can confuse such clients.

Only resource/collection owner or DAV administrator can change ':virtowneruid', ':virtownergid', ':virtpermissions' and ':virtacl'.

Only DAV administrator can change ':virtdet' property.

Errors

Table 24.20. Errors signalled by DAV_* functions

Error Code	Description
>=0	success
-1	The path (target of operation) is not valid
-2	The destination (path) is not valid
-3	Overwrite flag is not set and destination exists
-4	The target is resource, but source is collection (in copy move operations)
-5	Permissions are not valid
-6	uid is not valid
-7	gid is not valid
-8	Target is locked
-9	Destination is locked
-10	Property name is reserved (protected or private)
-11	Property does not exists
-12	Authentication failed
-13	Operation is forbidden (the authenticated user do not have a permissions for the action)
-14	the target type is not valid
-15	The umask is not valid
-16	The property already exists
-17	Invalid property value
-18	no such user
-19	no home directory

Examples

Example 24.77. Copy and move operations

Renaming of the resource and copy the folder. After executing the commands in the http://[host:port]/DAV/user/B/ we will show the resource B.txt

```
-- initial upload
SQL> select DB.DBA.DAV_COL_CREATE ('/DAV/user/','110100000R', 'dav','dav','dav','dav');
SQL> select DB.DBA.DAV_COL_CREATE ('/DAV/user/A/','110100000R','dav','dav','dav','dav');
SQL> select DB.DBA.DAV_RES_UPLOAD ('/DAV/user/A/A.txt','this is a test','text/plain','110100000R','dav','
SQL> DB.DBA.DAV_MOVE('/DAV/user/A/A.txt', '/DAV/user/A/B.txt', 1,'dav','dav');
SQL> DB.DBA.DAV_COPY('/DAV/user/A/', '/DAV/user/B/', 1, '110110000R','dav','dav','dav','dav');
```

See Also

DAV content add/update/delete functions

DAV lock manipulation functions

DAV search functions

DAV user management functions

DAV lock manipulation functions

DAV_LOCK , DAV_UNLOCK , DAV_IS_LOCKED , DAV_LIST_LOCKS — Operations on locks of DAV collections and resources

Synopsis

```
integer DAV_LOCK ( in path varchar ,
                   in locktype varchar ,
                   in scope varchar ,
                   in token varchar ,
                   in owner_name varchar ,
                   in owned_tokens varchar ,
                   in depth varchar ,
                   in timeout_sec integer ,
                   in auth_uname varchar ,
                   in auth_pwd varchar ) ;

any DAV_UNLOCK ( in path varchar ,
                 in token varchar ,
                 in auth_uname varchar ,
                 in auth_pwd varchar ) ;

integer DAV_IS_LOCKED ( in id any ,
                        in type char(1) ,
                        in owned_tokens any := 1 ) ;

integer DAV_LIST_LOCKS ( in id any ,
                         in type char(1) ) ;
```

Description

DAV_LOCK sets a new lock or refresh an existing lock or creates a lock object.

DAV_UNLOCK releases a lock.

DAV_IS_LOCKED reports whether the resource or collection is locked.

DAV_LIST_LOCKS returns a full list of all locks on a resource or collection.

It is strongly recommended to carefully study DAV standards before using these functions in applications. Improper use of DAV locks may result in subtle application errors that are hard to debug.

Parameters

path
: Resource or collection path. As usual, collection names must end with '/' and both sorts of path strings should begin with '/'. If the path specifies a non-existing resource in an existing collection then a lock object is created and only lock owner can create a resource with this name in future.

locktype
: The type of lock. Current set of DAV standards specifies only one type of locks, called 'D:write'; the parameter value is stored 'as is' and never used, so the recommended value is an empty string.

scope
: The scope of lock as a single-char string: 'S' for shared lock, 'X' for exclusive.

token
: The token to update. Empty string or NULL means creation of a new lock, not an update of the existing one.

owner_name
: Lock owner name as it will be displayed to other users that will query for existing locks. It is strongly recommended to provide commonly used names such as e-mails to let users ask each other to remove forgotten locks.

owned_tokens
: String of tokens owned by a user, to let user ignore his own tokens. The string should match syntax of DAV HTTP 'If' parameter. This can be also an integer, this will work as empty list of tokens.

depth
: This specifies the recursive depth of the lock. The string can be '0' to lock the specified collection only, '1' to lock the collection and its subcollections, 'infinity' to lock the whole subtree. When the lock is added to a resource, the value '0' is used, for obvious reason. If NULL is passed to the function then the effective value is 'infinity' for collection lock or '0' for resource lock.

timeout_sec
: Expiration interval for the lock, in seconds. If 0 or NULL is passed then HTTP DAV default is used that is equal to 604800 seconds (one week).

auth_uname
: DAV user name to authenticate access. Lock, Lock refresh and unlock operations on existing resource or collection require write permission. To create a lock object in a collection, user should have write permission on that collection. This may cause problem when user first locks a resource and then loose write permission for some reason: the user is unable to remove his own lock. Default is NULL that means public access.

auth_pwd
: User password. Default is NULL.

id
: Internal identifier of a resource or collection as returned by DAV_SEARCH_ID().

Examples

Example 24.78. Two concurrent users

The following example shows how two users can prevent each other from undesired operations.

The sample function DAV_LOCK_DEMO() uses a function DUMP_VEC() that print the content of vector in Virtuoso/PL syntax. Yon can find the implementation of this function as one of examples for `vector()` Virtuoso/PL function.

```
create procedure DAV_LOCK_DEMO (in dav_passwd varchar := 'dav')
{
  declare ACTION, RETVAL, token_s1, token_s2, token_x varchar;
  result_names (ACTION, RETVAL);
  result ('Adding group lock_demo_grp:',
    DAV_ADD_GROUP ('lock_demo_grp', 'dav', dav_passwd) );
  result ('Adding user lock_u1:',
    DAV_ADD_USER ('lock_u1', 'lock_u1_pwd', 'lock_demo_grp', '110110000T', 0, '/DAV/lock_demo_u1_home/',
  result ('Adding user lock_u2:',
    DAV_ADD_USER ('lock_u2', 'lock_u2_pwd', 'lock_demo_grp', '110110000T', 0, '/DAV/lock_demo_u2_home/',
  result ('Adding a collection:',
    DAV_COL_CREATE ('/DAV/lock_demo/', '110110000R', 'lock_u1', 'lock_demo_grp', 'dav', dav_passwd) );
  result ('Uploading /DAV/lock_demo/sample.htm:',
    DAV_RES_UPLOAD ('/DAV/lock_demo/sample.htm', '<html><body>This is /DAV/lock_demo/sample.htm</body></h
  token_s1 := DAV_LOCK ('/DAV/lock_demo/sample.htm', '', 'S', NULL, 'user1@example.com', '', '0', NULL, '
  result ('First user set a shared lock:', token_s1);
  token_s2 := DAV_LOCK ('/DAV/lock_demo/sample.htm', '', 'S', NULL, 'user2@example.com', '', '0', NULL, '
  result ('Second user set a shared lock:', token_s2);
  token_x := DAV_LOCK ('/DAV/lock_demo/sample.htm', '', 'X', NULL, 'user1@example.com', token_s1, '0', NU
```

```
  result ('Lock dump',
    DUMP_VEC (DAV_LIST_LOCKS (DAV_SEARCH_ID ('/DAV/lock_demo/sample.htm', 'R'), 'R')) );
  result ('First user tries to set an exclusive lock:', token_x);
  result ('Second user releases his lock:',
    DAV_UNLOCK ('/DAV/lock_demo/sample.htm', token_s2, 'lock_u2', 'lock_u2_pwd') );
  token_x := DAV_LOCK ('/DAV/lock_demo/sample.htm', '', 'X', NULL, 'user1@example.com', token_s1, '0', NU
  result ('First user sets an exclusive lock:', token_x);
  result ('First user releases his lock:',
    DAV_UNLOCK ('/DAV/lock_demo/sample.htm', token_s1, 'lock_u1', 'lock_u1_pwd') );
  token_x := DAV_LOCK ('/DAV/lock_demo/sample.htm', '', 'X', NULL, 'user1@example.com', token_s1, '0', NU
  result ('First user sets an exclusive lock:', token_x);
  result ('Second user tries to add a property to the resource:',
    DAV_PROP_SET ('/DAV/lock_demo/sample.htm', 'sample_prop', 'sample value', 'lock_u2', 'lock_u2_pwd') )
  result ('First user releases his shared:',
    DAV_UNLOCK ('/DAV/lock_demo/sample.htm', token_x, 'lock_u1', 'lock_u1_pwd') );
  result ('Second user adds a property to the resource:',
    DAV_PROP_SET ('/DAV/lock_demo/sample.htm', 'sample_prop', 'sample value', 'lock_u2', 'lock_u2_pwd') )
}

DAV_LOCK_DEMO ();
ACTION   RETVAL
VARCHAR  VARCHAR
_______________________________________________________________________________

Adding group lock_demo_grp:  107
Adding user lock_u1:  108
Adding user lock_u2:  109
Adding a collection:  171
Uploading /DAV/lock_demo/sample.htm:  1103
First user set a shared lock:  1545ce54-3599-11da-8697-8f3efbd86ef2
Second user set a shared lock:  15460590-3599-11da-8697-8f3efbd86ef2
Lock dump
vector (
vector ('', 'S', '1545ce54-3599-11da-8697-8f3efbd86ef2', 604800, 108, 'user1@example.com'),
vector ('', 'S', '15460590-3599-11da-8697-8f3efbd86ef2', 604800, 109, 'user2@example.com'))
First user tries to set an exclusive lock:  -8
Second user releases his lock:  15460590-3599-11da-8697-8f3efbd86ef2
First user sets an exclusive lock:  -8
First user releases his lock:  1545ce54-3599-11da-8697-8f3efbd86ef2
First user sets an exclusive lock:  1547251a-3599-11da-8697-8f3efbd86ef2
Second user tries to add a property to the resource:  -8
First user releases his shared:  1547251a-3599-11da-8697-8f3efbd86ef2
Second user adds a property to the resource:  1321
```

See Also

DAV content add/update/delete functions

DAV content manipulation functions

DAV search functions

DAV user management functions

DAV search functions

DAV_SEARCH_ID , DAV_SEARCH_PATH , DAV_DIR_LIST — Functions for searching a DAV collection or resource

Synopsis

```
integer DAV_SEARCH_ID (
                        in path varchar
                        ,
                        in what char(1)
                        ) ;

varchar DAV_SEARCH_PATH ( in id integer ,
                          in what char(1) ) ;

any DAV_DIR_LIST ( in path varchar ,
                   in recursive integer ,
                   in auth_uid varchar ,
                   in auth_pwd varchar ) ;
```

Description

DAV_SEARCH_ID() returns the RES_ID or COL_ID, depending on the '*what* ' parameter passed. ('R'esource or 'C'ollection or 'P'arent collection).

DAV_SEARCH_PATH() returns full path string of resource or collection, depending on parameter passed. ('R'esource or 'C'ollection or 'P'arent collection).

DAV_DIR_LIST() returns an array of arrays that contains the following information about the requested path:

fullpath
type ('r' or 'c') which indicates resource or collection.
length
modification time
id
permissions
group id
owner id
created
mime type

Parameters

path
: Name of DAV location to search.

what
: The type of DAV item to search for: 'R' for resource, 'C' for collection or 'P' for parent collection.

id
: Identifier of resource or collection, for example from DAV_SEARCH_ID().

recursive
: If non zero then recurse into subdirectories during the search. Default is 0 which causes a search in current path only.

auth_uid

Administration user capable of performing the operation. Default is null.

auth_pwd
: Password of Administrator. Default is null.

Errors

Table 24.21. Errors signalled by DAV_* functions

Error Code	Description
>=0	success
-1	The path (target of operation) is not valid
-2	The destination (path) is not valid
-3	Overwrite flag is not set and destination exists
-4	The target is resource, but source is collection (in copy move operations)
-5	Permissions are not valid
-6	uid is not valid
-7	gid is not valid
-8	Target is locked
-9	Destination is locked
-10	Property name is reserved (protected or private)
-11	Property does not exists
-12	Authentication failed
-13	Operation is forbidden (the authenticated user do not have a permissions for the action)
-14	the target type is not valid
-15	The umask is not valid
-16	The property already exists
-17	Invalid property value
-18	no such user
-19	no home directory

Examples

Example 24.79. Searching in WebDAV

The example shows retrieval of the resource id by given path string

```
SQL> select DB.DBA.DAV_SEARCH_ID ('/DAV/','c');
callret
VARCHAR
_______________________________________________________________________________

1

SQL> select DB.DBA.DAV_SEARCH_PATH (1,'c');
callret
VARCHAR
_______________________________________________________________________________

/DAV/
```

See Also

DAV content add/update/delete functions

DAV content manipulation functions

DAV lock manipulation functions

DAV user management functions

WebDAV Users & Groups administration

DAV_ADD_USER , DAV_DELETE_USER , DAV_HOME_DIR , DAV_ADD_GROUP , DAV_DELETE_GROUP — Functions for manipulating an existing DAV collection or resource

Synopsis

```
integer DAV_ADD_USER ( in uid varchar ,
                       in pwd varchar ,
                       in gid varchar ,
                       in permis varchar ,
                       in disable integer ,
                       in home varchar ,
                       in full_name varchar ,
                       in email varchar ,
                       in uid varchar ,
                       in auth_uid varchar ,
                       in auth_pwd varchar ) ;

varchar DAV_DELETE_USER ( in uid varchar ,
                          in auth_uid varchar ,
                          in auth_pwd varchar ) ;

varchar DAV_HOME_DIR ( in uid varchar ) ;

integer DAV_ADD_GROUP (   in gid varchar ,
                          in auth_uid
                          varchar ,
                          in auth_pwd
                          varchar ) ;

varchar DAV_DELETE_GROUP ( in gid varchar ,
                           in auth_uid varchar ,
                           in auth_pwd varchar ) ;
```

Description

DAV_ADD_USER() create a new WebDAV user with login name 'uid' and password 'pwd'. User will belong to the group named 'gid'. 'perms' are the default user permissions for creation of new resources. Additional user info supplied is 'home' directory, 'full name' and 'e-mail'.

DAV_DELETE_USER() remove the existing webDAV user named 'uid'.

DAV_HOME_DIR() returns the home folder for specified WebDAV user named 'uid'.

DAV_ADD_GROUP() create a new webDAV group named 'gid'.

DAV_DELETE_GROUP() remove the existing webDAV group named 'gid'.

Parameters

uid
: User identifier. Default is 'dav'.

pwd
: Password

gid
: Group identifier. Default is 'dav'.

perms
: Permissions

disable
: Disable flag

home
: The User's home directory path

full_name
: Full name of user

email
: User's email

auth_uid
: Administration user capable of performing the operation. Default is null.

auth_pwd
: Password of Administrator. Default is null.

Errors

Table 24.22. Errors signalled by DAV_* functions

Error Code	Description
>=0	success
-1	The path (target of operation) is not valid
-2	The destination (path) is not valid
-3	Overwrite flag is not set and destination exists
-4	The target is resource, but source is collection (in copy move operations)
-5	Permissions are not valid
-6	uid is not valid
-7	gid is not valid
-8	Target is locked
-9	Destination is locked
-10	Property name is reserved (protected or private)
-11	Property does not exists
-12	Authentication failed
-13	Operation is forbidden (the authenticated user do not have a permissions for the action)
-14	the target type is not valid
-15	The umask is not valid
-16	The property already exists
-17	Invalid property value
-18	no such user
-19	no home directory

Examples

Example 24.80. WebDAV users operations

The example will create a WebDAV user named 'davuser' belongs to the group 'davgroup'. No home directory supplied. The user account is enabled after operaion.

```
SQL> DB.DBA.DAV_ADD_GROUP ('davgroup', 'dav', 'dav');
SQL> DB.DBA.DAV_ADD_USER ('davuser', 'secret', 'davgroup', '110100000', 0, NULL, 'Test User Account', 'no
```

See Also

DAV content add/update/delete functions

DAV content manipulation functions

DAV lock manipulation functions

DAV search functions

DAV_EXP

DAV_EXP — Export a retrieved Web site to another WebDAV enabled server

Synopsis

```
WS.WS. DAV_EXP ( in host varchar ,
                 in url varchar ,
                 in root varchar ,
                 in dst varchar ) ;
```

Description

This function is used to export local content retrieved from a Web Robot Copy to the local file system.

Parameters

host
: The target host name

url
: start path on target

root
: local WebDAV collection that contains the retrieved content

dst
: absolute URL to the WebDAV folder to export content to

Return Types

The function returns a non-zero value (can be an error description) if an error occurred while exporting.

Errors

Table 24.23. Errors signalled by DAV_* functions

Error Code	Description
>=0	success
-1	The path (target of operation) is not valid
-2	The destination (path) is not valid
-3	Overwrite flag is not set and destination exists
-4	The target is resource, but source is collection (in copy move operations)
-5	Permissions are not valid
-6	uid is not valid
-7	gid is not valid
-8	Target is locked
-9	Destination is locked
-10	Property name is reserved (protected or private)
-11	Property does not exists
-12	Authentication failed
-13	Operation is forbidden (the authenticated user do not have a permissions for the action)
-14	the target type is not valid

Error Code	Description
-15	The umask is not valid
-16	The property already exists
-17	Invalid property value
-18	no such user
-19	no home directory

Examples

Example 24.81. Exporting to the Filesystem

```
WS.WS.LFS_EXP ('www.foo.com', '/help/', 'sites/www_foo_com', '/tmp/');
WS.WS.DAV_EXP ('www.foo.com', '/help/', 'sites/www_foo_com',
  'http://www.mydrive.com:8990/DAV/sites/');
```

See Also

`WS.WS.LFS_EXP()`

DAV content add/update/delete functions

DAV content manipulation functions

DAV search functions

DAV user management functions

dayname

dayname , monthname , dayofmonth , dayofweek , dayofyear , quarter , week , month , year , hour , minute , second , timezone — decompose a datetime to its components

Synopsis

dayname (in *dt* datetime);

monthname (in *dt* datetime);

dayofmonth (in *dt* datetime);

dayofweek (in *dt* datetime);

dayofyear (in *dt* datetime);

quarter (in *dt* datetime);

week (in *dt* datetime);

month (in *dt* datetime);

year (in *dt* datetime);

hour (in *dt* datetime);

minute (in *dt* datetime);

second (in *dt* datetime);

timezone (in *dt* datetime);

Description

These functions decompose a datetime to its components. These can be used on timestamps , datetimes , dates and times , all being the same internal data type.

```
:  dayname      name of day
:  monthname    name of month
:  dayofmonth   day of month
:  dayofweek    day of week
:  dayofyear    day since start of year
:  quarter      quarter number,
:  week                 week number
:  month                month number, starting at 1 for January
:  year                 year
:  hour                 hour
:  minute               minute
:  second               second
:  timezone     offset from UTC in minutes
```

Parameters

dt
: A datetime value.

Return Values

`monthname` and `dayname` return a varchar . The others return an integer .

Examples

Example 24.82. Simple example

```
SQL> select dayname(stringdate('2001-03-02'));
callret
VARCHAR
_______________________________________________________________________________

Friday

1 Rows. -- 3 msec.
```

See Also

`stringdate` , `datestring` , `datestring_GMT` , The TIMESTAMP datatype

dayofmonth

dayofmonth — get day of month from a datetime

Synopsis

dayofmonth (in *dt* datetime) ;

Description

`dayofmonth` takes a datetime and returns an integer containing day of the month represented by the datetime

Parameters

dt
: A datetime .

Return Values

An INTEGER containing the day of month.

Examples

Example 24.83. Simple example

Get current day of month.

See Also

`dayname` , `monthname` , `dayofweek` , `dayofyear` , `quarter` , `week` , `month` , `year` , `hour` , `minute` , `second` , `timezone`

dayofweek

dayofweek — get day of week from a datetime

Synopsis

dayofweek (in *dt* datetime);

Description

dayofweek takes a datetime and returns an integer containing a number representing the day of week of the datetime.

Parameters

dt
: A datetime .

Return Values

An INTEGER containing number of the day of week.

Examples

Example 24.84. Simple example

Get current day of week.

```
SQL> select dayofweek(now());
callret
INTEGER
_______________________________________________________________________________

6
```

See Also

dayname , monthname , dayofmonth , dayofyear , quarter , week , month , year , hour , minute , second , timezone

dayofyear

dayofyear — get day of year from a datetime

Synopsis

dayofyear (in *dt* datetime);

Description

dayofyear takes a datetime and returns an integer containing a number representing the day of year of the datetime.

Parameters

dt
: A datetime .

Return Values

An INTEGER containing number of the day of year.

Examples

Example 24.85. Simple example

Get current day of year.

```
SQL> select dayofyear (now ());
callret
INTEGER
_______________________________________________________________________________

282
```

See Also

dayname , monthname , dayofmonth , dayofweek , quarter , week , month , year , hour , minute , second , timezone

dbg_obj_print

dbg_obj_print — print to the Virtuoso system console

Synopsis

dbg_obj_print (in *arg1* any ,
...) ;

Description

dbg_obj_print prints a variable number of arguments onto the system console (stdout) of Virtuoso server, each argument in its own native format, on the same line, which is followed by one newline.

Parameters

dbg_obj_print takes a variable number of any type.

Return Values

None

Errors

Table 24.24. Errors signalled by

SQLState	Error Code	Error Text	Description

Examples

Example 24.86. Simple Use

```
...
declare vec any;
vec := vector ('a', 'b', 'c');
dbg_obj_print (vec)
...
```

Would output this on the console:

```
("a" "b" "c" )
```

See Also

dbg_printf

dbg_printf

dbg_printf — print formatted output onto the system console

Synopsis

dbg_printf (in *format* varchar ,
in *arg1* any ,
... ,
in *argn* any) ;

Description

`dbg_printf` prints a variable number (max. eight) of arguments to the system console of Virtuoso server, each argument formatted in C `printf` style, according to the format string specified in the first argument.

Parameters

format
: a C sprintf -style format string

argn
: The arguments to format and print in any type

Return Values

None

Examples

Example 24.87. Simple example

The frammitz just zilched!

```
if (frammitz_status 0)
{
dbg_printf ('Error: Frammitz status: %d.\nProgram terminated',
frammitz_status);
signal ('42666', 'Frammitz zilched');
}
```

Hitting these lines would cause something like this in a client:

```
*** Error 42666: [Virtuoso Driver][Virtuoso Server]Frammitz zilched at line 84 (84) of Top-Level:
zappi()
```

The console output would look like this:

```
Error: Frammitz status: 2.
Program terminated
```

See Also

sprintf()

log_message()

dbname

dbname — get current catalog

Synopsis

```
dbname (   void);
```

Description

Returns the current qualifier as set by the USE statement or default.

Parameters

None

Return Values

A varchar containing the current catalog name/qualifier.

Examples

Example 24.88. Simple Example

Get current catalog/qualifier

```
SQL> select dbname();
callret
VARCHAR
_______________________________________________________________________________

DB

1 Rows. -- 117 msec.
SQL> use Demo;

Done. -- 73 msec.
SQL> select dbname();
callret
VARCHAR
_______________________________________________________________________________

Demo

1 Rows. -- 4 msec.
SQL>
```

delay

delay — sleep for n seconds

Synopsis

delay (in *n_seconds* integer);

Description

This will halt calling procedure execution for specified interval in seconds.

Parameters

n_seconds
: INTEGER number of seconds to sleep.

Return Types

None

Examples

Example 24.89. Simple example

```
create procedure
waitsome (in _interval integer)
{
  delay (_interval);
}

SQL> waitsome (10);

Done. -- 10004 msec.
```

cl_exec

cl_exec — executes things on all nodes of a cluster

Synopsis

cl_exec (in *cmd* varchar ,
in *params* any := NULL ,
in *is_txn* int := 0) ;

Description

The cl_exec SQL function can be used for executing things on all nodes of a cluster.

Parameters

cmd
: A SQL string. If it contains parameter markers (?), the params array is used for assigning values, left to right.

params
: Any params.

is_txn
: If set to 1, the cl_exec makes a distributed transaction and does not automatically commit on locally on each node. Thus cl_exec can be used as part of a containing distributed transaction.

Examples

Example 24.90. Simple example

```
SQL> cl_exec ('shutdown')
--will shut all nodes.

SQL> cl_exec ('dbg_obj_print (?)', vector ('hello'));
--will print hello to the standard output of all the processes of the cluster.
```

See Also

Cluster Installation and Config

Backup and Restore Example

dict_dec_or_remove

dict_dec_or_remove — Decrement a counter in a dictionary of counters or deletes it if it becomes zero or negative.

Synopsis

dict_dec_or_remove (inout *dict* dictionary ,
in *key* any ,
in *value_decrement* integer) ;

Description

The function checks whether *dict* contains *key* . If it isn't so then the function checks the datatype of the value associated with the key. An error 42000 is signalled in case of non-integer value or a negative integer value. If the value is positive and greater than *value_decrement* then *value_decrement* is subtracted from it and the result become the new value associated with *key* in *dict* . If the value is positive and less than or equal to *value_decrement* then *key* is removed from *dict* . If key is not in the dictionary then the dictionary remains unchanged.

Informally, the function reverts the effect of dict_inc_or_add .

Parameters

dict
: Dictionary of counters. The NULL value is equivalent to an empty dictionary.

key
: Key of a dictionary item to process.

value decrement
: A nonnegative integer (typically 1) that is subtracted from the value associated with key .

Return Types

The function returns the changed value associated with the *key* , or zero in any other case.

See Also

dict_new()

dict_zap()

dict_put()

dict_get()

dict_remove()

dict_inc_or_put()

dict_size()

dict_duplicate()

dict_to_vector()

dict_list_keys()

dict_destructive_list_rnd_keys()

dict_iter_rewind()

dict_iter_next()

dict_duplicate

dict_duplicate — Creates a copy of the given dictionary.

Synopsis

```
dictionary dict_duplicate ( in dict dictionary );
```

Description

The function creates a full copy of the given dictionary.

Dictionary objects are always passed by reference, even if dictionary is passed as an 'in' argument of a function. If value of a variable is a dictionary and it is assigned to other variable then both variables refer to the same internal hash table. This function returns a really independent dictionary object.

This function duplicates the given dictionary but does not duplicate dictionaries that may be stored in key-value pairs.

Parameters

dcit
: The dictionary to be duplicated.

Return Types

The function returns a new dictionary object.

See Also

`dict_new()`

`dict_zap()`

`dict_put()`

`dict_get()`

`dict_remove()`

`dict_inc_or_put()`

`dict_dec_or_remove()`

`dict_size()`

`dict_to_vector()`

`dict_list_keys()`

`dict_destructive_list_rnd_keys()`

`dict_iter_rewind()`

`dict_iter_next()`

dict_get

dict_get — Returns the dependent part that corresponds to the given key in the given dictionary.

Synopsis

```
any dict_get ( in dict dictionary ,
               in key any ,
               in default_value any );
```

Description

The function returns the dependent part that corresponds to the given key in the given dictionary. The function returns *default_value* if the dictionary does not contain a pair whose key is equivalent to the given one. When only two arguments are passed to the function the default value is integer zero.

Parameters

dict
: Dictionary object to search in.

key
: Key whose dependent part should be found.

default_value
: The default value to be returned if the key is not found in the dictionary.

Return Types

The function can return a value of any type.

See Also

`dict_new()`

`dict_zap()`

`dict_put()`

`dict_remove()`

`dict_inc_or_put()`

`dict_dec_or_remove()`

`dict_size()`

`dict_duplicate()`

`dict_to_vector()`

`dict_list_keys()`

`dict_destructive_list_rnd_keys()`

`dict_iter_rewind()`

`dict_iter_next()`

dict_inc_or_put

dict_inc_or_put — Creates or increments an integer counter for a given key and a dictionary.

Synopsis

dict_inc_or_put (inout *dict* dictionary ,
in *key* any ,
in *value_increment* integer) ;

Description

The function checks whether *dict* contains *key* . If it isn't so then the function checks the datatype of the value associated with the key. An error 42000 is signalled in case of non-integer value or a negative integer value. If the value is positive then *value_decrement* is added to it and the result become the new value associated with *key* in *dict* . If key is not in the dictionary then a new item is added to the *dict* in order to associate the *key* with *value_increment* .

Parameters

dict
: Dictionary of counters. If the value is NULL then the function immediately returns zero.

key
: Key of a dictionary item to process.

value decrement
: A nonnegative integer (typically 1) that is added to the value associated with key or used as a starting value of a newly created counter.

Return Types

The function returns zero (for NULL *dict*) or the changed (or the added) value associated with the *key* .

Example

Example 24.91. Simple Use

The function is convenient to deal with multisets, i.e., sets with repeating elements. In this case the dictionary contains distinct items as keys and counts of duplicates as associated values. `dict_inc_or_add` is to add a member, `dict_dec_or_remove` is to remove. The following example gets an array of multisets and return the sum of them.

```
create function DB.DBA.SUM_MULTISETS (inout msets any) returns any
{
  declare sum_of_msets any;
  sum_of_msets := dict_new (17);
  foreach (any mset in msets) do
    {
      declare iter any;
      declare memb any;
      declare dup_count integer;
      iter := mset; --- unlike dict_duplicate() this does not make copy of mset so it's fast.
      dict_iter_rewind (iter);
      while (dict_iter_next (iter, memb, dup_count))
        dict_inc_or_put (sum_of_msets, memb, dup_count);
    }
  return sum_of_msets;
};
```

See Also

dict_new()

dict_zap()

dict_put()

dict_get()

dict_remove()

dict_dec_or_remove()

dict_size()

dict_duplicate()

dict_to_vector()

dict_list_keys()

dict_destructive_list_rnd_keys()

dict_iter_rewind()

dict_iter_next()

dict_iter_next

dict_iter_next — Fetches a pair of key and value from a dictionary iterator and moves the iterator to the next pair.

Synopsis

```
dict_iter_next ( inout dict dictionary ,
                 out ret_key any ,
                 out ret_value any ) ;
```

Description

The function gets the dictionary iterator initialized earlier by `dict_iter_rewind` and checks whether the iterator is still valid and not in conflict with any changes made by `dict_put` or the like. After that, if the iterator is in the position past the last item of the dictionary then zero is returned. If the iterator points to some item then *`ret_key`* and *`ret_value`* parameters are set to the key and value of the current item, the iterator is advanced to the next position (next item if present, otherwise past the end of the dictionary) and a nonzero integer is returned. If *`ret_value`* is a constant or an expression but not a plain variable then it is left unchanged but no error is signalled, so if the caller procedure needs only values of keys from dictionary then any constant like zero can be passed as a third parameter. Similarly, *`ret_key`* is not necessarily a variable.

Note that the values of *`ret_key`* and *`ret_value`* are left unchanged if the iterator points past the end of the dictionary. They are not filled in with NULLs or something like that.

Parameters

dict
: Dictionary iterator

ret_key
: The variable to be filled in with the key of the item. The parameter is ignored if it is not a plain variable.

ret_value
: The variable to be filled in with the value (dependant part) of the item. The parameter is ignored if it is not a plain variable.

Return Types

The function returns a nonzero integer if the item is successfully fetched, zero otherwise

Errors

This function can generate the following errors:

Table 24.25.

Error Code	Description
22023	SR630 Function dict_iter_next() tries to iterate a volatile dictionary changed after last `dict_iter_rewind` (). Not every change in the dictionary results in this error. If the dict_iter_next() and `dict_put` () (or similar function) are both called with same variable passed as *`dict`* parameter then the function might adjust the iterator to match the changed state of the dictionary so it remains valid. In addition, if `dict_put` () changes only the value associated with some key but does not extend the dictionary with a new item then it does not invalidate any iterators.

Example

Example 24.92. Simplest read throughout the dictionary.

The procedure creates a dictionary, puts couple of items into it and then print them to the server's console

```
create function dict_iterator_test ()
{
  declare dict, dkey, dvalue any;
  dict := dict_new (10);
  dict_put (dict, 'a', 1); dict_put (dict, 'b', 2);
  dict_iter_rewind (dict);
  while (dict_iter_next (dict, dkey, dvalue))
    dbg_obj_princ (' key is ' dkey, ', corresponding value is ', dvalue);
}
```

See Also

dict_new

dict_zap

dict_put

dict_get

dict_remove

dict_inc_or_put

dict_dec_or_remove

dict_size

dict_duplicate

dict_to_vector

dict_list_keys

dict_destructive_list_rnd_keys

dict_iter_rewind

dict_iter_rewind

dict_iter_rewind — Resets the iterator before fetching keys and values by dict_iter_next()

Synopsis

dict_iter_rewind (inout *dict* dictionary);

Description

The dictionary is always passed by reference, due to its (potentialy big) size. The reference variable contains not only pointer to the whole dictionary but also a forward-only iterator (a "cursor") that can be used to retrieve items of the dictionary one after the other. The function sets the iterator to the very first item of the dictionary.

Parameters

dict
: The reference to a dictionary to use as an iterator

Return Types

The function returns an integer that is the expected number of items in the dictionary; the value may not match to the number of items retrieved later by dict_iter_next() if a dictionary is edited by some thread.

Example

Example 24.93. Simplest read throughout the dictionary.

The procedure creates a dictionary, puts couple of items into it and then print them to the server's console

```
create function dict_iterator_test ()
{
  declare dict, dkey, dvalue any;
  dict := dict_new (10);
  dict_put (dict, 'a', 1); dict_put (dict, 'b', 2);
  dict_iter_rewind (dict);
  while (dict_iter_next (dict, dkey, dvalue))
    dbg_obj_princ (' key is ' dkey, ', corresponding value is ', dvalue);
}
```

See Also

dict_new

dict_zap

dict_put

dict_get

dict_remove

dict_inc_or_put

dict_dec_or_remove

dict_size

dict_duplicate

dict_to_vector

dict_list_keys

dict_destructive_list_rnd_keys

dict_iter_next

dict_list_keys

dict_list_keys — Returns an array of all keys stored in the given dictionary.

Synopsis

```
vector dict_list_keys ( inout dict dictionary ,
                        in destructive any ) ;
```

Description

The function returns an array of all keys stored in the given dictionary, ignoring dependent parts of key-value pairs. This is especially useful when dictionary is used to form a set of distinct keys, when dependent parts are fake (typically zeroes). If the *destructive* parameter is 1 or 2 then the function may avoid copying of keys to the resulting array by moving them out from the dictionary. This is faster but the dictionary will become empty at the end of operation. The *destructive* parameter equal to 1 does not have any effect if the dictionary is used as a value of more than one variable. Thus it is safe to make this parameter equal to 1 as soon as the variable passed to the function as *dict* is no longer in use after the function call and there's no need to inspect the whole program to check if other variables may be affected. The parameter equal to 2 ignores the presence of multiple variables so the operation may cause unexpected results in other fragments of code that share the dictionary.

The created array can not be longer than the system limit and even if it is below limit but longer than hundreds of thousands items then memory allocation can be inefficient. If the dictionary can be large and it is possible to process the content of the dictionary in parts, consider using dict_destructive_list_rnd_keys () instead.

Parameters

dict
: The dictionary to scan

destructive
: Flag that indicates whether the dictionary can be cleaned during the operation (0 = never clean, 1 = only if there is no other variables, 2 = clean unconditionally).

Return Types

The function returns a vector.

See Also

dict_new()

dict_zap()

dict_put()

dict_get()

dict_remove()

dict_inc_or_put()

dict_dec_or_remove()

dict_size()

dict_duplicate()

dict_to_vector()

dict_destructive_list_rnd_keys()

dict_iter_rewind()

dict_iter_next()

dict_destructive_list_rnd_keys

dict_destructive_list_rnd_keys — Returns all or some keys from the given dictionary

Synopsis

dict_destructive_list_rnd_keys (inout *dict* dictionary ,
in *batch_size* integer) ;

Description

The function returns an array of keys stored in the given dictionary, ignoring dependent parts of key-value pairs, but unlike `dict_list_keys` it may return only some keys, not all, and it always remove the returned keys from the dictionary. The returned array is *`batch_size`* items long (if the dictionary is big enough to fill it).

The order of items returned (and hence the content of the dictionary after function call) is undefined. It is not randomized artificially but it may vary from run to run depending on barely predictable factors. As a result, the most typical use of the function is calling it in a loop that ends when the size of returned array falls down to zero indicating that the dictionary is exausted.

Parameters

dict
: The dictionary where result keys come from.

batch_size
: The maximum size of array to be constructed. The value can not be less than 65535 and more than the maximum allowed length of array (1250000 for 64-bit machines, 25000000 for 32-bit machines). All other equal, it is preferable to keep it below 200000.

Return Types

The function always returns an array; the array is empty if *`dict`* is NULL or an empty dictionary.

Example

Example 24.94. Simple Use

The function gets a dictionary with RDF triples stored as keys and inserts all these triples into a given graph.

```
create function DB.DBA.INSERT_DICT_IN_GRAPH (in graph_iri varchar, in triples_dict any) returns varchar
{
  declare triples any;
  declare ins_count integer;
  ins_count := 0;
  while (dict_size (triples_dict) > 0)
    {
      triples := dict_destructive_list_rnd_keys (triples_dict, 80000);
      DB.DBA.RDF_INSERT_TRIPLES (graph_iri, triples, 1);
      ins_count := ins_count + length (triples);
    }
  return sprintf ('The function INSERT_DICT_IN_GRAPH has inserted %d triples in graph <%s>',
    ins_count, graph_iri );
}
;
```

See Also

`dict_new()`

`dict_zap()`

`dict_put()`

`dict_get()`

`dict_remove()`

`dict_inc_or_put()`

`dict_dec_or_remove()`

`dict_size()`

`dict_duplicate()`

`dict_to_vector()`

`dict_list_keys()`

`dict_iter_rewind()`

`dict_iter_next()`

dict_new

dict_new — Creates a new dictionary to store key-value pairs.

Synopsis

```
dictionary dict_new ( in size integer );
```

Description

This function creates a new dictionary. A dictionary is a memory-resident hash table that can store an arbitrary number of key-value pairs. Both key and dependent part can be of any type, including vectors. Two keys of different data types are always considered as different even if SQL '=' operator return 'true', e.g. integer zero and double precision 0.0 are two different keys. Vectors are equivalent if their corresponding members are either equal scalars or equivalent vectors. XML entities are equivalent if they refer to the same node or attribute in the same document.

Dictionary objects are always passed by reference, even if dictionary is passed as an 'in' argument of a function. If value of a variable is a dictionary and it is assigned to other variable then both variables refer to the same internal hashtable. To create two really independent dictionary objects, use `dict_duplicate ()`.

Parameters

size
: The guessed side of hashtable. overestimation of the size will result in needless memory consumption whereas underscore leads to a bit slower insertion, because the internal hash table will grow dynamically. The simplest approach is to omit the parameter at all if you don't know the number of keys to be stored.

Return Types

The function returns a new empty dictionary object.

See Also

`dict_zap()`

`dict_put()`

`dict_get()`

`dict_remove()`

`dict_inc_or_put()`

`dict_dec_or_remove()`

`dict_size()`

`dict_duplicate()`

`dict_to_vector()`

`dict_list_keys()`

`dict_destructive_list_rnd_keys()`

`dict_iter_rewind()`

`dict_iter_next()`

dict_put

dict_put — Adds a key-value pair to a dictionary

Synopsis

```
int dict_put ( inout dict dictionary ,
               in key any ,
               in value any );
```

Description

The function adds or updates a key-value pair in the given dictionary. If the dictionary contains a pair with the key part equivalent to the given one then old value is replaced with a new one. Otherwise, a new pair is added to the dictionary.

Parameters

dict
: The dictionary to be updated or extended.

key
: The key part of the dictionary pair to be added or updated.

value
: The dependent part of the pair.

Return Types

The function returns the total number of pairs stored in the dictionary as counted after the modification.

See Also

`dict_new()`

`dict_zap()`

`dict_get()`

`dict_remove()`

`dict_inc_or_put()`

`dict_dec_or_remove()`

`dict_size()`

`dict_duplicate()`

`dict_to_vector()`

`dict_list_keys()`

`dict_destructive_list_rnd_keys()`

`dict_iter_rewind()`

`dict_iter_next()`

dict_remove

dict_remove — Removes the given key and associated dependent value from the given dictionary.

Synopsis

```
int dict_remove ( inout dict dictionary ,
                  inout key any );
```

Description

The function removes the given key and the associated dependent value from the given dictionary. If the key-value pair is found (and removed) the function returns 1. If the dictionary does not contain a key equivalent to the given one the function returns zero.

Parameters

dict
: Dictionary object to process.

key
: Key of a key-value pair that should be removed from the dictionary.

Return Types

Integer flag indicating if a key-value pair has been found and removed.

See Also

`dict_new()`

`dict_zap()`

`dict_put()`

`dict_get()`

`dict_inc_or_put()`

`dict_dec_or_remove()`

`dict_size()`

`dict_duplicate()`

`dict_to_vector()`

`dict_list_keys()`

`dict_destructive_list_rnd_keys()`

`dict_iter_rewind()`

`dict_iter_next()`

dict_size

dict_size — Returns the number of items in the given dictionary.

Synopsis

dict_size (inout *dict* dictionary);

Description

The function returns the number of items in the *dict* dictionary. For NULL value of *dict* , zero is returned.

Parameters

dict
: Reference to the dictionary in question.

Return Types

The function returns an interger.

See Also

dict_new()

dict_zap()

dict_put()

dict_get()

dict_remove()

dict_inc_or_put()

dict_dec_or_remove()

dict_duplicate()

dict_to_vector()

dict_list_keys()

dict_destructive_list_rnd_keys()

dict_iter_rewind()

dict_iter_next()

dict_to_vector

dict_to_vector — Returns a get_keyword style vector of all items stored in the given dictionary.

Synopsis

```
vector dict_to_vector ( inout dict dictionary ,
                        in destructive any ) ;
```

Description

The function returns an array of all data stored in the given dictionary, every pair is represented as two consecutive element of the resulting array. Thus the dictionary of N pairs is converted into a vector of length 2N where keys are at positions 0, 2, 4, ..., 2N-1 and corresponding dependent data are at positions 1, 3, 5, ..., 2N. If keys are all scalars of same type then `get_keyword` function can be used to search in the resulting array.

If the *destructive* parameter is 1 or 2 then the function may avoid copying of keys and values to the resulting array by moving them out from the dictionary. This is faster but the dictionary will become empty at the end of operation. The *destructive* parameter equal to 1 does not have any effect if the dictionary is used as a value of more than one variable. Thus it is safe to make this parameter equal to 1 as soon as the variable passed to the function as *dict* is no longer in use after the function call and there's no need to inspect the whole program to check if other variables may be affected. The parameter equal to 2 ignores the presence of multiple variables so the operation may cause unexpected results in other fragments of code that share the dictionary.

Parameters

dict
: The dictionary to scan.

destructive
: Flag that indicates whether the dictionary can be cleaned during the operation (0 = never clean, 1 = only if there is no other variables, 2 = clean unconditionally).

Return Types

The function returns a vector of even length.

See Also

`dict_new()`

`dict_zap()`

`dict_put()`

`dict_get()`

`dict_remove()`

`dict_inc_or_put()`

`dict_dec_or_remove()`

`dict_size()`

`dict_duplicate()`

`dict_list_keys()`

```
dict_destructive_list_rnd_keys()

dict_iter_rewind()

dict_iter_next()
```

dict_zap

dict_zap — Removes all data from the given dictionary

Synopsis

dict_zap (inout *dict* dictionary ,
in *destructive* integer) ;

Description

The function removes all items from the given dictionary.

If the *destructive* parameter is 0 or 1 then an error 22023 is signalled if the dictionary is used as a value of more than one variable. The parameter equal to 2 forces the function to ignore the presence of multiple variables; in this case the operation may cause unexpected results in other fragments of code that share the dictionary.

In most of cases, this function is used specifically for its side effect on other variables. With only one variable in use, one may simply set that variable to the value returned by dict_new () (or to NULL if the dictionary is no longer needed). If the dictionary is no longer referenced by any variable, it is destroyed and the occupied memory is freed. However one may wish to zap the dictionary in order to not worry about number of variables that refers to the dictionary, e.g., when dictionary is used only in some part of a long procedure.

Parameters

dict
: The dictionary to zap. If it is NULL then the function call has no effect and returned value is zero.

destructive
: Flag that indicates whether the dictionary can be cleaned if multiple references exist (0 and 1 = zap only if there is no other variables and signal an error otherwise, 2 = clean unconditionally).

Return Types

The function returns the number of intems in the dictionary before the operation.

See Also

dict_new()

dict_put()

dict_get()

dict_remove()

dict_inc_or_put()

dict_dec_or_remove()

dict_size()

dict_duplicate()

dict_to_vector()

dict_list_keys()

dict_destructive_list_rnd_keys()

dict_iter_rewind()

dict_iter_next()

disconnect_user

disconnect_user — Disconnect client connections of a given user

Synopsis

disconnect_user (in *username_pattern* string) ;

Description

disconnect_user disconnects clients whose username matches the username_pattern string given as an argument, and returns an integer value giving the number of clients disconnected. This can be used after DELETE USER or REVOKE statement to make sure that the affected user has no open connections.

Parameters

username_pattern
: A string pattern to match users to be disconnected. SQL wildcards including Virtuoso extensions may be used:

Return Values

The number of clients disconnected is returned.

Errors

Parameter data type checking errors

Examples

Example 24.95. Disconnect users

This is how the DBA can close all sessions for all users who name starts with db, including 'dba' which is the DBA itself.

```
SQL> disconnect_user ('db*');

*** Error 08S01: [Virtuoso Driver]CL065: Lost connection to server
at line 7 (7) of Top-Level: disconnect_user ('db*')
bash$
```

See Also:

username

dt_set_tz

dt_set_tz — modifies the timezone component of a datetime

Synopsis

dt_set_tz (in *dt* datetime ,
in *tz* integer) ;

Description

This modifies the timezone component of a datetime. The value remains equal for purposes of comparison but will look different when converted to a string. The timezone component is an offset from UTC in minutes. It can be retrieved with the timezone function.

Timezoneless: The function returns its first argument with unchanged GMT value but new timezone offset. Unlike `adjust_timezone()` , if the argument is timezoneless then no error is signaled.

Parameters

dt
: The original DATETIME .

tz
: INTEGER new timezone offset.

Return Types

A DATETIME with the new timezone set

dvector

lvector , fvector , dvector — return an array of either long, float or double

Synopsis

```
array lvector ( elt1 ,
                .... ,
                elt-n );

array fvector ( elt1 ,
                .... ,
                elt-n );

array dvector ( elt1 ,
                .... ,
                elt-n );
```

Description

These functions are like vector but return an array of either long, float or double whereas vector returns a generic, untyped array.

Example 24.96. Examples:

```
aref (lvector (1, 2), 1) = 1    is true.
```

end_result

end_result — End the current result set.

Synopsis

```
end_result (   );
```

Description

The `result_names()` predefines variables to be used in a result set to follow. The variables must be previously declared, from which the column data types are ascertained. This assigns the meta data but does not send any results. The `result()` function sends its parameters as a single row of results. These parameters should be compatible with those in the previous `result_names()` . The `end_result()` function can be used to separate multiple result sets. The `result_names()` can then be used to alter the result set structure.

The `result_names()` call can be omitted if the application already knows what columns and their types are to be returned.

See Also

`result()`

`result_names()`

either

either — conditionally return one of specified parameters

Synopsis

either (in *cond* any ,
in *arg1* any ,
in *arg2* any) ;

Description

either returns a copy of arg1 if cond is something else than integer 0 (zero). Otherwise, a copy of arg2 is returned.

Parameters

cond
: Anything

arg1
: Anything

arg2
: Anything

Return values

A copy of arg1 or arg2, which could be of any type.

Errors

Table 24.26. Errors signalled by `either` and `stringdate`

SQLState	Error Code	Error Text	Description

Examples

Example 24.97. Simple Example

```
SQL> select either (mod (1, 2), 'odd', 'even');
callret
VARCHAR
_______________________________________________________________________________

odd

1 Rows. -- 33 msec.
SQL> select either (isnull (strstr ('Simurg', 'imu')), 'imu not found', 'imu found');
callret
VARCHAR
_______________________________________________________________________________

imu found

1 Rows. -- 4 msec.
```

elh_get_handler

elh_get_handler — get localization function handler

Synopsis

```
encodedlang_handler_t * elh_get_handler ( encoding_handler_t * encoding ,
                                          language_handler_t * language ) ;
```

Description

Gets localization.

Parameters

encoding
: Name of the encoding to be used.

language
: Name of the text language

Return Types

Returns "encoded language" handler for optimized processing of texts on given language with given encoding as described in section Adding New Languages and Encodings Into Virtuoso of the main documentation. If *encoding* is NULL, or if there's no optimized handler for given encoding/language combination, then NULL will be returned, indicating failure, otherwise combination's handler will be returned.

Examples

Example 24.98. Returning a Handler Pointer

```
encodedlang_handler_t *elh =
  elh_get_handler(eh_get_handler("MY-ENCODING"),
    lh_get_handler("x-my-language")
  );
```

See Also

elh_load_handler

lh_get_handler

lh_load_handler

Adding New Languages and Encodings Into Virtuoso

elh_load_handler

elh_load_handler — load encoding handler into system

Synopsis

```
int elh_load_handler ( encodedlang_handler_t * new_handler );
```

Description

Loads given handler in table of handlers bound to encoding specified in the handler, using ISO 639 and RFC 1766 language IDs of the handler as keys for future table lookups. If another handler was already specified for given RFC 1766 id, table entry will be updated and will refer to new handler. If another handler was already specified for given ISO 639 id, it will be replaced only if new handler has ISO 639 language ID equal to its RFC 1766 ID.

Please do not load custom versions of 'x-any' and 'x-ftq-x-any' handlers.

Parameters

new_handler
: Pointer to the structure that lists names of the handler and its callback functions.

See Also

elh_get_handler

lh_get_handler

lh_load_handler

encode_base64

encode_base64 , decode_base64 — base64-encode/decode a string

Synopsis

encode_base64 (in *str* varchar) ;

decode_base64 (in *str* varchar) ;

Description

These functions convert strings from/to base64-encoding.

Parameters

str
: A varchar value.

Return Values

`encode_base64` returns a varchar containing base64-encoded data. `decode_base64` returns a varchar containing the result of the base64-decoding.

Examples

Example 24.99. Simple Example

Base64-encode a string

```
SQL> select encode_base64('Rutabaga');
callret
VARCHAR
_______________________________________________________________________________

UnV0YWJhZ2E=

1 Rows. -- 25 msec.
```

Example 24.100. Web example

Using encode-base64 with serialize to persist complex data

```
declare n any; n := vector ('a', 3.14157, 4711, 'Hic Iacet Corpus.');
http (sprintf ('<input type="hidden" name="saved" value="%s">', encode_base64 (serialize (n))));
```

To retrieve this data in VSP context when handling a form submit:

```
declare n any; n := deserialize (decode_base64 (get_keyword ('saved', params, '')));
```

See Also

`serialize`

equ

lt , lte , gt , gte , equ , neq — comparison functions

Synopsis

lt (in *arg1* any ,
in *arg2* any) ;

lte (in *arg1* any ,
in *arg2* any) ;

gt (in *arg1* any ,
in *arg2* any) ;

gte (in *arg1* any ,
in *arg2* any) ;

equ (*arg1* any ,
arg2 any) ;

neq (in *arg1* any ,
in *arg2* any) ;

Description

These functions return 1 if their first argument is less than (lt), less than or equivalent (lte), greater than (gt), greater than or equivalent (gte), equivalent (equ), or not equivalent (neq) to the second argument, respectively. If the arguments are not of the same type, then an appropriate type coercion is done for them before comparison.

These functions correspond to SQL query operators <, <=, >, >=, = and <> and are needed because the SQL syntax does not allow these operators to be used on the left side of `FROM` keyword in a `SELECT` statement.

Parameters

arg1 , arg2
: integer , float , double precision , varchar or NULL .

Return Values

An integer value of 1 or 0 is returned.

Examples

Example 24.101. Simple Example

```
lt('pata','pato')  -> 1 (Yes, 'pata' is less than 'pato')
gt('barbar','bar')  -> 1 (Yes, 'barbar' is greater than 'bar')
equ(17,17)    -> 1 (seventeen is seventeen)
equ(17,17.0)    -> 1 (regardless of number format)
equ(atof('17.0'),17.0))  -> 1 (as it seems be)
equ(atof('17.1'),17.1))  -> 0 (But not always! Beware!)
gte(1234,NULL)    -> 0 (No, 1234 is not "greater"
        than or equal to NULL)
lt(1234,NULL)    -> 1 (Instead, it is "less" than NULL)
```

exec

exec — dynamic execution of SQL returning state and result set

Synopsis

```
exec ( in str varchar ,
       out state varchar ,
       out message varchar ,
       in params any ,
       in maxrows integer ,
       out metadata vector ,
       out rows vector ,
       out cursor_handle long ) ;
```

Description

This function provides dynamic SQL capabilities in Virtuoso PL. The first argument is an arbitrary SQL statement, which may contain parameter placeholders. The function returns as output parameters a SQL state, error message, column metadata and result set rows if the statement is a select.

A stored procedure can be invoked by exec but a procedure's result set will not be received in the rows output parameter but rather sent to the client.

Parameters

str
: A varchar containing arbitrary SQL using ?'s for parameter markers.

state
: An output parameter of type varchar set to the 5 character SQL state if the exec resulted an error. Not set if an error is not present.

message
: An output parameter of type varchar set to SQL error message associated with the error. Not set if an error is not present.

params
: A vector containing the parameters for the SQL being executed. Element 0 corresponding to first ?, etc.

maxrows
: The integer maximum number of rows to retrieve in case of a statement returning a result set.

metadata
: An output parameter of type vector returning the metadata of the statement and its result.

rows
: An output array with one element per result row containing an array with the leftmost column as element 0 and so forth.

cursor_handle
: The cursor handle for use with related functions.

Examples

Example 24.102. Procedure Example

This stored procedure returns 1 if a given table is empty. An error such as a timeout or deadlock would be reported back to the caller as an exception. Exec always returns, no matter the type of exception. Thus it is also useful as a universal error catcher.

```
create procedure tb_is_empty (in tb varchar)
{
  declare state, msg, descs, rows any;
  state := '00000';
  exec (sprintf ('select 1 from %s', tb), state,
    msg, vector (), 1, descs, rows);

  if (state <> '00000')
    signal (state, msg);

  if (length (rows) = 0)
    return 1;

  else
    return 0;
}
```

See Also

rexecute()

exec_next()

exec_close()

close

exec_close — Closes cursor created by exec()

Synopsis

exec_close (in *cursor_handle* long);

Description

Closes the cursor opened by the exec() function.

Parameters

cursor_handle
: The cursor handle used.

See Also

`exec_next()`

`exec()`

exec_next

exec_next — Get next result from a result set

Synopsis

```
integer retcode exec_next ( in cursor_handle long ,
                            out sql_state varchar ,
                            out sql_error varchar ,
                            out row_array vector ) ;
```

Description

Use `exec_next()` to iterate over a result set produced by a statement run with `exec` .

open a cursor with exec .
loop over the results with exec_next
if exec_next does not return an error or SQL_NO_DATA_FOUND continue the loop with exec_next, otherwise
close the cursor with exec_close

Parameters

cursor_handle
: The long cursor handle as obtained from exec().

row_array
: An output vector that will contain the result columns.

sql_state
: Optional varchar output parameter for SQL state.

sql_error
: Optional varchar output parameter for any error message.

Return Values

All data is returned in output parameters.

See Also

`exec()` , `exec_close()` .

exec_result

exec_result — Returns a result set row to the calling procedure context

Synopsis

any **exec_result** (in *res_values_array* any);

Description

This function returns a result set row to the calling procedure's context, whether it is the client, exec function or a procedure view). The row's values are the elements of the supplied res_values_array vector.

Parameters

res_values_array
: This parameter can be one of two things:

Return Types

The row's values are the elements of the supplied res_values_array vector. For example: `exec_result (vector (1, 'a'))` will return a row of two columns: 1 and 'a'. This is similar to the `result()` function, but it uses an array instead of parameter list

Examples

Example 24.103. Result set rows

```
create procedure XX1 ()
{
  declare meta, _dt any;
  declare inx integer;
  exec ('select U_ID, U_NAME from SYS_USERS', null, null, null, 0, meta, _dt);
  inx := 0;

  exec_result_names (meta[0]);
  while (inx < length (_dt))
    {
      exec_result (_dt[inx]);
      inx := inx + 1;
    }
};
```

See Also

`exec()`

`rexec()`

`exec_result_names()`

`result_names()`

exec_result_names

exec_result_names — Supplies column details for procedure result set output.

Synopsis

exec_result_names (in *res_names_array* any) ;

Description

This function allows you to define the column details for result sets returned by procedures, in particular for use with the `exec()` function. This function is similar to `result_names()` .

Parameters

res_names_array
: This parameter can be one of two things:

Return Types

None.

Examples

Example 24.104. Result set column names

The procedure below uses the metadata from exec() to generate result set column names.

```
create procedure XX1 ()
{
  declare meta, _dt any;
  declare inx integer;
  exec ('select U_ID, U_NAME from SYS_USERS', null, null, null, 0, meta, _dt);
  inx := 0;

  exec_result_names (meta[0]);
  while (inx < length (_dt))
    {
      exec_result (_dt[inx]);
      inx := inx + 1;
    }
};
```

See Also

`exec()`

`rexec()`

`exec_result()`

exec_metadata

exec_metadata — Compiles a SQL statement and returns the metadata

Synopsis

exec_metadata (in *str* varchar ,
out *state* varchar ,
out *message* varchar ,
out *metadata* vector) ;

Description

This function provides dynamic SQL capabilities in Virtuoso PL. The first argument is an arbitrary SQL statement, which may contain parameter placeholders. The function returns as output parameters a SQL state, error message, column metadata if the statement is a select.

Parameters

str
: A varchar containing arbitrary SQL using ?'s for parameter markers.

state
: An output parameter of type varchar set to the 5 character SQL state if the exec resulted an error. Not set if an error is not present.

message
: An output parameter of type varchar set to SQL error message associated with the error. Not set if an error is not present.

metadata
: An output parameter of type vector returning the metadata of the statement and its result.

Errors

The function will generate a SQL 22023 error value if a supplied parameter is not of the type expected.

See Also:
List of SQL 22023 errors.

Examples

Example 24.105. Simple Use

```
create procedure get_meta (in str varchar)
{
  declare state, message, mdta any;
  state := '00000';
  exec_metadata (str, state, message, mdta);
  if (state <> '00000')
    signal (state, message);
  return mdta;
}
```

See Also

`exec()`

`exec_next()`

```
exec_close()
```

exec_score

exec_score — Compiles a SQL statement and returns the estimate time cost

Synopsis

```
exec_score ( in str varchar ,
             out state varchar ,
             out message varchar ) ;
```

Description

This function provides dynamic SQL capabilities in Virtuoso PL. The first argument is an arbitrary SQL statement, which may contain parameter placeholders. The function returns as output parameters a SQL state, error message and returns the estimate time cost in milliseconds.

Parameters

str
: A varchar containing arbitrary SQL using ?'s for parameter markers.

state
: An output parameter of type varchar set to the 5 character SQL state if the exec resulted an error. Not set if an error is not present.

message
: An output parameter of type varchar set to SQL error message associated with the error. Not set if an error is not present.

Return Types

The function returns a float number which is the calculated estimate time cost for the query execution.

Errors

The function will generate a SQL 22023 error value if a supplied parameter is not of the type expected.

See Also:
List of SQL 22023 errors.

Examples

Example 24.106. Simple Use

```
SQL> select exec_score ('select * from T1');
```

See Also

`exec()`

`exec_metadata()`

`exec_next()`

`exec_close()`

exp

exp — calculate exponent

Synopsis

exp (in *x* double precision) ;

Description

The function raises e to the power of *x* , it works with double precision floating point numbers, converts its argument to an IEEE 64-bit float and returns a result of that type.

Parameters

x
: double precision

Return Values

`exp` returns a IEEE 64-bit float .

Examples

Example 24.107. Simple Examples

```
SQL> select concat ('the EXP of 0.513513 is: ', cast (exp (0.513513) as varchar));
callret
VARCHAR
_______________________________________________________________________________

the EXP of 0.513513 is: 1.671152

1 Rows. -- 3 msec.
```

See Also

`log` , `log10` , `power` , `sqrt` ,

explain

explain — describe SQL statement compilation

Synopsis

```
explain ( in text varchar ,
          in cursor_type integer ) ;
```

Description

The explain function compiles a SQL statement and returns a description of that compilation as a result set, return value or parse tree. The result set consists of one VARCHAR column with one line of the description in each row. Any given line may be quite long, even several hundred characters.

The output is not a complete disassembly of the query graph, but it is detailed enough to show the join order, the sub-query structure, and the order of evaluation of query predicates, as well as the splitting of a distributed VDB query over different data sources.

The optional cursor type can be one of the SQL_CURSOR_<xx> constants, or one of the special values listed below. The default is 0, for FORWARD ONLY. The special values each have special effect, as listed. If the statement is a SELECT and the cursor type is not FORWARD ONLY, the auxiliary SQL statements used by the cursor implementation are shown.

Table 24.31. Cursor Type

Cursor Type	Effect
3	SQL_CURSOR_STATIC
2	SQL_CURSOR_DYNAMIC
1	SQL_CURSOR_KEYSET_DRIVEN
0	SQL_CURSOR_FORWARD_ONLY
-1	A result set is printed. Same effect as 0.
-2	A parse tree is returned.
-5	A result set is printed, and a diagnostic dump is delivered to the server's standard output. This can be viewed if the server was started with +foreground option.
-7	Cost estimates are returned, associated with any different join orders tried during the compilation.

Parameters

text
: varchar SQL statement

cursor_type
: integer cursor type

Return Types

Returns types vary with the specified cursor type.

Table 24.32. Return Types

Cursor Type	Return Type
0, 1, 2, 3, -5, -1	Result set.
-7	Return value.
-2	Parse tree.

Examples

Example 24.108. Simple Example Usage

Execute from ISQL:

```
SQL> explain('select * from sys_users');
REPORT
VARCHAR
_______________________________________________________________________________

{
from DB.DBA.SYS_USERS by SYS_USERS         17 rows
Key SYS_USERS  ASC ($47 ".U_ID", $46 ".U_NAME", $45 ".U_IS_ROLE", $44 ".U_FULL_NAME", $43 ".U_E_MAIL", $4
N_TIME", $39 ".U_ACCOUNT_DISABLED", $38 ".U_DAV_ENABLE", $37 ".U_SQL_ENABLE", $36 ".U_DATA", $35 ".U_METH
".U_PASSWORD_HOOK", $31 ".U_PASSWORD_HOOK_DATA", $30 ".U_GET_PASSWORD", $29 ".U_DEF_QUAL", $28 ".U_OPTS")

Current of: <$49 "<DB.DBA.SYS_USERS >" spec 5>
Select ($47 ".U_ID", $46 ".U_NAME", $45 ".U_IS_ROLE", $44 ".U_FULL_NAME", $43 ".U_E_MAIL", $42 ".U_PASSWO
 ".U_ACCOUNT_DISABLED", $38 ".U_DAV_ENABLE", $37 ".U_SQL_ENABLE", $36 ".U_DATA", $35 ".U_METHODS", $34 ".
_HOOK", $31 ".U_PASSWORD_HOOK_DATA", $30 ".U_GET_PASSWORD", $29 ".U_DEF_QUAL", $28 ".U_OPTS", <$49 "<DB.D
}

8 Rows. -- 10 msec.
```

Example 24.109. Example with type -1

Execute from ISQL:

```
SQL> explain('sparql
select *
from <http://myopenlink.net/dataspace/person/kidehen#this>
where {?s ?p ?o} limit 10',-1);
REPORT
VARCHAR
_______________________________________________________________________________

{

Precode:
      0: $27 "hen#this" := Call iri_to_id_nosignal (<constant (http://myopenlink.net/dataspace/person/kid
      5: BReturn 0
from DB.DBA.RDF_QUAD by RDF_QUAD        0.4 rows
Key RDF_QUAD  ASC ($31 "s-1-1-t0.S", $30 "s-1-1-t0.P", $29 "s-1-1-t0.O")
 inlined <col=412 G = $27 "hen#this">

Current of: <$33 "<DB.DBA.RDF_QUAD s-1-1-t0>" spec 5>

After code:
      0: $34 "s" := Call id_to_iri ($31 "s-1-1-t0.S")
      5: $35 "p" := Call id_to_iri ($30 "s-1-1-t0.P")
      10: $36 "o" := Call __rdf_sqlval_of_obj ($29 "s-1-1-t0.O")
      15: BReturn 0
Select (TOP <constant (10)>) ($34 "s", $35 "p", $36 "o", <$33 "<DB.DBA.RDF_QUAD s-1-1-t0>" spec 5>)
}

18 Rows. -- 10 msec.
```

Example 24.110. Example with type -2

Execute from ISQL:

```
SQL> select dbg_obj_print(explain('sparql
select *
```

```
from <http://myopenlink.net/dataspace/person/kidehen#this>
where {?s ?p ?o} limit 10',-2));
callret
VARCHAR
_______________________________________________________________________________

0

1 Rows. -- 20 msec.
```

As result on the Server console will be shown this output:

```
(100 (122 0 10 0 0 0 ) ((21 (609 'id_to_iri' ((201 's-1-1-t0' 'S' ) ) ) 0 's' 0 ) (21 (609 'id_to_iri' ((
__rdf_sqlval_of_obj' ((201 's-1-1-t0' 'O' ) ) ) 0 'o' 0 ) ) 0 (106 ((107 (200 'DB.DBA.RDF_QUAD' 's-1-1-t0
 'iri_to_id_nosignal' ('http://myopenlink.net/dataspace/person/kidehen#this' ) ) 0 0 0 0 0 0 ) 0 0 0 0 (9
```

Example 24.111. Example with type -5

Execute from ISQL:

```
SQL> explain('sparql select * from <http://myopenlink.net/dataspace/person/kidehen#this> where {?s ?p ?o}
REPORT
VARCHAR
_______________________________________________________________________________

{

Precode:
      0: $27 "hen#this" := Call iri_to_id_nosignal (<constant (http://myopenlink.net/dataspace/person/kid
      5: BReturn 0
from DB.DBA.RDF_QUAD by RDF_QUAD        0.4 rows
Key RDF_QUAD  ASC ($31 "s-1-1-t0.S", $30 "s-1-1-t0.P", $29 "s-1-1-t0.O")
 inlined <col=412 G = $27 "hen#this">

Current of: <$33 "<DB.DBA.RDF_QUAD s-1-1-t0>" spec 5>

After code:
      0: $34 "s" := Call id_to_iri ($31 "s-1-1-t0.S")
      5: $35 "p" := Call id_to_iri ($30 "s-1-1-t0.P")
      10: $36 "o" := Call __rdf_sqlval_of_obj ($29 "s-1-1-t0.O")
      15: BReturn 0
Select (TOP <constant (10)>) ($34 "s", $35 "p", $36 "o", <$33 "<DB.DBA.RDF_QUAD s-1-1-t0>" spec 5>)
}

18 Rows. -- 40 msec.
```

As result on the Server console will be printed this output:

```
New best dt0 is:
sequence for dt0 cost       6.9 (0.0135757 msec):RDF_QUAD as t1  on RDF_QUAD ,
{
   dt dt0
  unit       6.9 (0.0135757 msecs) arity       0.4 reached       1
Out cols :
    call id_to_iri: (609 'id_to_iri' ((201 't1' 'S' ) ) )

    call id_to_iri: (609 'id_to_iri' ((201 't1' 'P' ) ) )

    call __rdf_sqlval_of_obj: (609 '__rdf_sqlval_of_obj' ((201 't1' 'O' ) ) )

    call iri_to_id_nosignal: (609 'iri_to_id_nosignal' ('http://myopenlink.net/dataspace/person/kidehen#t

    Table DB.DBA.RDF_QUAD(s-1-1-t0 t1) by RDF_QUAD   Reached         1 unit       4.6 (0.00907572 msecs)
  col preds:
        pred artm (201 't1' 'G' ) = (609 'iri_to_id_nosignal' ('http://myopenlink.net/dataspace/person/ki
      out cols:  O  P  S

    call id_to_iri: (609 'id_to_iri' ((201 't1' 'S' ) ) )
```

```
    call id_to_iri: (609 'id_to_iri' ((201 't1' 'P' ) ) )

    call __rdf_sqlval_of_obj: (609 '__rdf_sqlval_of_obj' ((201 't1' 'O' ) ) )
}
```

Example 24.112. Example with type -7

Execute from ISQL:

```
SQL> select explain('sparql
select *
from <http://myopenlink.net/dataspace/person/kidehen#this>
where {?s ?p ?o} limit 10',-7);
callret
VARCHAR
_______________________________________________________________________________

6.957722663879395

1 Rows. -- 0 msec.
```

Example 24.113. Example with vsp printing the result from executing explain:

```
<?vsp
  declare meta, data any;
  exec ('explain (?)', null, null, vector ('select * from sys_users'),  0, meta, data);
  foreach (any row in data) do
  {
     http_value (row[0], 'p');
  }
?>
```

Equivalent is:

```
<?vsp
  declare meta, data any;
  exec ('explain (?)', null, null, vector ('select * from sys_users'), 0, meta, data);
  for (declare i,l int, i := 0, l := length (data); i < l; i := i + 1)
   {
     http_value (data[i][0], 'p');
   }
?>
?>
```

As result, after accessing this sample vsp, the following output will be printed:

```
{

from DB.DBA.SYS_USERS by SYS_USERS 17 rows

Key SYS_USERS ASC ($47 ".U_ID", $46 ".U_NAME", $45 ".U_IS_ROLE", $44 ".U_FULL_NAME", $43 ".U_E_MAIL", $42

Current of: <$49 "<DB.DBA.SYS_USERS >" spec 5>

Select ($47 ".U_ID", $46 ".U_NAME", $45 ".U_IS_ROLE", $44 ".U_FULL_NAME", $43 ".U_E_MAIL", $42 ".U_PASSWO

}
?>
```

See Also

Query Plans.

file_delete

file_delete — Delete a file from the file system

Synopsis

```
integer file_delete ( in path varchar ,
                      in silent integer ) ;
```

Description

This function requires dba privileges.

This function is used to delete a file from file system. This function has a silent mode, where no errors will be signalled upon failure.

Parameters

path
: Path name of the file to delete.

silent
: Flag which if true (integer 1) will suppress signalling file system errors.

Errors

Parameter data type checking errors

file_open

file_open — returns the contents of a file.

Synopsis

```
varchar file_open ( in path varchar );
```

Description

Returns the contents of a file. The path is relative to the working directory of the database server.

Parameters

path
: Path name of the file to read.

Examples

Example 24.114. Simple Use

For loading N3 and Turtle files you can use:

```
SQL>DB.DBA.TTLP_MT (file_open('tmp/file1.n3'), 'base uri', 'graph uri', 255);
```

See Also

`file_delete()`

`DB.DBA.TTLP()`

`DB.DBA.TTLP_MT()`

`DB.DBA.TTLP_MT_LOCAL_FILE()`

ft_set_file

ft_set_file — associates table with tabular data structure represented by CSV document content with external CSV file.

Synopsis

```
varchar ft_set_file ( in tb varchar ,
                      in fname varchar ,
                      in delimiter varchar ,
                      in newline varchar (default '\n') ,
                      in esc varchar (default NULL) ,
                      in skip_rows integer (default 0) ) ;
```

Description

Associates table with tabular data structure represented by CSV document content with external CSV file.

Parameters

tb
: The name of the table that has tabular data structure represented by CSV document content.

fname
: External CSV file name

delimiter
: Delimiter to be used.

newline
: Set new line encoding. By default is '\n'.

esc
: Set escape character. By default is NULL.

skip_rows
: Use to skip n rows from the begining. By default is 0. When set to 1 for ex., treats the first row as header and skips it. When set to 2 skips 2 rows etc.

Examples

Example 24.115. Using a Table as a Placeholder for CSV Attachment

In this example a SQL TABLE is created and then associated with an external CSV file. This file is situated in a host operating system location that is accessible to the Virtuoso instance using relative (to instance .db file) or full path file name reference (constrained by `DirsAllowed` entry in `Virtuoso INI`).

CSV filename: contacts.csv (which can also be referred to as file:contacts.csv) .

File Content:

```
Id,Fname,Sname,Age
0,John,Smith,48
1,Anna,Clarks,62
2,Rojer,Danrette,27
3,Kate,Sigton,56
4,Tim,Craft,41
```

CSV File to SQL Table association steps:

1. CREATE an empty TABLE , using the SQL command:

```
CREATE TABLE csv.tutorial.contacts
  (
    Id INTEGER NOT NULL,
    Fname VARCHAR(20),
    Sname VARCHAR(20),
    Age INTEGER,
    CONSTRAINT demo_table_pk PRIMARY KEY (Id)
  );
```

   a. Notes:

      i. There is no special object type for File Tables corresponding to the tabular data structure represented by the CSV document content;
      ii. To create a File Table, a user must have DBA group privileges. In addition, access to the referenced file is subject the same file system access functions (e.g., file_to_string and DirsAllowed INI file settings) that govern the whole Virtuoso instance.

2. Use Virtuoso `ft_set_file` function to associate the above newly created internal TABLE (first procedure argument) with the external CSV file (second procedure argument, in the form of a file name relative to the Virtuoso instance); optional arguments may be used to specify delimiter, newline, and escape characters, respectively:

```
ft_set_file ('csv.tutorial.contacts', 'contacts.csv', ',', 1) ;
```

   a. Notes:

      i. The text in each field is parsed according to the data type declared for the column whose position in the CREATE TABLE statement corresponds to that field's position on the line. The parsing is as by the SQL CAST function from a VARCHAR value. If the CAST fails, the line is silently ignored.
      ii. The newline and escape characters must be single character strings. A newline or escape character following the escape character will be added to the parsed input as a literal character, without its special interpretation.
      iii. Each column in the CSV file is expected to end with the delimiter character. A field of zero length is considered a SQL NULL value; i.e., if two delimiters are adjacent, the field is considered NULL. Likewise, if a line begins with the delimiter, the first field is considered NULL.

3. Check the inserted data in the csv.tutorial.contacts table:

```
SQL>SELECT * FROM csv.tutorial.contacts;

Id                 Fname      Sname        F_AGE
INTEGER NOT NULL   VARCHAR    VARCHAR      INTEGER
_______________________________________________________________________

0                  John       Smith        48
1                  Anna       Clarks       62
2                  Rojer      Danrette     27
3                  Kate       Sigton       56
4                  Tim        Craft        41

5 Rows. -- 15 msec.
```

See Also

File Tables

`file_delete`

`DB.DBA.TTLP`

`DB.DBA.TTLP_MT`

`DB.DBA.TTLP_MT_LOCAL_FILE`

file_dirlist

file_dirlist — Returns list with content of file system directory

Synopsis

```
file_dirlist ( in path varchar ,
               in what integer ,
               out error any ) ;
```

Description

This function requires dba privileges.

This returns the list of the contents of a given file system directory.

Parameters

path
: string containing valid path to directory in file system

what
: flag designating what to return in the list: 0 - directories, 1 - regular files, 2 - symbolic links, 3 - all entries

error
: A variable for returning error messages. If supplied, errors are not signalled.

Return Types

an array of strings containing directory entries.

Errors

Table 24.33. Errors signalled by

SQLState	Error Code	Error Text	Description
42000	FA016	Access to %s is denied due to access control in ini file	
39000	FA017	Path string is too long.	
42000	FA018	Access to %s is denied due to access control in ini file	
39000	FA019	Path string is too long.	

See Also

`file_unlink()` `file_delete()` `file_mkdir()` `file_mkpath()` `file_stat()`

file_mkdir

file_mkdir — Creates a directory in the file system

Synopsis

```
file_mkdir ( in path varchar ,
             out error any ) ;
```

Description

This function requires dba privileges.

This function creates a file system directory.

Parameters

path
: A string containing path to the directory to be created, absolute or relative to server working directory.

error
: The error message is stored into this variable. If present, errors are not signalled.

Return Types

On success returns zero.

Errors

Table 24.34. Errors signalled by

SQLState	Error Code	Error Text	Description
42000	FA014	Access to %s is denied due to access control in ini file	
42000	FA015	Access to %s is denied due to access control in ini file	

See Also

`file_delete()` `file_unlink()` `file_mkpath()` `file_stat()`

file_mkpath

file_mkpath — Make a directory chain

Synopsis

```
file_mkpath ( in path varchar ,
              inout istest integer ) ;
```

Description

This function requires dba privileges.

This is to create a directory chain i.e. 'a/b/c/d', where one or more elements in the path may not exist.

Parameters

path
: string containing file system path to be crated as directory chain

istest
: Variable for returning the error message. If specified, errors are not signalled.

Return Types

On success returns zero.

Errors

Table 24.35. Errors signalled by

SQLState	Error Code	Error Text	Description
42000	FA014	Access to %s is denied due to access control in ini file	
42000	FA015	Access to %s is denied due to access control in ini file	
42000	FA116	Abnormally long path is passed as argument to sys_mkpath()	

See Also

`file_unlink() file_delete() file_mkdir() file_mkpath() file_stat()`

file_stat

file_stat — get various OS statistics about a file

Synopsis

file_stat (in *path* varchar ,
in *what* integer) ;

Description

`file_stat` returns various information for an OS file by calling stat () system call and converting the relevant value to a varchar . The path is relative to the server's working directory. The what is an integer value selecting what information to return. If you don't supply second argument to the function it defaults to mode = 0.

The DirsAllowed and DirsDenied lists in Parameters section of the virtuoso configuration file (virtuoso.ini by default) are not used to control disk access.

Parameters

path
: varchar relative path.

what
: integer info to be returned.

Examples

Example 24.116. Simple example

Get the size of the virtuoso INI file

```
SQL>select file_stat (virtuoso_ini_path(), 1);
callret
VARCHAR
_______________________________________________________________________________

958
```

See Also

`file_delete`

`os_chmod`

`os_chown`

`virtuoso_ini_path`

file_to_string

file_to_string — returns the contents of a file as a varchar

Synopsis

```
varchar file_to_string ( in path varchar );
```

Description

This function requires dba privileges.

Returns the contents of a file as a varchar value. The file's length is limited to 16 MB. The path is relative to the working directory of the database server.

Parameters

path
: Path name of the file to read.

Errors

Parameter data type checking errors

file_to_string_output

file_to_string_output — get contents of a file as a string output stream

Synopsis

file_to_string_output (in *file* varchar ,
in *from* integer ,
in *to* integer) ;

Description

This function requires dba privileges.

This function returns a string output stream initialized to contain the text of the file or its segment, on local file system path relative to the server's working directory.

`file_to_string_output` can handle longer files than `file_to_string` and the resulting string output, if too long to be converted into a varchar, can be stored inside a blob.

Access controls in the server configuration file apply. An attempt to access a file in a directory where access is not explicitly allowed will signal an error.

Parameters

file
: a varchar path relative to server's working directory.

from
: an optional integer byte offset of the start of the segment to extract. Defaults to 0.

to
: an optional integer byte offset of the end of the requested segment. Defaults to file length.

Examples

Example 24.117. Insert file contents into a table

This example shows how to insert file contents into a table `file_table` with two columns.

```
create table
file_table (
  ft_name varchar,
  ft_cont long varbinary,
  primary key (ft_name));

create procedure
insert_files (in fname varchar)
{
  declare strout_handle any;

  strout_handle := file_to_string_output (fname);
  insert into file_table (ft_name, ft_cont)
    values (fname, strout_handle);

  strout_handle := file_to_string_output (fname, 10);
  insert into file_table (concat (ft_name, '_1'), ft_cont)
    values (fname, strout_handle);

  strout_handle := file_to_string_output (fname, 10, 20);
  insert into file_table (concat (ft_name, '_2'), ft_cont)
    values (fname, strout_handle);
```

```
};

insert_file ('foo.dat');
```

See Also

`file_to_string`, `string_to_file`, `string_output`

file_unlink

file_unlink — Delete a file from the file system

Synopsis

file_unlink (in *path* varchar) ;

Description

This function requires dba privileges.

This function deletes a file from the file system. sys_unlink is a synonym of this function.

Parameters

path
: Path name of the file to delete.

Return Types

On success returns zero. Otherwise signals an error.

Errors

Table 24.36. Errors signalled by

SQLState	Error Code	Error Text	Description
42000	FA003	Access to %s is denied due to access control in ini file	
42000	FA004	Access to %s is denied due to access control in ini file	
42000	SR426	Permission is denied for the file '%s' in sys_unlink()	
42000	SR427	Path name '%s' too long in sys_unlink()	
42000	SR428	A directory component in '%s' does not exist or is a dangling symbolic link in sys_unlink()	
42000	SR429	A component used as a directory in '%s' is not, in fact, a directory in sys_unlink()	
42000	SR430	'%s' refers to a directory in sys_unlink()	
42000	SR431	Insufficient kernel memory was available in sys_unlink()	
42000	SR432	'%s' refers to a file on a read-only filesystem in sys_unlink()	
42000	SR433	Too many symbolic links were encountered in translating '%s' in sys_unlink()	
42000	SR434	An I/O error occurred in sys_unlink()	

See Also

`file_delete()` `file_mkdir()` `file_mkpath()` `file_stat()`

fk_check_input_values

fk_check_input_values — alter default foreign key checking behavior

Synopsis

DB.DBA. **fk_check_input_values** (in *mode* integer);

Description

mode = 1 - on

mode = 0 - off

Enforcing foreign key constraints is enabled by default. This function allows globally disabling it without however disabling all triggers. This may be useful for large data imports or other special circumstances. The return value is the previous state of this setting, 0 for off, 1, for on. The effect of this function is persistent and survives server restart.

floor

floor — Round a number to negative infinity.

Synopsis

floor (in *x* double precision);

Description

floor calculates the largest integer smaller than or equal to *x* .

Parameters

x
: double precision

Return Values

floor returns a 32-bit integer .

Examples

Example 24.118. Simple Examples

```
SQL> select floor (-0.513513), floor (0.513513), floor (123.4567);
callret   callret   callret
INTEGER   INTEGER   INTEGER
_______________________________________________________________________________

-1        0         123

1 Rows. -- 3 msec.
```

See also

ceiling

ftp_get

ftp_get — FTP get command; Virtuoso FTP client

Synopsis

```
integer ftp_get ( in server varchar ,
                  in user varchar ,
                  in pass varchar ,
                  in remote_file_name varchar ,
                  in local_file_name varchar ,
                  in is_pasv integer ) ;
```

Description

Virtuoso has FTP client functionality, that can be used inside Virtuoso/PL. This Virtuoso function mimics the FTP get command. As with any PL, this can be combined with Web Services and SOAP.

Parameters

server
: The remote server address or IP Address.

user
: The username for authentication at the FTP server.

pass
: The accompanying password for authentication at the FTP server.

remote_file_name
: Full path and file name to the file on the FTP server to be downloaded.

local_file_name
: The full path and file name of the local save point. This is relative to the server root directory. This directory must be included in DirsAllowed ACL list in the Virtuoso.INI file.

is_pasv
: Flag to specify whether to use passive mode. This can be one "1" for passive mode, or zero "0" for active mode. One "1" is assumed by default.

Return Types

This function returns the size of the file written to the local file system.

Errors

This function will return any errors returned by the remote FTP server.

Examples

Example 24.119. Retrieving a file from an FTP server

To get the file virtuoso30.tar.gz from the ftp server ftp.openlinksw.com, one can use:

```
select ftp_get ('ftp.openlinksw.com', 'user_name', 'password', 'virtuoso30.tar.gz', 'virtuoso30.tar.gz');
```

This will download the file which occurs on the root directory of the server, and save it to the local server root directory.

See Also

`ftp_put()`

`ftp_ls()`

ftp_ls

ftp_ls — FTP dir command; Virtuoso FTP client

Synopsis

```
any ftp_ls ( in server varchar ,
             in user varchar ,
             in pass varchar ,
             in remote_dir_name varchar ,
             in is_pasv integer ) ;
```

Description

Virtuoso has FTP client functionality, that can be used inside Virtuoso/PL. This Virtuoso function mimics the FTP dir command. As with any PL, this can be combined with Web Services and SOAP.

Parameters

server
: The remote server address or IP Address.

user
: The username for authentication at the FTP server.

pass
: The accompanying password for authentication at the FTP server.

remote_dir_name
: The full path of the remote directory to list the contents of.

is_pasv
: Flag to specify whether to use passive mode. This can be one "1" for passive mode, or zero "0" for active mode. One "1" is assumed by default.

Return Types

This function returns a vector of descriptions from the result of performing the dir command on the remote server. If errors occur then these will be returned instead.

Errors

This function will return any errors returned by the remote FTP server.

Examples

Example 24.120. Listing files on the remote FTP server

The following command will send the vector of the descriptions of the files in the virtuoso30 directory on the remote sever to the Virtuoso debug console, assuming the server was started with the -d or +debug option:

```
select dbg_obj_print(ftp_ls ('ftp.openlinksw.com', 'user_name', 'password', 'virtuoso30'));
```

See Also

`ftp_put()`

`ftp_get()`

ftp_put

ftp_put — FTP put command; Virtuoso FTP client

Synopsis

```
integer ftp_put ( in server varchar ,
                  in user varchar ,
                  in pass varchar ,
                  in local_file_name varchar ,
                  in remote_file_name varchar ,
                  in is_pasv integer ) ;
```

Description

Virtuoso has FTP client functionality, that can be used inside Virtuoso/PL. This Virtuoso function mimics the FTP put command. As with any PL, this can be combined with Web Services and SOAP.

Parameters

server
: The remote server address or IP Address.

user
: The username for authentication at the FTP server.

pass
: The accompanying password for authentication at the FTP server.

local_file_name
: The full path and file name of the local file to be uploaded. This is relative to the server root directory. This directory must be included in DirsAllowed ACL list in the Virtuoso.INI file.

remote_file_name
: Full path and file name to the file on the FTP server to be uploaded.

is_pasv
: Flag to specify whether to use passive mode. This can be one "1" for passive mode, or zero "0" for active mode. One "1" is assumed by default.

Return Types

This function returns either 1 for success, or the error returned from the server.

Errors

This function will return any errors returned by the remote FTP server.

Examples

Example 24.121. Uploading a file to an FTP server

To upload the file virtuoso30.tar.gz to the ftp server ftp.openlinksw.com, one can use:

```
select ftp_put ('ftp.openlinksw.com', 'user_name', 'password', 'virtuoso30.tar.gz', 'virtuoso30.tar.gz');
```

This will upload the file that occurs on the local server root directory, and save it to the remote servers root directory.

See Also

`ftp_put()`

`ftp_ls()`

gz_file_open

gz_file_open — returns the contents of a gzipped file

Synopsis

```
varchar gz_file_open ( in path varchar );
```

Description

Returns the contents of a file. The path is relative to the working directory of the database server.

Parameters

path
: Path name of the file to read.

Examples

Example 24.122. Simple Use

For loading gzipped N3 and Turtle files you can use:

```
SQL>DB.DBA.TTLP_MT (gz_file_open('path/myfile.n3.gz'), 'base uri', 'graph uri', 255);
```

Note that the last parameter means the parser uses relaxed parsing rules, the default value of 0 means strict syntax rules.

See Also

`DB.DBA.TTLP()`

`DB.DBA.TTLP_MT()`

`DB.DBA.TTLP_MT_LOCAL_FILE()`

get_csv_row

get_csv_row — This function takes a string_session containing CSV data, parses a row and returns a vector with field values.

Synopsis

get_csv_row (in *ses* any ,
in *delim* char ,
in *quote* char) ;

Description

This function takes a string_session containing CSV data, parses a row and returns a vector with field values.

Calling the function in a loop with same input will parse file line by line.

The function is also making a basic prediction of field types for varchar, integer and float values, so the result data may have one of these. If come other datatype is need to be produced the caller can convert the string using the BIF like cast sprintf_inverse and so on.

Parameters

ses
: string_session containing CSV data

delim
: Optional argument to handle tab, space delimited etc. formats.

quote
: Optional argument to handle tab, space delimited etc. formats.

Return Types

Returns a vector with field values.

Examples

Example 24.123. Simple Use

The following example prints on the Virtuoso Server console the CSV columns names:

```
SQL>create procedure my_csv (in file_name varchar)
{
   dbg_obj_print(get_csv_row(file_open('tmp/MyContacts.csv')));
}
;

Done. -- 10 msec.

SQL>select my_csv('tmp/MyContacts.csv');
callret
VARCHAR
_______________________

0

1 Rows. -- 10 msec.

-- and on the Virtuoso Server console will be shown:

('First Name;Last Name;Middle Name;Name;Nickname;E-mail Address;Home Street;Home City;Home Postal Code;Ho
```

```
Fax;Mobile Phone;Personal Web Page;Business Street;Business City;Business Postal Code;Business State;Busi
s Phone;Business Fax;Pager;Company;Job Title;Department;Office Location;Notes' )
```

See Also

csv_load_file()

csv_load()

csv_parse()

csv_cols_def()

csv_table_def()

csv_load_file

csv_load_file — Imports a CSV file into table.

Synopsis

csv_load_file (in *file_name* varchar ,
in *from_line* integer ,
in *to_line* integer ,
table_name varchar ,
in *trx_log_mode* integer ,
in *opts* any) ;

Description

Imports a CSV file into table.

Parameters

file_name
: Given as an input file_name will be parsed as CSV where it will insert into the table specified as table_name the lines between from_line and to_line offsets.

from_line
: Default - 0, This means counted from the begining.

to_line
: Default - null, This means counted to the end.

table_name
: The table the data to be inserted into.

trx_log_mode
: Default - 2. The trx_log_mode is used to do row auto commit and to log or not the transaction. The default mode is to do not log and do auto commit.

opts
: Default - null. The opts paramater is used to specify the delimiter and quote. It should look like this:

Errors

If table is not suitable in respect of number of columns or type of columns error will be signaled.

Examples

Example 24.124. Simple Use

```
SQL>create procedure my_csv_lf (in file_name varchar)
{
  declare tb_name varchar;
  declare ar any;

  -- to get the name of the table
  ar := csv_table_def(file_name);
  ar := split_and_decode(ar, 0,'\0\0 ');
  tb_name := trim (ar[2], '"');

  dbg_obj_print(tb_name);
```

```
  -- to create the table
  exec(csv_table_def(file_name));

  -- loading the data
  csv_load_file(file_name,0, null, concat('DB.DBA.', tb_name));
}
;

Done. -- 10 msec.

SQL>select my_csv_lf('tmp/MyContacts.csv');

callret
VARCHAR
_______________________________________________

0

1 Rows. -- 231 msec.

SQL>select * from DB.DBA.tmp_MyContacts_csv;
First_Name_Last_Name_Middle_Name_Name_Nickname_E_mail_Addre
VARCHAR
_______________________________________________________________________________

First Name;Last Name;Middle Name;Name;Nickname;E-mail Address;Home Street;Home City;Home Postal Code;Home
x;Mobile Phone;Personal Web Page;Business Street;Business City;Business Postal Code;Business State;Busine
Phone;Business Fax;Pager;Company;Job Title;Department;Office Location;Notes
;;; Virtuoso Nightly Builds on  oplusaix2 ;;virtuoso-nightly-builds@openlinksw.com;;;;;;;;;;;;;;;;;;;;;;;
......

202 Rows. -- 341 msec.
```

See Also

get_csv_row

csv_load

csv_parse

csv_cols_def

csv_table_def

attach_from_csv

attach_from_csv — Attaches CSV files as virtual tables in Virtuoso.

Synopsis

attach_from_csv (in *tb* varchar ,
in *fname* any ,
in *delimiter* varchar (default '') ,
in *newline* varchar (default '\n') ,
in *esc* varchar (default null) ,
in *skip_rows* int (default 1) ,
in *pkey_columns* any (default null)) ;

Description

This function attaches CSV files as virtual tables in Virtuoso, enabling them to be queried as if local to the Virtuoso database schema.

Parameters

tb
: Name of the Virtuoso SQL table to be associated with the attached CSV file

fname
: Name of the CSV file, which must be situated in a file system location within the scope of 'DirsAllowed' INI setting, in one of the following forms:

delimiter
: Specifies a single character to be treated as the field delimiter.

newline
: Specifies a single character to be treated as newline.

esc
: Specifies a single character to be treated as escape.

skip_rows
: Specifies a number of rows to be ignored as instance data. Default is '1', this typically being the CSV header row.

pkey_columns
: Specifies which ordinal column(s) of the CSV to use as the primary key of the virtual table.

Errors

Issues associated with CSV file structure, in regards to virtual table creation suitability, will be reported back as errors.

Examples

Example 24.125. Simplified CSV File Attachment

This example demonstrates how to directly attach external CSV file to Virtuoso that functions like any other internal TABLE:

1. Create CSV file named `contacts1.csv` :

```
$ cat > contacts1.csv
Id,Fname,Sname,Age
```

```
0,John,Smith,48
1,Anna,Clarks,62
2,Rojer,Danrette,27
3,Kate,Sigton,56
4,Tim,Craft,41
```

2. Place the created `contacts1.csv` file within scope of `DirsAllowed` INI setting.
3. Call the attach_from_csv function from iSQL or Virtuoso Conductor UI (http://host:port/conductor) . Note: As the pkey_columns parameter value is not specified, by default the virtual table will have no Primary Key:

```
SQL> attach_from_csv ('csv.tutorials.contacts1', 'file:contacts1.csv', ',', '\n', null, 1);
Done. -- 109 msec.
```

4. Check the inserted data in the csv.tutorials.contacts1 table:

```
SQL> SELECT * FROM csv.tutorials.contacts1 ;
Id          Fname         Sname       Age
INTEGER     VARCHAR       VARCHAR     INTEGER
_____________________________________________

0           John          Smith       48
1           Anna          Clarks      62
2           Rojer         Danrette    27
3           Kate          Sigton      56
4           Tim           Craft       41

5 Rows. -- 78 msec.
```

5. Check the Primary Key columns:

```
SQL> primarykeys csv.tutorials.contacts1;
Showing SQLPrimaryKeys of tables like 'csv.tutorials.contacts1', tabletype/colname like 'NULL'
TABLE_QUALIFIER  TABLE_OWNER  TABLE_NAME   COLUMN_NAME   KEY_SEQ   PK_NAME  ROOT_QUALIFIER  ROOT_C
VARCHAR          VARCHAR      VARCHAR      VARCHAR       SMALLINT  VARCHAR  VARCHAR         VARCHA
_____________________________________________________________________________________________

0 Rows. -- 47 msec.
```

Example 24.126. Simplified CSV File Attachment with Compound Key Generation

In this example a CSV File System is not only attached to Virtuoso, but as part of the process a Primary Key is generated using the first and second ordinal columns from the CSV file content:

1. Create CSV file named `contacts2.csv`:

```
$ cat > contacts2.csv
Fname,Sname,Age
John,Smith,48
Anna,Clarks,62
Rojer,Danrette,27
Kate,Sigton,56
Tim,Craft,41
```

2. Place the created `contacts2.csv` file within scope of `DirsAllowed` INI setting.
3. Call the attach_from_csv function from iSQL or Virtuoso Conductor UI (http://host:port/conductor):

```
SQL> attach_from_csv ('csv.tutorials.contacts2', 'file:contacts2.csv', ',', '\n', null, 1, vector(
Done. -- 109 msec.
```

4. Check the inserted data in the csv.tutorials.contacts2 table:

```
SQL> SELECT * FROM csv.tutorials.contacts2 ;
Fname         Sname       Age
VARCHAR       VARCHAR     INTEGER
_____________________________________________

John          Smith       48
Anna          Clarks      62
Rojer         Danrette    27
Kate          Sigton      56
Tim           Craft       41
```

```
5 Rows. -- 78 msec.
```

5. Check the Primary Key columns:

```
SQL> primarykeys csv.tutorials.contacts2;
Showing SQLPrimaryKeys of tables like 'csv.tutorials.contacts2', tabletype/colname like 'NULL'
TABLE_QUALIFIER  TABLE_OWNER  TABLE_NAME               COLUMN_NAME  KEY_SEQ   PK_NAME
VARCHAR          VARCHAR      VARCHAR                  VARCHAR      SMALLINT  VARCHAR
_______________________________________________________________________________

DB               DBA          csv.tutorials.contacts2  Fname        1         csv.tutorials.contac
DB               DBA          csv.tutorials.contacts2  Sname        2         csv.tutorials.contac

2 Rows. -- 32 msec.
```

Example 24.127. Simplified CSV File Attachment with Composite Key Generation

In this example a CSV File System is not only attached to Virtuoso, but as part of the process a Primary Key is generated using the second (`CompanyID`), third (`Fname`) and forth (`Sname`) ordinal columns. In this example the CompanyID column (which uniquely identifies a company) is combined with the `Fname` and `Sname` columns to create a primary key for each contact:

1. Create CSV file named `contacts3.csv`:

```
$ cat > contacts3.csv
CompanyName,CompanyID,Fname,Sname,Age
MLogistic,12,John,Smith,48
ZiAirLines,13,Anna,Clarks,62
MLogistic,12,Rojer,Danrette,27
MLogistic,12,Kate,Sigton,56
ZiAirLines,13,Tim,Craft,41
```

2. Place the created `contacts3.csv` file within scope of `DirsAllowed` INI setting.
3. Call the attach_from_csv function from iSQL or Virtuoso Conductor UI (http://host:port/conductor):

```
SQL> attach_from_csv ('csv.tutorials.contacts3', 'file:contacts3.csv', ',', '\n', null, 1, vector
Done. -- 109 msec.
```

4. Check the inserted data in the csv.tutorials.contacts3 table:

```
SQL> SELECT * FROM csv.tutorials.contacts3 ;
CompanyName     CompanyID   Fname     Sname       Age
VARCHAR               INTEGER     VARCHAR   VARCHAR     INTEGER
_____________________________________________________________

MLogistic       12          John      Smith       48
ZiAirLines      13          Anna      Clarks      62
MLogistic       12          Rojer     Danrette    27
MLogistic       12          Kate      Sigton      56
ZiAirLines      13          Tim       Craft       41

5 Rows. -- 78 msec.
```

5. Check the Primary Key columns:

```
SQL> primarykeys csv.tutorials.contacts3;
Showing SQLPrimaryKeys of tables like 'csv.tutorials.contacts3', tabletype/colname like 'NULL'
TABLE_QUALIFIER  TABLE_OWNER  TABLE_NAME               COLUMN_NAME  KEY_SEQ    PK_NAME
VARCHAR          VARCHAR      VARCHAR                  VARCHAR      SMALLINT   VARCHAR
_______________________________________________________________________________

DB               DBA          csv.tutorials.contacts3  CompanyID    1          csv.tutorials.conta
DB               DBA          csv.tutorials.contacts3  Fname        2          csv.tutorials.conta
DB               DBA          csv.tutorials.contacts3  Sname        3          csv.tutorials.conta

3 Rows. -- 47 msec.
```

See Also

`get_csv_row`

`csv_load_file`

`csv_parse`

`csv_cols_def`

`csv_table_def`

csv_load

csv_load — Import CSV file from string session.

Synopsis

csv_load (in *string_session_input* any ,
in *from_line* integer ,
in *to_line* integer ,
in *table_name* varchar ,
in *trx_log_mode* integer ,
in *opts* any) ;

Description

Import CSV file from string session.

Parameters

string_session_input
: String session containing CSV data will be parsed as CSV where it will insert into the table specified as table_name the lines between from_line and to_line offsets.

from_line
: Default - 0, This means counted from the begining.

to_line
: Default - null, This means counted to the end.

table_name
: The table the data to be inserted into.

trx_log_mode
: Default - 2. The trx_log_mode is used to do row auto commit and to log or not the transaction. The default mode is to do not log and do auto commit.

opts
: Default - null. The opts paramater is used to specify the delimiter and quote it should look like this:

Errors

If table is not suitable in respect of number of columns or type of columns error will be signaled.

Examples

Example 24.128. Simple Use

```
SQL>create procedure my_csv_load (in file_name varchar)
{
  declare tb_name varchar;
  declare sess any;
  declare ar any;

  sess := file_open(file_name);

  -- to get the name of the table
  ar := csv_table_def(file_name);
  ar := split_and_decode(ar, 0,'\0\0 ');
  tb_name := trim (ar[2], '"');
```

```
  dbg_obj_print(tb_name);

  -- to create the table
  exec(csv_table_def(file_name));

  -- loading the data
  csv_load(sess,0, null, concat('DB.DBA.', tb_name));
}
;

Done. -- 10 msec.

SQL>select my_csv_load('tmp/MyContacts.csv');
callret
VARCHAR
_______________________

0

1 Rows. -- 121 msec.

SQL>select * from DB.DBA.tmp_MyContacts_csv;
First_Name_Last_Name_Middle_Name_Name_Nickname_E_mail_Addre
VARCHAR
_______________________________________________________________________________

First Name;Last Name;Middle Name;Name;Nickname;E-mail Address;Home Street;Home City;Home Postal Code;Home
x;Mobile Phone;Personal Web Page;Business Street;Business City;Business Postal Code;Business State;Busine
Phone;Business Fax;Pager;Company;Job Title;Department;Office Location;Notes
;;; Virtuoso Nightly Builds on  oplusaix2 ;;virtuoso-nightly-builds@openlinksw.com;;;;;;;;;;;;;;;;;;;;;;
......

202 Rows. -- 341 msec.
```

See Also

get_csv_row

csv_load_file

csv_parse

csv_cols_def

csv_table_def

csv_parse

csv_parse — Parses a CSV file.

Synopsis

csv_parse (in *string_session_input* any ,
in *callback_sp_name* varchar ,
inout *callback_user_data* varchar ,
in *from_line* integer ,
in *to_line* integer ,
in *opts* any) ;

Description

Parses string session containing CSV data and then calls stored procedure by given callback_sp_name parameter. The call back will be invoked for lines between from_line and to_line argument's values. The default from/to is 0/null which means from begining to end. The callback function must take three arguments: the vector which contains parsed csv row, the line number, inout the callback_user_data.

Parameters

string_session_input
: String session containing CSV data will be parsed as CSV where it will insert into the table specified as table_name the lines between from_line and to_line offsets.

callback_sp_name
: The name of the stored procedure.

callback_user_data
: The user data.

from_line
: Default - 0, This means counted from the begining.

to_line
: Default - null, This means counted to the end.

opts
: Default - null. The opts paramater is used to specify the delimiter and quote it should look like this:

Examples

Example 24.129. Simple Use

```
create procedure y_csv_cb (inout r any, in inx int, inout cbd any)
{
 if (cbd is null)
   cbd := vector ();
 cbd := vector_concat (cbd, vector (r));
}
;

....
 h := null;
 csv_parse (ss, 'DB.DBA.y_csv_cb', h, 0, 10);
....
```

See Also

`get_csv_row`

`csv_load_file`

`csv_load`

`csv_cols_def`

`csv_table_def`

csv_cols_def

csv_cols_def — Guess columns and column types from CSV file.

Synopsis

csv_cols_def (in *file_name* varchar) ;

Description

This function will try to guess the columns by given CSV file. It would work if CSV file begins with a header row.

Parameters

file_name
: The name of the CSV file.

Examples

Example 24.130. Simple Use

```
SQL>create procedure my_csv_get_columns (in file_name varchar)
{
   dbg_obj_print(csv_cols_def(file_name));
}
;

Done. -- 10 msec.

SQL>select my_csv_get_columns('tmp/MyContacts.csv');
callret
VARCHAR
_______________________

0

1 Rows. -- 10 msec.
-- and on the Virtuoso Server console will be shown:

(('First_Name_Last_Name_Middle_Name_Name_Nickname_E_mail_Address_Home_Street_Home_City_Home_Postal_Code_H
_Fax_Mobile_Phone_Personal_Web_Page_Business_Street_Business_City_Business_Postal_Code_Business_State_Bus
ss_Phone_Business_Fax_Pager_Company_Job_Title_Department_Office_Location_Notes' 'VARCHAR' ) )
```

See Also

get_csv_row

csv_load_file

csv_load

csv_parse

csv_table_def

csv_table_def

csv_table_def — Guess table definition from CSV file.

Synopsis

csv_table_def (in *file_name* varchar) ;

Description

This function will return table definition appropriate for CSV file specified by file_name.

Parameters

file_name
: The name of the CSV file.

Examples

Example 24.131. Simple Use

```
SQL>select csv_table_def('tmp/MyContacts.csv');
callret
VARCHAR
_______________________________________________________________________________

CREATE TABLE "tmp_MyContacts_csv" (
        "First_Name_Last_Name_Middle_Name_Name_Nickname_E_mail_Address_Home_Street_Home_City_Home_Postal_
e_Home_Fax_Mobile_Phone_Personal_Web_Page_Business_Street_Business_City_Business_Postal_Code_Business_Sta
Business_Phone_Business_Fax_Pager_Company_Job_Title_Department_Office_Location_Notes" VARCHAR)

1 Rows. -- 10 msec.
```

See Also

`get_csv_row`

`csv_load_file`

`csv_load`

`csv_parse`

`csv_cols_def`

get_certificate_info

get_certificate_info — Returns information about the current client X509 certificate

Synopsis

```
integer or string get_certificate_info ( in type integer ,
                                         in cert_or_key_name varchar ,
                                         in in_format int ,
                                         in passwd varchar ,
                                         in ext_oid varchar ) ;
```

Description

This function will return information about the current client X509 certificate (if successfully verified). If there is no valid X509 certificate or the requested information is not available it will return NULL.

If the optional *cert_or_key_name* is supplied it should contain a encoded certificate (by default format is PEM). The certificate info is read from the first certificate in that string.

If the optional *cert_or_key_name* is supplied and *in_format* is equal to 3 (integer) the function will try to retrieve the certificate registered in the current user's key store by name contained in *cert_or_key_name* .

Parameters

type
: type must be an integer. It can be one of the following values:

cert_or_key_name
: A string containing an encoded X.509 certificate or name of a key from current user store.

in_format
: This is to designate encoding type of the value 'cert_or_key_name'

passwd
: Password to unlock PKCS#12 encoded certificate

ext_oid
: X.509 extension object identifier

Return Types

The requested information or NULL if the certificate is found to be invalid.

Examples

Example 24.132. Simple examples

```
CREATE PROCEDURE get_mail_example (in cert any := null, in cert_type int := 0)
{
  declare alts, mail any;
  mail := get_certificate_info (10, cert, cert_type, '', 'emailAddress');
  if (mail is null)
    {
      alts := get_certificate_info (7, cert, cert_type, '', '1.10.16.25');
      if (alts is not null)
        {
          alts := regexp_replace (alts, ',[ ]*', ',', 1, null);
          alts := split_and_decode (alts, 0, '\0\0,:');
```

```
          mail := get_keyword ('email', alts);
        }
    }
  return mail;
}
;
```

get_keyword

get_keyword — Find a value in keyword vector

Synopsis

get_keyword (*keyword* any ,
searched_array vector ,
default any ,
no_copy integer) ;

Description

`get_keyword` performs a case sensitive seek for the occurrence of *`keyword`* from each even position of *`searched_array`*. If found,this returns the element following the occurrence of the keyword. If the keyword is not found this returns the default argument or NULL if the default is not supplied.

Parameters

keyword
: String key value to be searched in the searched_array at even positions.

searched_array
: An array of even length to be searched. Each even position is a string to search. Each odd position can be any value that may then be returned.

default
: Any data to be returned if keyword is not matched in the searched_array.

no_copy
: By default no_copy is false (0). If passed as true (non-zero integer) then the element to return is the original content of the array and the place in the array from which the element came gets set to 0.

Errors

Parameter data type checking errors

Table 24.37. Errors signalled by `openxml`

SQL Code	Error Message	Virtuoso Code
22023	get_keyword expects a vector of even length, not of length <n>	SR057

Return Values

`get_keyword` returns the matched data if it is found. Otherwise it returns the *`default`*. When no *`default`* is defined, NULL is returned.

Examples

Example 24.133. Sample calls

```
get_keyword(2,vector(1,'primero',2,'segundo',
        3,'tercero'),NULL)              -> segundo
get_keyword('tercero',vector('primero',1,'segundo',
        2,'tercero',3), 'NOT FOUND!'))  -> 3

x := vector ('a', 2);
y := get_keyword ('a', x, -1, 1);
```

```
The first call returns 2
y := get_keyword ('a', x, -1, 1);
the second call returns 0 as the data was set to 0 by the previous call.
```

See Also

get_keyword_ucase

get_keyword_ucase

get_keyword_ucase — Find a value in keyword vector (search uppercase)

Synopsis

get_keyword_ucase (*keyword* anything ,
searched_array vector ,
default anything ,
no_copy integer) ;

Description

Identical to `get_keyword` except all comparisons are performed case insensitively.

See Also

get_keyword

get_timestamp

get_timestamp — returns the timestamp of the current transaction

Synopsis

```
timestamp get_timestamp ( );
```

Description

`get_timestamp` is merely an alias for now and is provided for backward compatibility.

getdate

getdate — returns the current transaction timestamp, alias of `now`

Synopsis

```
getdate (  );
```

Description

`getdate()` returns the timestamp associated with current transaction. This is an alias of `now()` .

See Also

Date & Time Manipulation section of the SQL Reference Chapter.

gvector_sort

gvector_sort — Performs quicksort of a given array.

Synopsis

gvector_sort (inout *data* vector ,
in *elements_in_group* integer ,
in *key_idx_in_group* integer ,
in *sort_ascending* integer) ;

Description

The function gets an vector that consists of groups of elements. All groups are of equal size, namely *elements_in_group* , so the total length of *data* vector should be whole multiple of *elements_in_group* . One element of each group is a key of sorting; it is identified by its zero-based position within the group, *key_idx_in_group* . The sorting procedure edits *data* and reorders groups in such a way that their keys become ordered ascending or descending, depending on *sort_ascending* flag.

The simplest case is plain sorting of a uniform array. In this case every element is an individual group, *elements_in_group* is 1 and *key_idx_in_group* is 0.

Other popular case is sorting of result of dict_to_vector (). In this case, every item of the original dictionary is represended in the vector by a pair of elements (*elements_in_group* is 2), so to sort items by their keys, the *key_idx_in_group* is 0 and to sort them by associated values, the *key_idx_in_group* is 1.

Similarly, *elements_in_block* is 2 for arrays like "lines" or "params" of a VSP page.

Parameters

data
: A group vector (gvector) to sort.

elements_in_group
: The number of vector elements in every group, usually 1, should be in range 1 to 1024.

key_idx_in_group
: Zero-based position of key element in group, should be nonnegative and less than elements_in_group .

sort_ascending
: Direction of sorting, nonzero for ascending sort, zero for descending.

Return Types

The function returns number of groups in the *data* vector .

See Also

gvector_digit_sort()

rowvector_digit_sort()

gvector_digit_sort

gvector_digit_sort — Performs a stable "digit" sort of a given array.

Synopsis

gvector_digit_sort (inout *data* vector ,
in *elements_in_group* integer ,
in *key_idx_in_group* integer ,
in *sort_ascending* integer) ;

Description

The function gets an vector that consists of groups of elements. All groups are of equal size, namely *elements_in_group* , so the total length of *data* vector should be whole multiple of *elements_in_group* . One element of each group is a key of sorting; it is identified by its zero-based position within the group, *key_idx_in_group* . The sorting procedure edits *data* and reorders groups in such a way that their keys become ordered ascending or descending, depending on *sort_ascending* flag.

The simplest case is plain sorting of a uniform array. In this case every element is an individual group, *elements_in_group* is 1 and *key_idx_in_group* is 0.

Other popular case is sorting of result of dict_to_vector (). In this case, every item of the original dictionary is represended in the vector by a pair of elements (*elements_in_group* is 2), so to sort items by their keys, the *key_idx_in_group* is 0 and to sort them by associated values, the *key_idx_in_group* is 1.

The performed sorting is stable. It means that it will not permutate groups in vain: it will preserve the relative order of any two groups that have equal keys. Using this property, one may sort groups by a "secondary" key and then by "primary" key. E.g., if each group contain elements for country code and province code then it is possible to make two-field sorting by sorting first by province and then by country; that will work even if province codes are not globally unique (say, if they're enumerated from 1 in each country).

This function supports only integer values of sorting keys. To sort by strings, floating-point numbers etc., use gvector_sort (). However, out of these two similar functions, only gvector_digit_sort() is stable-sort.

Parameters

data
: A group vector (gvector) to sort.

elements_in_group
: The number of vector elements in every group, usually 1, should be in range 1 to 1024.

key_idx_in_group
: Zero-based position of key element in group, should be nonnegative and less than elements_in_group .

sort_ascending
: Direction of sorting, nonzero for ascending sort, zero for descending.

Return Types

The function returns number of groups in the *data* vector .

See Also

gvector_sort()

rowvector_digit_sort()

gz_compress

gz_compress — Compress data using gzip algorithm

Synopsis

gz_compress (in *str* varchar) ;

Description

The gz_compress returns its argument compressed with the gzip algorithm. The argument and return values are arbitrary strings, possibly including any 8 bit characters.

Parameters

str
: The string containing data to be compressed.

Return Types

A string containing the compressed data.

Examples

Example 24.134. GZIP test 2

Just see how it compresses.

```
create procedure
gz_test_2 (in str varchar)
{
  declare res any;
  declare _out varchar;
  declare _len integer;

  result_names (_out, _len);

  res := string_output ();
  result (str, length (str));
  str := gz_compress (str);
  result ('binary', length (str));
  gz_uncompress (str, res);
  result (res, length (res));
}

SQL> gz_test_2 ('f00f f00f m0053 2 w3r h4x0r7 ch002 00000000000000000000');
_out                      _len
VARCHAR                   INTEGER
_______________________________________________________________________________

f00f f00f m0053 5 w3r h4x0r7 ch002 00000000000000000000  58
binary                    43
f00f f00f m0053 5 w3r h4x0r7 ch002 00000000000000000000  58

3 Rows. -- 10 msec.
SQL>
```

See Also

`gz_uncompress` , `string_output_gz_compress` , `string_output` .

gz_uncompress

gz_uncompress — Uncompress a string using gzip algorithm

Synopsis

gz_uncompress (in *str* varchar ,
out *str_out* string_output) ;

Description

`gz_uncompress` takes a string argument, uncompresses it using the gzip algorithm, writing it to a string_output given as the second argument.

Parameters

str
: A string to be uncompressed.

str_out
: A string_output where the uncompressed output should be written.

Return Types

Always returns NULL.

Errors

Examples

See `gz_compress` .

See Also

`gz_compress` , `string_output_gz_compress` , `string_output` .

ST_Affine

ST_Affine — performs standard 2d affine transformation

Synopsis

ST_Affine (in *XXa* any ,
in *XYb* any ,
in *YYe* any ,
in *Xoff* any ,
in *Yoff* any) ;

Description

performs standard 2d affine transformation with matrix:

```
XXa XXb 0
YXd YYe 0
0    0    0
```

and shift (Xoff, Yoff, 0)

Parameters

XXa
: matrix a

XYb
: matrix b

YYe
: matrix e

Xoff
: x

Yoff
: y

Return Types

Returns any.

Examples

Example 24.135. Simple Use

```
select st_affine (st_ewkt_read ('POLYGON((1 3,2 4,1 5,0 4,1 3),
                                         (1 3.5,1.5 4,1 4.5,0.5 4,1 3.5))'),
                  30, 3, 1, 0.3, 0.001, 0.002)
VARCHAR NOT NULL
--------------------------------------------------------------------------------

POLYGON((39.001000 1.902000,
         72.001000 3.202000,
         45.001000 2.502000,
         12.001000 1.202000,
         39.001000 1.902000),
        (40.501000 2.052000,
         57.001000 2.702000,
```

```
                43.501000 2.352000,
                27.001000 1.702000,
                40.501000 2.052000))
```

See Also

st_get_bounding_box()

st_point

st_x

st_y

ST_Z

st_distance

ST_SetSRID

st_astext

st_geomfromtext

st_intersects

st_contains

st_within

isgeometry

geo_insert

geo_delete

DB.DBA.RDF_GEO_ADD

DB.DBA.RDF_GEO_FILL

st_point

st_point — Returns a point geometry.

Synopsis

st_point (in *x* double precision ,
in *y* double precision) ;

Description

Returns a point geometry. The x coordinate corresponds to longitude.

Parameters

x
: X coordinate. Corresponds to longitude.

y
: Y coordinate.

Return Types

Returns a point geometry.

Examples

Example 24.136. Simple Use

```
SQL>select st_point (0, 52);
callret
VARCHAR
_______________________________________________________________________________

POINT(0 52)

1 Rows. -- 40 msec.

SQL>
SPARQL
SELECT ?m (bif:st_distance (?geo, bif:st_point (0, 52)))
WHERE
  {
    ?m geo:geometry ?geo .
    ?m a <http://dbpedia.org/ontology/City> .
    FILTER (bif:st_intersects (?geo, bif:st_point (0, 52), 100))
  }
ORDER BY DESC 2
LIMIT 20;
m                                                                    callret-1
VARCHAR                                                              VARCHAR
_______________________________________________________________________________

http://dbpedia.org/resource/Weston-on-Trent                          138.7082197019335
http://dbpedia.org/resource/Weston-on-Trent                          137.7213767969613
http://dbpedia.org/resource/Weston-on-Trent                          136.4597167847218
http://dbpedia.org/resource/Weston-on-Trent                          134.1807668663677
http://dbpedia.org/resource/Weston-on-Trent                          133.104337839536
http://dbpedia.org/resource/Weston-on-Trent                          133.104337839536
http://dbpedia.org/resource/Nonington                                132.7368236183588
http://dbpedia.org/resource/Nonington                                132.1339163200362
http://dbpedia.org/resource/Nonington                                132.1339163200362
http://dbpedia.org/resource/Nonington                                130.5478483560461
http://dbpedia.org/resource/Nonington                                130.1620410981843
```

```
http://dbpedia.org/resource/Nonington                                   129.8549842943355
http://dbpedia.org/resource/Nonington                                   129.6459280567849
http://dbpedia.org/resource/Nonington                                   129.4504858595742
http://dbpedia.org/resource/Nonington                                   129.2790713235814
http://dbpedia.org/resource/Nonington                                   128.9081040147881
http://dbpedia.org/resource/Nonington                                   128.8845164618929
http://dbpedia.org/resource/Nonington                                   128.6676189617872
http://dbpedia.org/resource/Nonington                                   128.2565253458452
http://dbpedia.org/resource/Nonington                                   128.2551696344652

20 Rows. -- 120 msec.
```

See Also

st_x()

st_y()

st_distance()

st_srid()

st_setsrid()

st_astext()

st_geomfromtext()

st_intersects()

st_contains()

st_within()

isgeometry()

geo_insert()

geo_delete()

DB.DBA.RDF_GEO_ADD()

DB.DBA.RDF_GEO_FILL()

st_x

st_x — Retrieves the x coordinate of a geometry.

Synopsis

st_x (in *g* any);

Description

Retrieves the x coordinate of a geometry.

Parameters

g
: Geometry point

Return Types

Returns double precision.

Examples

Example 24.137. Simple Use

```
SQL>SELECT st_x( st_point (10, 20));
callret
VARCHAR
_______________________________
 10
No. of rows in result: 1

SQL>SPARQL
SELECT DISTINCT (bif:st_x(?geo ))
WHERE
   {
     ?m geo:geometry ?geo .
   }
LIMIT 10;

callret-0
ANY
_______________________________
 -139.2666625976562
 -153.8333282470703
 -163
 -170.8000030517578
 -142.1759033203125
 -142.2581024169922
 -85.03309631347656
 -142.6853942871094
 -143.6544952392578
 -143.8195037841797
No. of rows in result: 10
```

See Also

st_point

st_y

st_distance

st_srid

st_setsrid

st_astext

st_geomfromtext

st_intersects

st_contains

st_within

isgeometry

geo_insert

geo_delete

DB.DBA.RDF_GEO_ADD

DB.DBA.RDF_GEO_FILL

st_y

st_y — Retrieves the y coordinate of a geometry.

Synopsis

st_y (in *g* any);

Description

Retrieves the y coordinate of a geometry.

Parameters

g
: Geometry

Return Types

Returns double precision.

Examples

Example 24.138. Simple Use

```
SQL>SELECT st_y( st_point (10, 20));
callret
VARCHAR
_______________________________
 20
No. of rows in result: 1

SQL>SPARQL
SELECT DISTINCT (bif:st_y(?geo ))
WHERE
   {
     ?m geo:geometry ?geo .
   }
LIMIT 10;

callret-0
ANY
_______________________________
 -90
 -86.96666717529297
 -85.41666412353516
 -85.11666870117188
 -85.05999755859375
 -85.04779815673828
 -85.03780364990234
 -85.03070068359375
 -84.99210357666016
 -84.98509979248047
No. of rows in result: 10
```

See Also

`st_point`

`st_x`

`st_distance`

st_srid

st_setsrid

st_astext

st_geomfromtext

st_intersects

st_contains

st_within

isgeometry

geo_insert

geo_delete

DB.DBA.RDF_GEO_ADD

DB.DBA.RDF_GEO_FILL

st_distance

st_distance — Returns the shortest distance between two points.

Synopsis

```
st_distance ( in g1 any ,
              in g2 any ) ;
```

Description

Returns the shortest distance between two points such that the first point is part of g1 and the second of g2. The srid of g1 and g2 must be the same. If the srid is 4326, the haversine function is used to compute a great circle distance in kilometers on Earth. Otherwise the distance is calculated as on a flat Euclidean plane.

Parameters

g1
: Geometry.

g2
: Geometry.

Return Types

Returns double precision.

Examples

Example 24.139. Simple Use

```
SQL>SELECT st_distance (st_point(0, 52),st_point(0, 70));
callret
VARCHAR
_______________________________
 2000.388915449352
No. of rows in result: 1
```

See Also

st_point

st_x

st_y

st_srid

st_setsrid

st_astext

st_geomfromtext

st_intersects

st_contains

st_within

isgeometry

geo_insert

geo_delete

DB.DBA.RDF_GEO_ADD

DB.DBA.RDF_GEO_FILL

ST_SRID

ST_SRID — returns SRID of shape's spatial reference system or 0 for shape on plane.

Synopsis

ST_SRID (in *g* any);

Description

Returns the SRID of a geometry.

Parameters

g
: Geometry.

Return Types

Returns int.

Examples

Example 24.140. Simple Use

```
SQL>SPARQL
SELECT (bif:ST_SRID (?geo))
WHERE
  {
    ?m geo:geometry ?geo .
  }
LIMIT 10;
callret-0
INTEGER
_______________________________
 4326
 4326
 4326
 4326
 4326
 4326
 4326
 4326
 4326
 4326
No. of rows in result: 10

SQL>select ST_SRID( st_point (10, 30));
callret
INTEGER
_______________________________
 4326
No. of rows in result: 1
```

See Also

st_point

st_x

st_y

ST_Z

st_distance

ST_SetSRID

st_astext

st_geomfromtext

st_intersects

st_contains

st_within

isgeometry

geo_insert

geo_delete

DB.DBA.RDF_GEO_ADD

DB.DBA.RDF_GEO_FILL

ST_SetSRID

ST_SetSRID — replaces the SRID of a shape but does not transform the shape or its coordinates from old SRID to a new one.

Synopsis

ST_SetSRID (in *g* any ,
in *SRID* int) ;

Description

The geometry given as argument is modified to have the specified SRID and the modified geometry is returned.

Parameters

g
: The geometry to be modified.

SRID
: The specified SRID.

Return Types

Returns any.

Examples

Example 24.141. Simple Use

```
SQL>SELECT ST_SetSRID( st_point (10,2), 4335);
callret
ANY
_______________________________
POINT(0.0197 51.1005)
No. of rows in result: 1
```

See Also

st_point

st_x

st_y

ST_Z

st_distance

ST_SRID

st_astext

st_geomfromtext

st_intersects

st_contains

st_within

isgeometry

geo_insert

geo_delete

DB.DBA.RDF_GEO_ADD

DB.DBA.RDF_GEO_FILL

st_astext

st_astext — Returns the well known text (WKT) representation of the geometry.

Synopsis

st_astext (in *g* any);

Description

Returns the well known text (WKT) representation of the geometry.

Parameters

g
: Geometry point

Return Types

Returns varchar.

Examples

Example 24.142. Simple Use

```
SQL>SPARQL
SELECT DISTINCT (bif:st_astext(?geo))
WHERE
  {
    ?m geo:geometry ?geo .

  }
LIMIT 10;
callret-0
ANY
_____________________________________
POINT(-139.267 -90)
POINT(-153.833 -86.9667)
POINT(-163 -85.4167)
POINT(-170.8 -85.1167)
POINT(-142.176 -85.06)
POINT(-142.258 -85.0478)
POINT(-85.0331 -85.0378)
POINT(-142.685 -85.0307)
POINT(-143.654 -84.9921)
POINT(-143.82 -84.9851)
No. of rows in result: 10
```

See Also

st_point

st_x

st_y

st_distance

st_srid

st_setsrid

st_geomfromtext

st_intersects

st_contains

st_within

isgeometry

geo_insert

geo_delete

DB.DBA.RDF_GEO_ADD

DB.DBA.RDF_GEO_FILL

st_geomfromtext

st_geomfromtext — Parses the string and returns the corresponding geometry.

Synopsis

st_geomfromtext (in *string* varchar) ;

Description

Parses the string and returns the corresponding geometry. The string is to be in well known text representation (WKT).

Parameters

string
: String for the corresponding geometry to be extracted from.

Return Types

Returns any.

Examples

Example 24.143. Simple Use

```
SQL>SELECT st_geomfromtext ('point (10.03 30.01)', 2000);
callret
VARCHAR
_______________________________
 POINT(10.03 30.01)
No. of rows in result: 1
```

See Also

st_point

st_x

st_y

st_distance

st_srid

st_setsrid

st_astext

st_intersects

st_contains

st_within

isgeometry

geo_insert

geo_delete

DB.DBA.RDF_GEO_ADD

DB.DBA.RDF_GEO_FILL

st_intersects

st_intersects — Returns intersects between two geometries.

Synopsis

st_intersects (in *g1* any ,
in *g2* any ,
in *prec* double precision) ;

Description

Returns intersects between two geometries. If prec is supplied, this is a tolerance for the matching in units of linear distance appropriate to the srid. Both geometries should have the same srid. st_intersects is true if there is at least one point in common.

Parameters

g1
: The first geometry.

g2
: The second geometry.

prec
: A tolerance for the matching in units of linear distance appropriate to the srid. Default is 0.

Return Types

Returns int.

Examples

Example 24.144. Simple Use

```
SQL>SELECT st_intersects (st_point (0, 51), st_point (0, 52), 100);
callret
INTEGER
_______________________________
 0
No. of rows in result: 1

SQL>SPARQL
SELECT ?c (bif:st_intersects (?geo, bif:st_point (0, 52), 100))
WHERE
   {
     ?m geo:geometry ?geo .
     ?m a ?c .

   }
GROUP BY ?c ?geo
ORDER BY desc 2
LIMIT 10;
c                                                                callret-1
ANY                                                              ANY
_____________________________________________________
http://xmlns.com/foaf/0.1/Person                                 1
http://rdf.alchemyapi.com/rdf/v1/s/aapi-schema#Disambiguation    1
http://rdf.alchemyapi.com/rdf/v1/s/aapi-schema#Disambiguation    1
http://www.w3.org/2003/12/exif/ns/IFD                            1
http://www.w3.org/2003/12/exif/ns/IFD                            1
http://www.w3.org/2003/12/exif/ns/IFD                            1
http://www.w3.org/2003/12/exif/ns/IFD                            1
http://www.w3.org/2003/12/exif/ns/IFD                            1
```

```
http://dbpedia.org/class/yago/HostCitiesOfTheCommonwealthGames   1
http://www.w3.org/2003/12/exif/ns/IFD                            1
No. of rows in result: 10
```

See Also

Querying Geometric Relations

st_point

st_x

st_y

st_distance

st_srid

st_setsrid

st_astext

st_geomfromtext

st_contains

st_within

isgeometry

geo_insert

geo_delete

DB.DBA.RDF_GEO_ADD

DB.DBA.RDF_GEO_FILL

st_contains

st_contains — Returns true if all points of a given geometry g2 are in another geometry g1.

Synopsis

st_contains (in *g1* any ,
in *g2* any ,
in *prec* double precision) ;

Description

Returns true if all points of a given geometry g2 are in another geometry g1. If prec is supplied, this is a tolerance for the matching in units of linear distance appropriate to the srid. Both geometries should have the same srid.

Parameters

g1
: The first geometry.

g2
: The second geometry.

prec
: A tolerance for the matching in units of linear distance appropriate to the srid. Default is 0.

Return Types

Returns int.

Examples

Example 24.145. Simple Use

```
SQL>SPARQL
SELECT ?c COUNT (*)
WHERE
  {
    ?m geo:geometry ?geo .
    ?m a ?c .
    FILTER (bif:st_contains (?geo, bif:st_point (0, 52)))
  }
GROUP BY ?c ?geo
ORDER BY DESC 2;

c                                                                    callret-1
ANY                                                                  ANY
_____________________________________________________________________________
 http://dbpedia.org/class/yago/Landmark108624891                     12
 http://www.w3.org/2002/07/owl#Thing                                 3
 http://dbpedia.org/ontology/Place                                   3
 http://dbpedia.org/ontology/PopulatedPlace                          2
 http://dbpedia.org/class/yago/TowersInTheNetherlands                2
 http://dbpedia.org/class/yago/TowersInGermany                       1
 http://dbpedia.org/class/yago/HillsOfWarwickshire                   1
 http://dbpedia.org/class/yago/PrehistoricSitesInPembrokeshire       1
 http://dbpedia.org/class/yago/HillsOfSomerset                       1
 http://dbpedia.org/class/yago/PeninsulasOfTheFalklandIslands        1
 http://dbpedia.org/class/yago/HillsOfWiltshire                      1
 http://dbpedia.org/class/yago/HillsOfOxfordshire                    1
 http://dbpedia.org/class/yago/HillsOfGloucestershire                1
 http://dbpedia.org/ontology/City                                    1
 http://dbpedia.org/class/yago/HillsOfWorcestershire                 1
```

```
        http://dbpedia.org/class/yago/GuyedMasts                              1
        http://dbpedia.org/ontology/Building                                  1
        http://dbpedia.org/class/yago/Cities,TownsAndVillagesInGelderland     1
        http://dbpedia.org/ontology/Area                                      1
        http://dbpedia.org/class/yago/FormerMunicipalitiesOfGelderland        1
      No. of rows in result: 20
```

See Also

Querying Geometric Relations

st_point

st_x

st_y

st_distance

st_srid

st_setsrid

st_astext

st_geomfromtext

st_intersects

st_within

isgeometry

geo_insert

geo_delete

DB.DBA.RDF_GEO_ADD

DB.DBA.RDF_GEO_FILL

st_within

st_within — Returns true if all points of a given geometry g1 are in another geometry g2.

Synopsis

st_within (in *g1* any ,
in *g2* any ,
in *prec* double precision) ;

Description

Returns true if all points of g1 are in g2. If prec is supplied, this is a tolerance for the matching in units of linear distance appropriate to the srid. Both geometries should have the same srid.

Parameters

g1
: The first geometry.

g2
: The second geometry.

prec
: A tolerance for the matching in units of linear distance appropriate to the srid. Default is 0.

Return Types

Returns int.

Examples

Example 24.146. Simple Use

```
SQL>SPARQL
SELECT ?c ?geo
WHERE
  {
    ?m geo:geometry ?geo .
    ?m a ?c .
    FILTER (bif:st_within(?geo, bif:st_point (0, 52), 100))
  }
LIMIT 10;
c                                                 geo
ANY                                               ANY
_____________________________________
http://linkedgeodata.org/vocabulary#node          POINT(0.0197 51.1005)
http://linkedgeodata.org/vocabulary#node          POINT(0.0207 51.1006)
http://linkedgeodata.org/vocabulary#node          POINT(0.0148 51.1006)
http://linkedgeodata.org/vocabulary#node          POINT(0.0217 51.1006)
http://linkedgeodata.org/vocabulary#node          POINT(0.0021 51.1008)
http://linkedgeodata.org/vocabulary#node          POINT(0.0029 51.1008)
http://linkedgeodata.org/vocabulary#node          POINT(0.0467 51.1007)
http://linkedgeodata.org/vocabulary#node          POINT(0.0217 51.1007)
http://linkedgeodata.org/vocabulary#node          POINT(0.0148 51.1007)
http://linkedgeodata.org/vocabulary#node          POINT(0.0217 51.1008)
No. of rows in result: 10
```

See Also

Querying Geometric Relations

`st_point`

`st_x`

`st_y`

`st_distance`

`st_srid`

`st_setsrid`

`st_astext`

`st_geomfromtext`

`st_intersects`

`st_contains`

`isgeometry`

`geo_insert`

`geo_delete`

`DB.DBA.RDF_GEO_ADD`

`DB.DBA.RDF_GEO_FILL`

isgeometry

isgeometry — Returns 1 if the argument is a geometry.

Synopsis

isgeometry (in *x* any);

Description

Returns 1 if the argument is a geometry.

Parameters

x
: The geometry value.

Return Types

Returns int.

Examples

Example 24.147. Simple example

```
SQL>SELECT isgeometry(st_point(0, 52));
callret
VARCHAR
_______________________________
 1
No. of rows in result: 1

SQL>SPARQL
SELECT ?m
WHERE
  {
    ?m a <http://dbpedia.org/ontology/City> .
    FILTER (bif:isgeometry(?m) = 0)
  }
LIMIT 10

m
VARCHAR
_______________________________________________________
http://dbpedia.org/resource/Berg%2C_Upper_Bavaria
http://dbpedia.org/resource/Grasbrunn
http://dbpedia.org/resource/Putzbrunn
http://dbpedia.org/resource/Ottobrunn
http://dbpedia.org/resource/Dietramszell
http://dbpedia.org/resource/Prutting
http://dbpedia.org/resource/Gauting
http://dbpedia.org/resource/Lucerne
http://dbpedia.org/resource/Hamburg
http://dbpedia.org/resource/Bavaria
No. of rows in result: 10
```

See Also

`st_point`

`st_x`

st_y

st_distance

st_srid

st_setsrid

st_astext

st_geomfromtext

st_intersects

st_contains

st_within

geo_insert

geo_delete

DB.DBA.RDF_GEO_ADD

DB.DBA.RDF_GEO_FILL

geo_insert

geo_insert — Inserts a geometry from an R tree index.

Synopsis

geo_insert (in *tb* any ,
in *geo* any ,
in *id* int) ;

Description

Inserts a geometry from an R tree index. The tb must be a fully qualified name of an R tree table. This function is typically used from triggers on a geometry column. The function is transactional and makes a transaction log record of the action.

Parameters

tb
: A fully qualified name of an R tree table.

geo
: A geometry.

id
: Geometry id.

Return Types

Returns int.

Examples

Example 24.148. Simple example

See Also

`st_point`

`st_x`

`st_y`

`st_distance`

`st_srid`

`st_setsrid`

`st_astext`

`st_geomfromtext`

`st_intersects`

`st_contains`

st_within

isgeometry

geo_delete

DB.DBA.RDF_GEO_ADD

DB.DBA.RDF_GEO_FILL

geo_delete

geo_delete — Deletes a geometry from an R tree index.

Synopsis

```
geo_delete ( in tb any ,
             in geo any ,
             in id int ) ;
```

Description

Deletes a geometry from an R tree index. The tb must be a fully qualified name of an R tree table. This function is typically used from triggers on a geometry column. The function is transactional and makes a transaction log record of the action.

Parameters

tb
: A fully qualified name of an R tree table.

geo
: A geometry.

id
: Geometry id.

Return Types

Returns int.

Examples

Example 24.149. Simple example

See Also

`st_point`

`st_x`

`st_y`

`st_distance`

`st_srid`

`st_setsrid`

`st_astext`

`st_geomfromtext`

`st_intersects`

`st_contains`

st_within

isgeometry

geo_insert

DB.DBA.RDF_GEO_ADD

DB.DBA.RDF_GEO_FILL

hour

hour — get hour from a datetime

Synopsis

hour (in *dt* datetime) ;

Description

`hour` takes a datetime and returns an integer containing a number representing the hour of the datetime.

Parameters

dt
: A datetime .

Return Values

An INTEGER containing the hour.

Examples

Example 24.150. Simple example

Get current hour.

```
SQL> select hour (now ());
callret
INTEGER
_______________________________________________________________________________

21
```

See Also

`dayname` , `dayofmonth` , `dayofweek` , `dayofyear` , `quarter` , `week` , `month` , `year` , `minute` , `second` , `timezone`

http

http — write to HTTP client or a string output stream

Synopsis

http (in *val_expr* any ,
in *stream* any) ;

Description

`http` writes *`val_expr`* to HTTP client or, if parameter *`stream`* is given, to the given string output stream.

`val_expr` may be any scalar object, i.e. string , date or number and will automatically be cast to varchar before further processing. http will print out the string without escapes. `http_value` uses HTML escapes and `http_url` URL escapes.

Parameters

val_expr
: A value expression. May be any scalar expression.

stream
: Optional parameter. If omitted or is 0 and the function is executed within a VSP context, the val_expr will be written to the HTTP client. If present and non-zero, val_expr will be written to the specified stream. If non-zero, the value must be a valid stream obtained from function string_output

Errors

Table 24.38. Errors signalled by `http`

SQLState	Error Code	Error Text	Description
22023	HT007	An interactive blob can't be passed as argument to http	
22023	HT008	http requires string, blob, or string session as argument 1	
37000	HT006	http output function outside of http context and no stream specified: %s.	

Examples

Example 24.151. HTTP output

Output of various flavours of http. See `string_output` and `string_output_string` for examples how to use http to write to streams other than the HTTP output.

```
http (' % <b>')      ' <b>
http_value (' % <b>')    % &lt;b&gt;
http_url (' % <b>')  +%25+<b>
http_value (12, 'li')  <li>12</li>
```

See Also

`http_value` , `http_url` , `string_output` , `http_rewrite` .

http_lock

http_lock — Locks the HTTP Web Server port

Synopsis

```
any http_lock (
    in pwd varchar
);
```

Description

This function takes as parameter the password of the dba user, locks the HTTP Web Server port and on any http access attempt shows the content of the html file given as value for the *MaintenancePage* parameter in the Virtuoso ini [HTTPServer] section:

```
-- Virtuoso ini
...
[HTTPServer]
...
MaintenancePage = atomic.html
..
```

Note: the html file should be placed in the HTTP root.

Parameters

pwd
: The password of the dba user.

Examples

Example 24.152. Simple Use

```
SQL> http_lock('password');

Done. -- 0 msec.
SQL>
```

See Also

`http_unlock()`

http_unlock

http_unlock — Unlocks the HTTP Web Server port

Synopsis

any **http_unlock** (in *pwd* varchar);

Description

This function takes as parameter the password of the dba user and unlocks the HTTP Web Server port

Parameters

pwd
: The password of the dba user.

Examples

Example 24.153. Simple Use

```
SQL> http_unlock('password');

Done. -- 0 msec.
SQL>
```

See Also

http_lock()

http_acl_set

http_acl_set — Set conditions against web server ACL's

Synopsis

```
integer http_acl_set ( in list_name varchar ,
                       in order integer ,
                       in client_ip varchar ,
                       in flag integer ,
                       in destination_ip_or_host varchar ,
                       in object_id integer ,
                       in read_write_flag integer ,
                       in rate integer ) ;
```

Description

This function can be used within application logic to set ACLs rule.

Parameters

list_name
: a string designating which list to be used for matching

order
: a string designating order

client_ip
: a string containing the IP number of client to set rules

flag
: 0 - allow, 1 - deny

destination_ip_or_host
: a string containing the IP number of destination requested by client

object_id
: is a integer designating a object to match

read_write_flag
: an integer designating action to match: 0 - read, 1 - write

rate
: When set, the filter will calculate hit rate average and will compare with limit for http acl rule. If it is larger then will drop connection. Once per day the statistics will be reset.

Return Types

The function returns the value of the HA_FLAG of the first matching rule. If no rule matches it returns -1.

See Also

`http` , `http_acl_get` , `http_acl_remove` , `http_url` , `string_output` , `http_rewrite` , General ACLs .

http_acl_get

http_acl_get — Test conditions against web server ACL's

Synopsis

```
integer http_acl_get ( in list_name varchar ,
                       in source_ip varchar ,
                       in destination_ip_or_host varchar ,
                       in object_id integer ,
                       in read_write_flag integer ) ;
```

Description

This function can be used within application logic to test that an internet client would not violate the ACL.

Parameters

list_name
: a string designating which list to be used for matching

source_ip
: a string containing the IP number of client to test against rules

destination_ip_or_host
: a string containing the IP number of destination requested by client

object_id
: is a integer designating a object to match

read_write_flag
: an integer designating action to match

Return Types

The function returns the value of the HA_FLAG of the first matching rule. If no rule matches it returns -1.

See Also

`http()` , `http_value()` , `http_url()` , `string_output()` , `http_rewrite()` , General ACLs .

http_acl_remove

http_acl_remove — Removes conditions against web server ACL's

Synopsis

```
integer http_acl_remove ( in list_name varchar ,
                          in order integer ,
                          in client_ip varchar ,
                          in flag integer ) ;
```

Description

This function can be used within application logic to remove ACLs rule.

Parameters

list_name
: a string designating which list to be used

order
: a string designating order

client_ip
: a string containing the IP number of client to remove rules

flag
: 0 - allow, 1 - deny

Return Types

The function returns the value of the HA_FLAG of the first matching rule. If no rule matches it returns -1.

See Also

`http` , `http_acl_get` , `http_acl_set` , `http_url` , `string_output` , `http_rewrite` , General ACLs .

http_body_read

http_body_read — Reads the HTTP body from the client HTTP connection and returns it as a string.

Synopsis

```
string http_body_read ( );
```

Description

This function reads the HTTP body from the client HTTP connection and returns it as a string output. This is suitable for processing POST requests with bodies encoded differently than `multipart/*` and `application/x-www-form-urlencoded` as in SOAP requests where the POST body is encoded as `text/xml`).

Example

Example 24.154. Reading a HTTP request entity body

Suppose we have the following HTTP request

```
POST /foo/bar.vsp HTTP/1.1
Content-Type: text/xml
Content-Length: ...

<document>abc</document>
```

The following VSP page will read the content and return an XML document back to the browser.

```
<?vsp
declare ses any;
ses := http_body_read ();
http_header ('Content-Type: text/xml\r\n');
http (string_output_string (ses));
?>
```

http_client

http_client — returns a varchar containing the body of the request uri

Synopsis

```
varchar http_client ( in url varchar ,
                      in uid varchar ,
                      in pwd varchar ,
                      in http_method varchar ,
                      in http_headers varchar ,
                      in body varchar ,
                      in cert_file varchar ,
                      in cert_pwd varchar ,
                      in timeout varchar ,
                      in proxy varchar ,
                      in ca_certs varchar ,
                      in insecure integer ,
                      in n_redirects integer ) ;
```

Description

This function is used to perform HTTP operations to retrieve generic content and/or to perform generic operations over HTTP/HTTPS protocols. It also supports HTTP authentication based on username/password credentials.

If the URL is https: an no x509 certificate given then it will operate as https client w/o client certificate.

Parameters

url
: target URL (http: or https: are supported)

uid
: user name

pwd
: user password

http_method
: 'GET', 'POST', 'HEAD' or 'PUT'

http_headers
: a string containing a HTTP headers supplied by application

body
: content to be sent

cert_file
: cpath to the certificate file where is stored x509 certificate, private key and CA certificates

cert_pwd
: password for opening the certificate file.

timeout
: use to set how many seconds to wait for reply from the client

proxy
: supports:

ca_certs
: Certificate CA. Default is null.

insecure
: Sets secure or not. Default is 0.

n_redirects
: By default int zero. If is given value greater than zero, so many redirects will be followed by client. Thus in pl no need to look at 30x response, just can add this number to follow the redirects.

http_client_ext

http_client_ext — returns a varchar containing the body of the request uri

Synopsis

```
varchar http_client_ext ( in url varchar ,
                          in uid varchar ,
                          in pwd varchar ,
                          in http_method varchar ,
                          in http_headers varchar ,
                          in body varchar ,
                          in cert_file varchar ,
                          in cert_pwd varchar ,
                          inout headers any ,
                          in timeout varchar ,
                          in proxy varchar ,
                          in ca_certs varchar ,
                          in insecure integer ,
                          in n_redirects integer ) ;
```

Description

This function is used to perform HTTP operations to retrieve generic content and/or to perform generic operations over HTTP/HTTPS protocols. It also supports HTTP authentication based on username/password credentials.

If the URL is https: an no x509 certificate given then it will operate as https client w/o client certificate.

Parameters

url
: target URL (http: or https: are supported)

uid
: user name

pwd
: user password

http_method
: 'GET', 'POST', 'HEAD' or 'PUT'

http_headers
: a string containing a HTTP headers supplied by application

body
: content to be sent

cert_file
: cpath to the certificate file where is stored x509 certificate, private key and CA certificates

cert_pwd
: password for opening the certificate file.

headers

return headers from the http reply.

timeout
: use to set how many seconds to wait for reply from the client

proxy
: supports:

ca_certs
: Certificate CA. Default is null.

insecure
: Sets secure or not. Default is 0.

n_redirects
: By default int zero. If is given value greater than zero, so many redirects will be followed by client. Thus in pl no need to look at 30x response, just can add this number to follow the redirects.

http_client_ip

http_client_ip — Returns the IP address of the calling client.

Synopsis

varchar **http_client_ip** (in *what* integer);

Description

This function is used to determine the IP address or DNS name of the calling client.

Please note that this function is slow when resolving a DNS names. It is advisable to use IP addresses to to make applications faster.

Parameters

what
: a flag designating what to return:

Return Types

By default, a string of the dotted IP address of the client user agent. If what=1 then the DNS name will be returned.

See Also

http , http_value , http_url , string_output , http_acl_get , http_rewrite .

Examples

Example 24.155. Obtaining the Client IP Address

```
<?vsp
  declare ip varchar;
  ip := http_client_ip ();
  http (sprintf ('<p>Your IP address is: %s</p>', ip));
?>
```

dns_txt_get

dns_txt_get — Use for DNS text record retrieval.

Synopsis

```
varchar dns_txt_get ( in dns_serevr_ip_or_null any ,
                      in dns_entry varchar ) ;
```

Description

This function can be used for DNS text record retrieval. It works on UNIX, Windows, Mac OS X w/o additional libraries.

Parameters

dns_serevr_ip_or_null
: DNS server IP or null. If null, then the system will try to find the system defined DNS servers. First argument null together with xenc_key_RSA_read() with 3d argument of 1 (int) can be used to write a PL procedure for DKIM signature check of mails.

dns_entry
: DNS entry.

Examples

Example 24.156. Obtaining DNS text record

```
SELECT dns_txt_get (null, 'x._domainkey.example.com');
dns_txt_get
VARCHAR
_______________________________________________________________________________

v=DKIM1; k=rsa; p=...
```

See Also

`http` , `http_client_ip` , `http_value` , `http_url` , `string_output` , `http_acl_get` , `http_rewrite` .

http_debug_log

http_debug_log — set WebDAV HTTP request logging

Synopsis

http_debug_log (in *file_path* varchar) ;

Description

When an valid path string is supplied and it is allowed in file ACL list, the WebDAV HTTP requests and responses will be logged in append mode in to that file. When an open logging session is encountered the second call will produce an error. Specifying a NULL instead of file_path string stops the logging.

The log file consists of lines with following fields:

- The request and response are marked by <<< and >>> signs
- Client IP address
- Date and time of request/response
- Timestamp (milliseconds)
- Request/response line

Errors

Table 24.39. Errors signalled by `http_debug_log`

SQLState	Error Code	Error Text	Description
42000	FA041	HTTP debug log is already being generated	
42000	FA040	Access to [file name] is denied due to access control in ini file	
39000	FA042	Can't open debug log file [file name], error : [OS error]	

Examples

Example 24.157. Logging DAV HTTP Requests

```
-- start the log session
SQL> http_debug_log ('dav.log');
... do some operations in WebDAV repository via WebFolders or other client ...
-- stop the session
SQL> http_debug_log (NULL);

Produces in the server working directory a file 'dav.log' with lines looking like this:
...
>>> [127.0.0.1] [02/Oct/2001 13:49:37 +0300] [40806] PROPFIND /DAV/css/one.css HTTP/1.1
<<< [127.0.0.1] [02/Oct/2001 13:49:37 +0300] [40896] HTTP/1.1 207 Multi-Status
...
```

http_enable_gz

http_enable_gz — Enable / Disable "Content-Encoding: gzip" for HTTP server

Synopsis

`integer` **`http_enable_gz`** `(` in *mode* integer `);`

Description

Enable (1)/ Disable (0) "Content-Encoding: gzip" for HTTP server.

Examples

Example 24.158. Using http_enable_gz()

```
SQL> select http_enable_gz (1);
callret
VARCHAR
_______________________________________________________________________________

1

1 Rows. -- 41 msec.
SQL>
```

http_file

http_file — Send a file to the HTTP client

Synopsis

```
varchar http_file ( in path varchar );
```

Description

This function causes the contents of the file specified by path to be sent as the response of the calling request. The file is not sent until the code calling this returns. Content types etc. are defaulted based on the file's extension. If this function is called, other output to the HTTP client by the caller is discarded.

Parameters

path
: Path to the file to send

Examples

Example 24.159. Sending A File To HTTP Client

```
<?vsp
http_file ('/index.html');
?>
```

http_flush

http_flush — Flush internal HTTP stream and disconnect client; Flush HTTP stream and try sending data in chunked mode.

Synopsis

http_flush (in *try_what* integer) ;

Description

This flushes the internal buffer where output of a VSP page is stored pending the execution of the page's code. This sends the content of the page output buffer along with headers and disconnects the client. The status is 200 OK by default, unless overridden by http_status. The purpose of this function is to allow a page to send output before terminating , thus the page can continue processing for an indefinite time without requiring the client to wait. This is useful for starting long running background tasks from HTTP clients.

VSP pages that use this function must be sure to supply appropriate content (or response headers) if needed before calling this function.

Virtuoso supports HTTP 1.1 Chunking Encoding which allows Virtuoso to send the user agent chunks of output as the page is still processing. Chunking is enabled by calling `http_flush(1)` within the VSP page. By default chunks are sent for every 4k worth of output generated, but in some cases the output needs to consist of smaller chunks, for example when run-time status needs to be shown in a status page. So to achieve this the http_flush (try_what=1) could be invoked in places where chunk must be flushed to the User agent.

Chunked mode requires the following conditions:

- no "Content-Length" header sent to the client using http_header()
- no "Content-Encoding" header sent to the client using http_header()
- use http_xslt() is not permitted
- The client supports HTTP 1.1

Failing these conditions, `http_flush(1)` will be a No-Operation.

If the function has actually switched to chunked mode it will return a non-zero integer. Otherwise and integer 0 will be returned.

Chunked mode is not supported for static content.

Parameters

try_what
: This optional parameter can be supplied the value one (1) to instruct Virtuoso to try sending the output of the VSP page in chunked mode.

Examples

Example 24.160. Using http_flush()

```
<?vsp
  http ('<p>Hit <a href="status.vsp">there</a> to go on status page</p>');
  http_flush ();
  long_task_procedure ();
?>
```

Example 24.161. Using http_flush() small chunks

The following example will render in browser at every loop iteration 'state=N'; so this will be visible at 'run-time' not when loop finished which may take a long.

```
<?vsp
  ....
  http_flush (1);

  while (i<1000)
    {
      process_some_item (); -- some procedure that takes a long time usually
      http ('state='||cast (i as varchar)||'<br>');
      http_flush (1);
      i := i + 1;
    }
?>
```

See Also

`http`, `http_value`, `http_url`, `string_output`, `http_rewrite`.

http_internal_redirect

http_internal_redirect — Performs internal redirect.

Synopsis

```
any http_internal_redirect ( in full_path varchar ,
                             in p_full_path varchar ,
                             in long_url varchar ,
                             in keep_lpath integer ) ;
```

Description

This function performs internal redirect. Can be used in handling situations where HTTP level redirection is inadequate. For example, 303 redirection offered when implementing hashless Linked Data URIs. Instead of sending a 303 Virtuoso will redirect to the URL in "Location:" and return 200 OK. Basically, this is like curl -iL instead of curl -i, but implemented inside Virtuoso.

The function can be used directly or via URI template syntax used to construct rewrite rules for Virtuoso virtual directories / web service endpoints.

Parameters

full_path
: Set new logical path to be redirected to.

p_full_path
: Set physical path to be redirected to.

long_url
: Reserved for internal usage.

keep_lpath
: Reserved for internal usage.

Examples

Example 24.162. Using http_internal_redirect()

```
create procedure my_redirect()
{
  declare full_path varchar;
      ...
      full_path := '/DAV/VAD/test/';
      http_internal_redirect (full_path);
      set_user_id ('demo');
      set http_charset='utf-8';
      http_header ('Content-Type: text/xml; charset=UTF-8\r\n');
      WS.WS.GET (path, pars, lines);
      return null;
    }
}
```

See Also

`http` , `http_value` , `http_url` , `string_output` , `http_rewrite` . `http_header` .

http_get

http_get — returns a varchar containing the body of the request uri

Synopsis

```
varchar http_get ( in target_uri varchar ,
                   out return_header any ,
                   in http_method varchar ,
                   in request_header varchar ,
                   in request_body varchar ,
                   in proxy varchar ) ;
```

Description

http_get returns a varchar containing the body of the requested target_uri or NULL if the body is not received.

Parameters

target_uri
: HTTP target in form http://<target_host>:<target_port>/<path> (if <target_port> is not given then 80 will be used by default)

return_header
: This output parameter is set to the array of HTTP response header lines from the target server, if the parameter is a constant it will be ignored.

http_method
: This parameter will be used to specify the HTTP request method. Possible values are: GET, POST, PUT, OPTIONS or see RFC2616[5.1.1] for details.

request_header
: This string will be sent to the target server together with other header fields. If more than one header field should be sent then header fields must be separated with CR/LF pair. (Warning: this string must not be terminated with CR/LF pair!).

request_body
: This string will be sent to the target server as the request body. The "Content-Length" header field is set to the length of this string.

proxy
: If this parameter is supplied the request will be passed through this HTTP proxy. The format is <proxy_host>:<proxy_port> .

Errors

Table 24.40. Errors signalled by

SQLState	Error Code	Error Text	Description
	HT001	Not valid host name or host IP address	
	HT002	Target host is unreachable	

Examples

Example 24.163. Using http_get()

```
declare header any;
page := http_get ('http://www.openlinksw.com/');
```

```
...
page := http_get ('http://www.openlinksw.com/',
                  header, 'GET', 'If-Modified-Since: Fri, 21 Jul 2000 11:19:45 GMT');
...
page := http_get ('http://example.com/some_link.html',
                  header, 'POST', '', 'a=1&b=2');
...
page := http_get ('http://www.openlinksw.com/',
                  NULL, 'GET', '', '', 'proxy.server.com:8080');
```

http_header

http_header — Specifies non-default HTTP response headers

Synopsis

http_header (in *head* varchar) ;

Description

This function is used to add additional HTTP header lines to the server response. The http_header parameter MUST finish with <CR><LF> characters. Warning: Each call of this function cancels the effect of the previous call. In order to add to previously set header lines, use the `http_header_get` function to retrieve the current headers.

A Content-Type or Media-Type header specified as a part of the headers given with this function will override the default. Otherwise the header lines set using this function add to but do not replace the default response headers. Note that this function cannot set the status line. Use http_request_status for that.

Examples

Example 24.164. Modifying the headers

```
<?vsp
http_header ('Content-type: text/plain\r\n');
-- the effect is browser will display content as text document
?>
```

http_header_get

http_header_get — returns header of current HTTP request

Synopsis

```
varchar http_header_get ( );
```

Description

Returns the response header associated with the current HTTP request. This will not return the default header lines, only those explicitly set with http_header.

This is useful for incrementally modifying response headers during processing of a URL.

Return Types

Varchar of the current headers

Examples

Example 24.165. Retrieving the current header

```
<?vsp
http_header (concat (http_header_get (), 'Location: default.html'));
?>
```

http_kill

http_kill — Kill VSP process whose details match parameter inputs

Synopsis

http_kill (in *client_IP_address* varchar ,
in *URL* varchar ,
in *process_request_id* integer) ;

Description

This function requires dba privileges.

This function is used to kill the process whose details match that of the input parameters. If Client's IP and URL are specified, then it will try to kill all matching pending HTTP requests for that peer requesting that URL. If all three parameters are given, then it will try to kill only that pending HTTP request.

Parameters

client_IP_address
: Client's IP as per the output of http_pending_req()

URL
: Process' URL as per the output of http_pending_req()

process_request_id
: The task ID of the request.

Errors

Table 24.41. Errors signalled by `http_kill`

SQLState	Error Code	Error Text	Description
42000	SR159	Function http_kill restricted to dba group	
22023	SR014	Function http_kill needs a string as argument 1, not an arg of type %s	
22023	SR014	Function http_kill needs a string as argument 2, not an arg of type %s	
22023	SR014	Function http_kill needs a string as argument 3, not an arg of type %s	

Examples

Example 24.166. Retrieving a list of VSP processes

```
declare ps any;

ps := http_pending_req ();

-- the ps value is (('127.0.0.1', '/long.vsp', 1234567), ('192.168.1.1', '/long.vsp', 345678))

http_kill ('192.168.1.1', '/long.vsp', 345678);
```

after this and calling http_pending_req() again we get (('127.0.0.1', '/long.vsp', 1234567)) only.

See Also

`http_pending_req`

`http`

http_listen_host

http_listen_host — Starts, stops and retrieves the state of a user-defined HTTP listener

Synopsis

```
integer http_listen_host ( in interface_address varchar ,
                           in action integer ,
                           in options vector ) ;
```

Description

This function requires dba privileges.

This function is used to start, stop or lookup the state of user-defined HTTP and HTTPS listeners. The return value is 0 or 1 and indicates state of the listener, 1 for started and 0 for stopped.

Parameters

interface_address
: IP address of interface to be started, stopped or queried for its current state.

action
: Can only take one of the following integer values:

options
: An array of name-value pairs for setting up a HTTPS listeners. This parameter only used for starting HTTPS listeners, and nothing more. The available options are:

Return Types

The return type is integer, and will be either 0 or 1 to indicate the state of the listener, 1 for started and 0 for stopped.

Examples

Example 24.167. Starting/stopping and state retrieval of a listener

```
SQL> http_listen_host ('127.0.0.1:7780', 0);
SQL> select http_listen_host ('127.0.0.1:7780', 2);
callret
VARCHAR
_______________________________________________________________________________

1

1 Rows. -- 1 msec.

SQL> http_listen_host ('127.0.0.1:7780', 1);
SQL> select http_listen_host ('127.0.0.1:7780', 2);
callret
VARCHAR
_______________________________________________________________________________

0

1 Rows. -- 1 msec.
```

http_map_table

http_map_table — Update internal HTTP mapping table

Synopsis

```
http_map_table ( in logical_path varchar ,
                 in physical_path varchar ,
                 in vhost varchar ,
                 in listen_host varchar ,
                 in stored_in_dav integer ,
                 in is_browseable integer ,
                 in default_page varchar ,
                 in security_restriction varchar ,
                 in authentication varchar ,
                 in auth_function varchar ,
                 in postprocess_function varchar ,
                 in execute_vsp_as varchar ,
                 in execute_soap_as varchar ,
                 in have_persistent_session_variables integer ,
                 in soap_options any ,
                 in auth_options any ) ;
```

Description

This function requires dba privileges.

This function inserts an entry defining a virtual directory into the HTTP maps table.

Parameters

logical_path
: The absolute path string which the user agent will pass to the server in path part of URI

physical_path
: The absolute path of the real content. For directories or WebDAV collections physical_path MUST end with a slash '/' character, otherwise the point will be treated as a file (or resource).

vhost
: The host name that will be sent to the user-agent in HTTP request. This MUST be valid fully-qualified host name or alias and port separated with semi-column ':' character. This parameter accept special value '*ini*' which will be replaced with hostname and port from INI file.

listen_host
: The fully-qualified host name or IP address and port which will be listened on. Warning: This is only used to make an in-memory mapping, and will not start listening (for starting and stopping a listener see http_listen_host).

stored_in_dav
: Determine if the physical location is a WebDAV resource or collection. Can accept zero or one (1) integer values.

is_browseable
: Determine if directory browsing is allowed for this location. Accepts integer values 0 or 1, treated as false and true respectively. If true (1) enabled and a default page is not specified, a GET request of an URL pointing to this location will generate a directory listing as a response to the user-agent.

default_page
: File name of default page that will be sent to the user-agent if physical_path is a directory.

security_restriction
: A keyword that denotes security type controlling access to the location. Can be 'Digest', 'SSL' or NULL. This value can be used in the auth_function hook using http_map_get .

authentication
: A string value that will be passed as a parameter to the auth_function hook

auth_function
: Fully qualified name of a PL procedure that will perform HTTP authentication. The function must accept one input parameter of type VARCHAR and MUST return integer 0 or 1 as false or true, respectively. A zero return value from the authentication function will cause the HTTP request to be rejected.

postprocess_function
: Fully qualified name of a PL procedure that will be called every time after page processing. Usual purpose is to store session variables in a session table.

execute_vsp_as
: The name of DB user, as whom VSP pages will be executed. If the user is not specified (is null), execution is forbidden.

execute_soap_as
: The name of DB user, as whom SOAP calls will be executed. If null, execution of SOAP calls is forbidden.

have_persistent_session_variables
: Flag that determines if the location has persistent session variables. The value of this flag can be retrieved with http_map_get .

soap_options
: A vector with keyword/value pairs. Currently, valid keywords are 'Namespace' and 'ServiceName'. Namespace is a string defining the namespace for the SOAP service. ServiceName is a string containing name of the SOAP service. See example.

auth_options
: The value of this parameter can be used in the authentication hook. In practice an array of keyword/value pairs would be the input but a single string could be supplied. The user-specific authentication hook can retrieve the options by calling the http_map_get('auth_opts') function.

Examples

Example 24.168. Inserting A Map Entry

```
http_map_table ('/vdir', '/admin/', 'www.foo.com', 'host.foo.com');
```

Example 24.169. Create entry for a SOAP service

```
http_map_table ('/soapapp', '/soapapp/', 'www.foo.com', 'host.foo.com', NULL,
                NULL, NULL, 'SSL', 'SOAP_APP', 'mysoapapp_auth_hook', 'mysoapapp_post_processor',
                NULL, 'mysoapapp_user', 1, vector ('NameSpace', 'http://example.com/soap/v11/',
                                                   'ServiceName','soapApp'));
```

See Also

`http_map_get`

`http_listen_host`

HTTP Authentication in main documentation

HTTP Session Management in main documentation .

http_map_get

http_map_get — get values from HTTP virtual host path mapping table

Synopsis

http_map_get (in *element_name* varchar) ;

Description

Retrieves information associated with the virtual host / HTTP path mapping in effect for the VSP page being processed. Values valid in current connection or URL context may be retrieved by *element_name* .

Calling http_map_get has no use outside of http context. In this case an error will be signalled.

Parameters

element_name
: The possible values for element_name are: 'vsp_uid', 'soap_uid', 'persist_ses_vars', 'default_page', 'browseable', 'security_level' , 'auth_opts', 'soap_opts', 'domain', 'mounted'.

Return Types

Value returned depends of *element_name* :

Table 24.42. Values returned by http_map_get

element_name	Datatype	Return value
vsp_uid	VARCHAR	Which SQL user active content will execute as
soap_uid	VARCHAR	Which SQL user SOAP calls will be executed as
persist_ses_vars	INTEGER	Is persisting of session variables enabled?
default_page	VARCHAR	Default page for current mapping
browseable	INTEGER	Is directory browsing enabled?
security_level	VARCHAR	Security level ('DIGEST', 'SSL', 'BASIC')
auth_opts	ANY	Application-specific authentication options
soap_opts	ANY	Application-specific SOAP options
domain	VARCHAR	Directory path that is the logical start point of current mapping.
mounted	VARCHAR	Physical path that is the physical start point of current mapping

Examples

Example 24.170. Retrieving Mapping Details

```
<?vsp
declare def_page varchar;
def_page := http_map_get ('default_page');
http (sprintf ('<p>The default page for this path is: %s </p>', def_page));
?>
```

Example 24.171. Warn users if non-SSL connection

```
<HTML>
  <HEAD>
    <TITLE>TEST</TITLE>
```

```
  </HEAD>
  <BODY>
    <P>We're executing as <?= http_map_get ('vsp_uid') ?></P>
<?vsp
    if (http_map_get ('security_level') <> 'SSL')
      {
?>
    Warning: This page is not protected by transport level encryption
    and may be viewable by anybody in the middle with a network
    sniffer.
<?vsp
      }
?>
  </BODY>
</HTML>
```

http_param

http_param — returns value of a HTML form parameter in VSP context

Synopsis

http_param (in *name* varchar) ;

Description

This function is used to return value of a HTML form parameter in VSP context. It's almost like call get_keyword ('name', params) used in VSP programming. When 'name' parameter is not supplied, the result of http_param() call will be all parameters, as they are contained in 'params' parameter of the VSP pages. This function is useful in HTTP authentication PL hook, as in this place there is no 'params' argument.

Parameters

name
: A string with name of the parameter to return

Return Types

Returns value of a HTML form parameter, or array of all parameters.

See Also

VSP pages `vhost_define()`

http_path

http_path — returns the absolute path to the logical path location of the current http request

Synopsis

```
http_path (   );
```

Description

This function returns the absolute path to the logical path location of current HTTP request.

Examples

Example 24.172. Retrieving Current Path of Request

```
<?vsp
  http (sprintf ('<p>The current location is : %s </p>', http_path ()));
?>
```

http_pending_req

http_pending_req — return array describing running VSP threads

Synopsis

```
http_pending_req (  );
```

Description

This function requires dba privileges.

`http_pending_req` returns an array of arrays of data on running VSP requests. Each array contains the Client IP Address, URL, and Process Request ID. These values can be used with the `http_kill()` function.

Return Types

An array of 3-part-arrays. Each 3-part-array consists of client_IP_address (string), URL (string), and process request ID (number).

The request ID's can be used to terminate requests with http_kill.

Errors

Table 24.43. Errors signalled by `http_pending_req`

SQLState	Error Code	Error Text	Description
42000	SR159	Function http_pending_req restricted to dba group	

Examples

Example 24.173. Retrieving a list of VSP processes

```
declare ps any;

ps := http_pending_req ();

-- the ps value is (('127.0.0.1', '/long.vsp', 1234567), ('192.168.1.1', '/long.vsp', 345678))
```

See Also

`http_kill`

`http`

http_physical_path

http_physical_path — returns the physical path location of the requested URL

Synopsis

http_physical_path ( );

Description

This function returns the absolute path to the physical path location of current HTTP request

Examples

Example 24.174. Retrieving Current Path of Request

```
<?vsp
  http (sprintf ('<p>The physical location is : %s </p>', http_physical_path ()));
?>
```

http_proxy

http_proxy — proxy request to another host and return content to calling client

Synopsis

http_proxy (in *host* varchar ,
in *header* any ,
in *content* varchar) ;

Description

This function is used to retrieve content from a foreign host and send the response to the HTTP client of the page calling this. This is useful for re-routing a request to another server in the middle of a VSP page.

Parameters

host
: The fully-qualified host name or alias. If a target port is specified the semi-column ':' character MUST be used as the separator. ('www.foo.com:8080')

header
: an array consisting of the HTTP request header lines. Warning: Each line MUST finish with <CR><LF> characters. This header lines will be sent to the target server without any conversion.

content
: In the case of posting of forms this parameter can contain the form data as specified in HTML standards.

Examples

Example 24.175. Retrieving Current Path of Request

```
<?vsp
  http_proxy ('www.foo.com', vector ('GET / HTTP/1.0'), null);
?>
```

http_request_header

http_request_header — returns array of HTTP request header lines

Synopsis

```
any http_request_header ( in array any ,
                          in field_name varchar ,
                          in attr-name varchar ,
                          in default_value varchar ) ;
```

Description

This function provides access to the HTTP request header lines.

Return Types

If no input parameters are supplied then this function returns a copy of the lines vector of VSP pages. If array, field_name and mime_attribute is supplied then this function will attempt to extract the associated value, if is cannot be found then the default_value input will be returned. If attr-name is supplied then the mime attribute from the field_name will be returned.

Note that when field value is requested then it will be treated as a MIME field and only value will be returned if found. For example consider presence of following header line: "Content-Type: text/plain; charset=utf-8", then if 'Content-Type' is requested, only "text/plain" will be returned. Use http_request_header_full to obtain the full field value.

Examples

Example 24.176. Accessing the HTTP request headers

```
<p>Your request follows</p>
  <?vsp
    declare header any;
    declare ix, len integer;
    declare host varchar;
    header := http_request_header ();
    len := length (header);
    ix := 0;
    while (ix < len)
    {
      http (aref (header, ix));
                 ix := ix + 1;
    }
    host := http_request_header(lines, 'Host', null, '*** NO HOST IN REQUEST ***')
  ?>
```

http_request_header_full

http_request_header_full — returns array of HTTP request header lines

Synopsis

```
any http_request_header_full ( in array any ,
                               in field_name varchar ,
                               in default_value varchar ) ;
```

Description

This function provides access to the HTTP request header lines.

Return Types

If no input parameters are supplied then this function returns a copy of the lines vector of VSP pages. If array and field_name is supplied then this function will attempt to extract the associated field value, if is cannot be found then the default_value input will be returned.

Examples

Example 24.177. Accessing the HTTP request headers

```
<p>Your request follows</p>
  <?vsp
    http (http_request_header_full(lines, 'Content-Type', '*none*'));
  ?>
```

http_request_status

http_request_status — set the status sent to the client in an HTTP response

Synopsis

http_request_status (in *status_line* varchar) ;

Description

This allows a VSP page to control the status sent to the client in the HTTP response. The argument will be presented as the first line of the reply instead of the default "HTTP/1.1 200 OK". The string should not contain a CR or LF at the end.

This allows a page to issue redirects, authentication challenges etc.

Use it with `http_headers` to control the content of the HTTP reply headers.

Parameters

status_line
: String conforming to HTTP/1.1 (see RFC2616). Examples of possible status lines are: 'HTTP/1.1 200 OK', 'HTTP/1.1 500 Internal Server Error', 'HTTP/1.1 401 Not found' or 'HTTP/1.1 400 Bad request' etc.

http_request_get

http_request_get — Access to the HTTP request line

Synopsis

http_request_get (in *variable_name* varchar) ;

Description

This function is used to access the HTTP request line within VSP or VSPX context. It returns 'CGI' style variables for protocol version, HTTP method and query string.

Parameters

variable_name
: A string designating what to return.

http_rewrite

http_rewrite — Clears output written to a string output or to an HTTP

Synopsis

http_rewrite (in *stream* any);

Description

This clears any previous output to the stream. If the stream is omitted or 0 the stream is the HTTP client stream of the calling procedure.

All output from VSP page procedures is buffered into a local string stream before being sent out. This is done so as to support the HTTP/1.1 required content length and to allow recovery from errors.

Parameters

stream
: Optional stream to clear. Null or zero (0) implies the default HTTP client stream.

Using http_rewrite()

Example 24.178. A VSP page excerpt.

```
<p>Nope. You won't see this.</p>

<?vsp
http ('<p>This line will contribute to your local entropy.</p>');
http_rewrite ();
http ('<p>Now THIS is what you wanted to see, isn't it?</p>');
?>
```

See Also

`http_value`, `http_url`, `string_output`.

http_root

http_root — Returns the absolute path of the server root directory.

Synopsis

```
varchar http_root ( );
```

Description

Returns the absolute path of the server root directory.

Return Types

String of the absolute path of the server root directory.

Examples

Example 24.179. Obtaining the Client IP Address

```
<?vsp
  http (file_to_string (sprintf ('%s/banner.html', http_root())));
?>
```

See Also

`http`, `http_value`, `http_url`, `string_output`, `http_rewrite`.

http_value

http_value — write to HTTP client or string output stream with HTML escapes

Synopsis

http_value (in *val_expr* any ,
in *tag* varchar ,
in *stream* any) ;

Description

The http_value is used to write to an HTTP client (when in a VSP context) or a specified string output stream. http_value uses HTML-escapes for characters that should be escaped according to the HTML spec.

Parameters

val_expr
: A value expression. May be any string expression. If val_expr is an XML entity, a serialization of the entity is written to the stream . This is not the string value of the XML fragment, but a serialization of the XML fragment as text including all the markup, i.e. elements, attributes, namespaces, text nodes, etc. To get the string value of an XML entity, convert it to a varchar using cast . Casting as varchar will only produce a concatenation of the text nodes in the XML fragment, leaving out elements, attributes, name spaces, etc.

tag
: Optional. If present and is a string, the output will be enclosed in tags named as the string content of tag . If the expression evaluates to 0 or null, it will be ignored.

stream
: Optional parameter. If omitted or is 0 and the function is executed within a VSP context, the val_expr will be written to the HTTP client. If present and non-zero, val_expr will be written to the specified stream. If non-zero, the value must be a valid stream obtained from function string_output

Return Values

None

Errors

Table 24.45. Errors signalled by http_value

SQLSTATE	Error Code	Error Text	Description
37000	HT006	http output function outside of http context and no stream specified: %s	occurs if called outside VSP and no string session is given
22023	SR066	Unsupported case in CONVERT (<data_type_of 1st arg> -> VARCHAR): %s	if the value passed as the 1st argument can't be converted to a VARCHAR value and it's not a XML/XQUERY tree

Using http_value()

Example 24.180. HTTP output

Output of various flavours of http . See string_output and string_output_string for examples how to use http to write to streams other than the HTTP output.

```
http (' % <b>')      ' <b>
```

```
http_value (' % <b>')    % &lt;b&gt;
http_url (' % <b>')  +%25+<b>
http_value (12, 'li')  <li>12</li>
```

See Also

`http`, `http_url`, `string_output`, `http_rewrite`.

json_parse

json_parse — Takes json string and returns parse tree.

Synopsis

```
integer json_parse ( in txt varchar );
```

Description

This function takes json string and returns parse tree.

Parameters

txt
: a json string

Return Types

The function returns tree.

Examples

Example 24.181. Simple example

```
SQL>create procedure json_tree_example ()
{
  declare qr, url, cnt, tree, xt, hdr any;
  cnt := '
{ "head": { "link": [], "vars": ["s", "p", "o"] },
  "results": { "distinct": false, "ordered": true, "bindings": [
    { "s": { "type": "uri", "value": "http://example.org/node" }        , "p": { "type": "uri", "value":
    { "s": { "type": "uri", "value": "http://example.org/node" }        , "p": { "type": "uri", "value":
  tree := json_parse (cnt);
  tree := get_keyword ('results', tree);
  tree := get_keyword ('bindings', tree);
  return tree[0][1];
}
;

Done.
SQL> SELECT json_tree_example();
json_tree_example
VARCHAR
 structure
No. of rows in result: 1
```

See Also

`http`, `http_value`, `http_url`, `string_output`, `http_rewrite`,

http_url

http_url — write to HTTP client or string output stream with URL escapes

Synopsis

http_url (in *val_expr* any ,
in *tag* varchar ,
in *stream* any) ;

Description

The `http_url` is used to write to an HTTP client (when in a VSP context) or a specified string output stream. `http_url` uses URL escapes for special characters.

Parameters

val_expr
: A value expression. May be any scalar expression. If val_expr is an XML entity, a serialization of the entity is written to the stream . This is not the XML as a text string, but a serialization of the internal representation of parsed XML data.

tag
: Optional. If present and is a string, the output will be enclosed in tags named by the string content of tag . If the expression evaluates to 0 or null, it will be ignored.

stream
: Optional parameter. If omitted or is 0 and the function is executed within VSP context, the val_expr will be written to the HTTP client. If present and non-zero, val_expr will be written to the specified stream. If non-zero, the value must be a valid stream obtained from function string_output

Return Values

None

Errors

Table 24.46. Errors signalled by `http_url`

SQLState	Error Code	Error Text	Description
37000	HT006	http output function outside of http context and no stream specified: %s	
22023	SR066	Unsupported case in CONVERT (%s -> VARCHAR)	

Examples

Example 24.182. HTTP output

Output of various flavours of `http` . See `string_output` and `string_output_string` for examples how to use http to write to streams other than the HTTP output.

```
http (' % <b>')      ' <b>
http_value (' % <b>')   % &lt;b&gt;
http_url (' % <b>')  +%25+<b>
http_value (12, 'li')  <li>12</li>
```

See Also

`http`, `http_value`, `string_output`, `http_rewrite`.

http_xslt

http_xslt — applies an XSLT stylesheet to the output of a VSP page

Synopsis

http_xslt (in *xslt_uri* varchar ,
in *params* any) ;

Description

This function can be called inside a VSP page to apply an XSLT stylesheet to the output of the page once the page is complete. This function will return immediately and the stylesheet will not be applied until the page is successfully formed. Any errors arising in the stylesheet processing will be reported to the web client.

The stylesheet does not have to be previously defined. The URI supplied will be used to locate the stylesheet. This can be a file, an HTTP URL or a virt:// URI for a stylesheet stored in a local table. Virtuoso will cache the stylesheet after first use. You can clear the cache entry with the `xslt_stale()` function.

For this to work the text generated by the VSP page should be well-formed XML.

This function is only valid in a VSP context. The *`xsl:output`* element will control the Content-Type sent to the user agent.

Parameters

xslt_uri
: Absolute URI of the XSL stylesheet

params
: Even length array of name/value pairs.

Examples

Example 24.183. Performing XSLT Transformation of VSP Output

```
<?vsp
  http ('<a><b>simple XML document</b></a>');
  http_xslt ('file://');
?>
```

URLREWRITE_CREATE_REGEX_RULE

URLREWRITE_CREATE_REGEX_RULE — Creates regex rules.

Synopsis

```
URLREWRITE_CREATE_REGEX_RULE ( in rule_iri varchar ,
                               in allow_update integer ,
                               in nice_match varchar ,
                               in nice_params any ,
                               in nice_min_params integer ,
                               in target_compose varchar ,
                               in target_params any ,
                               in target_expn varchar ,
                               in accept_pattern varchar ,
                               in do_not_continue integer ,
                               in http_redirect_code integer ,
                               in http_header_lines varchar ) ;
```

Description

Creates regex rules.

Parameters

rule_iri
: The rule's name / identifier

allow_update
: Indicates whether the rule can be updated. 1 indicates yes; 0 indicates no. The update is subject to the following rules:

nice_match
: A regex match expression to parse the URL into a vector of occurrences.

nice_params
: A vector of the names of the parsed parameters. The length of the vector should be equal to the number of '(...)' specifiers in the format string.

nice_min_params
: Used to specify the minimum number of sprintf format patterns to be matched in order to trigger the given rule. In existing versions of Virtuoso it only affects sprintf rules and has no effect for regex rules.

target_compose
: A regex compose expression for the URL of the destination page.

target_params
: A vector of names of parameters that should be passed to the compose expression (target_compose) as $1, $2 and so on.

target_expn
: Optional SQL text that should be executed instead of a regex compose call.

accept_pattern
: A regex expression to match the HTTP Accept header

do_not_continue

If the given rule satisfies the match conditions, 1 signifies do not try the next rule from same rule list, and 0 signifies try the next rule.

http_redirect_code
: NULL or the integer values 301, 302, 303, or 406, are currently allowed. If a 3xx redirect code is given, an HTTP redirect response will be sent back to client. If NULL is specified, the server will process the redirect internally.

http_header_lines
: Additional header lines to be added to the return value.

Return Types

The return value is not specified.

Examples

Example 24.184. Example 1

```
DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'demo_nw_rule1',
    1,
    '(/[^#]*)',
    vector('path'),
    1,
    '/sparql?query=CONSTRUCT+{+%%3Chttp%%3A//^{URIQADefaultHost}^%U%%23this%%3E+%%3Fp+%%3Fo+}+FROM+%%3Cht
    vector('path', 'path', '*accept*'),
    null,
    '(text/rdf.n3)|(application/rdf.xml)',
    0,
    null
    );
```

Example 24.185. Example 2

```
DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
    'demo_nw_rule2',
    1,
    '^/~(.*)',
    vector('uname'),
    1,
    '/home/%s',
    vector('uname'),
    null,
    null,
     2,
     null,
     'MS-Author-Via: DAV'
    );
```

identity_value

identity_value — returns the last assigned identity column value

Synopsis

```
integer identity_value ( );
```

Description

This function returns the value assigned to an identity column by the previous insert statement. Insert statements that do not assign identity columns do not affect this. Note that tables that have no primary key have n invisible identity column called _IDN. The scope is the connection. This function may be called from a client or from a stored procedure and will return the last given identity column value wherever it was given. The value stays set until overwritten by the next insert operation. This value is not set by rexecute or inserts to remote tables with autoincrement columns declared on the remote database since there is no standard way of getting this information from remote data sources.

The same value can be more efficiently accessed from clients using the SQLGetStmtOption ODBC call with the option SQL_GETLASTSERIAL. In this case the value is of type SQLINTEGER.

Return Types

The value assigned to an identity column by the previous insert or 0 if no identity column was assigned.

import_clr

import_clr — This function automatically creates the SQL Type wrappers based on the CLR Reflection API.

Synopsis

```
any import_clr ( in assemblies_vector any ,
                 in classes_vector any ,
                 in security_mode integer ) ;
```

Description

This function automatically creates the SQL Type wrappers based on the CLR Reflection API.

Parameters

assemblies_vector
: a vector of assembly names (as VARCHAR) to look into (or null).

classes_vector
: a vector of type names to create SQL type wrappers for (or null to mark all the types in the assemblies specified by assemblies_vector. In that case the assemblies_vector cannot be NULL).

security_mode
: This optional parameter defines the access mode as follows:

Examples

Example 24.186. Importing a Class

Here is a simple C# program that we can import and use with Virtuoso. This example requires that you are running Virtuoso with CLR support.

Using a text editor create a C# source file in the server root directory called sanity.cs, with the following contents:

```
using System;

public class sanity
{
    public static string test(string  name) {
       return "Hello "+ name + ", from Virtuoso";
    }
}
```

This sample needs to be compiled into bytecode assembly before it can be used. Use a command prompt that is suitably set up to find .NET utilities in its path, the .NET Framework SDK installation provides a shortcut in the Start menu to a command prompt that is preconfigured. From the command prompt change directory to the Virtuoso server root containing the C# source file. Execute:

```
C:\Program Files\OpenLink\Virtuoso 3.0\bin>csc /target:library sanity.cs
Microsoft (R) Visual C# .NET Compiler version 7.00.9466
for Microsoft (R) .NET Framework version 1.0.3705
Copyright (C) Microsoft Corporation 2001. All rights reserved.
```

Now this library must be introduce to the Virtuoso Server. Using ISQL use the following commands to test the CLR:

```
C:\Program Files\OpenLink\Virtuoso 3.0\bin>isql 1112
Connected to OpenLink Virtuoso
Driver: 03.00.2315 OpenLink Virtuoso ODBC Driver
OpenLink Interactive SQL (Virtuoso), version 0.9849b.
Type HELP; for help and EXIT; to exit.
```

```
SQL> DB..import_clr (vector ('sanity'), vector ('sanity'));

Done. -- 300 msec.
SQL> select sanity::test('Rob');
callret
VARCHAR
_______________________________________________

Hello Rob, from Virtuoso

1 Rows. -- 60 msec.
```

Congratulations, you have proven that your Virtuoso server can run .NET classes.

See Also

The Runtime Hosting Chapter

unimport_clr

import_jar

import_jar — Creates SQL wrapper types of selected Java classes

Synopsis

```
void import_jar ( in files any ,
                  in classes any ,
                  in security_mode integer ) ;
```

Description

This function calls the `jvm_ref_import()` function to produce the XML, then transforms it to a set of CREATE TYPE statements and executes them. The SQL names of the types are generated by retrieving the fully qualified name of the Java class and substituting the . with _ (e.g.: java.lang.System becomes java_lang_System). The names of the static members observer functions are composed by prepending the name of the static member with 'get' so the static Java member stat_m is mapped to a SQL static method getstat_m(). As a result it creates SQL type wrappers for the specified Java classes.

Parameters

files
: Null, string or vector of strings. This can contain one or many Java .class files, .zip or .jar files to pick classes from. Null means only the classes specified in the classes argument regardless of their physical location.

classes
: Wildcard string or a vector of strings specifying which classes to describe in the resulting XML.

security_mode
: This optional parameter defines the access mode as follows:

Examples

Example 24.187. Simple example

```
SQL> import_jar (null, vector ('java.lang.System'));
SQL> select java_lang_System::getProperty('java.vm.name');
callret
NVARCHAR
_______________________________________________________________________________

Java HotSpot(TM) Client VM
```

See Also

`unimport_jar()`

`jvm_ref_import()`

initcap

initcap — returns its argument with the first letter capitalized

Synopsis

initcap (*str* string);

Description

initcap returns a copy of string str with the first character, if it is a lowercase letter, converted to the corresponding uppercase letter. Otherwise, an identical copy of the string is returned. Notes about ucase apply also here.

```
initcap('simurg!')  -> 'Simurg!'
```

internal_to_sql_type

internal_to_sql_type — returns the integer standard SQL type of its argument

Synopsis

internal_to_sql_type (*internal_type* integer) ;

Description

internal_to_sql_type returns an integer value representing the standard SQL type converted from internal_type given as its argument.

```
internal_to_sql_type(182)       -> 12 (VARCHAR)
```

internal_type

internal_type — returns internal integer datatype of its argument

Synopsis

internal_type (*arg* anything);

Description

internal_type returns an integer value representing the internal type of its argument. These values are the same as what Virtuoso uses in the column COL_DTP of the system table SYS_COLS for keeping the track of the default types of each column of each table.

```
internal_type(space(5))          -> 182 (A long string)
```

internal_type_name

internal_type_name , dv_type_title — returns the internal type name of the argument

Synopsis

internal_type_name (*internal_type* integer) ;

dv_type_title (*internal_type* integer) ;

Description

internal_type_name returns a string which is a human-readable name for an internal_type integer given as its argument. The function `dv_type_title()` is an alias of `internal_type_name()` .

```
internal_type_name (internal_type ('kumikala'))
                -> 'SHORT_STRING'
select internal_type_name(COL_DTP) from SYS_COLS;
```

isarray

isarray — Check for a valid array

Synopsis

```
boolean isarray ( in x any );
```

Description

isarray is true if the argument is a valid argument to aref. This is the case for any string or vector.

Parameters

x
: The variable to be checked.

Return Values

1 - if arg is an array (vector()) or string (varchar, char) otherwise return 0.

Errors

No error, it always succeeds.

isblob

isblob — returns true if its argument is of type long varchar

Synopsis

isblob (*arg* anything);

Description

isblob returns one if its argument as a handle to an object of the type LONG VARCHAR, zero otherwise.

isbinary

isbinary — returns true if its argument is of type varbinary

Synopsis

```
boolean isbinary ( arg anything );
```

Description

isbinary returns one if its argument is of type VARBINARY, zero otherwise.

Parameters

arg
: Some variable to be assessed.

Examples

Example 24.188. Simple Use

```
SQL>select isbinary(0x1213);
-&gt; 1 (Yes it is a VARBINARY)
```

isdouble

isdouble — returns true is argument is a double

Synopsis

isdouble (*arg* anything) ;

Description

isdouble returns one if its argument is of type double precision float, zero otherwise.

```
isdouble(0.0)           -> 1 (Decimal literals are by
                             default converted to
                             double precision numbers)
```

isentity

isentity — returns true if its argument is an XML entity

Synopsis

```
boolean isentity ( in x any );
```

Description

`isentity` is true if the argument is an XML entity object, such as that returned from XPATH expressions etc.

isfloat

isfloat — returns true if its argument is a float

Synopsis

isfloat (*arg* anything);

Description

`isfloat` returns one if its argument is of type single float, zero otherwise.

```
isfloat(0.0)            -> 0 (No it is not, because decimal
                             literals are by default
                             converted to double precision
                             numbers)
isfloat(atof('0.0'))    -> 1 (Only with explicit atof we get
                             a single float)
```

isinteger

isinteger — returns true if its argument is of type integer

Synopsis

isinteger (*arg* anything) ;

Description

`isinteger` returns one if its argument is of type integer, zero otherwise.

```
isinteger(0)            -> 1 (Yes it is)
```

isnull

isnull — returns true if its argument is NULL

Synopsis

isnull (*arg* anything);

Description

isnull returns one if its argument is NULL, zero otherwise.

isnumeric

isnumeric — returns true if argument is of numeric type

Synopsis

isnumeric (*arg* anything) ;

Description

isnumeric returns one if its argument is of type integer, single float or double precision floating point number, zero otherwise.

isfinitenumeric

isfinitenumeric — returns 1 if its argument is of numeric type and its value is plain valid numeric value.

Synopsis

isfinitenumeric (*arg* anything) ;

Description

isfinitenumeric returns 1 if its argument is of numeric type and its value is plain valid numeric value, not a peculiarity like infinity or not-a-number.

Examples

Example 24.189. Simple Use

```
SQL>  SPARQL
INSERT INTO GRAPH <http://mygraph.com>
  {
    <http://mygraph2.com/pricing/#QVal1> <price1>  1 ;    <price2> "";    <price3> 5 .
    <http://mygraph2.com/pricing/#QVal2> <price1>  "" ;   <price2> 1 ;    <price3> 4 .
    <http://mygraph2.com/pricing/#QVal3> <price1>  6 ;    <price2> 2 ;    <price3> "".
  };

Insert into <http://mygraph.com>, 9 (or less) triples -- done
No. of rows in result: 1

SQL> SPARQL SELECT ?s (bif:isfinitenumeric(?pr1))
FROM <http://mygraph.com>
WHERE
  {
    ?s <price1> ?pr1 ;
       <price2> ?pr2 ;
       <price3> ?pr3
  };

Query result:
s                                    callret-1
VARCHAR                                    INTEGER
_______________________________________________
http://mygraph2.com/pricing/#QVal2       0
http://mygraph2.com/pricing/#QVal1       1
http://mygraph2.com/pricing/#QVal3       1

No. of rows in result: 3
```

isstring

isstring — returns true if its argument is of type varchar

Synopsis

```
boolean isstring ( arg anything );
```

Description

isstring returns one if its argument is of type VARCHAR, zero otherwise.

Parameters

arg
: Some variable to be assessed.

Examples

Example 24.190. Simple Use

```
SQL>select isstring('Cadena de los patos amarillos')
                -&gt; 1 (Yes it is a string)
```

iszero

iszero — returns true if argument is numeric zero

Synopsis

iszero (*arg* any);

Description

iszero returns one if its argument is an integer 0, a float 0.0 or a double 0.0 For any other arguments, of whatever type, it will return zero.

```
iszero(0)               -> 1 (Yes it is)
iszero(0.0)             -> 1 (Double precision zero also
                             is a zero)
iszero(atof('0.0'))     -> 1 (As well as single
                             precision floating point)
iszero(1)               -> 0 (No, it's not)
iszero('Cifra')         -> 0 (neither is this one)
iszero(NULL)            -> 0 (nor this one)
```

__min

__min — Returns minimum value among all its arguments.

Synopsis

```
any __min ( arg1 any ,
            arg2 any ,
            ... ,
            argN any ) ;
```

Description

The function returns the minimum value among all values in all its arguments.

Parameters

argI
: Values which can be with type float, integer etc.

Examples

Example 24.191.

Example 1

```
SQL> select __min(12, 2.34, 5, 20);
__min
VARCHAR
_______________________________________________________________________________

2.34

1 Rows. -- 32 msec.
```

Example 2

```
SQL> SPARQL
INSERT INTO GRAPH <http://mygraph2.com>
  {
    <http://mygraph2.com/pricing/#QVal1> <price1>  1 ; <price2> 9 ; <price3> 5 .
    <http://mygraph2.com/pricing/#QVal2> <price1>  3 ; <price2> 1 ; <price3> 4 .
    <http://mygraph2.com/pricing/#QVal3> <price1>  6 ; <price2> 2 ; <price3> 2 .
  };

Insert into <http://mygraph.com>, 9 (or less) triples -- done
No. of rows in result: 1

SQL> SPARQL SELECT ?s (bif:__min(?pr1, ?pr2, ?pr3))
FROM <http://mygraph2.com>
WHERE
  {
    ?s <price1> ?pr1 ;
       <price2> ?pr2 ;
       <price3> ?pr3
  };

Query result:
s                                       callret-1
VARCHAR                                       VARCHAR
_______________________________________________________
http://mygraph2.com/pricing/#QVal3              2
http://mygraph2.com/pricing/#QVal2              1
http://mygraph2.com/pricing/#QVal1              1
```

```
No. of rows in result: 3
```

See Also

__max()

__min_notnull()

__max_notnull()

__max

__max — Returns maximum value among all its arguments.

Synopsis

```
any __max ( arg1 any ,
            arg2 any ,
            ... ,
            argN any ) ;
```

Description

The function returns the maximum value among all values in all its arguments.

Parameters

argI
: Values which can be with type float, integer etc.

Examples

Example 24.192.

Example 1

```
SQL> select __max(12, 2.34, 5, 20.5);
__max
VARCHAR
_______________________________________________________________________________

20.5

1 Rows. -- 16 msec.
```

Example 2

```
SQL> SPARQL
INSERT INTO GRAPH <http://mygraph2.com>
  {
    <http://mygraph2.com/pricing/#QVal1> <price1>  1 ; <price2> 9 ; <price3> 5 .
    <http://mygraph2.com/pricing/#QVal2> <price1>  3 ; <price2> 1 ; <price3> 4 .
    <http://mygraph2.com/pricing/#QVal3> <price1>  6 ; <price2> 2 ; <price3> 2 .
  };

Insert into <http://mygraph2.com>, 9 (or less) triples -- done
No. of rows in result: 1

SQL> SPARQL SELECT ?s (bif:__max(?pr1, ?pr2, ?pr3))
FROM <http://mygraph2.com>
WHERE
  {
    ?s <price1> ?pr1 ;
       <price2> ?pr2 ;
       <price3> ?pr3
  };

Query result:
s                                     callret-1
VARCHAR                                     VARCHAR
_______________________________________________________
http://mygraph2.com/pricing/#QVal3          6
http://mygraph2.com/pricing/#QVal2          4
http://mygraph2.com/pricing/#QVal1          9
```

```
No. of rows in result: 3
```

See Also

__min()

__min_notnull()

__max_notnull()

__max_notnull

__max_notnull — Returns maximum value among all its arguments, ignoring NULLs entirely.

Synopsis

```
any __max_notnull ( arg1 any ,
                    arg2 any ,
                    ... ,
                    argN any ) ;
```

Description

The function returns the maximum value among all values in all its arguments by ignoring NULLs entirely.

Parameters

argI
: Values which can be with type float, integer etc.

Examples

Example 24.193.

Example 1

```
SQL> select __max_notnull (null, 1, 0);
__max_notnull
VARCHAR
_______________________________________________________________________________

1

1 Rows. -- 16 msec.

SQL> select __max (null, 1, 0);
__max
VARCHAR
_______________________________________________________________________________

NULL

1 Rows. -- 0 msec.
```

Example 2

```
SQL> SPARQL
INSERT INTO GRAPH <http://mygraph.com>
  {
    <http://mygraph2.com/pricing/#QVal1> <price1>  1 ;                    <price3> 5 .
    <http://mygraph2.com/pricing/#QVal2>                  <price2> 1 ;    <price3> 4 .
    <http://mygraph2.com/pricing/#QVal3> <price1>  6 ;    <price2> 2                 .
  };

Insert into <http://mygraph.com>, 6 (or less) triples -- done
No. of rows in result: 1

SQL> SPARQL SELECT ?s, (bif:__max_notnull (?pr1, ?pr2, ?pr3))
FROM <http://mygraph.com>
WHERE
  {
    {
      SELECT DISTINCT ?s
      WHERE
```

```
        {
          ?s ?p ?pr .
          FILTER (?p in (<price1>, <price2>, <price3>))
        }
    }
    OPTIONAL { ?s <price1> ?pr1 }
    OPTIONAL { ?s <price2> ?pr2 }
    OPTIONAL { ?s <price3> ?pr3 }
  };

Query result:
s                                       callret-1
VARCHAR                                       VARCHAR
_______________________________________________________
http://mygraph2.com/pricing/#QVal2       4
http://mygraph2.com/pricing/#QVal1       5
http://mygraph2.com/pricing/#QVal3       6

No. of rows in result: 2
```

See Also

__max()

__min()

__min_notnull()

__min_notnull

__min_notnull — Returns minimum value among all its arguments, ignoring NULLs entirely.

Synopsis

```
any __min_notnull ( arg1 any ,
                    arg2 any ,
                    ... ,
                    argN any ) ;
```

Description

The function returns the minimum value among all values in all its arguments by ignoring NULLs entirely.

Parameters

argI
: Values which can be with type float, integer etc.

Examples

Example 24.194.

Example 1

```
SQL> select __min_notnull (null, 1, 0);
__min_notnull
VARCHAR
_______________________________________________________________________________

0

1 Rows. -- 0 msec.
SQL> select __min (null, 1, 0);
__min
VARCHAR
_______________________________________________________________________________

NULL

1 Rows. -- 0 msec.
```

Example 2

```
SQL> SPARQL
INSERT INTO GRAPH <http://mygraph.com>
  {
    <http://mygraph2.com/pricing/#QVal1> <price1>  1 ;                    <price3> 5 .
    <http://mygraph2.com/pricing/#QVal2>                  <price2> 1 ;    <price3> 4 .
    <http://mygraph2.com/pricing/#QVal3> <price1>  6 ;    <price2> 2                 .
  };

Insert into <http://mygraph.com>, 6 (or less) triples -- done
No. of rows in result: 1

SQL> SPARQL SELECT ?s, (bif:__min_notnull (?pr1, ?pr2, ?pr3))
FROM <http://mygraph.com>
WHERE
  {
    {
      SELECT DISTINCT ?s
      WHERE
        {
```

```
          ?s ?p ?pr .
          FILTER (?p in (<price1>, <price2>, <price3>))
        }
    }
    OPTIONAL { ?s <price1> ?pr1 }
    OPTIONAL { ?s <price2> ?pr2 }
    OPTIONAL { ?s <price3> ?pr3 }
  };

Query result:
s                                     callret-1
VARCHAR                                     VARCHAR
_______________________________________________
http://mygraph2.com/pricing/#QVal2       1
http://mygraph2.com/pricing/#QVal1       1
http://mygraph2.com/pricing/#QVal3       2

No. of rows in result: 2
```

See Also

__max()

__min()

__max_notnull()

java_call_method

java_call_method

Synopsis

java_call_method (in *class_name* varchar ,
in *instance_obj* varchar ,
in *method_name* varchar ,
in *method_ret_sig* varchar ,
arg1, ...) ;

Description

Calls a class method using the supplied parameters (if any) and returns the return value (if any). If instance_obj is supplied (not NULL) then this function searches for a non-static method otherwise for static.

Parameters

class_name
: The name of the Java class.

instance_obj
: the Java VM class instance reference value (for example returned by java_new_object VSEI). Can be NULL.

method_name
: the name of the method to call

metjod_ret_name
: the JNI type signature of the method's return value (or V for void).

arg1, ...
: method parameters (as in java_new_object).

See Also

`java_vm_attach()`

`java_set_property()`

`java_get_property()`

`java_load_class()`

`java_new_object()`

`java_vm_detach()`

java_set_property

java_set_property — Sets a Java class property

Synopsis

```
java_set_property ( in class_name varchar ,
                    in instance_obj any ,
                    in field_name varchar ,
                    in field_ret_type_sig varchar ,
                    in field_new_value any ) ;
```

Description

Assigns a new value to either a Java class instance property referenced by instance_obj, or if instance_obj is not supplied (NULL) then sets a static Java class property.

Parameters

class_name
: the name of the java class to set property for.

instance_obj
: the Java VM class instance reference value (for example returned by java_new_object VSEI). Can be NULL.

field_name
: the name of the field in the java class.

field_ret_type_sig
: the JNI type signature of the field.

field_new_value
: the value to be set.

See Also

`java_call_method()`

`java_vm_attach()`

`java_get_property()`

`java_load_class()`

`java_new_object()`

`java_vm_detach()`

java_get_property

java_get_property — Gets a property value from a Java class instance.

Synopsis

java_get_property (in *class_name* varchar ,
in *instance_obj* varchar ,
in *field_name* varchar ,
in *field_ret_type_sig* varchar) ;

Description

Gets a property value from a Java class instance referenced by instance_obj. If instance_obj is not supplied, ie NULL, then it is returned as a static Java class property value.

Parameters

class_name
: the name of the java class to get the property from

instance_obj
: the Java VM class instance reference value (for example returned by java_new_object VSEI). Can be NULL.

field_name
: the name of the field in the java class

field_ret_type_sig
: the JNI type signature of the field.

Examples

Example 24.195. static value - Java's PI value returned as DOUBLE PRECISION

```
java_get_property ('java.lang.Math', NULL, 'PI', 'D');
```

Example 24.196. default time zone name in tz_name

```
tz := java_call_method ('java.util.TimeZone', NULL, 'getDefault', 'Ljava/util/TimeZone;');
tz_name := java_get_property ('java.util.TimeZone', tz, 'getDisplayName', 'Ljava/lang/String;');
```

See Also

`java_call_method()`

`java_set_property()`

`java_vm_attach()`

`java_load_class()`

`java_new_object()`

`java_vm_detach()`

java_load_class

java_load_class — Declares a Java class to a Java VM

Synopsis

java_load_class (in *java_class_name* varchar ,
in *java_class_bytecode* varchar) ;

Description

Defines a java class into the running Java VM. This is useful for loading .class/.jar/.zip files from a BLOB column or from the Virtuoso WebDAV repository.

Parameters

java_class_name
: the name under which the class is to be defined

java_class_bytecode
: the contents of the .class file.

Examples

Example 24.197. Loading a Java Class

Some sample Java code:

```
java_server.java:

  public class java_server {
    public static int property;
  }
```

Compiling it makes a java_server.class. Loading it in Virtuoso is as follows:

```
  java_load_class ('java_server', file_to_string ('java_server.class'));
```

See Also

`java_call_method()`

`java_set_property()`

`java_get_property()`

`java_vm_attach()`

`java_new_object()`

`java_vm_detach()`

java_new_object

java_new_object — Creates and instance of a Java class.

Synopsis

```
any java_new_object ( in class_name varchar ,
                      in arg1 any ,
                      ... ) ;
```

Description

Creates an instance of a java class, makes a global reference in the Java VM and returns it to Virtuoso as a PL object reference value.

Parameters

class_name
: the name of the java class to be instantiated. (eg: java.util.Date)

arg1, ...
: the arguments to the constructor called.

Return Types

Virtuoso object reference value.

Examples

Example 24.198. Creating new Java class instances

creates a new java.util.Date class instance and initializes it with the current time from Virtuoso.

```
cal := java_new_object ('java.util.Date', vector ('L', msec_time()));
```

creates a new java.util.Date class instance and initializes it with the current time from java VM.

```
cal := java_new_object ('java.util.Date');
```

See Also

`java_call_method()`

`java_set_property()`

`java_get_property()`

`java_load_class()`

`java_vm_attach()`

`java_vm_detach()`

java_vm_attach

java_vm_attach

Synopsis

java_vm_attach (in *classpath* varchar ,
in *vm_options* any) ;

Description

Ensures that the current working thread is attached to the Java VM. It operates as follows:

- if the Java VM is not created it creates it.
- if the java VM is running, but the current working thread is not attached as a Java VM thread it attaches it.
- if none of the above it returns.

The `java_vm_attach()` function is called implicitly in each of the other VSEs, and also when allocating, copying or deleting a Virtuoso/PL reference to a Java VM object values.

If the Java VM is already initialized and the classpath is supplied it will throw a SQL error. If the Java VM is not initialized, but it is required to execute a statement the server will implicitly call `java_vm_attach (NULL);` . The Virtuoso Java VM integration binary works with JDK 1.2 and later.

Parameters

classpath
: The classpath string to be supplied to the Java VM. If classpath is NULL the server OS environment variable CLASSPATH will be used instead.

vm_options
: A vector of name/value pairs for the Java VM initialization parameters. The format and values of the parameters is described in the JNI Enhancements Introduced in version 1.2 of the JavaTM 2 SDK document - the description of JNI_CreateJavaVM Invocation API function.

Examples

Example 24.199. Initializing the Java VM

This example initializes the Java VM by supplying a classpath of: `CLASSPATH=/usr/local/virtuoso/classes:/usr/local/jakarta-tomcat-3.3.1/lib/tomcat.jar` and sets the `tomcat.home` Java system property to `/usr/local/jakarta-tomcat-3.3.1` It is the equivalent of executing JDK/JRE 1.2 unix java tool using: `java -cp /usr/local/virtuoso/classes:/usr/local/jakarta-tomcat-3.3.1/lib/tomcat.jar -Dtomcat.home=/usr/local/jakarta-tomcat-3.3.1`

```
java_vm_attach ('/usr/local/virtuoso/classes:/usr/local/jakarta-tomcat-3.3.1/lib/tomcat.jar',
  vector ('-Dtomcat.home=/usr/local/jakarta-tomcat-3.3.1, 0);
```

See Also

`java_call_method()`

`java_set_property()`

`java_get_property()`

`java_load_class()`

`java_new_object()`

`java_vm_detach()`

java_vm_detach

java_vm_detach — Detaches the current Virtuoso working thread from the Java VM.

Synopsis

```
java_vm_detach (  );
```

Description

Detaches the current Virtuoso working thread from the Java VM.

See Also

`java_call_method()`

`java_set_property()`

`java_get_property()`

`java_load_class()`

`java_new_object()`

`java_vm_attach()`

jvm_ref_import

jvm_ref_import — Creates XML description of Java class

Synopsis

```
varchar jvm_ref_import ( in files any ,
                         in classes any ) ;
```

Description

This function will returns an XML description of the selected classes from the source files.

The XML produced by the JVM_REF_IMPORT can be supplied to the predefined XSL style sheet `__javavm_type` to produce the CREATE TYPE statements:

```
select xslt ('__javavm_type', xml_tree_doc (JVM_REF_IMPORT (files, classes)));
```

This can also be achieve directly using a single call to:

```
IMPORT_JAR (in files any, in classes any)
```

Parameters

files
: Null, string or vector of strings. This can contain one or many Java .class files, .zip or .jar files to pick classes from. Null means only the classes specified in the classes argument regardless of their physical location.

classes
: Wildcard string or a vector of strings specifying which classes to describe in the resulting XML.

Return Types

A varchar of XML that describes the selected Java classes.

Examples

Example 24.200. Simple Use

```
JVM_REF_IMPORT (NULL, vector ('java.lang.Object', 'java.lang.System'));
```

This will return an XML describing the Java classes java.lang.Object and java.lang.System. Because the java.lang.System class is a subclass of java.lang.Object the XML description for java.lang.System will contain a reference to java.lang.Object and only the methods/members defined in java.lang.System (because the methods/members of java.lang.Object will be inherited).

```
JVM_REF_IMPORT (NULL, 'java.lang.System');
```

As opposed to the above example this will create an XML description only for the java.lang.System class, but will also add the inherited methods/members (from java.lang.Object) as if they were methods/members of the java.lang.System class.

See Also

`import_jar()`

lcase

lcase — Converts string argument to lower case

Synopsis

lcase (*str* string);

Description

lcase returns a copy of string str with all the uppercase alphabetical characters converted to corresponding lowercase letters. This includes also the diacritic letters present in the ISO 8859/1 standard in range 192 - 222 decimal, excluding the character 223, German double-s which stays the same.

lower is an alias for lcase.

```
lcase('AbracadabrA')  -> 'abracadabra'
```

ldap_search

ldap_search — Search in an LDAP server.

Synopsis

```
any ldap_search ( in server_url varchar ,
                  in try_tls integer ,
                  in base varchar ,
                  in filter varchar ,
                  in username varchar ,
                  in password varchar ) ;
```

Description

This function performs a search in the LDAP server. It returns control to the Virtuoso/PL environment only after all of the search results have been sent by the server or if the search request is timed out by the server. The result of the search (attributes, names of the attributes, etc.) will be returned as an array result. Options to the LDAP search can be passed as an array.

Parameters

server_url
: The server URL has three parts, <protocol>://<host>:<port>. Missing parameters will be defaulted to <ldap://localhost:389> .

try_tls
: try_tls is a flag that tells the client to perform a handshake with the LDAP server using a secure connection. This is only applicable to the ldap:// protocol and not ldaps://. If a secure connection cannot be made, the connection will be insecure.

base
: base is a string representing the DN base of the search.

filter
: Filter is a string representation of the filter to apply in the search. Simple filters can be specified as attributetype=attributevalue . More complex filters are specified using a prefix notation according to the following BNF:

username
: username authorization credential

password
: password authorization credential

Return Types

This function returns an array consisting of the following elements:

```
<entry type>, (<attribute name>, (<value 1>, <value 2> ...))
```

The *entry type* can be the keyword 'entry' for search entry, 'reference' for search reference, 'extended' for extended result, or 'result' for result from search. When you specify 'result', the returned array consists of 'error' and 'error message' keywords corresponding to error codes and error descriptions.

Errors

Table 24.47. Errors signalled by ldap_search

SQLState	Error Code	Error Text	Description

SQLState	Error Code	Error Text	Description
2E000	LD020	Failed to load the wldap32.dll	
2E000	LD005	Failed to initialize LDAP connection: <reason> (<reason code>)	
2E000	LD006	Failed to set LDAP version option: <reason> (<reason code>)	
2E000	LD016	Failed to start TLS: <reason> (<reason code>)	
28000	LD007	Failed to bind synchronous LDAP connection: <reason> (<reason code>)	
42000	LD008	Failed to search	
39000	LD002	Failed to parse LDAP reference response	
39000	LD003	Failed to parse LDAP extended result	
39000	LD004	Failed to parse LDAP extended partial result	

Examples

Example 24.201. Using ldap_search

```
....
declare result any;

-- without authentication
result := ldap_search ('ldap://localhost', 0, 'c=US', '(cn=SomeBody*)', NULL);

or

-- with authentication
result := ldap_search ('ldap://localhost', 0, 'c=US', '(cn=SomeBody*)',
  'cn=root,o=opl,c=US', 'secret');

-- the result may be following array:
-- ("entry"
     ("dn" "cn="John Atanasov",mail=hellraisor@hotmail.com,c=US,o=hotmail.com"
      "mail" ("hellraisor@hotmail.com" )
      "cn" ("John Atanasov" )
      "o" ("hotmail.com" )
      "l" ("SOFIA" )
      "givenName" ("John" )
      "surname" ("Atanasov" ))
     "result"
       ("error" "0" "error message" "Success" ))

...
```

See Also

`ldap_add()`, `ldap_delete()`, `ldap_modify()`

ldap_delete

ldap_delete — Remove a leaf entry in the LDAP Directory Information Tree.

Synopsis

```
int ldap_delete ( in server_url varchar ,
                  in try_tls integer ,
                  in entrydn varchar ,
                  in username varchar ,
                  in password varchar ) ;
```

Description

This function removes a leaf entry in the LDAP Directory Information Tree.

Parameters

server_url
: The server URL has three parts, <protocol>://<host>:<port>. Missing parameters will be defaulted to <ldap://localhost:389> .

try_tls
: try_tls is a flag that tells the client to perform a handshake with the LDAP server using a secure connection. This is only applicable to the ldap:// protocol and not ldaps://. If a secure connection cannot be made, the client will fall back to using an insecure connection.

entrydn
: entrydn is a qualified string which represents the LDAP DN entry to be deleted.

username
: username authorization credential

password
: password authorization credential

Return Types

Zero for success or error code.

Errors

Table 24.48. Errors signalled by `ldap_delete`

SQLState	Error Code	Error Text	Description
2E000	LD005	Failed to initialize LDAP connection: <reason> (<reason code>)	
2E000	LD006	Failed to set LDAP version option: <reason> (<reason code>)	
2E000	LD016	Failed to start TLS: <reason> (<reason code>)	
28000	LD007	Failed to bind synchronous LDAP connection: <reason> (<reason code>)	
39000	LD012	Failed to delete the DN entry: <reason> (<reason code>)	

Examples

Example 24.202. Using ldap_delete

```
if (not ldap_delete ('ldap://localhost', 0,
  'cn=John Atanasov,mail=hellraisor@hotmail.com,c=US,o=hotmail.com', NULL))
  {
    -- the entry removed successfully, we can perform some other action
  }
```

See Also

`ldap_add()`, `ldap_modify()`, `ldap_search()`

ldap_add

ldap_add — Adds a new entry to an LDAP directory.

Synopsis

```
int ldap_add ( in server_url varchar ,
               in try_tls integer ,
               in data varchar ,
               in username varchar ,
               in password varchar ) ;
```

Description

This function adds a new entry to the LDAP directory.

Parameters

server_url
: The server URL has three parts, <protocol>://<host>:<port>. Missing parameters will be defaulted to: <ldap://localhost:389> .

try_tls
: try_tls is a flag that tells the client to perform a handshake with the LDAP server using a secure connection. This is only applicable to the ldap:// protocol and not ldaps://. If the secure connection attempt fails, the client will fall back to insecure connect.

data
: data is an array with name/value pairs representing the data to be added.

username
: username authorization credential

password
: password authorization credential

Return Types

The error status code is returned. Zero for success.

Errors

Table 24.49. Errors signalled by `ldap_add`

SQLState	Error Code	Error Text	Description
2E000	LD005	Failed to initialize LDAP connection: <reason> (<reason code>)	
2E000	LD006	Failed to set LDAP version option: <reason> (<reason code>)	
2E000	LD016	Failed to start TLS: <reason> (<reason code>)	
28000	LD007	Failed to bind synchronous LDAP connection: <reason> (<reason code>)	
2E000	LD004	The DN must be supplied	
2E000	LD004	Failed to modify err=<reason code (<reason>)	

Examples

Example 24.203. Using ldap_add

```
create procedure
sam_ldap_add (in s1 varchar, in s2 varchar)
{
  declare res, vec any;
  declare _from_add integer;

  vec := vector ('dn', concat ('cn=', s1, ' ', s2, ',o=opl,c=BG'),
         'cn', vector (concat (s1, ' ', s2)),
         'mail', vector('mail@techno-link.com'),
         'sn', vector(s2), 'telephoneNumber', vector('032-947020', '032-633710', '048 850 760'),
         'objectClass', vector('inetorgperson' ));

  _from_add := ldap_add ('ldap://buba:389', 0, vec, NULL);

  return (_from_add);
};
```

See Also

`ldap_delete()`, `ldap_modify()`, `ldap_search()`

ldap_modify

ldap_modify — Modifies an existing LDAP directory.

Synopsis

```
int ldap_modify ( in server_url varchar ,
                  in try_tls integer ,
                  in data varchar ,
                  in username varchar ,
                  in password varchar ) ;
```

Description

This function modifies an existing LDAP directory entry.

Parameters

server_url
: The server URL has three parts, <protocol>://<host>:<port>. Missing parameters will be defaulted to: <ldap://localhost:389> .

try_tls
: try_tls is a flag that tells the client to perform a handshake with the LDAP server using a secure connection. This is only applicable to the ldap:// protocol and not ldaps://.

data
: data is an array with name/value pairs representing the data of the modified entry.

username
: username authorization credential

password
: password authorization credential

Return Types

The error status code is returned.

This function will return zero on success or non-zero in case of a failure. In case of a failure the signal will be raised according to SQL error codes and Virtuoso internal error codes.

The reserved prefix for internal errors is 'LD'.

Errors

Table 24.50. Errors signalled by `ldap_modify`

SQLState	Error Code	Error Text	Description
2E000	LD005	Failed to initialize LDAP connection: <reason> (<reason code>)	
2E000	LD006	Failed to set LDAP version option: <reason> (<reason code>)	
2E000	LD016	Failed to start TLS: <reason> (<reason code>)	
28000	LD007	Failed to bind synchronous LDAP connection: <reason> (<reason code>)	
2E000	LD004	The DN must be supplied	
2E000	LD004	Failed to modify err=<reason code (<reason>)	

Examples

Example 24.204. Using ldap_modify

```
create procedure
sam_ldap_modify (in s1 varchar, in s2 varchar)
{
  declare res, vec any;
  declare _from_modify integer;

  vec := vector ('dn', concat ('cn=', s1, ' ', s2, ',o=opl,c=BG'), 'mail',
    vector('new_mail@techno-link.com'), 'telephoneNumber',
    vector('123-45-00', '032-633710', '048 850 760') );

  _from_modify := ldap_modify ('ldap://localhost:389', 0, vec, NULL);

  return (_from_modify);
};
```

See Also

`ldap_add()`, `ldap_delete()`, `ldap_search()`

left

left — returns substring taken from left of string argument

Synopsis

left (*str* string ,
count integer) ;

Description

left returns a subsequence of string str, taking count characters from the beginning of the string.

If count is zero an empty string '' is returned.

If length of str is less than count then a copy of the whole str is returned.

```
left('AbracadabrA',4) -> 'Abra'
```

length

length — Get length of argument

Synopsis

```
integer length ( in arg any );
```

Description

Returns the length of its argument.

Table 24.51. Value returned by `length` by argument datatype

Argument type	Value returned
string	number of characters
array	number of elements
NULL	zero
LOB	length if known (see below)
string_session	number of bytes written into the session
composite	sum of byte lengths of elements consisting the composite

Note

In case of a LOB that does not have a length as in the case of one supplied by a client with SQLPutData and not yet stored, a zero is returned.

Parameters

arg
: Any type that can be tested for length.

Return Values

Integer length of the argument

Errors

Table 24.52. Errors signalled by `length`

sqlstate	error code	error text	
22023	SR015	Function length is not applicable to XML tree entity	`Length` cannot calculate a length for a non-persistent XML entity.
22023	SR016	Function length does not handle arguments of type x	An invalid data type was passed as an argument to `length`

Examples

Example 24.205. Some uses of length()

```
SQL> select length('abacus'), length (NULL);
callret   callret
INTEGER   INTEGER
_____________________________________________________________________________

6         0
```

```
1 Rows. -- 11 msec.
SQL> select max (length (ProductName)) from Demo.demo.Products;
flag
INTEGER
_______________________________________________________________________________

32

1 Rows. -- 61 msec.
```

LFS_EXP

LFS_EXP — Export retrieved web site to the local file system

Synopsis

```
WS.WS. LFS_EXP ( in host varchar ,
                 in url varchar ,
                 in root varchar ,
                 in dst varchar ) ;
```

Description

This function is used to export local content retrieved from a Web Robot Copy to the local file system.

Parameters

host
: The target host name

url
: start path on target

root
: local WebDAV collection that contains the retrieved content

dst
: absolute path to the file system directory to export content to

Return Types

The function returns a non-zero value (can be an error description) if an error occurred while exporting.

Examples

Example 24.206. Exporting to the Filesystem

```
WS.WS.LFS_EXP ('www.foo.com', '/help/', 'sites/www_foo_com', '/tmp/');
WS.WS.DAV_EXP ('www.foo.com', '/help/', 'sites/www_foo_com',
  'http://www.mydrive.com:8990/DAV/sites/');
```

See Also

WS.WS.DAV_EXP

lh_get_handler

lh_get_handler — Returns language handler

Synopsis

```
lang_handler_t * lh_get_handler ( const char * language_name );
```

Description

Returns language handler

Parameters

language_name
: Name of language handler.

Return Types

Returns language handler for given language, or universal 'x-any' handler if no matches found. Unlike elh_get_handler, it will never return NULL.

See Also

elh_get_handler

elh_load_handler

lh_load_handler

lh_load_handler

lh_load_handler — Loads given handler in global table of the server

Synopsis

```
int lh_load_handler ( lang_handler_t * new_handler );
```

Description

Loads given handler in global table of the server, using ISO 639 and RFC 1766 language IDs of the handler as keys for future table lookups. If another handler was already specified for given RFC 1766 id, table entry will be updated and will refer to new handler. If another handler was already specified for given ISO 639 id, it will be replaced only if new handler has ISO 639 language ID equal to its RFC 1766 ID.

Please do not load custom versions of 'x-any' and 'x-ftq-x-any' handlers.

Parameters

new_handler
: Pointer to the structure that lists names of the handler and its callback functions.

See Also

elh_get_handler()

elh_load_handler()

lh_get_handler()

locate

locate — returns the starting position of the first occurrence of an substring in a string

Synopsis

```
integer locate ( in string_exp1 varchar ,
                 in string_exp2 varchar ,
                 in start integer ) ;
```

Description

Returns the starting position of the first occurrence of string_exp1 within string_exp2. The search for the first occurrence of string_exp1 begins with the first character position in string_exp2 unless the optional argument, start, is specified. If start is specified, the search begins with the character position indicated by the value of start. The first character position in string_exp2 is indicated by the value 1. If string_exp1 is not found within string_exp2, the value 0 is returned.

log

log — calculate natural logarithm of an expression

Synopsis

log (in *x* double precision);

Description

log calculates the natural logarithm of its argument and returns it as a IEEE 64-bit float.

Parameters

x
: double precision

Return Values

log.

Examples

Example 24.207. Simple Examples

```
SQL> select concat ('the LOG of 0.513513 is: ', cast (log (0.513513) as varchar));
callret
VARCHAR
_______________________________________________________________________________

the LOG of 0.513513 is: -0.666480

1 Rows. -- 4 msec.
```

See Also

exp, log10, power, sqrt

log10

log10 — Calculate 10-based logarithms

Synopsis

log10 (in *x* double precision);

Description

log10 calculates the 10-based logarithm of its argument and returns it as a IEEE 64-bit float.

Parameters

x
: double precision

Return Values

Double precision.

Examples

Example 24.208. Simple Examples

```
SQL> select concat ('the LOG10 of 0.513513 is: ', cast (log10 (0.513513) as varchar));
callret
VARCHAR
_______________________________________________________________________________

the LOG10 of 0.513513 is: -0.289449

1 Rows. -- 3 msec.
```

See Also

log , exp , power , sqrt ,

log_enable

log_enable — controls transaction logging and in-statement autocommit

Synopsis

log_enable (in *bits* integer ,
in *quiet* integer) ;

Description

The `log_enable` function allows enabling or disabling regular transaction logging or autocommit after every changed row. The parameter *`bits`* is a bitmask.

Bit1
: Bit1 controls whether the transactions are written to the transaction log file (virtuoso.trx) or not.

Bit2
: Bit2 controls manual or autocommit mode.

See Also

`log_text`

log_message

log_message — print output into the system log file

Synopsis

log_message (in *str* varchar);

Description

log_message Appends a line to the system log file (typically virtuoso.log).

Parameters

str
: An informative string to append to the log file.

Return Values

None

Examples

Example 24.209. Simple example

```
SQL> log_message('1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62
68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97
Done. -- 0 msec.
SQL> ^D


bash$ tail virtuoso.log
16:05:48 INFO: PL LOG: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
66 67 68 69 70
```

See Also

dbg_printf

log_text

log_text — inserts statements into the roll forward log

Synopsis

```
log_text ( in text varchar ,
           in arg_1 any ,
           ... ) ;
```

Description

The log_text function can be used to insert a SQL statement into the roll forward log.

The log_text function causes the SQL text given as first argument to be executed at roll forward time with the following arguments as parameters, bound from left to right to the parameter markers in the statement ('?'). There can be a maximum of 8 parameters but these can be arrays.

Parameters

text
: VARCHAR SQL statement to be added in the transaction log.

arg1..arg8
: Parameters to be passed to logged statement.

Examples

Example 24.210. Log a procedure call

Log a procedure call instead of its effects.

```
create procedure xx ()
{
  log_text ('xx (?)', arg);
  log_enable (0);
  ... action code
  log_enable (1);
}
```

See Also

`log_enable`

read_log

read_log — reads Virtuoso log

Synopsis

read_log (in *file* varchar ,
in *pos* integer) ;

Description

The `read_log` function reads from the Virtuoso Transactions log file from a given position.

Parameters

file
: Virtuoso transaction log file.

pos
: A given position from which the read to start from.

Return Types

The function returns array of transaction values.

Examples

Example 24.211. Sample example

```
-- insert sample data so to change the rdf_quad index:
SQL> SPARQL INSERT INTO <g> { <s> <p> <o> };

-- create the following example procedure
SQL>
create procedure rlt (in f any, in inpos int := 0)
{
  declare h, op, g, s, p, o any;
  declare pos int;
  result_names (op, g, s, p, o);
  h := file_open (f, inpos);
  declare r, rr any;
  while ((rr := read_log (h, pos)) is not null)
    {
      declare rw, k any;
      declare i int;
      rw := null;
      k := null;
      for (i := 1; i < length (rr); i := i + 1)
        {
          r := rr[i];
          if (r[0] = 13) -- key insert
            {
              rw := r[2];
              op := 'I';
            }
          else if (r[0] in (1,8,9)) -- insert,soft,replacing
            {
              rw := r[1];
              op := 'I';
            }
          else if (r[0] in (3,14)) -- delete
            {
              rw := r[1];
              op := 'D';
```

```
            }
          if (rw is not null)
            {
              k := rw[0];
              if (k = 273) -- RDF_QUAD, should check with SYS_KEYS
                {
                  result (op, __ro2sq (rw[1]), __ro2sq (rw[2]), __ro2sq (rw[3]), __ro2sq (rw[4]));
                }
            }
        }
    }
  result (pos + inpos, '', '', '', '');
}
;

-- Call the procedure:

-- in case of no changes to the rdf_quad index are done, it will return:
SQL> rlt('tmp/Virtuoso.trx');
Query result:
op        g   s   p    o
ANY   ANY ANY ANY  ANY
8403
No. of rows in result: 1

-- in case of changes to the rdf_quad index are done ( example with the short INSERT we did above), it wi
SQL> rlt('tmp/Virtuoso.trx');
Query result:
op         g   s   p    o
ANY    ANY ANY ANY  ANY
I      g   s   p    o
71446
No. of rows in result: 2
```

See Also

log_enable

log_text

log

log10

lower

lower — returns a lower case version of its argument

Synopsis

lower (*str* string);

Description

lcase returns a copy of string str with all the uppercase alphabetical characters converted to corresponding lowercase letters. This includes also the diacritic letters present in the ISO 8859/1 standard in range 192 - 222 decimal, excluding the character 223, German double-s which stays the same.

lower is just an alias for lcase.

```
lcase('AbracadabrA')  -> 'abracadabra'
```

ltrim

ltrim — removes specific characters from a string

Synopsis

ltrim (*str* string ,
trimchars string) ;

Description

ltrim returns a copy of subsequence of string str with all the characters present in trimchars trimmed off from the beginning. If the second argument is omitted, it is a space ' ' by default.

rtrim is similar except that it trims from the right.

trim trims from both ends.

```
concat('*',trim('   SIMURG   '),'*') -> '*SIMURG*'
ltrim('AbracadabrA','bAr')  -> 'acadabrA'
rtrim('AbracadabrA','bAr')  -> 'Abracada'
trim('AbracadabrA','bAr')    -> 'acada'
```

__dbf_set

__dbf_set — Sets vector execution parameters.

Synopsis

__dbf_set (in *flag_name* varchar ,
out *flag_value* any) ;

Description

Sets at run time by a dba group user the vector execution parameters.

Parameters

flag_name
: Name of the flag.

flag_value
: Value of the flag.

Examples

Example 24.212. Simple Use

Set the maximum number of outstanding parallel work units to 16:

```
SQL> __dbf_set ('enable_qp', 16);
```

stat_import

stat_import — Imports database statistics

Synopsis

stat_import (in *st_name* varchar) ;

Description

This function imports produced database statistics.

Parameters

st_name
: Name of the statistics file.

Examples

Example 24.213. Simple Use

```
-- produce statistics for a given database:
SQL> string_to_file('stat.dv', serialize(stat_export()), -2);

-- import the produced statistics:
SQL> stat_import (deserialize (file_to_string ('stat.dv')));
```

See Also

`stat_export()`

stat_export

stat_export — Produces a statistics summary.

Synopsis

```
stat_export (   );
```

Description

This function produces a statistics summary that can be read back into another database with the `stat_import()` function

Parameters

Examples

Example 24.214. Simple Use

```
-- produce statistics for a given database:
SQL> string_to_file('stat.dv', serialize(stat_export()), -2);
```

See Also

`stat_import()`

make_array

make_array — returns a new array

Synopsis

```
array make_array ( in length integer ,
                   in content varchar ) ;
```

Description

This returns an array of length elements with the content element type. The initial content of the array is undefined.

Parameters

length
: Number of elements to be allocated in the resultant array.

content
: String that specifies the data type of the array to make. Valid strings are 'float', 'double', 'long' or 'any'. These correspond respectively to the C types long (32 bit signed), float (IEEE 32-bit), double (IEEE 64-bit) and untyped. The untyped array may hold a heterogeneous collection of any Virtuoso data types, including other arrays. The initial content of the array is undefined.

Return Values

An array of specified length and data type

Errors

Parameter data type checking errors

Table 24.54. Errors signalled by make_array()

sqlstate	error code	error text
22003	SR021	make_array called with an invalid count <num>
22023	SR022	Type for make_array must be float, double, long or any

make_string

make_string — make a string

Synopsis

make_string (in *count* integer) ;

Description

make_string returns a new string of length count, filled with binary zeros.

If count is zero, an empty string '' is returned.

Parameters

count
: Length of the string to be generated.

Return Values

A string with defined length is returned.

Examples

Example 24.215. Using make_string and aref

Make a string and fill it with character sequence containing the alphabet upper case characters from A to Z.

```
SQL> create procedure
alphabet_string ()
{
  declare _inx integer;
  declare _str varchar;
  _str := make_string (26);

  while (_inx < 26)
    {
      aset (_str, _inx, _inx + 65);
      _inx := _inx + 1;
    }
  return (_str);
}
;

Done. -- 6 msec.
SQL> select alphabet_string ();
callret
VARCHAR NOT NULL
_________________________________________________________________________

ABCDEFGHIJKLMNOPQRSTUVWXYZ

1 Rows. -- 4 msec.
```

See Also

vector(),aref,aset

md5

md5 — returns the md5 checksum of its argument

Synopsis

```
checksum md5 ( in str varchar );
```

Description

`md5` calculates the MD5 checksum of its argument. The md5 message digest algorithm is defined in RFC1321 .

Parameters

str
: A string or string_output containing the data for calculating the message digest.

Return Types

A string of 32 lowercase alphanumeric characters.

Errors

Table 24.55. Errors signalled by

SQLState	Error Code	Error Text	Description

Examples

Example 24.216. Simple example

```
SQL> select md5 ('blah blah');
callret
VARCHAR
_______________________________________________________________________________

ae661d08d1ca1576a6efcb82b7bc502f

1 Rows. -- 4 msec.
```

See Also

RFC1321

md5_init

md5_init — returns the string serialization of a new md5 context

Synopsis

```
new md5 checksum
context md5_init (       );
```

Description

This function initializes an MD5_CTX, converts it into varchar form and returns this representation. Should be used with md5_update/md5_final.

See Also

md5_update , md5_final

md5_update

md5_update — returns the updated md5 context serialized as varchar

Synopsis

```
md5 context update md5_update ( in ctx varchar ,
                                in data varchar ) ;
```

Description

This function updates MD5_CTX with data parameter and returns the (deserialized from ctx parameter) updated context.

See Also

md5_init md5_final ,

md5_final

md5_final — returns the md5 checksum given an initialized md5 context

Synopsis

md5_final (in *ctx* varchar) ;

Description

This function finalizes the MD5_CTX and returns the final checksum.

Parameters

ctx
: A MD5_CTX

Return Types

A string md5 checksum.

See Also

`md5_init` `md5_update` ,

mime_body

mime_body — used to compose multipart/mixed MIME message body

Synopsis

```
varchar mime_body ( in mime_parts any );
```

Description

The `mime_body()` procedure is used to compose a multipart/mixed MIME body. it takes only one parameter, mime_parts, an array of `mime_part` elements such as those produced by the `mime_part()` function.

Parameters

mime_parts
: Array of mime parts such as those produced by the mime_part() function.

Return Types

The function returns a multipart mixed MIME body as text.

Examples

Example 24.217. Creating a MIME message body

This example will show the output in ISQL of producing a two part MIME message using the mime_body() and mime_part() functions.

```
SQL> select MIME_BODY (vector (MIME_PART (null,null,null,'this is a first'),
MIME_PART (null,null,'base64','this is a second')));
callret
VARCHAR
_______________________________________________________________________________

Date: Tue, 25 Jun 2002 11:13:05 GMT
Content-Type: multipart/mixed; boundary="----43c7f8b88e14a87dc3f2d840db93c731"
Mime-Version: 1.0
X-Mailer: Virtuoso

This is a multi-part message in MIME format.

------43c7f8b88e14a87dc3f2d840db93c731
Content-Type: text/plain
Content-Transfer-Encoding: 8bit

this is a first
------43c7f8b88e14a87dc3f2d840db93c731
Content-Type: text/plain
Content-Transfer-Encoding: base64

dGhpcyBpcyBhIHNlY29uZA==
------43c7f8b88e14a87dc3f2d840db93c731--

1 Rows. -- 21 msec.
```

See Also

`mime_part()`

mime_body()

mime_part

mime_part — used to compose a MIME message body part.

Synopsis

```
any mime_part ( in content_type varchar ,
                in content_disposition varchar ,
                in transfer_encoding varchar ,
                in data varchar ) ;
```

Description

This function is used to make a MIME part that can be used with the `mime_body()` function.

Parameters

content_type
: This parameter is used to specify the media type and subtype of data in the body of a message and to fully specify the native representation of such data. This header embodies much of the power of MIME. The IETF can add new official content types. Additionally, private content-type values can be defined by anyone. Such private content types have values of "x-something" or "X-something", where "something" can take on any value.

content_disposition
: This parameter provides information about how to present a body part. When a body part is to be treated as an attached file, the Content-Disposition header should include a file name parameter.

transfer_encoding
: The encoding method to use, such as base64.

data
: The content of the message.

Return Types

The function returns a MIME part for use with the `mime_body()` function.

Examples

Example 24.218. Creating a MIME message body

This example will show the output in ISQL of producing a two part MIME message using the mime_body() and mime_part() functions.

```
SQL> select MIME_BODY (vector (MIME_PART (null,null,null,'this is a first'),
MIME_PART (null,null,'base64','this is a second')));
callret
VARCHAR
_______________________________________________________________________________

Date: Tue, 25 Jun 2002 11:13:05 GMT
Content-Type: multipart/mixed; boundary="----43c7f8b88e14a87dc3f2d840db93c731"
Mime-Version: 1.0
X-Mailer: Virtuoso

This is a multi-part message in MIME format.

------43c7f8b88e14a87dc3f2d840db93c731
Content-Type: text/plain
Content-Transfer-Encoding: 8bit
```

```
this is a first
------43c7f8b88e14a87dc3f2d840db93c731
Content-Type: text/plain
Content-Transfer-Encoding: base64

dGhpcyBpcyBhIHNlY29uZA==
------43c7f8b88e14a87dc3f2d840db93c731--

1 Rows. -- 21 msec.
```

See Also

mime_part()

mime_body()

mime_tree

mime_tree — parses MIME messages into an array structure

Synopsis

```
array mime_tree (
                 in message_text
                 string ,
                 in flag integer ) ;
```

Description

This function is intended to parse MIME (RFC2045) messages (coming from a RFC822 or HTTP sources). It parses the text and produces an array structure representing the structure of the MIME message. It copies into the structure MIME headers, but for the MIME bodies it only stores start and end offsets, thus optimizing space usage.

The parameters to mime_tree are:

message_text (string, required) - contains the text of the message to be parsed.

flag (integer, optional) - a flag describing the format of the "root" object in the text.
If flag is 1, the "root" message follows RFC822. This means mime_tree will unfold the attributes, will scan for MIME registered header fields and will take their attributes. Alternately this can be a MIME message which needs no unfolding and has attributes separated with semicolon.

If flag is 2, the "root" message follows RFC2045. This means mime_tree will scan for MIME attributes.

In either cases mime_tree will look for the Content-Type header field and will parse the "message/rfc822" and "multipart/digest" MIME bodies as nested messages.

mime_tree will return an array of 3 elements (message descriptor) with the following structure:

index 0: an array of attributes, parameters and their values, which can be passed to get_keyword or get_keyword_ucase functions.

index 1: an array of four elements: (0 - starting index of the body into the source_message text (for aref), 1 - ending index of the body, 2 - an message descriptor if the body is recognized to contain a valid RFC822 message; 0 otherwise, 3 - an array of two elements with starting and ending offset if the body is recognized as a valid RFC822 multipart message and has some things after the last MIME boundary, 0 otherwise).

index 2: an array of subpart message descriptors, if the MIME message is recognized as multipart MIME message, 0 otherwise.

Examples

consider the following message text

```
Form: Somebody <someuser@somehost>
Mime-Version: 1.0
Content-Type: "multipart/mixed";
        boundary="--the boundary"
To: self@localhost

This is a multipart MIME message
----the boundary
Content-Type: image/gif; filename="the_big_picture.gif"

GIF........
----the boundary
Content-Type: message/rfc822

From: Ford@Perfect
To: vogon
Mime-Version: 1.0
Content-Type: multipart/alternate; boundary="--sub-boundary"
```

```
This is some Message
----sub-boundary
Content-Type: text/plain

Hi
----sub-boundary
Content-Type: text/html

<P>Hi</P>
----sub-boundary--
Some garbage
----the boundary
Content-Type: text/plain

Some additional text
----the boundary--
Some additional garbage
```

MIME_TREE(the_text, 1) will produce:

```
--- the main message start
(
 ("From", "Somebody <someuser@somehost>",
      "Mime-Version", "1.0", "Content-Type",
      "multipart/mixed",
      "boundary", "--the boundary",
      "To", "self@localhost"),          --- main attributes
 (n1, n2, 0, (mg1, mg2)),               --- main message body
                                            ("This is a multipart MIME message")
 (                                      --- Sub-parts array start
  (                                     --- Sub-Part 1
   ("Content-Type", "image/gif",
        "filename",
        "the_big_picture.gif"),         --- Attributes
   (s2, e2, 0, 0),                      --- body
   0                                    --- no sub parts of the GIF
  ),
  (                                     --- Sub-Part 2
   ("Content-Type", "message/rfc822"),  --- Attributes
   (s3, e3,                             --- the body offsets
    (                                   --- the body is recognized as a message,
                                            so parse it
     ("From", "Ford@Perfect", "To", "vogon",
          "Mime-Version",
          "multipart/alternate",
          "boundary",
          "--sub-boundary"),            --- The body's Attributes
     (ss1, se1, 0, (g2, ge2)),          --- the body's body ("This is some message").
                                            The message has the text
                                            "Some additional garbage"
                                            marked by g2, ge2 offsets
     (                                  --- body's parts
      (                                 --- body's SubPart 1
       ("Content-Type", "text/plain"),  --- attributes
       (ss2, se2, 0, 0),                --- the text "Hi"
       0                                --- no subparts
      ),
      (                                 --- body's SubPart 2
       ("Content-Type", "text/html"),   --- attributes
       (ss3, se3, 0, 0),                --- the HTML paragraph "Hi"
       0                                --- no subparts
      )
     )
    ),                                  --- end of the body's structure
    0                                   --- no trailers
   ),
   0                                    --- no subparts
  ),
  (                                     --- SubPart 3
   ("Content-Type", "text/plain"),      --- attributes
   (s4, e4, 0, (g1, ge1)),              --- the text "Some additional text"
                                            and "Some additional garbage"
   0                                    --- no subparts
```

```
  )
 )                                    --- end of subparts array of the main message
)
```

where the n1, n2, mg1, mg2, s2, e2, s3, e3, ss1, se1, g2, ge2, ss2, se2, ss3, se3, s4, e4, g1, ge1 are offsets, denoting starts and ends of the appropriate pieces within the source message, which can be used by the subseq function:

```
subseq (the_text, g1, ge1) returns the string "Some additional garbage"
```

minute

minute — get minute from a datetime

Synopsis

minute (in *dt* datetime) ;

Description

minute takes a datetime and returns an integer containing a number representing the minute of the datetime.

Parameters

dt
: A datetime .

Return Values

An INTEGER containing the minute.

Examples

Example 24.219. Simple example

Get current minute.

```
SQL> select minute (now ());
callret
INTEGER
_______________________________________________________________________________

58
```

See Also

dayname , dayofmonth , dayofweek , dayofyear , quarter , week , month , year , hour , second , timezone

mod

mod — returns the modulus of its arguments

Synopsis

mod (*dividend* integer ,
divisor integer) ;

Description

mod returns the modulus (i.e. remainder) of the division dividend/divisor. If the divisor is zero the SQL error 22012 "Division by zero" is generated.

```
mod(35,3)                       -> 2
mod(35,-3)                      -> 2
mod(-35,3)                      -> -2
mod(-35,-3)                     -> -2
mod(3,35)                       -> 3
mod(0,7)                        -> 0
mod(60,3)                       -> 0
```

month

month — get number of month from a datetime

Synopsis

month (in *dt* datetime);

Description

month takes a datetime and returns an integer containing a number representing the month of year of the datetime.

Parameters

dt
: A datetime .

Return Values

An INTEGER containing number of the month of year.

Examples

Example 24.220. Simple example

Get current month of year.

```
SQL> select month (now ());
callret
INTEGER
_______________________________________________________________________________

10
```

See Also

dayname , monthname , dayofmonth , dayofweek , dayofyear , quarter , week , year , hour , minute , second , timezone

monthname

monthname — get name of month from a datetime

Synopsis

monthname (in *dt* datetime);

Description

monthname takes a datetime and returns a string containing name of the month represented by the datetime

Parameters

dt
: A datetime .

Return Values

A VARCHAR containing name of the month.

Examples

Example 24.221. Simple example

Get name of current month.

```
SQL> select monthname(now());
callret
VARCHAR
_______________________________________________________________________________

October
```

See Also

dayname , dayofmonth , dayofmonth , dayofweek , dayofyear , quarter , week , month , year , hour , minute , second , timezone

msec_time

msec_time — Get number of milliseconds from system epoch

Synopsis

msec_time (void);

Description

msec_time returns the number of milliseconds since system epoch. It is useful for benchmarking purposes, timing operations, etc.

Parameters

No parameters
: The function does not take parameters.

Return Values

A 32-bit integer no. of milliseconds since system epoch.

Examples

Example 24.222. Simple example

Time a function

```
create procedure
fib (in n integer)
{
  if (n <= 2) return 1;
  return fib (n - 1) + fib (n - 2);
}
;

create procedure
time_fib (in n integer)
{
  declare t,i integer;
  declare msg varchar;

  result_names (msg);

  t := msec_time();
  i := fib (n);
  result (sprintf ('fib (%d) is %d, got it in %d milliseconds.',
                   n, i, msec_time() - t));
}
;

SQL> time_fib(10);
msg
VARCHAR NOT NULL
_______________________________________________________________________________

fib (10) is 55, got it in 10 milliseconds.

1 Rows. -- 21 msec.
```

See Also

now

mts_connect

mts_connect — connects Virtuoso server to MS DTC.

Synopsis

mts_connect (in *reconnect* integer) ;

Description

If reconnect flag is set to non-zero value then forces Virtuoso to connect even it have been connected already. Returns zero if succeeds. If MS DTC service is not available (W2K) or MTS is not running (NT4.0) signals error with code "MX000".

mts_get_timeout

mts_get_timeout — returns timeout of distributed transaction in milliseconds.

Synopsis

```
mts_get_timeout (   );
```

Description

If reconnect flag is set to non-zero value then forces Virtuoso to connect even it have been connected already. Returns zero if succeeds. If MS DTC service is not available (W2K) or MTS is not running (NT4.0) signals error with code "MX000".

mts_set_timeout

mts_set_timeout — sets timeout of distributed transaction.

Synopsis

mts_set_timeout (in *timeout* integer) ;

Description

sets distributed transactions timeout. 'timeout' parameter indicates amount of timeout in milliseconds. If it equals -1 then default timeout of Virtuoso transaction is used (SQL_QUERY_TIMEOUT). This function must be called directly after "SET MTS_2PC=1". The time countdown begins at moment of changing first branch.

mts_status

mts_status — checks status of current transaction or server.

Synopsis

mts_status (in *subject* varchar) ;

Description

Checks status of subject. Subject can be either 'MTS' or 'TRANSACTION'. In the first case this checks if the server is connected to MTS. In the second case, checks if 2pc control is enabled for the current transaction. This function returns status string. For 'MTS' it could be either 'connected' or 'disconnected'. For 'TRANSACTION' - either '2pc enabled' or '2pc disabled'.

sha1_digest

sha1_digest — returns string containing sha1 digest encoded in base64

Synopsis

sha1_digest (in *str* varchar);

Description

Returns string containing sha1 digest encoded in base64. This function has an alias: xenc_sha1_digest ().

Parameters

str
: A varchar value.

Return Values

`sha1_digest` returns a varchar containing sha1 digest encoded in base64.

Examples

Example 24.223. Simple Example

Base64-encode a string

```
SQL> select sha1_digest('demodemo');
callret
VARCHAR
_______________________________________________________________________________

WGPZ5MvfUi6qYuB0f86xxbJJuhM=

1 Rows. -- 25 msec.
```

See Also

`encode_base64`

`md5`

`serialize`

name_part

name_part — Returns portion of dotted name such as a fully qualified table name.

Synopsis

```
varchar name_part ( in string varchar ,
                    in idx integer ,
                    in dstring varchar ) ;
```

Description

The `name_part()` can be used to dissecting parts of a three part names (string where items are divided by dots ".") such as table names or columns names. The table name "DB"."DBA"."SYS_USERS" contains three parts which can be extracted individually using this function providing the correct index from a 0 base: 0 would return "DB", 1 would return "DBA", 2 would return "SYS_USERS".

Parameters

string
: The string to be disected.

idx
: The part index starting from 0.

dstring
: The default value if the found value is null.

Return Types

A string will be returned containing the text from the specified index.

Errors

Table 24.56. Errors signalled by

SQLState	Error Code	Error Text	Description
22023	SR014	Function name_part needs a string as argument 1, not an arg of type [type]	
22023	SR008	Function name_part needs an integer as argument 2, not an arg of type [type]	

Examples

Example 24.224. Using the name_part() function

This simple example shows the 3 parts of a table name being retrieved using the name_part() function from ISQL.

```
SQL> select name_part('"DB"."DBA"."SYS_TABLES"', 0);
callret
VARCHAR
_______________________________________________________________

"DB"

1 Rows. -- 20 msec.
SQL> select name_part('"DB"."DBA"."SYS_TABLES"', 1);
callret
VARCHAR
_______________________________________________________________
```

```
"DBA"

1 Rows. -- 30 msec.
SQL> select name_part('"DB"."DBA"."SYS_TABLES"', 2);
callret
VARCHAR
_______________________________________________________________

SYS_TABLES"

1 Rows. -- 20 msec.
```

nntp_auth_get

nntp_auth_get — returns information about an NNTP server with authorization

Synopsis

```
array nntp_auth_get ( in server varchar ,
                      in user varchar ,
                      in password varchar ,
                      in command varchar ,
                      in group varchar ,
                      in first_message integer ,
                      in last_message integer ) ;
```

Description

The `nntp_auth_get()` is used to retrieve messages from a server requiring authorization. See nntp_get for more information.

Parameters

server
: The host to connect with. IP address or hostname:port. There is no default for port, so to connect to the standard port for NNTP, use <hostname/IP address>:119

user
: The username.

password
: The user password.

command
: Command string . Valid values are:

group
: A string containing name of the news group.

Return Types

A vector of vectors the content depends of command.

Errors

Table 24.57. Errors signalled by nntp_auth_get

SQLState	Error Code	Error Text	Description
22023	NN008	the command is not recognized	
2E000	NN002	Invalid address for News Server at [host]	
08001	NN003	Unable to Contact News Server at [host]	
08006	NN004	Lost connection with NNTP server	
08006	NN008	Misc. error in connection in nntp_auth_get	

nntp_auth_post

nntp_auth_post — Post message to NNTP server with authorization

Synopsis

```
array nntp_auth_post ( in server varchar ,
                       in user varchar ,
                       in password varchar ,
                       in body varchar ) ;
```

Description

`Nntp_auth_post` is used to post a message to the server require authorization.

Parameters

server
: The host to connect with. IP address or hostname:port. There is no default for port, so to connect to the standard port for NNTP, use <hostname/IP address>:119

user
: The username.

password
: The user password.

body
: The body string The structure of the message must comply with RFC 850 (Standard for Interchange of USENET Messages).

Return Types

1 on success or error

Errors

Table 24.58. Errors signalled by

SQLState	Error Code	Error Text	Description
2E000	NN015	Invalid address for News Server at [host]	
08001	NN016	Unable to Contact News Server at [host]	
08006	NN017	Misc. error in connection in nntp_auth_post	

nntp_get

nntp_get — Returns information about an NNTP server.

Synopsis

```
array nntp_get ( in server varchar ,
                 in command varchar ,
                 in group varchar ,
                 in first_message integer ,
                 in last_message integer ) ;
```

Description

`nntp_get()` is used to retrieve messages from a server running the Network News Transfer Protocol (NNTP) as defined in RFC977 . It returns an array whose structure depends on the *command* parameter, thus:

list : an array of vectors, each of which contain the name of the news group, the first message number, the last message number, and a single character 'y' or 'n' as a flag for posting.

group : a vector of three elements: total number of messages in the group, the number of the first message, and the number of the last message.

stat : an array of vectors, each of which contains the number of the message and the message id.

article, body, head : an array of vectors each containing the message number and the requested part of the message.

Parameters

server
: The IP address or <hostname:port> of the host with which to connect. There is no default for port , so to connect to the standard port for NNTP, use <hostname/IP address>:119.

command
: Command string . Valid values are:

group
: A string containing the name of the newsgroup.

Return Types

A vector of vectors the content of which depends on the *command* parameter.

Errors

Table 24.59. Errors signalled by

SQLState	Error Code	Error Text
22023	NN006	the command is not recognized
22023	NN001	Large ID in nntp_id_get
2E000	NN002	Invalid address for News Server at [host]
08001	NN003	Unable to Contact News Server at [host]
08006	NN004	Lost connection with NNTP server
08006	NN005	Misc. error in connection in nntp_get

Examples

Example 24.225. Get remote messages

This example retrieves messages from a remote NNTP server and stores them in a table.

```
create table my_news (m_id integer, m_group varchar, m_text long varchar,
  primary key (m_id, m_group));

create procedure get_my_news (in server varchar, in grp varchar)
{
   declare res, ent any;
   declare i, l integer;
   res := nntp_get (server, 'article', grp, 0, 1000);
   i := 0; l := length (res);
   while (i < l)
         {
     ent := res [i];
           insert replacing my_news (m_id, m_group, m_text)
       values (ent[0], grp, ent[1]);
     i := i + 1;
         }
}
```

Here is a test run. We extract the article subject with the `mail_header()` function.

```
SQL> get_my_news ('news.techno-link.com:119', 'comp.lang.java.api');

SQL> select m_id, mail_header (m_text, 'Subject') from my_news;
m_id              callret
INTEGER NOT NULL  VARCHAR
_______________________________________________________________________________

2                 New java method modifier "partial":  not quite abstract, not quite concrete
3                 Senior Design Project Ideas
4                 java & dummy terminals
5                 Re: What is the equivalent of WORD in Java
6                 Re: What is the equivalent of WORD in Java
```

nntp_post

nntp_post — Post message to NNTP server

Synopsis

```
array nntp_post ( in server varchar ,
                  in body varchar ) ;
```

Description

`Nntp_post` is used to post a message to the server running the Network News Transfer Protocol as defined in the rfc977.

Parameters

server
: The host to connect with. IP address or hostname:port. There is no default for port, so to connect to the standard port for NNTP, use <hostname/IP address>:119

body
: The body string The structure of the message must comply with RFC 850 (Standard for Interchange of USENET Messages).

Return Types

1 on success or error

Errors

Table 24.60. Errors signalled by

SQLState	Error Code	Error Text
2E000	NN011	Invalid address for News Server at [host]
08001	NN012	Unable to Contact News Server at [host]
08006	NN013	Misc. error in connection in nntp_post

Examples

Example 24.226. Post message to remote NNTP

This example post message to a remote NNTP server.

```
SQL> set MACRO_SUBSTITUTION off;
SQL> select nntp_post ('news.openlinksw.co.uk:119',
  'From: "Demo User" user@demo.com\r\nSubject: Test Virtuoso nntp_post function\r\nNewsgroups: openlink.p
callret
VARCHAR
_______________________________________________________________________________

1

1 Rows. -- 782 msec.
SQL>
```

now

now — returns the current transaction timestamp

Synopsis

```
now (   );
```

Description

Now returns the timestamp associated with current transaction as a DATETIME . This value is guaranteed to differ from the timestamp of any other transaction.

Examples

Example 24.227. Get a timestamp

Get a timestamp in human-readable form.

```
SQL> select datestring(now()), cast (now() as VARCHAR);
callret                  callret
VARCHAR                  VARCHAR
_______________________________________________________________________________

2001-10-08 16:31:49.000000  2001-10-08 16:31:49.000000

1 Rows. -- 99 msec.
```

Example 24.228. Table example

Store update time in a column

```
SQL> update test_table set TIME_CHANGED = now();

Done. -- 37 msec.
SQL> select cast (TIME_CHANGED as VARCHAR) from test_table;
callret
VARCHAR
_______________________________________________________________________________

2001-10-08 16:34:28.000000

1 Rows. -- 3 msec.
```

Parameters

`now` has no parameters.

Return Types

A DATETIME timestamp.

Errors

`now` does not return errors.

See Also

`datestring()` , casting , `curdate()` , `curdatetime()` , `curtime()`

DB.DBA.AMAZON_START_INSTANCE

DB.DBA.AMAZON_START_INSTANCE — Starts an existing AMI instance that user has previously stopped.

Synopsis

DB.DBA.AMAZON_START_INSTANCE (in *aws_access_key* varchar ,
in *instance_id* varchar ,
in *secret_key* varchar ,
in *http_proxy* varchar (default null)) ;

Description

Starts an existing AMI that user has previously stopped.

Parameters

aws_access_key
: Amazon Access Key ID.

instance_id
: Image instance ID (check Amazon list of instances).

secret_key
: AWS Security Token.

http_proxy
: Proxy server, can be null or empty.

Examples

Example 24.229. Simple Use

```
create procedure simple_test()
{
  declare access_key, sec_token, p_name varchar;

  access_key := 'AKIAJI7ZL427TI5EDF5A'; -- amazon manager site access key
  sec_token  := 'CGI/UMaXf2LRUctaj/HGJ57UNy/t7fNCshh8wpJg'; -- amazon manager site secret token
  p_name     := 'my_instance_of_linux';

  DB.DBA.AMAZON_START_INSTANCE (access_key, p_name, sec_token);

}
;
```

See Also

DB.DBA.AMAZON_RUN_INSTANCE

DB.DBA.AMAZON_STOP_INSTANCE

DB.DBA.AMAZON_TERMINATE_INSTANCE

DB.DBA.AMAZON_DEREGISTER_IMAGE

DB.DBA.AMAZON_CREATE_IMAGE

DB.DBA.AMAZON_CREATE_VOLUME

DB.DBA.AMAZON_CREATE_SNAPSHOT

DB.DBA.AMAZON_DELETE_SNAPSHOT

DB.DBA.AMAZON_DELETE_VOLUME

DB.DBA.AMAZON_DESCRIBE_IMAGES

DB.DBA.AMAZON_DESCRIBE_INSTANCES

DB.DBA.AMAZON_IMPORT_KEY_PAIR

ApiReference--StartInstances

DB.DBA.AMAZON_RUN_INSTANCE

DB.DBA.AMAZON_RUN_INSTANCE — Runs new AMI instance of image.

Synopsis

DB.DBA.AMAZON_RUN_INSTANCE (
in *aws_access_key* varchar ,
in *image_id* varchar ,
in *secret_key* varchar ,
in *key_name* varchar ,
in *MinCount* integer (default 1) ,
in *MaxCount* integer (default 1) ,
in *http_proxy* varchar (default null)) ;

Description

Runs new AMI instance of image.

Parameters

aws_access_key
: Amazon Access Key ID.

image_id
: The Image ID of the AMI.

secret_key
: AWS Security Token.

key_name
: Name of public key to use (can be used for Linux images). Can be empty or null.

MinCount
: The minimum number of instances to launch. If the value is more than Amazon EC2 can launch, no instances are launched at all. Default is 1.

MaxCount
: The maximum number of instances to launch. If the value is more than Amazon EC2 can launch, the largest possible number above MinCount will be launched instead. Default is 1.

http_proxy
: Proxy server, can be null or empty.

Examples

Example 24.230. Simple Use

```
create procedure simple_test()
{
  declare access_key, sec_token, key_name, p_name varchar;

  access_key := 'AKIAJI7ZL427TI5EDF5A'; -- amazon manager site access key
  sec_token  := 'CGI/UMaXf2LRUctaj/HGJ57UNy/t7fNCshh8wpJg'; -- amazon manager site secret token
  key_name   := 'my_key';
  p_name     := 'my_instance_of_linux';

  DB.DBA.AMAZON_RUN_INSTANCE (access_key, p_name, sec_token, key_name);

}
```

```
;
```

See Also

DB.DBA.AMAZON_START_INSTANCE

DB.DBA.AMAZON_STOP_INSTANCE

DB.DBA.AMAZON_TERMINATE_INSTANCE

DB.DBA.AMAZON_DEREGISTER_IMAGE

DB.DBA.AMAZON_CREATE_IMAGE

DB.DBA.AMAZON_CREATE_VOLUME

DB.DBA.AMAZON_CREATE_SNAPSHOT

DB.DBA.AMAZON_DELETE_SNAPSHOT

DB.DBA.AMAZON_DELETE_VOLUME

DB.DBA.AMAZON_DESCRIBE_IMAGES

DB.DBA.AMAZON_DESCRIBE_INSTANCES

DB.DBA.AMAZON_IMPORT_KEY_PAIR

ApiReference--RunInstances

DB.DBA.AMAZON_STOP_INSTANCE

DB.DBA.AMAZON_STOP_INSTANCE — Stops an existing AMI instance.

Synopsis

DB.DBA.AMAZON_STOP_INSTANCE (in *aws_access_key* varchar ,
in *instance_id* varchar ,
in *secret_key* varchar ,
in *http_proxy* varchar (default null)) ;

Description

Stops an existing AMI instance. Each time user performs transition an instance from stopped to started, OpenLink charges a full instance hour, even if transitions happen multiple times within a single hour.

Parameters

aws_access_key
: Amazon Access Key ID.

instance_id
: Image instance ID (check Amazon list of instances).

secret_key
: AWS Security Token.

http_proxy
: Proxy server, can be null or empty.

Examples

Example 24.231. Simple Use

```
create procedure simple_test()
{
  declare access_key, sec_token, p_name varchar;

  access_key := 'AKIAJI7ZL427TI5EDF5A'; -- amazon manager site access key
  sec_token  := 'CGI/UMaXf2LRUctaj/HGJ57UNy/t7fNCshh8wpJg'; -- amazon manager site secret token
  p_name     := 'my_instance_of_linux';

  DB.DBA.AMAZON_STOP_INSTANCE (access_key, p_name, sec_token);

}
;
```

See Also

DB.DBA.AMAZON_START_INSTANCE

DB.DBA.AMAZON_RUN_INSTANCE

DB.DBA.AMAZON_TERMINATE_INSTANCE

DB.DBA.AMAZON_DEREGISTER_IMAGE

DB.DBA.AMAZON_CREATE_IMAGE

DB.DBA.AMAZON_CREATE_VOLUME

DB.DBA.AMAZON_CREATE_SNAPSHOT

DB.DBA.AMAZON_DELETE_SNAPSHOT

DB.DBA.AMAZON_DELETE_VOLUME

DB.DBA.AMAZON_DESCRIBE_IMAGES

DB.DBA.AMAZON_DESCRIBE_INSTANCES

DB.DBA.AMAZON_IMPORT_KEY_PAIR

ApiReference--StopInstances

DB.DBA.AMAZON_TERMINATE_INSTANCE

DB.DBA.AMAZON_TERMINATE_INSTANCE — Terminates (shuts down) an already created AMI instance.

Synopsis

DB.DBA.AMAZON_TERMINATE_INSTANCE (in *aws_access_key* varchar ,
in *instance_id* varchar ,
in *secret_key* varchar ,
in *http_proxy* varchar (default null)) ;

Description

Terminates (shuts down) an already created instance. This operation is idempotent; if you terminate an instance more than once, each call will succeed.

Parameters

aws_access_key
: Amazon Access Key ID.

instance_id
: Image instance ID (check Amazon list of instances).

secret_key
: AWS Security Token.

http_proxy
: Proxy server, can be null or empty.

Examples

Example 24.232. Simple Use

```
create procedure simple_test()
{
  declare access_key, sec_token, p_name varchar;

  access_key := 'AKIAJI7ZL427TI5EDF5A'; -- amazon manager site access key
  sec_token  := 'CGI/UMaXf2LRUctaj/HGJ57UNy/t7fNCshh8wpJg'; -- amazon manager site secret token
  p_name     := 'my_instance_of_linux';

  DB.DBA.AMAZON_TERMINATE_INSTANCE (access_key, p_name, sec_token);

}
;
```

See Also

DB.DBA.AMAZON_START_INSTANCE

DB.DBA.AMAZON_RUN_INSTANCE

DB.DBA.AMAZON_STOP_INSTANCE

DB.DBA.AMAZON_DEREGISTER_IMAGE

DB.DBA.AMAZON_CREATE_IMAGE

DB.DBA.AMAZON_CREATE_VOLUME

DB.DBA.AMAZON_CREATE_SNAPSHOT

DB.DBA.AMAZON_DELETE_SNAPSHOT

DB.DBA.AMAZON_DELETE_VOLUME

DB.DBA.AMAZON_DESCRIBE_IMAGES

DB.DBA.AMAZON_DESCRIBE_INSTANCES

DB.DBA.AMAZON_IMPORT_KEY_PAIR

ApiReference--TerminateInstances

DB.DBA.AMAZON_CREATE_VOLUME

DB.DBA.AMAZON_CREATE_VOLUME — Creates an Amazon EBS volume that can be attached to any Amazon EC2 instance in the same Availability Zone.

Synopsis

DB.DBA.AMAZON_CREATE_VOLUME (in *aws_access_key* varchar ,
in *availabilityZone* varchar ,
in *secret_key* varchar ,
in *snapshotId* varchar (default '') ,
in *volume_size* integer (default 1) ,
in *http_proxy* varchar (default null)) ;

Description

Creates an Amazon EBS volume that can be attached to any Amazon EC2 instance in the same Availability Zone. Any AWS Marketplace product codes from the snapshot are propagated to the volume.

Parameters

aws_access_key
: Amazon Access Key ID.

availabilityZone
: The Availability Zone for the new volume.

secret_key
: AWS Security Token.

snapshotId
: The snapshot from which to create the new volume.

volume_size
: The size of the volume, in GiBs.

http_proxy
: Proxy server, can be null or empty.

Examples

Example 24.233. Simple Use

```
create procedure simple_test()
{
  declare access_key, sec_token, zone varchar;

  access_key := 'AKIAJI7ZL427TI5EDF5A'; -- amazon manager site access key
  sec_token  := 'CGI/UMaXf2LRUctaj/HGJ57UNy/t7fNCshh8wpJg'; -- amazon manager site secret token
  zone       := 'us-east-1a';

  DB.DBA.AMAZON_CREATE_VOLUME (access_key, zone, sec_token);

}
;
```

See Also

DB.DBA.AMAZON_START_INSTANCE

DB.DBA.AMAZON_RUN_INSTANCE

DB.DBA.AMAZON_STOP_INSTANCE

DB.DBA.AMAZON_TERMINATE_INSTANCE

DB.DBA.AMAZON_DEREGISTER_IMAGE

DB.DBA.AMAZON_CREATE_SNAPSHOT

DB.DBA.AMAZON_CREATE_IMAGE

DB.DBA.AMAZON_DELETE_SNAPSHOT

DB.DBA.AMAZON_DELETE_VOLUME

DB.DBA.AMAZON_DESCRIBE_IMAGES

DB.DBA.AMAZON_DESCRIBE_INSTANCES

DB.DBA.AMAZON_IMPORT_KEY_PAIR

ApiReference--CreateVolume

DB.DBA.AMAZON_DEREGISTER_IMAGE

DB.DBA.AMAZON_DEREGISTER_IMAGE — Deregisters the specified AMI.

Synopsis

DB.DBA.AMAZON_DEREGISTER_IMAGE (in *aws_access_key* varchar ,
in *image_id* varchar ,
in *secret_key* varchar ,
in *http_proxy* varchar (default null)) ;

Description

Deregisters the specified AMI. Once deregistered, the AMI cannot be used to launch new instances.

Parameters

aws_access_key
: Amazon Access Key ID.

image_id
: Image ID (check Amazon list of images).

secret_key
: AWS Security Token.

http_proxy
: Proxy server, can be null or empty.

Examples

Example 24.234. Simple Use

```
create procedure simple_test()
{
  declare access_key, sec_token, image_id varchar;

  access_key := 'AKIAJI7ZL427TI5EDF5A'; -- amazon manager site access key
  sec_token  := 'CGI/UMaXf2LRUctaj/HGJ57UNy/t7fNCshh8wpJg'; -- amazon manager site secret token
  image_id   := 'my_new_image';

 DB.DBA.AMAZON_DEREGISTER_IMAGE (access_key, image_id, sec_token);

}
;
```

See Also

DB.DBA.AMAZON_START_INSTANCE

DB.DBA.AMAZON_RUN_INSTANCE

DB.DBA.AMAZON_STOP_INSTANCE

DB.DBA.AMAZON_TERMINATE_INSTANCE

DB.DBA.AMAZON_CREATE_IMAGE

DB.DBA.AMAZON_CREATE_VOLUME

DB.DBA.AMAZON_CREATE_SNAPSHOT

DB.DBA.AMAZON_DELETE_SNAPSHOT

DB.DBA.AMAZON_DELETE_VOLUME

DB.DBA.AMAZON_DESCRIBE_IMAGES

DB.DBA.AMAZON_DESCRIBE_INSTANCES

DB.DBA.AMAZON_IMPORT_KEY_PAIR

ApiReference--DeregisterImage

DB.DBA.AMAZON_CREATE_IMAGE

DB.DBA.AMAZON_CREATE_IMAGE — Creates an Amazon EBS-backed AMI from an Amazon EBS-backed instance that is either running or stopped.

Synopsis

DB.DBA.AMAZON_CREATE_IMAGE (in *aws_access_key* varchar ,
in *instance_id* varchar ,
in *image_name* varchar ,
in *secret_key* varchar ,
in *http_proxy* varchar (default null)) ;

Description

Creates an Amazon EBS-backed AMI from an Amazon EBS-backed instance that is either running or stopped.

Parameters

aws_access_key
: Amazon Access Key ID.

instance_id
: Image instance ID (check Amazon list of instances).

image_name
: New image ID (check Amazon list of images).

secret_key
: AWS Security Token.

http_proxy
: Proxy server, can be null or empty.

Examples

Example 24.235. Simple Use

```
create procedure simple_test()
{
  declare access_key, sec_token, p_name, image_id varchar;

  access_key := 'AKIAJI7ZL427TI5EDF5A'; -- amazon manager site access key
  sec_token  := 'CGI/UMaXf2LRUctaj/HGJ57UNy/t7fNCshh8wpJg'; -- amazon manager site secret token
  image_id   := 'my_new_image';
  p_name     := 'my_instance_of_linux';

  DB.DBA.AMAZON_CREATE_IMAGE (access_key, p_name, image_id, sec_token);

}
;
```

See Also

DB.DBA.AMAZON_START_INSTANCE

DB.DBA.AMAZON_RUN_INSTANCE

DB.DBA.AMAZON_STOP_INSTANCE

DB.DBA.AMAZON_TERMINATE_INSTANCE

DB.DBA.AMAZON_DEREGISTER_IMAGE

DB.DBA.AMAZON_CREATE_SNAPSHOT

DB.DBA.AMAZON_CREATE_VOLUME

DB.DBA.AMAZON_DELETE_SNAPSHOT

DB.DBA.AMAZON_DELETE_VOLUME

DB.DBA.AMAZON_DESCRIBE_IMAGES

DB.DBA.AMAZON_DESCRIBE_INSTANCES

DB.DBA.AMAZON_IMPORT_KEY_PAIR

ApiReference--CreateImage

DB.DBA.AMAZON_CREATE_SNAPSHOT

DB.DBA.AMAZON_CREATE_SNAPSHOT — Creates a snapshot of an Amazon EBS volume and stores it in Amazon S3.

Synopsis

DB.DBA.AMAZON_CREATE_SNAPSHOT (in *aws_access_key* varchar ,
in *volumeId* varchar ,
in *secret_key* varchar ,
in *description* varchar (default 'new+snapshot') ,
in *http_proxy* varchar (default null)) ;

Description

Creates a snapshot of an Amazon EBS volume and stores it in Amazon S3. You can use snapshots for backups, to make copies of instance store volumes, and to save data before shutting down an instance.

Parameters

aws_access_key
: Amazon Access Key ID.

volumeId
: The ID of the Amazon EBS volume.

secret_key
: AWS Security Token.

description
: A description for the snapshot.

http_proxy
: Proxy server, can be null or empty.

Examples

Example 24.236. Simple Use

```
create procedure simple_test()
{
  declare access_key, sec_token, vol_id varchar;

  access_key := 'AKIAJI7ZL427TI5EDF5A'; -- amazon manager site access key
  sec_token  := 'CGI/UMaXf2LRUctaj/HGJ57UNy/t7fNCshh8wpJg'; -- amazon manager site secret token
  vol_id     := 'vol-1a2b3c4d';

  DB.DBA.AMAZON_CREATE_SNAPSHOT (access_key, vol_id, sec_token);

}
;
```

See Also

DB.DBA.AMAZON_START_INSTANCE

DB.DBA.AMAZON_RUN_INSTANCE

DB.DBA.AMAZON_STOP_INSTANCE

DB.DBA.AMAZON_TERMINATE_INSTANCE

DB.DBA.AMAZON_DEREGISTER_IMAGE

DB.DBA.AMAZON_CREATE_IMAGE

DB.DBA.AMAZON_CREATE_VOLUME

DB.DBA.AMAZON_DELETE_SNAPSHOT

DB.DBA.AMAZON_DELETE_VOLUME

DB.DBA.AMAZON_DESCRIBE_IMAGES

DB.DBA.AMAZON_DESCRIBE_INSTANCES

DB.DBA.AMAZON_IMPORT_KEY_PAIR

ApiReference--CreateSnapshot

DB.DBA.AMAZON_DELETE_SNAPSHOT

DB.DBA.AMAZON_DELETE_SNAPSHOT — Deletes a snapshot of an Amazon EBS volume.

Synopsis

DB.DBA.AMAZON_DELETE_SNAPSHOT (in *aws_access_key* varchar ,
in *snapshotId* varchar ,
in *secret_key* varchar ,
in *http_proxy* varchar (default null)) ;

Description

Deletes a snapshot of an Amazon EBS volume.

Parameters

aws_access_key
: Amazon Access Key ID.

snapshotId
: The ID of the Amazon EBS snapshot.

secret_key
: AWS Security Token.

http_proxy
: Proxy server, can be null or empty.

Examples

Example 24.237. Simple Use

```
create procedure simple_test()
{
  declare access_key, sec_token, snapshotId varchar;

  access_key := 'AKIAJI7ZL427TI5EDF5A'; -- amazon manager site access key
  sec_token  := 'CGI/UMaXf2LRUctaj/HGJ57UNy/t7fNCshh8wpJg'; -- amazon manager site secret token
  snapshotId := 'snap-1a2b3c4d';

  DB.DBA.AMAZON_DELETE_SNAPSHOT (access_key, snapshotId, sec_token);

}
;
```

See Also

DB.DBA.AMAZON_START_INSTANCE

DB.DBA.AMAZON_RUN_INSTANCE

DB.DBA.AMAZON_STOP_INSTANCE

DB.DBA.AMAZON_TERMINATE_INSTANCE

DB.DBA.AMAZON_DEREGISTER_IMAGE

DB.DBA.AMAZON_CREATE_IMAGE

DB.DBA.AMAZON_CREATE_VOLUME

DB.DBA.AMAZON_CREATE_SNAPSHOT

DB.DBA.AMAZON_DELETE_VOLUME

DB.DBA.AMAZON_DESCRIBE_IMAGES

DB.DBA.AMAZON_DESCRIBE_INSTANCES

DB.DBA.AMAZON_IMPORT_KEY_PAIR

ApiReference--DeleteSnapshot

DB.DBA.AMAZON_DELETE_VOLUME

DB.DBA.AMAZON_DELETE_VOLUME — Deletes an Amazon EBS volume.

Synopsis

DB.DBA.AMAZON_DELETE_VOLUME (in *aws_access_key* varchar ,
in *volumeId* varchar ,
in *secret_key* varchar ,
in *http_proxy* varchar (default null)) ;

Description

Deletes an Amazon EBS volume. The volume must be in the available state (not attached to an instance).

Parameters

aws_access_key
: Amazon Access Key ID.

volumeId
: The ID of the Amazon EBS volume.

secret_key
: AWS Security Token.

http_proxy
: Proxy server, can be null or empty.

Examples

Example 24.238. Simple Use

```
create procedure simple_test()
{
  declare access_key, sec_token, vol_id varchar;

  access_key := 'AKIAJI7ZL427TI5EDF5A'; -- amazon manager site access key
  sec_token  := 'CGI/UMaXf2LRUctaj/HGJ57UNy/t7fNCshh8wpJg'; -- amazon manager site secret token
  vol_id     := 'vol-1a2b3c4d';

  DB.DBA.AMAZON_DELETE_VOLUME (access_key, vol_id, sec_token);

}
;
```

See Also

DB.DBA.AMAZON_START_INSTANCE

DB.DBA.AMAZON_RUN_INSTANCE

DB.DBA.AMAZON_STOP_INSTANCE

DB.DBA.AMAZON_TERMINATE_INSTANCE

DB.DBA.AMAZON_DEREGISTER_IMAGE

DB.DBA.AMAZON_CREATE_IMAGE

DB.DBA.AMAZON_CREATE_VOLUME

DB.DBA.AMAZON_CREATE_SNAPSHOT

DB.DBA.AMAZON_DELETE_VOLUME

DB.DBA.AMAZON_DESCRIBE_IMAGES

DB.DBA.AMAZON_DESCRIBE_INSTANCES

DB.DBA.AMAZON_IMPORT_KEY_PAIR

ApiReference--DeleteVolume

DB.DBA.AMAZON_DESCRIBE_IMAGES

DB.DBA.AMAZON_DESCRIBE_IMAGES — Describes one or more of the images (AMIs, AKIs, and ARIs) available to you.

Synopsis

DB.DBA.AMAZON_DESCRIBE_IMAGES (in *aws_access_key* varchar ,
in *secret_key* varchar ,
in *http_proxy* varchar (default null)) ;

Description

Describes one or more of the images (AMIs, AKIs, and ARIs) available to you. Images available to you include: public images, private images that you own, and private images owned by other AWS accounts but for which you have explicit launch permissions.

Parameters

aws_access_key
: Amazon Access Key ID.

secret_key
: AWS Security Token.

http_proxy
: Proxy server, can be null or empty.

Examples

Example 24.239. Simple Use

```
create procedure simple_test()
{
  declare access_key, sec_token varchar;
  declare images, image_item any;

  access_key := 'AKIAJI7ZL427TI5EDF5A'; -- amazon manager site access key
  sec_token  := 'CGI/UMaXf2LRUctaj/HGJ57UNy/t7fNCshh8wpJg'; -- amazon manager site secret token
  images     := xtree_doc (DB.DBA.AMAZON_DESCRIBE_IMAGES (access_key, sec_token));
  image_item := xpath_eval('/DescribeImagesResponse/imagesSet/item', images, 0);

}
;
```

See Also

DB.DBA.AMAZON_START_INSTANCE

DB.DBA.AMAZON_RUN_INSTANCE

DB.DBA.AMAZON_STOP_INSTANCE

DB.DBA.AMAZON_TERMINATE_INSTANCE

DB.DBA.AMAZON_DEREGISTER_IMAGE

DB.DBA.AMAZON_CREATE_IMAGE

DB.DBA.AMAZON_CREATE_VOLUME

DB.DBA.AMAZON_CREATE_SNAPSHOT

DB.DBA.AMAZON_DELETE_SNAPSHOT

DB.DBA.AMAZON_DELETE_VOLUME

DB.DBA.AMAZON_DESCRIBE_INSTANCES

DB.DBA.AMAZON_IMPORT_KEY_PAIR

ApiReference--DescribeImages

DB.DBA.AMAZON_DESCRIBE_INSTANCES

DB.DBA.AMAZON_DESCRIBE_INSTANCES — Describes one or more of your instances.

Synopsis

DB.DBA.AMAZON_DESCRIBE_INSTANCES (in *aws_access_key* varchar ,
in *secret_key* varchar ,
in *http_proxy* varchar (default null)) ;

Description

Describes one or more of your instances.

Parameters

aws_access_key
: Amazon Access Key ID.

secret_key
: AWS Security Token.

http_proxy
: Proxy server, can be null or empty.

Examples

Example 24.240. Simple Use

```
create procedure simple_test()
{
  declare access_key, sec_token varchar;
  declare amis, instances any;

  access_key := 'AKIAJI7ZL427TI5EDF5A'; -- amazon manager site access key
  sec_token  := 'CGI/UMaXf2LRUctaj/HGJ57UNy/t7fNCshh8wpJg'; -- amazon manager site secret token
  amis       := xtree_doc (DB.DBA.AMAZON_DESCRIBE_INSTANCES (access_key, sec_token));
  instances  := xpath_eval('/DescribeInstancesResponse/reservationSet/item/instancesSet/item', amis, 0);

}
;
```

See Also

DB.DBA.AMAZON_START_INSTANCE

DB.DBA.AMAZON_RUN_INSTANCE

DB.DBA.AMAZON_STOP_INSTANCE

DB.DBA.AMAZON_TERMINATE_INSTANCE

DB.DBA.AMAZON_DEREGISTER_IMAGE

DB.DBA.AMAZON_CREATE_IMAGE

DB.DBA.AMAZON_CREATE_SNAPSHOT

DB.DBA.AMAZON_CREATE_VOLUME

DB.DBA.AMAZON_DELETE_SNAPSHOT

DB.DBA.AMAZON_DELETE_VOLUME

DB.DBA.AMAZON_DESCRIBE_IMAGES

DB.DBA.AMAZON_IMPORT_KEY_PAIR

ApiReference--DescribeInstances

DB.DBA.AMAZON_IMPORT_KEY_PAIR

DB.DBA.AMAZON_IMPORT_KEY_PAIR — Uploads public key to Amazon.

Synopsis

DB.DBA.AMAZON_IMPORT_KEY_PAIR (in *aws_access_key* varchar ,
in *key_name* varchar ,
in *public_key_material* varchar ,
in *secret_key* varchar ,
in *http_proxy* varchar (default null)) ;

Description

Uploads public key to Amazon

Parameters

aws_access_key
: Amazon Access Key ID.

key_name
: New name of key, which will be used on Amazon lists.

public_key_material
: base64 encoded public key.

secret_key
: AWS Security Token.

http_proxy
: Proxy server, can be null or empty.

Examples

Example 24.241. Simple Use

```
create procedure simple_test()
{
  declare access_key, sec_token, key_name varchar;
  declare pub_key, fingers, fingerPrint any ;

  access_key  := 'AKIAJI7ZL427TI5EDF5A'; -- amazon manager site access key
  ec_token    := 'CGI/UMaXf2LRUctaj/HGJ57UNy/t7fNCshh8wpJg'; -- amazon manager site secret token
  key_name    := 'my_key';
  pub_key     := encode_base64 (cast (xenc_pubkey_DER_export (key_name) as varchar));
  pub_key     := encode_base64(pub_key);
  fingers     := xtree_doc(DB.DBA.AMAZON_IMPORT_KEY_PAIR (access_key, key_name,   pub_key, sec_token));
  fingerPrint := xpath_eval('/ImportKeyPairResponse/keyFingerprint', fingers, 0);

  if (fingerPrint is not null)
    return 1;
  else
    return 0;
}
;
```

See Also

DB.DBA.AMAZON_START_INSTANCE

DB.DBA.AMAZON_RUN_INSTANCE

DB.DBA.AMAZON_STOP_INSTANCE

DB.DBA.AMAZON_TERMINATE_INSTANCE

DB.DBA.AMAZON_DEREGISTER_IMAGE

DB.DBA.AMAZON_CREATE_IMAGE

DB.DBA.AMAZON_CREATE_VOLUME

DB.DBA.AMAZON_CREATE_SNAPSHOT

DB.DBA.AMAZON_DELETE_SNAPSHOT

DB.DBA.AMAZON_DELETE_VOLUME

DB.DBA.AMAZON_DESCRIBE_IMAGES

DB.DBA.AMAZON_DESCRIBE_INSTANCES

ApiReference--ImportKeyPair

os_chmod

os_chmod — changes the file access mode of a file

Synopsis

os_chmod (in *path* varchar ,
in *new_mod* integer) ;

Description

`os_chmod` set the file mode bits by calling the system function chmod() with the arguments supplied.

Not all the host OSes support all the file modes. That's why the function will not throw an SQL error if the function fails.

It will return DB NULL value when the function was called successfully and the error message as a string if it failed.

In most OSes the mode is a bitmask, so one would usually use the `file_stat` to get the current mode and `bit_or` /`bit_xor` to make the new mode.

Typical bit layout (from Linux 'man 2 chmod') is :

bit 0 (mask 1) - execute/search by others
bit 1 (mask 2) - write by others
bit 2 (mask 4) - read by others
bit 3 (mask 8) - execute/search by group
bit 4 (mask 16) - write by group
bit 5 (mask 32) - read by group
bit 6 (mask 64) - execute/search by owner
bit 7 (mask 128) - write by owner
bit 8 (mask 256) - read by owner
bit 9 (mask 512) - sticky bit
bit 10 (mask 1024) - set group ID on execution
bit 12 (mask 2048) - set user ID on execution

The DirsAllowed and DirsDenied lists in Parameters section of the virtuoso configuration file (virtuoso.ini by default) are used to control disk access.

Parameters

path
: varchar relative path.

new_mod
: integer new mode for the file.

Examples

Example 24.242. Simple example

Make the virtuoso INI file readable by the others

```
SQL>select os_chmod (virtuoso_ini_path(), bit_or (cast (file_stat (virtuoso_ini_path(), 2) as integer), 4
callret
VARCHAR
_______________________________________________________________________________
```

```
NULL
```

See Also

file_delete

file_stat

os_chown

virtuoso_ini_path

bit_or

bit_xor

os_chown

os_chown — changes the owning group/user of a file

Synopsis

os_chown (in *path* varchar ,
in *username* varchar ,
in *groupname* varchar) ;

Description

This function requires dba privileges.

os_chown set the owning user/group of an OS file by calling the system function chown() with the arguments supplied.

Not all the host OSes support the concept of owner users and groups. That's why the function will not throw an SQL error if it fails.

It will return DB NULL value when the function was executed successfully and the error message as a string if it failed.

The function first looks up the user and group name supplied into the OS database to get the user and group IDs. it then calls the system function chown().

The DirsAllowed and DirsDenied lists in Parameters section of the virtuoso configuration file (virtuoso.ini by default) are used to control disk access.

Parameters

path
: varchar relative path.

username
: varchar the name of the OS user to own the file.

groupname
: varchar the name of the OS group to own the file.

Examples

Example 24.243. Simple example

Sets the ownership of the virtuoso INI file to virtuoso user and group

```
SQL>select os_chown (virtuoso_ini_path(), 'virtuoso', 'virtuoso');
callret
VARCHAR
_______________________________________________________________________________

NULL
```

See Also

file_delete

file_stat

os_chown

virtuoso_ini_path

pem_certificates_to_array

pem_certificates_to_array — converts a PEM file to an array of PEM strings

Synopsis

pem_certificates_to_array (in *pem_string* varchar) ;

Description

This gets a PEM file with (possibly) many X509 certificates among others and constructs an array containing each X509 certificate as a separate PEM string. This can serve for splitting a PEM file containing multiple certificates (for example CA file) to single certificate entries so it can be examined with get_certificate_info function. Note that the array can contain NULL elements in places where in the PEM file there are blocks other than X509 PEM certificates.

Parameters

pem_string
: text of the PEM file

Return Types

Array of PEM strings

Errors

Table 24.61. Errors signalled by pem_certificates_to_array

SQLState	Error Code	Error Text	Description
42000	CR011	Can't allocate temp space. SSL error : <error text>	
42000	CR012	Can't read certificates. SSL error : <error text>	

See Also

smime_sign

smime_verify

pldbg_stats

pldbg_stats — Makes an array of line/count information based on current coverage.

Synopsis

```
any pldbg_stats ( in name varchar ,
                  in add_line_info integer ) ;
```

Description

This function is used to make an array of line/count information based on the current coverage statistics. If the procedure `name` is given (first param), then the result will contain only coverage statistic for that procedure. if the procedure name is not supplied or supplied as NULL then the result will contain coverage data for all procedures having statistic. The `add_line_info` flag is used to add code excerpt on line info.

Parameters

name
: Optional name of procedure for producing selective information.

outdir
: Optional flag to control output existence. If this flag is set to 1 then code excerpts will be contained in the output.

Return Types

An array (or vector) is returned containing the line/count information of the selected coverage data. The format is as follows:

```
(
  ("PRODECURE_NAME" "file_name" <times executed> <total lines> ) -- procedure info
  ((<line no> <no executed> "<line excerpt>" ) .... )            -- line statistics
  (("CALLER PROCEDURE" <counts> ) .... )                         -- caller statistics
  ...
)
```

See Also

`cov_load()`

`cov_store()`

`cov_report()`

`pldbg_stats_load()`

pldbg_stats_load

pldbg_stats_load — Used to load a coverage of a single procedure record as an array.

Synopsis

pldbg_stats_load (in *info* any);

Description

This function is used to load coverage information for a single procedure record. The expected format is an array. The data expected is exactly that which can be produced by `pldbg_stats()` . Only one record will can be accepted, an array of several procedure records will not be acceptable.

Parameters

info
: An array with the following format:

Return Types

None.

See Also

`cov_load()`

`cov_store()`

`cov_report()`

`pldbg_stats()`

pop3_get

pop3_get — get messages from a POP3 server

Synopsis

```
array pop3_get ( in host varchar ,
                 in user varchar ,
                 in password varchar ,
                 in buffer_size integer ,
                 in command varchar ,
                 in exclude_uidl_list vector ) ;
```

Description

`Pop3_get` is used to retrieve and delete messages from a server running the Post Office Protocol version 3 as defined in rfc1725. In its default form it returns a vector of vectors containing messages retrieved from the POP3 server. Each vector within the vector contains a pair of VARCHAR UIDL and VARCHAR Message body, i.e. to get the message body of the second message retrieved, one would use `aref (aref (msg_vec, 1), 1)` . Total length of messages retrieved will not exceed the value of *`buffer_size`* parameter in bytes.

The optional parameter *`command`* can be used to control output or delete messages. When *`command`* is passed a VARCHAR 'uidl', `pop3_get` outputs single vector containing VARCHAR UIDLs. The *`buffer_size`* constraint is effective here. Thus, the vector will only contain UIDLs of messages whose total message text length does not exceed *`buffer_size`* bytes. These message lengths are accumulated in the order returned by the POP3 server.

Command 'delete' will cause retrieved messages to be deleted from the server after retrieval.

Parameters

host
: The host to connect with. IP address or hostname:port. There is no default for port, so to connect to the standard port for POP3, use <hostname/IP address>:110

user
: string user id in remote host.

password
: string password in remote host.

buffer_size
: integer maximum total length of message text for messages/uidls to be retrieved.

command
: Command string . Valid values are empty, 'uidl' or 'delete'

exclude_uidl_list
: A vector containing UIDLs. A message whose UIDL appears in this list will not be retrieved or deleted.

Return Types

A vector of vectors containing UIDL/Message text strings or a 'flat' vector containing UIDL strings .

Errors

Table 24.62. Errors signalled by

SQLState	Error Code	Error Text	Description
2E000	PO001	Cannot resolve host in pop3_get	
08001	PO002	Cannot connect in pop3_get	
08006	PO003	No response from remote POP3 server	
08006	PO004	Not valid user in remote POP3 server	
08006	PO005	UIDL command to remote POP3 server failed	
08006	PO006	Could not get output of UIDL from remote POP3 server.	
08006	PO007	LIST command to remote POP3 server failed.	
08006	PO008	Could not get output of LIST from remote POP3 server.	
	PO009		
08006	PO010	Failed reading output of LIST command on remote POP3 server	
08006	PO011	Could not DELE messages from remote POP3 server	
08006	PO012	Could not QUIT from remote POP3 server	
08000	PO013	Argument 6 to pop3_get must be a vector	
08006	PO014	Misc. error in connection in pop3_get	

Examples

Example 24.244. Get messages from remote POP3

This example retrieves messages from a remote POP3 server and stores them in a table.

```
create table MY_MSGS (MSG_ID INTEGER IDENTITY,
                      MSG_HOST VARCHAR,
                      MSG_UIDL VARCHAR,
                      MSG_TEXT LONG VARCHAR,
                      primary key (MSG_ID, MSG_HOST, MSG_UIDL));

create procedure
get_msgs (in pop_host varchar, in pop_uid varchar, in pop_pwd varchar)
{
  declare msg_vec any;
  declare inx integer;

  msg_vec := pop3_get (concat (pop_host, ':110'),
                               pop_uid,
                               pop_pwd,
                               10000000,
                               'delete');

  inx := 0;

  while (inx < length (msg_vec))
    {
      insert into MY_MSGS (MSG_HOST, MSG_UIDL, MSG_TEXT)
             values (pop_host,
                     aref (aref (msg_vec, inx), 0),
                     aref (aref (msg_vec, inx), 1));

      inx := inx + 1;
    }
}
```

Here is a test run. Just for the fun, let's get the message subjects, too.

```
SQL> get_msgs('pop.xs4all.nl', 'ghard', '|_337h4x0R');

SQL> select MSG_UIDL, length (MSG_TEXT), get_keyword ('Subject', aref (mime_tree (MSG_TEXT), 0)) from MY_
MSG_UIDL          callret   callret
VARCHAR NOT NULL  INTEGER   VARCHAR
_______________________________________________________________________________

1003930514.maildrop7.14798  3482      [Fwd: Linux Expo]
1003930555.maildrop7.15349  7683      [Fwd: SOAP options example]
```

```
2 Rows. -- 8 msec.
```

See Also

`mime_tree` , RFC1725

imap_get

imap_get — get messages from a IMAP4 mail server.

Synopsis

```
array imap_get ( in host varchar ,
                 in user varchar ,
                 in password varchar ,
                 in command varchar ,
                 in folder_id varchar ,
                 in list vector ,
                 in use_ssl integer ) ;
```

Description

`imap_get` is used used to work with mail server running the IMAP4 version 4rev1 as defined in rfc3501.

Parameters

host
: The host to connect with. IP address or hostname:port. There is no default for port, so to connect to the standard port for IMAP, use <hostname/IP address>:143

user
: string user id in remote host.

password
: string password in remote host.

command
: Command string . Valid values are ' list ', 'delete ', 'create ', 'select ', 'expunge ', 'rename ', 'fetch ', ' message_delete ', ' message_copy '.

folder_id
: IMAP4 mail box ID (e.g. 'INBOX', 'Draft' and etc) to work with.

list
: A list of items to work with, depends on the command.

use_ssl
: Use SSL connection, 1 - use, 0 - do not use.

Return Types

Depends on the command value:

- *list* : - list of mail boxes in selected mail box (needs folder_id, if folder_id = '' work with root of mail account). list - empty vector.

  Returns: A vector of strings containing name attributes, hierarchy delimiter, name of mailbox or empty vector.

  See more specification details .

  Example

```
SQL> select imap_get ('imap.gmail.com:993','ods.check','openlink','list','INBOX', vector (), 1);
```

```
imap_get
VARCHAR
_______________________________________________________________________________

vector(0x00491f08)

1 Rows. -- 2090 msec.

SQL> select imap_get ('imap.gmail.com:993','ods.check','openlink','list','INBOX', vector (), 1)[0]
aref
VARCHAR
_______________________________________________________________________________

(\HasNoChildren) "/" "INBOX"

1 Rows. -- 1997 msec.
```

- *delete* : deletes the selected mail box (needs folder_id). list - empty vector.

  Returns: Empty vector.

  See more specification details .

  Example

```
SQL> select imap_get ('imap.gmail.com:993','ods.check','openlink','delete','SomeBox', vector (), 1
imap_get
VARCHAR
_______________________________________________________________________________

vector()

1 Rows. -- 2090 msec.
```

- *create* : creates a new mail box in the selected mail box (needs folder_id). list - empty vector.

  Returns: Empty vector.

  See more specification details .

  Example

```
SQL> select imap_get ('imap.gmail.com:993','ods.check','openlink','create','INBOX/SomeBox', vector
imap_get
VARCHAR
_______________________________________________________________________________

vector()

1 Rows. -- 2090 msec.
```

- *select* : lists all messages in the selected mail box (needs folder_id). list - empty vector.

  This command gets the header fields of messages:

```
UID FLAGS INTERNALDATE RFC822.SIZE BODY.PEEK[HEADER.FIELDS (DATE FROM SUBJECT TO CC MESSAGE-ID REF
```

  Returns: A vector of vectors containing: UID, header fields and data separated by ' ', Data of BODY.PEEK information (the last requested parameter - BODY.PEEK[HEADER.FIELDS (DATE FROM SUBJECT TO CC MESSAGE-ID REFERENCES CONTENT-TYPE CONTENT-DESCRIPTION IN-REPLY-TO REPLY-TO LINES LIST-POST X-LABEL)]).

  Example

```
SQL> select imap_get ('imap.gmail.com:993','ods.check','openlink','select', 'INBOX', vector (), 1)
imap_get
VARCHAR
_______________________________________________________________________________

vector(0x004435a0)
```

```
1 Rows. -- 2293 msec.
SQL> select imap_get ('imap.gmail.com:993','ods.check','openlink','select', 'INBOX', vector (), 1)
aref
VARCHAR
_______________________________________________________________________________

vector(139,0x00443d50,0x00494758)

1 Rows. -- 2293 msec.
SQL> select imap_get ('imap.gmail.com:993','ods.check','openlink','select', 'INBOX', vector (), 1)
aref
VARCHAR
_______________________________________________________________________________

RFC822.SIZE 89126 INTERNALDATE "09-Nov-2011 22:47:59 +0000" FLAGS (NonJunk \Seen) BODY[HEADER.FIEL

1 Rows. -- 2230 msec.
SQL> select imap_get ('imap.gmail.com:993','ods.check','openlink','select', 'INBOX', vector (), 1)
aref
VARCHAR
_______________________________________________________________________________

Message-ID: <4EBB0319.1050200@openlinksw.com>
Date: Wed, 09 Nov 2011 23:47:53 +0100
From: demo <demo@openlinksw.com>
To: "ods.check@gmail.com" <ods.check@gmail.com>
Subject: test from OL account with attached image
Content-Type: multipart/signed; protocol="application/pkcs7-signature"; micalg=sha1; boundary="---

1 Rows. -- 2200 msec.
```

- *expunge* : expunges (deletes messages marked for deletion) the selected mail box (needs folder_id). list - empty vector.

  Returns: An empty vector.

  Example

```
SQL> select imap_get ('imap.gmail.com:993','ods.check','openlink','expunge', 'INBOX', vector (), 1
imap_get
VARCHAR
_______________________________________________________________________________

vector()

1 Rows. -- 3541 msec.
```

- *rename* : renames selected mail box. folder_id is not used, folder_id = ''. list - vector of 2 names: old and new.

  Returns: Empty vector.

  See more specification details .

  Example

```
SQL> select imap_get ('imap.gmail.com:993','ods.check','openlink','rename', '', vector ('Travel',
imap_get
VARCHAR
_______________________________________________________________________________

vector()

1 Rows. -- 2013 msec.
```

- *fetch* : fetches messages from the selected mail box (needs folder_id). folder_id is id of selected mail box, list - vector of UIDs of messages.

  Returns: A vector of vectors containing: UID and data of BODY.PEEK[] information (full message data).

  Example Get the list and UIDs of messages:

```
SQL> select imap_get ('imap.gmail.com:993','ods.check','openlink','select', 'INBOX', vector (), 1)
aref
VARCHAR
_______________________________________________________________________________

vector(139,0x00443d50,0x00494758)

1 Rows. -- 2215 msec.
```

Then get full data of message by its mail box id and UID:

```
SQL> select imap_get ('imap.gmail.com:993','ods.check','openlink','fetch', 'INBOX', vector (139),
imap_get
VARCHAR
_______________________________________________________________________________

vector(0x00442410)

1 Rows. -- 3807 msec.

SQL> select imap_get ('imap.gmail.com:993','ods.check','openlink','fetch', 'INBOX', vector (139),
aref
VARCHAR
_______________________________________________________________________________

Warning 01004: [OpenLink][Virtuoso ODBC Driver]CL077: Data truncated in column 1 of the result-set
at line 1 of Top-Level: select imap_get ('imap.gmail.com:993','ods.check','openlink','fetch', 'INB
Delivered-To: ods.check@gmail.com
Received: by 43.18.75.3 with SMTP id um9cs133471icb; Wed, 9 Nov 2011 14:48:00 -0800 (PST)
Received: by 101.43.13.258 with SMTP id d19mr8062565vdh.48.1320878879524; Wed, 09 Nov 2011 14:47:5
Return-Path: <demo@openlinksw.com>
Received: from mail.openlinksw.com (mail.openlinksw.com. [25.138.12.16])by mx.google.com with ESMT
Received-SPF: pass (google.com: best guess record for domain of demo@openlinksw.com designates 81.
Authentication-Results: mx.google.com; spf=pass (google.com: best guess record for domain of demo@
Received: from example.com ([152.114.12.11] helo=[152.114.12.146]) by mail.openlinksw.com with esm
Message-ID: <4EBB0319.1050200@openlinksw.com>
Date: Wed, 09 Nov 2011 23:47:53 +0100
From: demo <demo@openlinksw.com>
User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:7.0.1) Gecko/20110929 Thunderbird/7.0.1
MIME-Version: 1.0
To: "ods.check@gmail.com" <ods.check@gmail.com>
Subject: test from OL account with attached image
Content-Type: multipart/signed; protocol="application/pkcs7-signature"; micalg=sha1; boundary="---

This is a cryptographically signed message in MIME format.

--------------ms000205010302050500060406
Content-Type: multipart/mixed; boundary="------------080604090408010101040005"

This is a multi-part message in MIME format.
--------------080604090408010101040005
Content-Type: multipart/alternative; boundary="------------060001020305030405060601"

--------------060001020305030405060601
Content-Type: text/plain; charset=ISO-8859-1; format=flowed
Content-Transfer-Encoding: quoted-printable

*simple test*

--------------060001020305030405060601
Content-Type: text/html; charset=ISO-8859-1
Content-Transfer-Encoding: quoted-printable

<html>
  <head>
    <meta http-equiv=3D"content-type" content=3D"text/html; charset=3DISO=-8859-1">
  </head>
  <body bgcolor=3D"#FFFFFF" text=3D"#000000">
    <b>simple test</b><br>
  </body>
</html>

--------------060001020305030405060601--
```

```
--------------080604090408010101040005
Content-Type: image/png; name="a1.png"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="a1.png"

iVBORw0KGgoAAAANSUhEUgAAAyAAAAFNCAIAAAB355ySAAAACXBIWXMAAAsTAAALEwEAmpwY
AAAgAElEQVR4nOydeYBOVR/Hf+ecuz7nfmc3/mec3/3d8+COOcAgBACAADg
9Zbe9nqVSqUQIesw+/Isdz/L+8edmYaeEaUMnk9o5j7nufec3/mec3/3d8+COOcAgBACAADg
nGOM80vjn/ywOa84lpUWCKhyWdwK6cqg7i2aZqf5CYQQAGB79KPvNv24sfCMgR1bN84CIRBC
QgjLoW9/uX5HaWzMwE659SIAAmMMAYAJmkXVkkSJEiRYoUhwfId5UAwPe0dvOELMdjHg+F1JoHq30s
3y3jjGOCq8+AEKp211KkSJEiRYoUhwHwnQpnVfgAAAAASUVORK5CYII=

1 Rows. -- 3338 msec.
```

- *message_delete* : deletes messages from mail box (needs folder_id). folder_id is id of selected mail box, list - vector of UIDs of messages.

  Returns: empty vector.

  Example

```
SQL> select imap_get ('imap.gmail.com:993','ods.check','openlink','message_delete', 'INBOX', vecto
imap_get
VARCHAR
_______________________________________________________________________________

vector()

1 Rows. -- 2262 msec.
```

- *message_copy* : copies messages from select mail box to another. folder_id is id of selected mail box, list - vector of: first vector item is an id of target mail box, and UIDs of messages.

  Returns: empty vector.

  Example

```
SQL> select imap_get ('imap.gmail.com:993','ods.check','openlink','message_copy', 'INBOX', vector(
imap_get
VARCHAR
_______________________________________________________________________________

vector()

1 Rows. -- 2262 msec.
```

Errors

Table 24.63. Errors signalled by all commands

SQLState	Error Code	Error Text	Description
	IM001	Cannot resolve host in imap_get	
	IM002	Cannot connect in imap_get	
	IM003		

SQLState	Error Code	Error Text	Description
		No response from remote IMAP server	
	IM004	Could not login to remote IMAP server. Please check user or password parameters.	
	IM005	CAPABILITY command to remote IMAP server failed	
	IM010	Failed reading output of LIST command on remote IMAP server	
	IM007	Cannot resolve host in imap_get	
	IM008	Cannot resolve host in imap_get	

Table 24.64. Errors signalled by command "delete":

SQLState	Error Code	Error Text	Description
	IM011	There must be folder name to delete (5th argument)	
	IM012	DELETE command to remote IMAP server failed	

Table 24.65. Errors signalled by command "create":

SQLState	Error Code	Error Text	Description
	IM015	There must be folder name to create (5th argument)	
	IM016	CREATE command to remote IMAP server failed	

Table 24.66. Errors signalled by command "select":

SQLState	Error Code	Error Text	Description
	IM019	SELECT command to remote IMAP server failed	
	IM027	Failed reading output of FETCH command on remote IMAP server	

Table 24.67. Errors signalled by command "expunge":

SQLState	Error Code	Error Text	Description
	IM020	EXPUNGE command to remote IMAP server failed	

Table 24.68. Errors signalled by command "rename":

SQLState	Error Code	Error Text	Description
	IM028	There must be 2 string items in vector of argument 6 (old folder name to rename and a new name)	
	IM029	There must be 2 string items in vector of argument 6 (old folder name to rename and a new name)	
	IM030	command to remote IMAP server failed	

Table 24.69. Errors signalled by command "fetch":

SQLState	Error Code	Error Text	Description
	IM032	SELECT command to remote IMAP server failed	

SQLState	Error Code	Error Text	Description
	IM033	No messages in list	
	IM034	There must be integer items in vector of argument 6	

Table 24.70. Errors signalled by command "delete":

SQLState	Error Code	Error Text	Description
	IM035	Error in IMAP command UID STORE	

Table 24.71. Errors signalled by incorrect command value:

SQLState	Error Code	Error Text	Description
	IM042	No such command (5th parameter) in protocol	

See Also

`pop3_get`

RFC3501

position

position — returns the index of an element within an array or string

Synopsis

position (*element* any ,
array any ,
start_index integer (optional, default 1) ,
step integer (optional, default 1)) ;

Description

position returns the one-based index for the first occurrence of element within an array or string , beginning from offset start_index and iterating with step. If the second argument is a string , the first argument must be a string with a single character. If the second argument is an array of any , then depending of type of it's elements same type is expected as the first argument.

Parameters

element
: any , depends of type of elements in second argument .

array
: array , vector or string .

start_index
: integer index beginning from 1; from where to start search.

step
: integer the step between elements to be tested; default is 1 (every element)

Return Values

An integer index in array starting from 1; if not found returns zero.

Errors

Table 24.72. Errors signalled by position

SQLState	Error Code	Error Text	Description
22003	SR061	position: expects a vector whose length is divisible by [step], not of length [actual len].	
22023	SR000	position expects an array or vector, not an arg of type %d.	

Examples

Example 24.245. Simple examples

```
SQL> select position ('b', 'Abacus');
callret
VARCHAR
_______________________________________________________________________________

2

1 Rows. -- 3 msec.
SQL> select position (2, vector ('Primero', null, 2, 3.333));
callret
VARCHAR
_______________________________________________________________________________
```

```
3

1 Rows. -- 4 msec.

SQL> select position ('a', 'babaab', 1, 2);
                                ^--- matches 5-th as other 'a' are skipped as step is 2 chars
callret
INTEGER
_______________________________________________________________________________

5

1 Rows. -- 4 msec.
```

See Also

vector aset aref ascii

power

power — return value of expression raised to specified power.

Synopsis

power (in *x* double precision ,
in *y* double precision) ;

Description

`power` raises x to the yth power and returns the value as a IEEE 64-bit float.

Parameters

x
: double precision

y
: double precision

Return Values

`power` returns a IEEE 64-bit float.

Examples

Example 24.246. Stored Procedure Example

Calculate squares of integers between 2 numbers.

```
SQL> create procedure
calc_pow2s (in _from integer, in _to integer)
{
  declare i integer;
  declare result double precision;

  result_names (i, result);

  i := _from;
  while (i < _to)
   {
     result (i, power (i, 2));
     i := i + 1;
   }
}
;
Done. -- 5 msec.
SQL> calc_pow2s (1, 10);
i                 result
INTEGER NOT NULL  DOUBLE PRECISION NOT NULL
_____________________________________________________________________________

1                                  1.000000
2                                  4.000000
3                                  9.000000
4                                 16.000000
5                                 25.000000
6                                 36.000000
7                                 49.000000
8                                 64.000000
9                                 81.000000

9 Rows. -- 7 msec.
```

See Also

`log`

prof_enable

prof_enable — Control virtuoso profiling

Synopsis

prof_enable (in *flag* integer) ;

Description

`prof_enable` is used to enable or disable profiling of execution times, SQL statements and web requests. Passing flag value 1, enables profiling, causing times of statement executions, which begin and end while profiling is on, to be accumulated.

When called with a flag of 0, the accumulation is stopped and results gathered so far are written into file named virtprof.out in the server's working directory. For a description of the file, see section about SQL Execution Profiling in Performance tuning part of Virtuoso documentation.

Parameters

flag
: An INTEGER . Valid values are 1 or 0.

Return Types

None.

Errors

No errors are signalled by `prof_enable`

Examples

Example 24.247. Simple example

Enable profiling.

```
SQL> prof_enable(1);
```

See Also

SQL Execution profiling.

prof_sample()

prof_sample

prof_sample — Adds a profiling sample to a profile being accumulated.

Synopsis

prof_sample (in *desc* varchar ,
in *time_spent* integer ,
in *flag* integer) ;

Description

`prof_sample` is used to adds a profiling sample to a profile being accumulated.

The first argument is the name of the sampled section, the times called and cumulative times will be totaled under this heading. The second argument is the time in milliseconds. The third argument is a flag indicating whether the section was successfully executed. 0 indicates success, 1 indicates execute of the statement, 2 indicates fetch on a statement's resultset, 4 indicates error. For more description of profiling capabilities see the section about SQL Execution Profiling in Performance tuning part of Virtuoso documentation.

Parameters

desc
: A VARCHAR . Name of the sampled section.

time_spent
: An INTEGER . Time in milliseconds.

flag
: An INTEGER . flag indicating whether the section was successfully executed. 0 - success, 1 - execute of statement, 2 - fetch on a statement's resultset, 4 - error.

Return Types

None.

Example 24.248. Example

```
create procedure do_prof_sample()
{
  declare stime integer;
  for(declare i integer;i < 5;i := i + 1){
    stime := msec_time();
    for(select * from Demo.demo.Customers) do sprintf('1');
    for(select * from Demo.demo.Employees) do sprintf('1');
    for(select * from Demo.demo.Order_Details) do sprintf('1');
    prof_sample('3 selects execute',msec_time() - stime,1);

  };

};

prof_enable(1);
select do_prof_sample();
prof_enable(0);
```

This will produce virtprof.out file of the sort:

```
Query Profile (msec)
Real 168, client wait 313, avg conc 1.863095 n_execs 6 avg exec  52
```

```
100 % under 1 s
0 % under 2 s
0 % under 5 s
0 % under 10 s
0 % under 30 s

2 stmts compiled 1 msec, 0 % prepared reused.

 %  total n-times n-errors
50 % 157      1        0     select do_prof_sample
49 % 156      5        0     3 selects execute
```

See Also

SQL Execution profiling.

prof_enable()

quarter

quarter — get number of quarter of year from a datetime

Synopsis

quarter (in *dt* datetime);

Description

quarter takes a datetime and returns an integer containing a number representing the quarter of year of the datetime.

Parameters

dt
: A datetime .

Return Values

An INTEGER containing number of the quarter of year.

Examples

Example 24.249. Simple example

Get current quarter of year.

```
SQL> select quarter (now ());
callret
INTEGER
_______________________________________________________________________________

4
```

See Also

dayname , monthname , dayofmonth , dayofweek , dayofyear , week , month , year , hour , minute , second , timezone

quote_dotted

quote_dotted — Returns an quoted identifier.

Synopsis

```
varchar quote_dotted ( in dsn varchar ,
                       in identifier varchar ) ;
```

Description

The `quote_dotted()` function will return the identifier (table name or column name) appropriately quoted for the remote data source. This function will obtain the appropriate quote characters from the remote data source. This function can be used in conjunction with rexecute function.

Parameters

dsn
: The remote DSN name.

identifier
: The string containing the identifier. The identifier can be a one, two or three part name, separated with the dot, '.', character.

Return Types

A string will be returned containing the quoted identifier.

Errors

Table 24.73. Errors signalled by

SQLState	Error Code	Error Text	Description
22023	VD011	Supplied DSN [name] is not valid	

Examples

Example 24.250. Using the quote_dotted() function

This simple example shows an identifier from a remote Oracle data source being correctly quoted using the quote_dotted() function from ISQL.

```
SQL> select quote_dotted ('oracle', 'DEMO.EMP');
callret
VARCHAR
_______________________________________________________________________________

"DEMO"."EMP"

1 Rows. -- 2 msec.
```

See Also

`rexecute()`

randomize

randomize — initializes the random number generator

Synopsis

randomize (in *seed* integer) ;

Description

The rnd function returns a random number between zero and n - 1, inclusive.

randomize initializes the random number generator.

The random number generator is initialized after the clock at first usage, so the produced sequences will be different each time unless specifically initialized.

rclose

rclose — Closes cursor created by rexecute()

Synopsis

rclose (in *cursor_handle* long);

Description

Closes the cursor opened to a remote DSN by the rexecute() function.

Parameters

cursor_handle
: The cursor handle used.

regexp_match

regexp_match — returns a substring matching the regular expression to the supplied string

Synopsis

```
varchar regexp_match ( in pattern any ,
                       inout str any ,
                       in change_the_str integer ) ;
```

Description

The regexp_match function returns a copy of substring of string *str* which matches the regular expression pattern.

Previous behavior of this function would cut the first characters of *str* until the end of regular expression matched substring. In this way *str* could be passed to this function again to find the next occurrence of substring that matches the regular expression. By default this behavior is not adopted by this function, but can be enabled for pre 3.2 compatibility by supplying the optional third parameter.

If either the *pattern* or *str* parameter contain wide characters this function will operate in wide mode, first converting any narrow characters to wide and returning a wide character response. Otherwise this function operates in narrow mode by default.

Parameters

pattern
: The regular expression to match.

str
: The string to be searched. In compatibility mode (change_the_str =1) this string in modified, removing the matched substring.

change_the_str
: This new parameter allows this function to operate in pre 3.2 compatibility mode which modified the original string. By default this parameter is set to "0" so that the original string is never altered.

Returns

This function returns the a substring matching the regular expression.

Examples

Example 24.251. Examples

```
CREATE PROCEDURE all_tokens(IN pattern VARCHAR, IN str VARCHAR)
{
        DECLARE wrd VARCHAR;
        DECLARE ans VARCHAR;
        DECLARE str_i VARCHAR;

        ans:='';
        str_i := str;
        wrd := regexp_match(pattern,str_i);
        WHILE (wrd IS NOT NULL)
        {
                ans := concat(ans,',',wrd);
                wrd := regexp_match(pattern, str_i, 1);
        };
        RETURN ans;
};
```

Compatibility:
The regular expressions supported here follow version 7.9 of the Perl Compatible Regular Expression (PCRE) syntax.

1. Wikipedia
2. PCRE.org

See Also

`regexp_parse()`

`regexp_like()`

`regexp_instr()`

`regexp_replace()`

`regexp_substr()`

regexp_parse

regexp_parse — returns substrings that match the regular expression in supplied string after an offset

Synopsis

```
index_vector regexp_parse ( in pattern string ,
                            in target_string string ,
                            in offset integer ) ;
```

Description

It applies regular expression to target_str with offset. This function returns a vector containing 2 elements for first match of the pattern and if there a sub expressions : 2 elements for each of subexpression match. The first (even numbered) element of each pair is the start index and the second (odd numbered) is the end index of the match. The regexp_parse function is more efficient than regexp_match and regexp_substr, because it doesn't allocate strings.

```
SQL> select regexp_parse('(2[34]).*(2[35])','22232225222323', 0);
callret
VARCHAR
_______________________________________________________________________________

lvector(2,14,2,4,12,14)

1 Rows. -- 10 msec.
```

Where: 2-14 is a range matched by whole expression, 2-4 is a range where '(2[34])' is matched , and 12-14 is a range where '(2[35])' subexpression matched.

Example 24.252. Examples

```
CREATE PROCEDURE all_tokens2 (IN pattern VARCHAR,IN str VARCHAR, IN offs INTEGER)
{

        DECLARE vec ANY;
        DECLARE i INTEGER;
        DECLARE out_str VARCHAR;

        vec:=regexp_parse(pattern,str,offs);
        IF ((vec IS NOT NULL) AND (length(vec)>1))
        {
                out_str:='';
                i:=0;
                WHILE ( (length(vec)/2) > i )
                {
                        out_str:=concat(out_str,'/',
                                subseq(str,aref(vec,(i)*2),
                                aref(vec,(i)*2+1)));
                        i:=i+1;
                };
                RETURN concat(out_str,test_parsing(pattern,str,aref(vec,1)+1));
        }
        return NULL;
};
```

Compatibility:
The regular expressions supported here follow version 7.9 of the Perl Compatible Regular Expression (PCRE) syntax.

1. Wikipedia
2. PCRE.org

See Also

`regexp_match()`

`regexp_like()`

`regexp_instr()`

`regexp_replace()`

`regexp_substr()`

regexp_substr

regexp_substr — returns a single captured substring from matched substring

Synopsis

```
(varchar|nvarchar) regexp_substr ( in pattern (varchar|nvarchar) ,
                                   in str (varchar|nvarchar) ,
                                   in offset integer ) ;
```

Parameters

pattern
: the regexp pattern string

str
: the data into which 1 (the first) or 0 matching substrings will be searched for

offset
: the number of regexp subexpression who's match to return : 0 for the whole matching substring, 1 for the substring matching the first regexp subexpression and so on

Description

This function will return the whole string value of the first substring in "str" that matches the regexp in "pattern" or a sub part of the first match. The regexp syntax allows subexpressions to be marked in the regular expression (using the braces syntax). An example of such type of expression will be: '(2[34]).*(2[35])' which means a regular expression having two subexpressions: '2[34]' and '2[35]'.

Examples

Example 24.253. Simple example

Let's apply the above regexp to the following source string: 22232225222323

```
SQL> select regexp_substr('(2[34]).*(2[35])','22242226222527', 0);
callret
VARCHAR
_______________________________________________________________________________

2422262225

1 Rows. -- 0 msec.
```

This returns the whole matched string from the expression.

```
SQL> select regexp_substr('(2[34]).*(2[35])','22242226222527', 1);
callret
VARCHAR
_______________________________________________________________________________

24

1 Rows. -- 0 msec.
```

This returns what has been matched for the first ('2[34]') regexp subexpression out of the whole matched substring (see above) : basically denoted as \1 in perl

```
SQL> select regexp_substr('(2[34]).*(2[35])','22242226222527', 2);
```

```
callret
VARCHAR
_______________________________________________________________________________

25

1 Rows. -- 10 msec.
```

This returns what has been matched for the second ('2[35]') regexp subexpression out of the whole matched substring. This is \2 in perl. Note that the text '22' (that corresponds to '.*' part of the regexp) is not returned by the above call because it is not contained in the second pair of braces.

Compatibility:

The regular expressions supported here follow version 7.9 of the Perl Compatible Regular Expression (PCRE) syntax.

1. Wikipedia
2. PCRE.org

See Also

`regexp_match()`

`regexp_like()`

`regexp_parse()`

`regexp_replace()`

`regexp_instr()`

DB.DBA.RDF_AUDIT_METADATA

DB.DBA.RDF_AUDIT_METADATA — Fix RDF metadata corruption.

Synopsis

DB.DBA.RDF_AUDIT_METADATA (in *audit_mode* integer ,
in *str_value* varchar ,
in *graph_iri* varchar) ;

Description

This function can detect and automatically fix most popular sorts of metadata corruption.

Parameters

audit_mode
: (0|1|2|3 default 0) - 0 = read-only audit operation; 1 = audit and fix; 2 = audit, fix and garbage collection for all incomplete declarations; 3 = the routine erases data it can't fix and it makes up to 10 iterations trying to erase all invalid/incomplete metadata. The bigger the number, the more errors can be fixed, at the price of more metadata changed if needed for the fix.

str_value
: Default '' is to stop if the metadata graph contains marks that some storages are being edited.

graph_iri
: Default is system metadata graph.

Examples

Example 24.254. Simple example

Automatically fix the corrupted metadata:

```
SQL> DB.DBA.RDF_AUDIT_METADATA(1, '*');
```

See Also

`DB.DBA.RDF_BACKUP_METADATA`

`DB.DBA.RDF_RESTORE_METADATA`

DB.DBA.RDF_GRAPH_SECURITY_AUDIT

DB.DBA.RDF_GRAPH_SECURITY_AUDIT — Fix graph-level security errors.

Synopsis

DB.DBA.RDF_GRAPH_SECURITY_AUDIT (in *recovery* integer) ;

Description

This function can detect and automatically fix many sorts of errors in the configuration of the graph-level security.

Parameters

recovery
: (0|1 default 0) - 0 = read-only diagnostics ; 1 = diagnostics +recovery, and makes a result-set of messages (such as found errors). The result set consists of 6 columns:

Examples

Example 24.255. Simple example

Automatically fix the corrupted metadata:

```
SQL> DB.DBA.RDF_GRAPH_SECURITY_AUDIT(1);
SEVERITY  GRAPH_IID     GRAPH_IRI  USER_ID     USER_NAME  MESSAGE
VARCHAR  VARCHAR       VARCHAR  INTEGER     VARCHAR  VARCHAR
_______________________________________________________________________________

         NULL          NULL     NULL        NULL     Inspecting caches of IRI_IDs of IRIs mentioned in se
         NULL          NULL     NULL        NULL     Inspecting completeness of IRI cache for graph group
         NULL          NULL     NULL        NULL     Inspecting completeness of IRI cache for graph group
         NULL          NULL     NULL        NULL     Check for mismatches between graph group IRIs and gr
         NULL          NULL     NULL        NULL     Inspecting caching of list of private graphs...
         NULL          NULL     NULL        NULL     Inspecting permissions of users...
         NULL          NULL     NULL        NULL     No errors found in RDF security

7 Rows. -- 16 msec.
```

See Also

DB.DBA.RDF_AUDIT_METADATA

DB.DBA.RDF_DEFAULT_USER_PERMS_SET

DB.DBA.RDF_DEFAULT_USER_PERMS_DEL

DB.DBA.RDF_GRAPH_USER_PERMS_SET

DB.DBA.RDF_GRAPH_USER_PERMS_DEL

DB.DBA.RDF_ALL_USER_PERMS_DEL

DB.DBA.RDF_GRAPH_GROUP_CREATE

DB.DBA.RDF_GRAPH_GROUP_INS

DB.DBA.RDF_DEFAULT_USER_PERMS_SET

DB.DBA.RDF_DEFAULT_USER_PERMS_SET — Sets default permissions of user.

Synopsis

DB.DBA.RDF_DEFAULT_USER_PERMS_SET (in *uname* varchar ,
in *perms* integer ,
in *set_private* integer (default 0)) ;

Description

This function sets default permissions of user named uname to the value specified by perms.

Parameters

uname
: User name.

perms
: If perms is null, then the procedure acts as DB.DBA.RDF_DEFAULT_USER_PERMS_DEL (uname, set_private) ;

set_private
: (0|1 default 0). If set_private is true then the permission is set as default for "private" graphs otherwise it is set for "world" graphs.

Examples

Example 24.256. Example

```
-- Set 'demo' user/role to have no access to private graphs:
DB.DBA.RDF_DEFAULT_USER_PERMS_SET ('demo', 0, 1);
-- Set 'demo' user/role ability to read, update, sponge other graphs, assuming bit_or (1,2,4) = 7 :
DB.DBA.RDF_DEFAULT_USER_PERMS_SET ('demo', 7, 0);
```

See also Example for Blogs and Resource Sharing

See Also

DB.DBA.RDF_AUDIT_METADATA

DB.DBA.RDF_GRAPH_SECURITY_AUDIT

DB.DBA.RDF_DEFAULT_USER_PERMS_DEL

DB.DBA.RDF_GRAPH_USER_PERMS_SET

DB.DBA.RDF_GRAPH_USER_PERMS_DEL

DB.DBA.RDF_ALL_USER_PERMS_DEL

DB.DBA.RDF_GRAPH_GROUP_CREATE

DB.DBA.RDF_GRAPH_GROUP_INS

DB.DBA.RDF_DEFAULT_USER_PERMS_DEL

DB.DBA.RDF_DEFAULT_USER_PERMS_DEL — Removes the setting of default permissions of user.

Synopsis

DB.DBA.RDF_DEFAULT_USER_PERMS_DEL (in *uname* varchar ,
in *set_private* integer (default 0)) ;

Description

This function removes the setting of default permissions of user named uname. As the result, the default permission of "nobody" is used for the specified user, the future changes in permissions of "nobody" will change the access control for the user.

If set_private is true then the permission in question is the default permission for "private" graphs otherwise it is for "world" graphs.

The function works for disabled or DAV users as well as for active SQL users, so a user can be banned first and then restricted in RDF access rights or visa versa. However the function will signal an error if it is called for "nobody" or "dba" users. Default permissions for these two users can be changed but can not deleted.

Parameters

uname
: User name.

set_private
: (0|1 default 0). If set_private is true then the permission in question is the default permission for "private" graphs otherwise it is for "world" graphs.

Examples

Example 24.257. Example

See Example for Blogs and Resource Sharing

See Also

`DB.DBA.RDF_AUDIT_METADATA`

`DB.DBA.RDF_GRAPH_SECURITY_AUDIT`

`DB.DBA.RDF_DEFAULT_USER_PERMS_SET`

`DB.DBA.RDF_GRAPH_USER_PERMS_SET`

`DB.DBA.RDF_GRAPH_USER_PERMS_DEL`

`DB.DBA.RDF_ALL_USER_PERMS_DEL`

`DB.DBA.RDF_GRAPH_GROUP_CREATE`

`DB.DBA.RDF_GRAPH_GROUP_INS`

DB.DBA.RDF_BACKUP_METADATA

DB.DBA.RDF_BACKUP_METADATA — Makes a backup copy of RDF metadata (i.e., descriptions of Linked Data Views and the like).

Synopsis

DB.DBA.RDF_BACKUP_METADATA (in *save_to_file* integer := 0 ,
in *backup_name* varchar := null) ;

Description

This saves all RDF metadata to file or to some graph in RDF storage, depending on first argument (nonzero for file, zero for graph). If the function gets nonzero as first argument then it creates two files, called, say, 'example.ttl', and 'example-DEBUG.ttl'. It's enough to save only the former file in order to restore metadata later, but the latter contains important additional debug information so both files should be attached to any bug report related to RDF metadata.

There are no functions to make copies of backups. File backup consists of plain TURTLE RDF files only. Backup graph is just a regular RDF graph in the default storage so it can be serialized into a file or copied into other graph. However it is important to remember that the system graph, whose name is returned by JSO_SYS_GRAPH() stored procedure, should not be edited directly. The content of the system graph is not necessarily equal to the content of a backup graph and it should be touched only by API function calls.

Parameters

save_to_file
: Value of 1 to backup to a files, 0 to backup into graph.

backup_name
: Name of destination graph or file. Graph should be empty before backup, but file with the specified name may exist before backup and will be silently overwritten. When the destination is file, the '.ttl' extension is concatenated to the backup_name. If the argument is null then the backup name will contain date and time of backup creation, like '2007-12-31T23:59.59'.

Return Types

The function returns the actual name of the created backup as a string.

See Also

`DB.DBA.RDF_AUDIT_METADATA()`

`DB.DBA.RDF_RESTORE_METADATA()`

DB.DBA.RDF_VOID_STORE

DB.DBA.RDF_VOID_STORE — Collects statistics for a given graph and saves it.

Synopsis

`DB.DBA.RDF_VOID_STORE` `(` in *graph* varchar ,
in *to_graph_name* varchar `)` ;

Description

Collects statistics for the first given graph and saves the result into the second graph. For correct results, the first graph should be loaded in the database before executing the function.

Parameters

graph
: the graph iri string to take statistics from

to_graph_name
: the graph iri string to write the void data.

Examples

Example 24.258. Simple example

```
SQL>DB.DBA.RDF_VOID_STORE('http://www.openlinksw.com/blog/~kidehen','http://example.com');
Done. -- 201 msec.

SQL>sparql
select *
from <http://example.com>
where {?s ?p ?o}
limit 30;

s                                                                                          p                 o
VARCHAR                                                                                    VARCHAR           VARCHAR
_________________________________________________________________________________________

http://www.openlinksw.com/blog/~kidehen#Dataset                                            http://www.w3.org/1999/
http://www.openlinksw.com/blog/~kidehen#Dataset                                            http://www.w3.org/2000/
http://www.openlinksw.com/blog/~kidehen#Dataset                                            http://rdfs.org/ns/void
http://www.openlinksw.com/blog/~kidehen#Dataset                                            http://rdfs.org/ns/void
http://www.openlinksw.com/blog/~kidehen#Dataset                                            http://rdfs.org/ns/void
http://www.openlinksw.com/blog/~kidehen#Dataset                                            http://rdfs.org/ns/void
http://www.openlinksw.com/blog/~kidehen#Dataset                                            http://rdfs.org/ns/void
http://www.openlinksw.com/blog/~kidehen#Dataset                                            http://rdfs.org/ns/void
http://www.openlinksw.com/blog/~kidehen#Dataset                                            http://rdfs.org/ns/void
http://www.openlinksw.com/blog/~kidehen#Dataset                                            http://rdfs.org/ns/void
http://www.openlinksw.com/blog/~kidehen#Dataset                                            http://rdfs.org/ns/void
http://www.openlinksw.com/blog/~kidehen#Dataset                                            http://rdfs.org/ns/void
http://www.openlinksw.com/blog/~kidehen#Dataset                                            http://rdfs.org/ns/void
http://www.openlinksw.com/blog/~kidehen#Dataset                                            http://rdfs.org/ns/void
http://www.openlinksw.com/blog/~kidehen#Dataset                                            http://rdfs.org/ns/void
http://www.openlinksw.com/blog/~kidehen#Dataset                                            http://rdfs.org/ns/void
http://www.openlinksw.com/blog/~kidehen#Dataset                                            http://rdfs.org/ns/void
http://www.openlinksw.com/blog/~kidehen#Dataset                                            http://rdfs.org/ns/void
http://www.openlinksw.com/blog/~kidehen#Dataset                                            http://rdfs.org/ns/void
http://www.openlinksw.com/blog/~kidehen#Dataset                                            http://rdfs.org/ns/void
http://www.openlinksw.com/blog/~kidehen#Dataset                                            http://rdfs.org/ns/void
http://www.openlinksw.com/blog/~kidehen#Dataset                                            http://rdfs.org/ns/void
http://www.openlinksw.com/blog/~kidehen#Stat                                               http://www.w3.org/1999/
http://www.openlinksw.com/blog/~kidehen#Stat                                               http://www.w3.org/1999/
http://www.openlinksw.com/blog/~kidehen#Stat                                               http://purl.org/NET/sco
http://www.openlinksw.com/blog/~kidehen#sameAsLinks                                        http://www.w3.org/1999/
http://www.openlinksw.com/blog/~kidehen#sameAsLinks                                        http://rdfs.org/ns/void
```

```
http://www.openlinksw.com/blog/~kidehen#sameAsStat                    http://www.w3.org/1999/
http://www.openlinksw.com/blog/~kidehen#sameAsStat                    http://www.w3.org/1999/
http://www.openlinksw.com/blog/~kidehen#sameAsStat                    http://purl.org/NET/sco
http://www.openlinksw.com/blog/~kidehen#sameAsType                    http://www.w3.org/1999/

30 Rows. -- 340 msec.
SQL>
```

See Also

DB.DBA.RDF_AUDIT_METADATA

DB.DBA.RDF_BACKUP_METADATA

DB.DBA.RDF_LOAD_RDFXML_MT

DB.DBA.TTLP

DB.DBA.TTLP_MT

DB.DBA.RDF_TRIPLES_TO_RDF_XML_TEXT

rdf_load_stop

rdf_load_stop — Stops RDF Loader threads.

Synopsis

```
rdf_load_stop (  );
```

Description

All RDF loader threads can be stopped using the command rdf_load_stop(), at which point all currently running threads will be allowed to complete and then exit.

Parameters

Examples

Example 24.259. Simple example

```
SQL> rdf_load_stop();
```

See Also

Bulk Loading RDF Source Files into one or more Graph IRIs

ld_dir()

ld_dir_all()

rdf_loader_run()

rdf_loader_run

rdf_loader_run — Performs bulk load.

Synopsis

```
rdf_loader_run ( in max_files integer := NULL ,
                 in log_enable int := 2 ) ;
```

Description

Virtuoso Bulk Loader function that performs bulk load. The function recognizes several file types, including .ttl, .nt, .xml, .rdf, .owl, .nq, .n4, and others. Internally the function uses ttlp or db..rdf_load_rdfxml, as appropriate. Please see more details for what file formats and file extensions are supported.

Note: The Virtuoso Bulk Loader functions must be present. They are pre-loaded starting with commercial version 06.02.3129 and open source version 6.1.3, but must be manually loaded into older versions.

Parameters

max_files
: Maximum files to be loaded.

log_enable
: For the default value 2 triggers are disabled, to speed the loading of data. If triggers are required (e.g., for RDF Graph replication between nodes), then the log_enable mode should be set to 3.

Examples

Example 24.260. Simple example

```
SQL> rdf_loader_run();
```

See Also

Bulk Loading RDF Source Files into one or more Graph IRIs

ld_dir()

ld_dir_all()

rdf_load_stop()

ld_dir_all

ld_dir_all — Loads from the specified directory, including any and all subdirectories.

Synopsis

ld_dir_all (in *dir_path* varchar ,
in *file_mask* varchar ,
in *target_graph* varchar) ;

Description

Adds files from the specified directory, including any and all subdirectories, to control list set up in the virtuoso.ini file.

Parameters

dir_path
: path to the folder where the files will be loaded

file_mask
: SQL like pattern to match against the files in the directory

target_graph
: target graph IRI, parsed triples will appear in that graph.

Return Types

The return value is not specified and may be changed in future versions.

Examples

Example 24.261. Simple Use

```
ld_dir_all ('/data8/', '%.gz', 'http://bsbm.org');
```

would load the RDF in all files ending in .gz from the directory (and its children) given as first parameter. The RDF would be loaded in the http://bsbm.org graph.

If NULL is given for the graph, each file may go to a different graph specified in a separate file with the name of the RDF source file plus the extension .graph.

A .graph file contains the target graph URI without any other content or whitespace.

See Also

RDF Bulk Load Utility

`ld_dir()`

`DB.DBA.RDF_AUDIT_METADATA()`

`DB.DBA.RDF_BACKUP_METADATA()`

`DB.DBA.RDF_LOAD_RDFXML()`

`DB.DBA.RDF_LOAD_RDFXML_MT()`

```
DB.DBA.TTLP_MT()

DB.DBA.TTLP()

DB.DBA.TTLP_MT_LOCAL_FILE()
```

DB.DBA.RDF_RESTORE_METADATA

DB.DBA.RDF_RESTORE_METADATA — Restores RDF metadata (descriptions of Linked Data Views and the like) from previously saved backup.

Synopsis

DB.DBA.RDF_RESTORE_METADATA (in *read_from_file* integer ,
in *backup_name* varchar) ;

Description

This restores RDF metadata from specified file or graph in RDF storage. The file or graph should be previously created by DB.DBA.RDF_BACKUP_METADATA () . It is usually safe to restore metadata from backup made by some previous version of Virtuoso server but it is good idea to call DB.DBA.RDF_AUDIT_METADATA () after such restore.

When the procedure is executed on the server that continues to serve user requests during the maintenance then SPARQL query compiler may interrupt query compilations or create queries that will return incomplete result sets. These queries may be cached till the end of metadata update procedure but the cache is flushed when the update is complete, so possible errors will be transient. If even transient errors are not appropriate then do RDF metadata update with all precautions usual for changing database schema of an application.

Parameters

read_from_file
: This flag specifies the type of the origin: it should be nonzero for file, zero for graph.

backup_name
: Name of backup file or graph as it is returned by the call of DB.DBA.RDF_BACKUP_METADATA () that created the backup.

Return Types

The return value is not specified and may be changed in future versions.

See Also

DB.DBA.RDF_AUDIT_METADATA()

DB.DBA.RDF_BACKUP_METADATA()

DB.DBA.RDF_LOAD_RDFXML_MT

DB.DBA.RDF_LOAD_RDFXML_MT — Parses the content of large RDF/XML text as a sequence of separate RDF triples.

Synopsis

DB.DBA.RDF_LOAD_RDFXML_MT (in *strg* varchar ,
in *base* varchar ,
in *graph* varchar ,
in *log_mode* integer ,
in *threads* integer ,
in *transactional* integer) ;

Description

For loading large resources when transactional integrity is not important (loading of a single resource may take more than one transaction). The function may or may not leave a transaction log, depending on `log_mode` . Hence, after successful load, one may need to execute the checkpoint statement to make sure that a server restart does not wipe out the results.

Parameters

strg
: text of the resource.

base
: base IRI to resolve relative IRIs to absolute.

graph
: target graph IRI, parsed triples will appear in that graph.

log_mode
: detail level of writing the effect of loading to the transaction log. 0 means log nothing, 1 means log only allocations of internal IDs for new IRIs and literals, 2 means log everything. If database crashes when the loading is in progress or after the loading but before checkpoint is made, 0 will means that the database become inconsistent, 1 means that the database is consistent but loaded quads may disappear so the loading should be repeated and log replay may produce wrong results if actions in it depend on the content of quad store, 2 means no danger (so the default is 2). Hence loading with mode 1 and especially mode 0 are faster than usual mode 2 but they require checkpoints after data loadings and mode 0 additionally requires a checkpoint and database backup right before the loading.

threads
: number of threads that insert quads into the database. It should not be less than 1, obviously; it is better to not set it greater than ((N-2)/k)-1 where N is the number of available CPU cores and k is the number of loadings that happen at the same time.

transactional
: controls the transaction mode which is "0" by default i.e. off and can be turned on by setting it to "1".

Return Types

The return value is not specified and may be changed in future versions.

Examples

Example 24.262. Usage Example

The following example demonstrates importing data from the rdf file resource Kingsley_Idehen.rdf which can be downloaded from here

```
SQL> DB.DBA.RDF_LOAD_RDFXML_MT (file_to_string_output ('tmp/Kingsley_Idehen.rdf'), '', 'http://example.co
Done. -- 61 msec.
SQL>sparql
select *
from <http://example.com>
where {?s ?p ?o};

s             p                     o
VARCHAR       VARCHAR               VARCHAR
_______________________________________________________________________________

http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#based_near      http://www.w3.org/1999/
http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#based_near      http://www.w3.org/2003/
http://www.openlinksw.com/dataspace/person/kidehen@openlinksw.com#based_near      http://www.w3.org/2003/
http://www.openlinksw.com/dataspace/organization/vdb#this                        http://www.w3.org/1999/
http://www.openlinksw.com/dataspace/organization/vdb#this                        http://www.w3.org/1999/
http://www.openlinksw.com/dataspace/organization/vdb#this                        http://xmlns.com/foaf/0
http://www.openlinksw.com/dataspace/organization/vdb#this                        http://xmlns.com/foaf/0
http://www.openlinksw.com/dataspace/organization/dav#this                        http://www.w3.org/1999/
http://www.openlinksw.com/dataspace/organization/dav#this                        http://www.w3.org/1999/
http://www.openlinksw.com/dataspace/organization/dav#this                        http://xmlns.com/foaf/0

10 Rows. -- 30 msec.
```

The following example demonstrates importing data from the rdf resource with URI: http://www.w3.org/People/Berners-Lee/card

```
SQL>create procedure MY_LOAD_FILE (in full_uri varchar, in in_resultset integer := 0)
{
  declare REPORT varchar;
  declare graph_uri, dattext varchar;
  declare app_env any;
  app_env := null;
  whenever sqlstate '*' goto err_rep;
  if (not in_resultset)
    result_names (REPORT);
  dattext := cast (XML_URI_GET_AND_CACHE (full_uri) as varchar);
  MY_SPARQL_REPORT (sprintf ('Downloading %s: %d bytes',
      full_uri, length (dattext) ) );
  graph_uri := full_uri;
  delete from RDF_QUAD where G = DB.DBA.RDF_MAKE_IID_OF_QNAME (graph_uri);
  DB.DBA.RDF_LOAD_RDFXML_MT (dattext, full_uri, graph_uri);
  return graph_uri;
err_rep:
  result (sprintf ('%s: %s', __SQL_STATE, __SQL_MESSAGE));
  return graph_uri;
}
;

Done. -- 0 msec.

SQL>create procedure MY_SPARQL_REPORT(in strg varchar)
{
  if (__tag(strg) <> 182)
    strg := cast (strg as varchar) || sprintf (' -- not a string, tag=%d', __tag(strg));
  strg := replace (strg, SPARQL_DAV_DATA_URI(), '\044{SPARQL_DAV_DATA_URI()}');
  strg := replace (strg, SPARQL_DAV_DATA_PATH(), '\044{SPARQL_DAV_DATA_PATH()}');
  strg := replace (strg, SPARQL_FILE_DATA_ROOT(), '\044{SPARQL_FILE_DATA_ROOT()}');
  result (strg);
}
;

Done. -- 0 msec.

SQL> MY_LOAD_FILE('http://www.w3.org/People/Berners-Lee/card');
REPORT
VARCHAR
_______________________________________________________________________________

Downloading http://www.w3.org/People/Berners-Lee/card: 17773 bytes

1 Rows. -- 4046 msec.

SQL>sparql
```

```
select *
from <http://www.w3.org/People/Berners-Lee/card>
where {?s ?p ?o} ;

s                                          p                                                o
VARCHAR                                    VARCHAR                                          VARCHAR
_______________________________________________________________________________________________

http://bblfish.net/people/henry/card#me    http://xmlns.com/foaf/0.1/name                   Henry Story
http://www.w3.org/People/Berners-Lee/card#i http://www.w3.org/1999/02/22-rdf-syntax-ns#type http://xmln
http://www.w3.org/People/Berners-Lee/card#i http://www.w3.org/1999/02/22-rdf-syntax-ns#type http://www.
http://www.w3.org/People/Berners-Lee/card#i http://xmlns.com/foaf/0.1/nick                 TimBL
http://www.w3.org/People/Berners-Lee/card#i http://xmlns.com/foaf/0.1/nick                 timbl
http://www.w3.org/People/Berners-Lee/card#i http://xmlns.com/foaf/0.1/mbox                 mailto:timb
http://www.w3.org/People/Berners-Lee/card#i http://xmlns.com/foaf/0.1/mbox_sha1sum         965c47c5a70
http://www.w3.org/People/Berners-Lee/card#i http://xmlns.com/foaf/0.1/knows                http://bblf
http://www.w3.org/People/Berners-Lee/card#i http://xmlns.com/foaf/0.1/knows                http://home
http://www.w3.org/People/Berners-Lee/card#i http://xmlns.com/foaf/0.1/knows                http://danb
http://www.w3.org/People/Berners-Lee/card#i http://xmlns.com/foaf/0.1/knows                http://norm
http://www.w3.org/People/Berners-Lee/card#i http://xmlns.com/foaf/0.1/knows                http://www.
http://www.w3.org/People/Berners-Lee/card#i http://xmlns.com/foaf/0.1/knows                http://www.
http://www.w3.org/People/Berners-Lee/card#i http://xmlns.com/foaf/0.1/knows                http://www.
http://www.w3.org/People/Berners-Lee/card#i http://xmlns.com/foaf/0.1/knows                http://dig.
..........
```

See Also

DB.DBA.RDF_AUDIT_METADATA()

DB.DBA.RDF_BACKUP_METADATA()

DB.DBA.RDF_LOAD_RDFXML()

DB.DBA.TTLP()

DB.DBA.TTLP_MT()

DB.DBA.RDF_LOAD_RDFXML

DB.DBA.RDF_LOAD_RDFXML — Parses the content of RDF/XML text as a sequence of separate RDF triples.

Synopsis

DB.DBA.RDF_LOAD_RDFXML (in *rdfxml_text* varchar ,
in *omt_top_rdf* varchar ,
in *graph_uri* varchar) ;

Description

Parses the content of RDF/XML text as a sequence of separate RDF triples.

Parameters

rdfxml_text
: text of XML document

omt_top_rdf
: base IRI to resolve relative IRIs

graph_uri
: the IRI of destination graph

Return Types

The return value is not specified and may be changed in future versions.

Examples

Example 24.263. Simple Use

Load from local file:

```
SQL>sparql drop graph <http://www.biopax.org/prototype#sample-closure>;
Done. -- 2 msec.
SQL>DB.DBA.RDF_LOAD_RDFXML (file_to_string ('MatInf-for-Alan/Reactome_109581.owl'), '', 'http://www.biopa
Done. -- 106 msec.
```

Load from URI:

```
SQL> DB.DBA.RDF_LOAD_RDFXML (http_get('http://lod.taxonconcept.org/ontology/txn.owl'), '', 'http://lod.ta
Done. -- 109 msec.

-- Check total triples retrieved:
SQL>SPARQL
SELECT COUNT(*)
from <http://lod.taxonconcept.org/ontology/txn.owl#>
WHERE {?s ?p ?o};
callret-0
INTEGER
_______________________________________________________________________________

495

1 Rows. -- 16 msec.
```

See Also

DB.DBA.RDF_AUDIT_METADATA()

DB.DBA.RDF_BACKUP_METADATA()

DB.DBA.RDF_LOAD_RDFXML_MT()

DB.DBA.TTLP()

DB.DBA.TTLP_MT()

DB.DBA.RDF_LOAD_RDFA

DB.DBA.RDF_LOAD_RDFA — Parses the content of RDF embedded as RDFa text as a sequence of separate RDF triples.

Synopsis

DB.DBA.RDF_LOAD_RDFA (in *rdfa_text* varchar ,
in *omt_top_rdf* varchar ,
in *graph_uri* varchar) ;

Description

Parses the content of RDF embedded as RDFa text as a sequence of separate RDF triples.

Parameters

rdfa_text
: text of document containing RDFa data

omt_top_rdf
: base IRI to resolve relative IRIs

graph_uri
: the IRI of destination graph

Return Types

The return value is not specified and may be changed in future versions.

Examples

Example 24.264. Simple Use

Load from local file:

```
SQL>sparql clear graph <http://virtuoso.openlinksw.com/example>;
Done. -- 2 msec.
SQL>DB.DBA.RDF_LOAD_RDFA (file_to_string ('pricing.html'), 'http://virtuoso.openlinksw.com/pricing/#', 'h
Done. -- 106 msec.
```

Load from URI:

```
SQL>sparql clear graph <http://virtuoso.openlinksw.com/example>;
SQL> DB.DBA.RDF_LOAD_RDFA (http_get('http://virtuoso.openlinksw.com/pricing/'), 'http://virtuoso.openlink
Done. -- 109 msec.

-- Check total triples retrieved:
SQL>SPARQL
SELECT COUNT(*)
from <http://virtuoso.openlinksw.com/example>
WHERE {?s ?p ?o};
callret-0
INTEGER
_______________________________________________________________________________

1181

1 Rows. -- 16 msec.
```

See Also

DB.DBA.RDF_LOAD_RDFXML()

DB.DBA.RDF_AUDIT_METADATA()

DB.DBA.RDF_BACKUP_METADATA()

DB.DBA.RDF_LOAD_RDFA_MT()

DB.DBA.TTLP()

DB.DBA.TTLP_MT()

ld_dir

ld_dir — Adds files to control list set up in the virtuoso.ini file.

Synopsis

```
ld_dir ( in dir_path varchar ,
         in file_mask varchar ,
         in target_graph varchar ) ;
```

Description

Adds files to control list set up in the virtuoso.ini file.

Parameters

dir_path
: path to the folder where the files will be loaded

file_mask
: SQL like pattern to match against the files in the directory

target_graph
: target graph IRI, parsed triples will appear in that graph.

Return Types

The return value is not specified and may be changed in future versions.

Examples

Example 24.265. Simple Use

```
ld_dir ('/data8/2848260', '%.gz', 'http://bsbm.org');
```

would load the RDF in all files ending in .gz from the directory given as first parameter. The RDF would be loaded in the http://bsbm.org graph.

If NULL is given for the graph, each file may go to a different graph specified in a separate file with the name of the RDF source file plus the extension .graph.

A .graph file contains the target graph URI without any other content or whitespace.

See Also

DB.DBA.RDF_AUDIT_METADATA()

DB.DBA.RDF_BACKUP_METADATA()

DB.DBA.RDF_LOAD_RDFXML()

DB.DBA.RDF_LOAD_RDFXML_MT()

DB.DBA.TTLP_MT()

DB.DBA.TTLP()

```
DB.DBA.TTLP_MT_LOCAL_FILE()
```

RDF Bulk Load Utility

DB.DBA.TTLP

DB.DBA.TTLP — parses TTL (TURTLE or N3 resource) and places its triples into DB.DBA.RDF_QUAD.

Synopsis

DB.DBA.TTLP (in *strg* any ,
in *base* varchar ,
in *graph* varchar ,
in *flags* integer) ;

Description

Parses TTL (TURTLE or N3 resource) and places its triples into DB.DBA.RDF_QUAD.

Parameters

strg
: text of the resource

base
: base IRI to resolve relative IRIs to absolute

graph
: target graph IRI, parsed triples will appear in that graph.

flags
: bitmask of parsing flags. Permits some sorts of syntax errors in resource. Default is 0, meaning no permitted deviations from the spec. Other supported bits are:

Return Types

The return value is not specified and may be changed in future versions.

Examples

Example 24.266.

For loading a file of any greater length, it is most practical to use the file_to_string_output function.

It is important the file to be accessible for the Virtuoso server. You need to have set properly the DirsAllowed parameter value in section [Parameters] of the Virtuoso database INI file. For example on Windows it could be:

```
virtuoso.ini file:
[Parameters]
...
DirsAllowed =  .\tmp
...
```

So in the example the file you want to import from, should be in the tmp folder or in its subfolder. Note that this example folder is a subfolder of the Virtuoso Server working directory.

Sample Example 1

```
SQL> ttlp (file_to_string_output ('.\tmp\data.ttl'), '', 'http://my_graph', 0);
```

Forth item in record of NQuad format is optional. If present then it is used as a graph. If missed then the default graph is used. The purpose is to make SPARQL dataset serialization possible in a "natural" way: the content of default graph is written without any

graph name specified whereas the content of named graphs is written with fouth field in place. Similarly, TriG uses default graph for triples outside graph blocks. In all these cases, base IRI is used to resolve all relative IRIs of the document, no matter what is the destination graph. Here is a simple example:

Sample Example 2 Suppose we have the simple nquad.nq file:

```
<http://www.w3.org/2002/01/tr-automation/tr.rdf> <http://purl.org/dc/elements/1.1/title> "W3C Standards a
<http://www.w3.org/People/Berners-Lee/card> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xml
<http://www.w3.org/People/Berners-Lee/card> <http://creativecommons.org/ns#license> <http://creativecommo
<http://www.w3.org/People/Berners-Lee/card> <http://purl.org/dc/elements/1.1/title> "Tim Berners-Lee's FO
<http://www.w3.org/People/Berners-Lee/card> <http://xmlns.com/foaf/0.1/maker> <http://www.w3.org/People/B
<http://www.w3.org/People/Berners-Lee/card> <http://xmlns.com/foaf/0.1/primaryTopic> <http://www.w3.org/P
<http://www.w3.org/People/Berners-Lee/card#cm> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://
<http://www.w3.org/People/Berners-Lee/card#cm> <http://www.w3.org/2000/01/rdf-schema#seeAlso> <http://www
<http://www.w3.org/People/Berners-Lee/card#cm> <http://xmlns.com/foaf/0.1/mbox> <mailto:coralie@w3.org> <
<http://www.w3.org/People/Berners-Lee/card#cm> <http://xmlns.com/foaf/0.1/name> "Coralie Mercier" <http:/
```

Load the file:

```
SQL>DB.DBA.TTLP (file_to_string_output ('./nquad.nq'), '', 'http://example.com/DAV/test', 512);
Done. -- 48 msec.
```

Select all triples from the graph with URI the fourth provenance URI from the NQuad file i.e.: <http://www.w3.org/People/Berners-Lee/card>:

```
SQL> sparql select * from <http://www.w3.org/People/Berners-Lee/card> where {?s ?p ?o};
s                                              p                                                  o
VARCHAR                                        VARCHAR                                            VARCH
_______________________________________________

http://www.w3.org/2002/01/tr-automation/tr.rdf  http://purl.org/dc/elements/1.1/title             W3C S
http://www.w3.org/People/Berners-Lee/card       http://www.w3.org/1999/02/22-rdf-syntax-ns#type    http:
http://www.w3.org/People/Berners-Lee/card       http://xmlns.com/foaf/0.1/primaryTopic             http:
http://www.w3.org/People/Berners-Lee/card       http://purl.org/dc/elements/1.1/title              Tim B
http://www.w3.org/People/Berners-Lee/card       http://xmlns.com/foaf/0.1/maker                    http:
http://www.w3.org/People/Berners-Lee/card       http://creativecommons.org/ns#license              http:
http://www.w3.org/People/Berners-Lee/card#cm    http://www.w3.org/1999/02/22-rdf-syntax-ns#type    http:
http://www.w3.org/People/Berners-Lee/card#cm    http://xmlns.com/foaf/0.1/mbox                     mailt
http://www.w3.org/People/Berners-Lee/card#cm    http://xmlns.com/foaf/0.1/name                     Coral
http://www.w3.org/People/Berners-Lee/card#cm    http://www.w3.org/2000/01/rdf-schema#seeAlso       http:

10 Rows. -- 7 msec.
```

See Also

DB.DBA.RDF_AUDIT_METADATA()

DB.DBA.RDF_BACKUP_METADATA()

DB.DBA.RDF_LOAD_RDFXML()

DB.DBA.RDF_LOAD_RDFXML_MT()

DB.DBA.TTLP_MT()

DB.DBA.TTLP_MT_LOCAL_FILE()

DB.DBA.TTLP_MT

DB.DBA.TTLP_MT — parses TTL (TURTLE), TTL(N-Triple) or NQ (N-quad) resources, and places its triples into DB.DBA.RDF_QUAD on multiple threads.

Synopsis

DB.DBA.TTLP_MT (in *strg* any ,
in *base* varchar ,
in *graph* varchar ,
in *flags* integer ,
in *log_mode* integer ,
in *threads* integer ,
in *transactional* integer) ;

Description

Loads the TTL (TURTLE or N3 resource) file on multiple threads, using parallel I/O and multiprocessing if available. The function commit partial transactions while it runs so the transaction log may contain part of loading. Moreover, the function may or may not leave a transaction log, depending on `log_mode` . Hence, after successful load, one may need to execute the checkpoint statement to make sure that a server restart does not wipe out the results.

Parameters

strg
: text of the resource

base
: base IRI to resolve relative IRIs to absolute

graph
: target graph IRI, parsed triples will appear in that graph.

flags
: bitmask of parsing flags. Permits some sorts of syntax errors in resource. Default is 0, meaning no permitted deviations from the spec. Other supported bits are:

log_mode
: detail level of writing the effect of loading to the transaction log. 0 means log nothing, 1 means log only allocations of internal IDs for new IRIs and literals, 2 means log everything. If database crashes when the loading is in progress or after the loading but before checkpoint is made, 0 will means that the database become inconsistent, 1 means that the database is consistent but loaded quads may disappear so the loading should be repeated and log replay may produce wrong results if actions in it depend on the content of quad store, 2 means no danger (so the default is 2). Hence loading with mode 1 and especially mode 0 are faster than usual mode 2 but they require checkpoints after data loadings and mode 0 additionally requires a checkpoint and database backup right before the loading.

threads
: number of threads that insert quads into the database. It should not be less than 1, obviously; it is better to not set it greater than ((N-2)/k)-1 where N is the number of available CPU cores and k is the number of loadings that happen at the same time.

transactional
: controls the transaction mode which is "0" by default i.e. off and can be turned on by setting it to "1".

Return Types

The return value is not specified and may be changed in future versions.

Examples

Example 24.267.

Example 1 Load ttl from local file:

```
SQL> DB.DBA.TTLP_MT (file_to_string_output ('tmp/users.ttl'), '', 'http://example.com');

Done. -- 381 msec.
SQL>sparql
select *
from <http://example.com>
where {?s ?p ?o}
limit 10;
s             p                   o
VARCHAR       VARCHAR             VARCHAR
_______________________________________________________________________________

http://www.openlinksw.com/virtrdf-data-formats#SysUsers-GranteeId-format          http://www.w3.org/1999/
http://www.openlinksw.com/virtrdf-data-formats#SysUsers-GranteeId-format          http://www.openlinksw.c
http://www.openlinksw.com/virtrdf-data-formats#SysUsers-GranteeId-format          http://www.openlinksw.c
http://www.openlinksw.com/virtrdf-data-formats#SysUsers-GranteeId-format          http://www.openlinksw.c
http://www.openlinksw.com/virtrdf-data-formats#SysUsers-GranteeId-format          http://www.openlinksw.c
http://www.openlinksw.com/virtrdf-data-formats#SysUsers-UserId-format             http://www.w3.org/1999/
http://www.openlinksw.com/virtrdf-data-formats#SysUsers-UserId-format             http://www.openlinksw.c
http://www.openlinksw.com/virtrdf-data-formats#SysUsers-UserId-format             http://www.openlinksw.c
http://www.openlinksw.com/virtrdf-data-formats#SysUsers-UserId-format             http://www.openlinksw.c
http://www.openlinksw.com/virtrdf-data-formats#SysUsers-UserId-format             http://www.openlinksw.c

10 Rows. -- 30 msec.
```

Example 2 Usage of the ttlp_mt function in a procedure:

```
SQL>create procedure SPARQL_DAWG_LOAD_REMOTE_DATFILE (in full_uri varchar, in in_resultset integer := 0)
{
  declare REPORT varchar;
  declare graph_uri, dattext varchar;
  declare app_env any;
  app_env := null;
  whenever sqlstate '*' goto err_rep;
  if (not in_resultset)
    result_names (REPORT);
  dattext := cast (XML_URI_GET_AND_CACHE (full_uri) as varchar);
  SPARQL_REPORT (sprintf ('Downloading %s: %d bytes',
      full_uri, length (dattext) ) );
  graph_uri := full_uri;
  delete from RDF_QUAD where G = DB.DBA.RDF_MAKE_IID_OF_QNAME (graph_uri);
  if ((full_uri like '%.ttl') or (full_uri like '%.nt') or (full_uri like '%.n3'))
    DB.DBA.TTLP_MT (dattext, full_uri, graph_uri);
  else -- if (rel_path like '%.rdf')
    DB.DBA.RDF_LOAD_RDFXML_MT (dattext, full_uri, graph_uri);
  return graph_uri;
err_rep:
  result (sprintf ('%s: %s', __SQL_STATE, __SQL_MESSAGE));
  return graph_uri;
}
;
Done. -- 891 msec
SQL> select SPARQL_DAWG_LOAD_REMOTE_DATFILE('http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/w
REPORT
VARCHAR
_______________________________________________________________________________

Downloading http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%20%5B
```

```
2 Rows. -- 1382 msec.
```

Example 3 Fourth item in record of NQuad format is optional. If present then it is used as a graph. If missed then the default graph is used. The purpose is to make SPARQL dataset serialization possible in a "natural" way: the content of default graph is written without any graph name specified whereas the content of named graphs is written with fourth field in place. Similarly, TriG uses default graph for triples outside graph blocks. In all these cases, base IRI is used to resolve all relative IRIs of the document, no matter what is the destination graph. Suppose the following scenario:

There is a simple nquad.nq file:

```
<http://www.w3.org/2002/01/tr-automation/tr.rdf> <http://purl.org/dc/elements/1.1/title> "W3C Standards a
<http://www.w3.org/People/Berners-Lee/card> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xml
<http://www.w3.org/People/Berners-Lee/card> <http://creativecommons.org/ns#license> <http://creativecommo
<http://www.w3.org/People/Berners-Lee/card> <http://purl.org/dc/elements/1.1/title> "Tim Berners-Lee's FO
<http://www.w3.org/People/Berners-Lee/card> <http://xmlns.com/foaf/0.1/maker> <http://www.w3.org/People/B
<http://www.w3.org/People/Berners-Lee/card> <http://xmlns.com/foaf/0.1/primaryTopic> <http://www.w3.org/P
<http://www.w3.org/People/Berners-Lee/card#cm> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://
<http://www.w3.org/People/Berners-Lee/card#cm> <http://www.w3.org/2000/01/rdf-schema#seeAlso> <http://www
<http://www.w3.org/People/Berners-Lee/card#cm> <http://xmlns.com/foaf/0.1/mbox> <mailto:coralie@w3.org> <
<http://www.w3.org/People/Berners-Lee/card#cm> <http://xmlns.com/foaf/0.1/name> "Coralie Mercier" <http:/
```

Load the file:

```
SQL>DB.DBA.TTLP_MT (file_to_string_output ('./nquad.nq'), '', 'http://example.com/DAV/test', 512);
Done. -- 48 msec.
```

Select all triples from the graph with URI the fourth provenance URI from the NQuad file i.e.: <http://www.w3.org/People/Berners-Lee/card>:

```
SQL> sparql select * from <http://www.w3.org/People/Berners-Lee/card> where {?s ?p ?o};
s                                              p                                                 o
VARCHAR                                        VARCHAR                                           VARCH
_______________________________________________

http://www.w3.org/2002/01/tr-automation/tr.rdf  http://purl.org/dc/elements/1.1/title             W3C S
http://www.w3.org/People/Berners-Lee/card       http://www.w3.org/1999/02/22-rdf-syntax-ns#type   http:
http://www.w3.org/People/Berners-Lee/card       http://xmlns.com/foaf/0.1/primaryTopic            http:
http://www.w3.org/People/Berners-Lee/card       http://purl.org/dc/elements/1.1/title             Tim B
http://www.w3.org/People/Berners-Lee/card       http://xmlns.com/foaf/0.1/maker                   http:
http://www.w3.org/People/Berners-Lee/card       http://creativecommons.org/ns#license             http:
http://www.w3.org/People/Berners-Lee/card#cm    http://www.w3.org/1999/02/22-rdf-syntax-ns#type   http:
http://www.w3.org/People/Berners-Lee/card#cm    http://xmlns.com/foaf/0.1/mbox                    mailt
http://www.w3.org/People/Berners-Lee/card#cm    http://xmlns.com/foaf/0.1/name                    Coral
http://www.w3.org/People/Berners-Lee/card#cm    http://www.w3.org/2000/01/rdf-schema#seeAlso      http:

10 Rows. -- 7 msec.
```

Example 4 Usage of the ttlp_mt function with transactional param set to 1

```
-- Setting the "isql" tool into "manual" commit mode with the command:
SQL> set AUTOCOMMIT MANUAL;

SQL> sparql drop silent graph <data.nt>;
Done. -- 1 msec.

SQL> sparql select count(*) from <data.nt> where {?s ?p ?o};
callret-0
INTEGER
_______________________________________________________________________________

0

1 Rows. -- 1 msec.

-- Perform data load with TTLP_MT()
SQL> DB.DBA.TTLP_MT (file_to_string_output ('./data.nt'), '', 'data.nt', 512, 1, 1, 1);

Done. -- 3 msec.
SQL> sparql select count(*) from <data.nt> where {?s ?p ?o};
```

```
callret-0
INTEGER
_______________________________________________________________________________

100

1 Rows. -- 1 msec.

-- Commit or Rollback the transaction/work as required with the commit work or rollback work commands
SQL> rollback work;

Done. -- 0 msec.
SQL> sparql select count(*) from <data.nt> where {?s ?p ?o};
callret-0
INTEGER
_______________________________________________________________________________

0

1 Rows. -- 1 msec.
SQL> DB.DBA.TTLP_MT (file_to_string_output ('./data.nt'), '', 'data.nt', 512, 1, 1, 1);

Done. -- 3 msec.
SQL> sparql select count(*) from <data.nt> where {?s ?p ?o};
callret-0
INTEGER
_______________________________________________________________________________

100

1 Rows. -- 0 msec.
SQL> commit work;

Done. -- 1 msec.
SQL> sparql select count(*) from <data.nt> where {?s ?p ?o};
callret-0
INTEGER
_______________________________________________________________________________

100

1 Rows. -- 0 msec.
SQL> rollback work;

Done. -- 0 msec.
SQL> sparql select count(*) from <data.nt> where {?s ?p ?o};
callret-0
INTEGER
_______________________________________________________________________________

100

1 Rows. -- 0 msec.
SQL>
```

See Also

DB.DBA.RDF_AUDIT_METADATA()

DB.DBA.RDF_BACKUP_METADATA()

DB.DBA.RDF_LOAD_RDFXML()

DB.DBA.RDF_LOAD_RDFXML_MT()

DB.DBA.TTLP()

DB.DBA.TTLP_MT_LOCAL_FILE

DB.DBA.TTLP_MT_LOCAL_FILE — parses TTL (TURTLE or N3 resource) and places its triples into DB.DBA.RDF_QUAD on multiple threads.

Synopsis

```
DB.DBA.TTLP_MT_LOCAL_FILE ( in path varchar ,
                            in base varchar ,
                            in graph varchar ,
                            in flags integer ,
                            in log_mode integer ,
                            in threads integer ) ;
```

Description

Loads the TTL (TURTLE or N3 resource) file on multiple threads, using parallel I/O and multiprocessing if available. The function commit partial transactions while it runs so the transaction log may contain part of loading. Moreover, the function may or may not leave a transaction log, depending on `log_mode` . Hence, after successful load, one may need to execute the checkpoint statement to make sure that a server restart does not wipe out the results.

Parameters

path
: path to the file in the local filesystem

base
: base IRI to resolve relative IRIs to absolute

graph
: target graph IRI, parsed triples will appear in that graph.

flags
: bitmask of parsing flags. Permits some sorts of syntax errors in resource. Default is 0, meaning no permitted deviations from the spec. Other supported bits are:

log_mode
: detail level of writing the effect of loading to the transaction log. 0 means log nothing, 1 means log only allocations of internal IDs for new IRIs and literals, 2 means log everything. If database crashes when the loading is in progress or after the loading but before checkpoint is made, 0 will means that the database become inconsistent, 1 means that the database is consistent but loaded quads may disappear so the loading should be repeated and log replay may produce wrong results if actions in it depend on the content of quad store, 2 means no danger (so the default is 2). Hence loading with mode 1 and especially mode 0 are faster than usual mode 2 but they require checkpoints after data loadings and mode 0 additionally requires a checkpoint and database backup right before the loading.

threads
: number of threads that insert quads into the database. It should not be less than 1, obviously; it is better to not set it greater than ((N-2)/k)-1 where N is the number of available CPU cores and k is the number of loadings that happen at the same time.

Return Types

The return value is not specified and may be changed in future versions.

Examples

Example 24.268.

Sample Example 1

```
SQL>DB.DBA.TTLP_MT_LOCAL_FILE ('tmp/users.ttl', '', 'http://example.com');

Done. -- 381 msec.
SQL>sparql
select *
from <http://example.com>
where {?s ?p ?o}
limit 10;
s             p                   o
VARCHAR       VARCHAR             VARCHAR
_______________________________________________________________________________

http://www.openlinksw.com/virtrdf-data-formats#SysUsers-GranteeId-format          http://www.w3.org/1999/
http://www.openlinksw.com/virtrdf-data-formats#SysUsers-GranteeId-format          http://www.openlinksw.c
http://www.openlinksw.com/virtrdf-data-formats#SysUsers-GranteeId-format          http://www.openlinksw.c
http://www.openlinksw.com/virtrdf-data-formats#SysUsers-GranteeId-format          http://www.openlinksw.c
http://www.openlinksw.com/virtrdf-data-formats#SysUsers-GranteeId-format          http://www.openlinksw.c
http://www.openlinksw.com/virtrdf-data-formats#SysUsers-UserId-format             http://www.w3.org/1999/
http://www.openlinksw.com/virtrdf-data-formats#SysUsers-UserId-format             http://www.openlinksw.c
http://www.openlinksw.com/virtrdf-data-formats#SysUsers-UserId-format             http://www.openlinksw.c
http://www.openlinksw.com/virtrdf-data-formats#SysUsers-UserId-format             http://www.openlinksw.c
http://www.openlinksw.com/virtrdf-data-formats#SysUsers-UserId-format             http://www.openlinksw.c

10 Rows. -- 30 msec.
```

Sample Example 2

```
SQL>create procedure SPARQL_DAWG_LOAD_REMOTE_DATFILE (in full_uri varchar, in in_resultset integer := 0)
{
  declare REPORT varchar;
  declare graph_uri, dattext varchar;
  declare app_env any;
  app_env := null;
  whenever sqlstate '*' goto err_rep;
  if (not in_resultset)
    result_names (REPORT);
  dattext := cast (XML_URI_GET_AND_CACHE (full_uri) as varchar);
  SPARQL_REPORT (sprintf ('Downloading %s: %d bytes',
      full_uri, length (dattext) ) );
  graph_uri := full_uri;
  delete from RDF_QUAD where G = DB.DBA.RDF_MAKE_IID_OF_QNAME (graph_uri);
  if ((full_uri like '%.ttl') or (full_uri like '%.nt') or (full_uri like '%.n3'))
    DB.DBA.TTLP_MT_LOCAL_FILE (dattext, full_uri, graph_uri);
  else -- if (rel_path like '%.rdf')
    DB.DBA.RDF_LOAD_RDFXML_MT (dattext, full_uri, graph_uri);
  return graph_uri;
err_rep:
  result (sprintf ('%s: %s', __SQL_STATE, __SQL_MESSAGE));
  return graph_uri;
}
;
Done. -- 891 msec
SQL> select SPARQL_DAWG_LOAD_REMOTE_DATFILE('http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/w
REPORT
VARCHAR
_______________________________________________________________________________

Downloading http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%
http://www.openlinksw.com/dataspace/kidehen@openlinksw.com/weblog/kidehen@openlinksw.com%27s%20BLOG%20%5B

2 Rows. -- 1382 msec.
```

See Also

DB.DBA.RDF_AUDIT_METADATA()

DB.DBA.RDF_BACKUP_METADATA()

DB.DBA.RDF_LOAD_RDFXML()

DB.DBA.RDF_LOAD_RDFXML_MT()

DB.DBA.TTLP()

DB.DBA.TTLP_MT()

DB.DBA.RDF_DATATYPE_OF_OBJ

DB.DBA.RDF_DATATYPE_OF_OBJ — Returns the datatype of a given argument.

Synopsis

DB.DBA.RDF_DATATYPE_OF_OBJ (in *arg* any ,
in *type_val* any) ;

Description

Returns the datatype of a given argument.

Parameters

arg
: The argument which datatype should be returned.

type_val
: The value to be returned for untyped literal arguments. When omitted, its default value http://www.w3.org/2001/XMLSchema#string is used.

Return Types

any

Examples

Example 24.269. Usage Examples

```
# Explicit typecast (insert)
SQL> sparql insert into <test_datatype> { <a> <string> "string 1"^^xsd:string . };
callret-0
VARCHAR
_______________________________________________________________________________

Insert into <test_datatype>, 1 (or less) triples -- done

1 Rows. -- 94 msec.

#Not explicit typecast (insert)
SQL> sparql insert into <test_datatype> { <a> <string> "string 2". };
callret-0
VARCHAR
_______________________________________________________________________________

Insert into <test_datatype>, 1 (or less) triples -- done

1 Rows. -- 16 msec.

SQL> SPARQL
SELECT ?o (iri(sql:RDF_DATATYPE_OF_OBJ(?o, 'untyped!')))
FROM <test_datatype> { <a> <string> ?o} ;
o                     callret-1
VARCHAR               VARCHAR
_______________________________________________________________________________

string 1              http://www.w3.org/2001/XMLSchema#string
string 2              untyped!

2 Rows. -- 16 msec.
SQL>
```

DB.DBA.RDF_TRIPLES_TO_RDF_XML_TEXT

DB.DBA.RDF_TRIPLES_TO_RDF_XML_TEXT — Serializes vector of triples into a session, in RDF/XML syntax.

Synopsis

DB.DBA.RDF_TRIPLES_TO_RDF_XML_TEXT (inout *triples* any ,
in *print_top_level* any ,
inout *ses* any) ;

Description

Serializes vector of triples into a session, in TURTLE syntax. In current version, every triple is printed in separate top-level record (say, in rdf:Description tag), without any pretty-print or nesting optimization.

Parameters

triples
: vector of triples in 'long valmode'.

print_top_level
: zero if only rdf:Description tags should be written, non-zero if the rdf:RDF top-level element should also be written

ses
: an output stream in server default encoding

Examples

Example 24.270. Simple example

```
create procedure simple_test(in query varchar, in dflt_graph varchar)
{
  declare ses, rset, triples any;
  declare txt varchar;
  ses := string_output ();
  rset := DB.DBA.SPARQL_EVAL_TO_ARRAY (query, dflt_graph, 1);
  triples := dict_list_keys (rset[0][0], 1);
  DB.DBA.RDF_TRIPLES_TO_RDF_XML_TEXT (triples, 1, ses);
  txt := string_output_string (ses);
  dump_large_text (txt);
}
;
```

See Also

DB.DBA.RDF_AUDIT_METADATA

DB.DBA.RDF_BACKUP_METADATA

DB.DBA.RDF_LOAD_RDFXML_MT

DB.DBA.TTLP

DB.DBA.TTLP_MT

DB.DBA.RDF_TRIPLES_TO_TTL

DB.DBA.RDF_TRIPLES_TO_TTL

DB.DBA.RDF_TRIPLES_TO_TTL — Serializes vector of triples into a session, in TURTLE syntax.

Synopsis

DB.DBA.RDF_TRIPLES_TO_TTL (inout *triples* any ,
inout *ses* any) ;

Description

Serializes vector of triples into a session, in TURTLE syntax. In current version, every triple is printed in separate top-level record (say, in rdf:Description tag), without any pretty-print or nesting optimization.

Parameters

triples
: vector of triples in 'long valmode'.

ses
: an output stream in server default encoding

Examples

Example 24.271. Simple example

```
SQL>create procedure RDFXML2N3 (in _rdfxml varchar)
{
  declare triples, ses any;
  declare STRG varchar;
  result_names (STRG);
  delete from RDF_QUAD where G=DB.DBA.RDF_MAKE_IID_OF_QNAME ('http://fake.example.org/graph/');
  DB.DBA.RDF_LOAD_RDFXML_MT (_rdfxml, 'http://fake.example.org/base/', 'http://fake.example.org/graph/' )
  for (sparql define output:valmode "LONG" select ?s ?p ?o where { graph <http://fake.example.org/graph/>
    {
      ses := string_output ();
      DB.DBA.RDF_TRIPLES_TO_TTL (vector (vector ("s", "p", "o")), ses);
      result (sprintf ('%s\t%d %d', replace (string_output_string (ses), '\n', ' '), rdf_box_type ("o"),
    }
}

Done. -- 0 msec.

SQL>RDFXML2N3('
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         xmlns:ex="http://example.org/stuff/1.0/">

<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <ex:editor>
    <rdf:Description>
      <ex:fullName>Dave Beckett</ex:fullName>
    </rdf:Description>
  </ex:editor>
</rdf:Description>

<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <ex:editor>
    <rdf:Description>
      <ex:homePage>
        <rdf:Description rdf:about="http://purl.org/net/dajobe/">
        </rdf:Description>
      </ex:homePage>
    </rdf:Description>
```

```
  </ex:editor>
</rdf:Description>

<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <dc:title>RDF/XML Syntax Specification (Revised)</dc:title>
</rdf:Description>

</rdf:RDF>')
STRG
VARCHAR
_______________________________________________________________________________

<http://www.w3.org/TR/rdf-syntax-grammar> <http://example.org/stuff/1.0/editor> _:b1000010000 .         2
<http://www.w3.org/TR/rdf-syntax-grammar> <http://example.org/stuff/1.0/editor> _:b1000010001 .         2
<http://www.w3.org/TR/rdf-syntax-grammar> <http://purl.org/dc/elements/1.1/title> "RDF/XML Syntax Specifi
_:b1000010000 <http://example.org/stuff/1.0/fullName> "Dave Beckett" .  257 182
_:b1000010001 <http://example.org/stuff/1.0/homePage> <http://purl.org/net/dajobe/> .   257 243

5 Rows. -- 0 msec.
```

See Also

DB.DBA.RDF_AUDIT_METADATA

DB.DBA.RDF_BACKUP_METADATA

DB.DBA.RDF_LOAD_RDFXML_MT

DB.DBA.TTLP

DB.DBA.TTLP_MT

DB.DBA.RDF_TRIPLES_TO_RDF_XML_TEXT

DB.DBA.RDF_64BIT_UPGRADE

DB.DBA.RDF_64BIT_UPGRADE — Alters IRI_ID columns to get IRI_ID_8.

Synopsis

```
DB.DBA.RDF_64BIT_UPGRADE (  );
```

Description

There are two data types. IRI_ID is 4-byte and IRI_ID_8 is 8-byte. Initially, IRI_ID columns are created.
DB.DBA.RDF_64BIT_UPGRADE() alters them to get IRI_ID_8.

Note that this is to hold more that 4G of distinct IRIs, number of distinct quads is not limited by 4G even without the upgrade.

The function should be called once.

It shutdowns the server at completion upgrade so it should not be called from web-application.

Then the server should be restarted manually.

The server log should be checked for diagnostics before restart.

RDF_VIEW_SYNC_TO_PHYSICAL

RDF_VIEW_SYNC_TO_PHYSICAL — This function generates data synchronization triggers and/or actual RDF quad store data from transient Linked Data views. Basically, it enables you share physical and transient Linked Data views typically generated from 3rd party ODBC/JDBC accessible data sources.

Synopsis

```
RDF_VIEW_SYNC_TO_PHYSICAL ( in vgraph varchar ,
                            in load_data int := 0 ,
                            in pgraph varchar := null ,
                            in transaction_mode int := 1 ,
                            in atomicity int := 1 ) ;
```

Description

This function create new Transient and Materialized View pairs with option (where load = 1/0) to load data or wait for updates -- from triggers at execute time. 0 for non-transaction mode, and atomicity 0/1 determines server accessiblity where 1 indicates inaccessiblity during run. Note: applicable for Virtuoso 6.0 and higher.

Parameters

vgraph
: The transient (or virutal) named graph to which synchronization triggers are to be associated, or the actual source of data used to populate a named graph in the quad store.

load_data
: Flag that determines which operations are performed:

pgraph
: Designated quad store named graph IRI.

transaction_mode
: Integer values (0, 1, 2, or 3) which determine the transaction mode in place during synchronization. This is basically a call to log_enable () with the same values.

atomicity
: Enforces atomic mode during data synchronization. This mode of operation performs a checkpoint when completed, and is the recommended behavior for handling transient Linked Data views derived from large SQL large tables.

Return Types

any

Examples

Example 24.272.

Example 1

Using the command-line iSQL utility or its variant hosted in the HTML based Conductor, and assuming named graphs denoted by the following IRIs:

1. <http://www.openlinksw.com/schemas/odbcCustomers#> -- a named graph associated with virtual (and transient) RDF statements (triples) mapped to an ODBC accessible data source e.g., a "Customers" database.
2. <urn:rdf:materialized:view:odbc:Customers> -- a named graph to be associated with RDF statements (triples) materialized from virtual triples in the designated source named graph.

```
SQL> RDF_VIEW_SYNC_TO_PHYSICAL ('http://www.openlinksw.com/schemas/odbcCustomers#', 1, 'urn:rdf:materiali

# which is equivalent to:

SQL> RDF_VIEW_SYNC_TO_PHYSICAL ('http://www.openlinksw.com/schemas/odbcCustomers#', 1, 'urn:rdf:materiali
```

Example 2

A variant of the prior example where with ACID transaction modality enabled:

```
SQL> RDF_VIEW_SYNC_TO_PHYSICAL ('http://www.openlinksw.com/schemas/odbcCustomers#', 0, 'urn:rdf:materiali
```

See Also

DB.DBA.RDF_AUDIT_METADATA()

DB.DBA.RDF_BACKUP_METADATA()

DB.DBA.RDF_LOAD_RDFXML_MT()

DB.DBA.TTLP()

DB.DBA.TTLP_MT()

DB.DBA.RDF_TRIPLES_TO_RDF_XML_TEXT()

DB.DBA.SPARQL_REXEC()

DB.DBA.SPARQL_REXEC_TO_ARRAY()

DB.DBA.SPARQL_REXEC_WITH_META()

DB.DBA.RDF_QUAD_URI()

DB.DBA.RDF_QUAD_URI_L_TYPED()

DB.DBA.RDF_TTL2HASH()

DB.DBA.RDF_CONVERT_RDFXML_TO_TTL

DB.DBA.RDF_CONVERT_RDFXML_TO_TTL — Converts rdf xml to ttl.

Synopsis

DB.DBA.RDF_CONVERT_RDFXML_TO_TTL (in *strg* varchar ,
in *base* varchar ,
inout *ttl_ses* any) ;

Description

Converts rdf xml to ttl.

Parameters

strg
: text of the rdf resource

base
: base IRI to resolve relative IRIs to absolute

ttl_ses
: ttl resource

Return Types

The return value is not specified and may be changed in future versions.

Examples

Example 24.273. Simple example

```
create procedure DB.DBA.RDF_XML_IRI_TO_TTL (inout obj any, inout ses any)
{
  declare res varchar;
  if (isiri_id (obj))
    {
      if (obj >= min_bnode_iri_id ())
        {
          if (obj >= #ib0)
            http (sprintf ('_:bb%d ', iri_id_num (obj) - iri_id_num (#ib0)), ses);
          else
            http (sprintf ('_:b%d ', iri_id_num (obj)), ses);
        }
      else
        {
          res := coalesce (id_to_iri (obj), sprintf ('_:bad_iid_%d', iri_id_num (obj)));
          http ('<', ses);
          http_escape (res, 12, ses, 1, 1);
          http ('> ', ses);
        }
    }
  else if (__tag of varchar = __tag (obj))
    {
      if ("LEFT" (obj, 9) = 'nodeID://')
        {
          http ('_:', ses);
          http (subseq (obj, 9), ses);
          http (' ', ses);
        }
      else
        {
```

```
          http ('<', ses);
          http_escape (obj, 12, ses, 1, 1);
          http ('> ', ses);
        }
    }
  else
    {
      http ('<', ses);
      http_escape (cast (obj as varchar), 12, ses, 1, 1);
      http ('> ', ses);
    }
}
;

create procedure DB.DBA.RDF_XML_OBJ_TO_TTL (
  inout o_val any, inout o_type varchar, inout o_lang varchar,
  inout ses any)
{
  declare res varchar;
  if (isiri_id (o_val))
    {
      if (o_val >= min_bnode_iri_id ())
        {
          if (o_val >= #ib0)
            http (sprintf ('_:bb%d ', iri_id_num (o_val) - iri_id_num (#ib0)), ses);
          else
            http (sprintf ('_:b%d ', iri_id_num (o_val)), ses);
        }
      else
        {
          res := coalesce (id_to_iri (o_val), sprintf ('_:bad_iid_%d', iri_id_num (o_val)));
          http ('<', ses);
          http_escape (res, 12, ses, 1, 1);
          http ('> ', ses);
        }
      return;
    }
  http ('"', ses);
  if (__tag of XML = o_val)
    http_escape (serialize_to_UTF8_xml (o_val), 11, ses, 1, 1);
  else
    http_escape (o_val, 11, ses, 1, 1);
  if (isstring (o_type))
    {
      http ('"^^<', ses);
      http_escape (o_type, 12, ses, 1, 1);
      http ('> ', ses);
    }
  else if (isstring (o_lang))
    {
      http ('"@', ses);
      http (o_lang, ses);
      http (' ', ses);
    }
  else
    http ('" ', ses);
}
;

create procedure DB.DBA.RDF_CONVERT_RDFXML_TO_TTL_EV_NEW_BLANK (inout g varchar, inout app_env any, inout
{
;
}
;

create procedure DB.DBA.RDF_CONVERT_RDFXML_TO_TTL_EV_TRIPLE (
  inout g_iid IRI_ID, inout s_uri varchar, inout p_uri varchar,
  inout o_uri varchar,
  inout app_env any )
{
  DB.DBA.RDF_XML_IRI_TO_TTL (s_uri, app_env);
  DB.DBA.RDF_XML_IRI_TO_TTL (p_uri, app_env);
  DB.DBA.RDF_XML_IRI_TO_TTL (o_uri, app_env);
```

```
  http ('.\n', app_env);
}
;

create procedure DB.DBA.RDF_CONVERT_RDFXML_TO_TTL_EV_TRIPLE_L (
  inout g_iid IRI_ID, inout s_uri varchar, inout p_uri varchar,
  inout o_val any, inout o_type varchar, inout o_lang varchar,
  inout app_env any )
{
  DB.DBA.RDF_XML_IRI_TO_TTL (s_uri, app_env);
  DB.DBA.RDF_XML_IRI_TO_TTL (p_uri, app_env);
  DB.DBA.RDF_XML_OBJ_TO_TTL (o_val, o_type, o_lang, app_env);
  http ('.\n', app_env);
}
;

create procedure DB.DBA.RDF_CONVERT_RDFXML_TO_TTL (in strg varchar, in base varchar, inout ttl_ses any)
{
  rdf_load_rdfxml (strg, 0,
    'http://example.com',
    vector (
      'DB.DBA.TTLP_EV_NEW_GRAPH',
      'DB.DBA.RDF_CONVERT_RDFXML_TO_TTL_EV_NEW_BLANK',
      'DB.DBA.TTLP_EV_GET_IID',
      'DB.DBA.RDF_CONVERT_RDFXML_TO_TTL_EV_TRIPLE',
      'DB.DBA.RDF_CONVERT_RDFXML_TO_TTL_EV_TRIPLE_L',
      '' ),
    ttl_ses,
    base );
}
;

create procedure DB.DBA.RDF_CONVERT_RDFXML_FILE_TO_TTL_FILE (in rdfxml_source_filename varchar, in base v
{
  declare in_ses, out_ses any;
  in_ses := file_to_string_output (rdfxml_source_filename);
  out_ses := string_output ();
  DB.DBA.RDF_CONVERT_RDFXML_TO_TTL (in_ses, base, out_ses);
  string_to_file (ttl_target_filename, out_ses, -2);
}
;

-- Sample call:
DB.DBA.RDF_CONVERT_RDFXML_FILE_TO_TTL_FILE ('open_source_projects.rdf', 'http://example.com', 'open_sourc
```

See Also

DB.DBA.RDF_AUDIT_METADATA

DB.DBA.RDF_BACKUP_METADATA

DB.DBA.RDF_LOAD_RDFXML

DB.DBA.RDF_LOAD_RDFXML_MT

DB.DBA.TTLP

DB.DBA.RDF_GRAPH_GROUP_CREATE

DB.DBA.RDF_GRAPH_GROUP_CREATE — Creates graph group.

Synopsis

DB.DBA.RDF_GRAPH_GROUP_CREATE (in *group_name* varchar ,
in *is_silent* integer) ;

Description

Creates graph group.

Parameters

group_name
: The name of the graph group

is_silent
: 0 (default) or 1. When set to 0, and there is already group with the given name, error will be raised. When set to 1, there will be no error message.

Return Types

The return value is not specified and may be changed in future versions.

Examples

Example 24.274. Simple example

```
SQL>DB.DBA.RDF_GRAPH_GROUP_CREATE ('TestGroup',1);
Done. -- 50 msec.
SQL> select * from DB.DBA.RDF_GRAPH_GROUP ;
RGG_IID           RGG_IRI            RGG_MEMBER_PATTERN  RGG_COMMENT
VARCHAR NOT NULL  VARCHAR NOT NULL   VARCHAR             VARCHAR
_______________________________________________________________________________

#i1020000         TestGroup          NULL                NULL

1 Rows. -- 10 msec.
```

See Also

DB.DBA.RDF_GRAPH_GROUP_INS

DB.DBA.RDF_GRAPH_GROUP_INS

DB.DBA.RDF_GRAPH_GROUP_INS — Inserts graph into group.

Synopsis

DB.DBA.RDF_GRAPH_GROUP_INS (in *group_name* varchar ,
in *graph_uri* varchar) ;

Description

Inserts graph into a given group.

Parameters

group_name
: The name of the graph group

graph_uri
: graph uri IRI string or IRI_ID

Return Types

The return value is not specified and may be changed in future versions.

Examples

Example 24.275. Simple example

```
SQL>DB.DBA.RDF_GRAPH_GROUP_CREATE ('TestGroup',1);
Done. -- 50 msec.
SQL>DB.DBA.RDF_GRAPH_GROUP_INS ('TestGroup','http://myopenlink.net');
Done. -- 50 msec.
SQL>DB.DBA.RDF_GRAPH_GROUP_INS ('TestGroup','http://demo.openlinksw.com');
Done. -- 50 msec.
```

See Also

DB.DBA.RDF_GRAPH_GROUP_CREATE

DB.DBA.RDF_GRAPH_USER_PERMS_SET

DB.DBA.RDF_GRAPH_USER_PERMS_SET — Sets permissions of an the user on a graph.

Synopsis

DB.DBA.RDF_GRAPH_USER_PERMS_SET (in *graph_iri* varchar ,
in *uname* varchar ,
in *perms* integer) ;

Description

sets permissions of the user named uname on graph specified by graph_iri to the value specified by perms.

Parameters

graph_iri
: Graph IRI.

uname
: User name.

perms
: If perms is null then the function acts as DB.DBA.RDF_GRAPH_USER_PERMS_DEL (graph_iri, uname);

Examples

Example 24.276. Example

See Example for Blogs and Resource Sharing

See Also

DB.DBA.RDF_AUDIT_METADATA

DB.DBA.RDF_GRAPH_USER_PERMS_GET

DB.DBA.RDF_GRAPH_SECURITY_AUDIT

DB.DBA.RDF_DEFAULT_USER_PERMS_SET

DB.DBA.RDF_DEFAULT_USER_PERMS_DEL

DB.DBA.RDF_GRAPH_USER_PERMS_DEL

DB.DBA.RDF_ALL_USER_PERMS_DEL

DB.DBA.RDF_GRAPH_GROUP_CREATE

DB.DBA.RDF_GRAPH_GROUP_INS

DB.DBA.RDF_GRAPH_USER_PERMS_GET

DB.DBA.RDF_GRAPH_USER_PERMS_GET — Gets permissions of an the user on a graph.

Synopsis

DB.DBA.RDF_GRAPH_USER_PERMS_GET (in *graph_iri* varchar ,
in *uname* varchar) ;

Description

Gets permissions of the user named uname on graph specified by graph_iri.

Parameters

graph_iri
: Graph IRI.

uname
: User name.

Return Types

Returns integer or null.

If the returned value is not null (hence an integer) then the specified user should be an active SQL user or "nobody".

Four sorts of access are returned, specified by four bits of an integer "permission bit-mask", plain old UNIX style:

- Bit 1 permits read access.
- Bit 2 permits write access via SPARUL and it's basically useless without bit 1 set.
- Bit 4 permits write access via "RDF sponge" methods and it's basically useless without bits 1 and 2 set.
- Bit 8 allows to obtain list of members of graph group; an IRI can be used as graph IRI and as graph group IRI at the same time so bit 8 can be freely combined with any of bits 1, 2 or 4.

For more details see our documentation section "Understanding Default Permissions"

Examples

Example 24.277. Example

Suppose the following Example for Blogs and Resource Sharing .

Get Ana's permissions on a given graph:

```
SQL> select DB.DBA.RDF_GRAPH_USER_PERMS_GET ('http://example.com/Anna/system', 'Anna');
Query result:
RDF_GRAPH_USER_PERMS_GET
INTEGER
 1
No. of rows in result: 1
```

See Also

DB.DBA.RDF_AUDIT_METADATA

DB.DBA.RDF_GRAPH_USER_PERMS_SET

DB.DBA.RDF_GRAPH_SECURITY_AUDIT

DB.DBA.RDF_DEFAULT_USER_PERMS_SET

DB.DBA.RDF_DEFAULT_USER_PERMS_DEL

DB.DBA.RDF_GRAPH_USER_PERMS_DEL

DB.DBA.RDF_ALL_USER_PERMS_DEL

DB.DBA.RDF_GRAPH_GROUP_CREATE

DB.DBA.RDF_GRAPH_GROUP_INS

DB.DBA.RDF_GRAPH_USER_PERMS_DEL

DB.DBA.RDF_GRAPH_USER_PERMS_DEL — Removes the setting of permissions of an user named uname on a given graph.

Synopsis

DB.DBA.RDF_GRAPH_USER_PERMS_DEL (in *graph_iri* varchar ,
in *uname* varchar) ;

Description

This function removes the setting of permissions of user named uname on graph specified by graph_iri. As the result, the permission of "nobody",is used for the specified user, the future changes in permissions of "nobody" will change the access control for the user on the graph in question.

The function works for disabled or DAV users as well as for active SQL users, so a user can be banned first and then restricted in RDF access right or visa versa.

Unlike DB.DBA.RDF_DEFAULT_USER_PERMS_DEL , the procedure will not signal an error if called for "nobody" or "dba" users.

Parameters

graph_iri
: Graph IRI.

uname
: User name.

Examples

Example 24.278. Example

See Example for Blogs and Resource Sharing

See Also

DB.DBA.RDF_AUDIT_METADATA

DB.DBA.RDF_GRAPH_SECURITY_AUDIT

DB.DBA.RDF_DEFAULT_USER_PERMS_SET

DB.DBA.RDF_DEFAULT_USER_PERMS_DEL

DB.DBA.RDF_GRAPH_USER_PERMS_SET

DB.DBA.RDF_ALL_USER_PERMS_DEL

DB.DBA.RDF_GRAPH_GROUP_CREATE

DB.DBA.RDF_GRAPH_GROUP_INS

DB.DBA.RDF_ALL_USER_PERMS_DEL

DB.DBA.RDF_ALL_USER_PERMS_DEL — Removes all the setting of permissions of an user, both default permissions and permissions on specific graphs.

Synopsis

```
DB.DBA.RDF_ALL_USER_PERMS_DEL ( in uname varchar ,
                                in uid integer := null ) ;
```

Description

This function removes all the setting of permissions of user named uname, both default permissions and permissions on specific graphs. The only exception is that it does not remove default permissions if the specified user is "nobody" or "dba", but all graph-specific permissions are always removed.

The procedure can deal with disabled users. Moreover it can be used with entirely deleted users as soon as user id is known. It is possible to pass NULL instead of user name and an integer user ID as a second argument.

Parameters

uname
: User name

uid
: The user U_ID in the system table DB.DBA.SYS_USERS.

Examples

Example 24.279. Example

See Example for Blogs and Resource Sharing

See Also

DB.DBA.RDF_AUDIT_METADATA

DB.DBA.RDF_GRAPH_SECURITY_AUDIT

DB.DBA.RDF_DEFAULT_USER_PERMS_SET

DB.DBA.RDF_DEFAULT_USER_PERMS_DEL

DB.DBA.RDF_GRAPH_USER_PERMS_SET

DB.DBA.RDF_GRAPH_USER_PERMS_DEL

DB.DBA.RDF_GRAPH_GROUP_CREATE

DB.DBA.RDF_GRAPH_GROUP_INS

rdfs_rule_set

rdfs_rule_set — Adds the applicable facts of the graph into a rule set.

Synopsis

rdfs_rule_set (in *rule_name* varchar ,
in *graph_iri* varchar ,
in *remove* integer) ;

Description

This function associates a rule-name with a source named graphs identified by a named graph IRI. This named graph is contains RDF statements that constitute the desired inference rule to be created. You can use this procedure to add RDF statements from other named graphs as part as part of a rule-enhancement effort.

Parameters

rule_name
: Rule name (literal that identifies the rule).

graph_iri
: A named graph IRI that identifies the source of RDF statements to be used as inference rules,

remove
: (0|1 default 0) - where 1 indicates that the specified graph is to be removed from the rule set.

Examples

Example 24.280. Simple Usage Example

Google and Bloomberg Stock Quote Services return HTML5+Microdata based Linked Open Data using terms from Schema.org. As will be quite common, there is a case sensitive issue in regards to the HTTP URIs that identify the "Financial Quote" class. For Bloomberg we have: <http://schema.org/intangible/financialquote> and for Google: <http://schema.org/Intangible/FinancialQuote> . What follows in an *owl:equivalentProperty* relation that enables a processor of this data (e.g., SPARQL Query Processor) apply reasoning and inference based on the semantics of the *owl:equivalentProperty* relation. Using Virtuoso's SPARQL processor, you can create an inference rule declaratively via `rdfs_rule_set()` :

```
-- Suppose a ttl file inference-rule-for-stock-quotes.ttl has the following content:
<http://schema.org/intangible/financialquote>
  owl:equivalentProperty <http://schema.org/Intangible/FinancialQuote> .

-- Load the data to a Named Graph that's implicitly (in the case of Virtuoso)
-- identified by the RDF Document URL that contains the RDF statements that
-- constitute the rule being created
SQL> LOAD <http://example.com/inference-rule-for-stock-quotes.ttl> ;
Done. -- 15 msec.

-- Via SQL run the procedure:
SQL> RDFS_RULE_SET ('schema_stock_quotes', 'http://example.com/inference-rule-for-stock-quotes.ttl');
```

Example 24.281. Create Rule example

Create rule set and verify rule's existence:

```
SQL> rdfs_rule_set('myset', 'http://mygraph.com');
Done. -- 15 msec.
SQL> SELECT * FROM DB.DBA.SYS_RDF_SCHEMA ;
RS_NAME                              RS_URI                                                                RS_G
```

```
VARCHAR NOT NULL                    VARCHAR NOT NULL                                    VARCHAR
_______________________________________________________________________________

http://example.com/dataspace        http://example.com/dataspace/inf                    NULL
http://www.w3.org/2002/07/owl#      http://www.w3.org/2002/07/owl#                      NULL
myset                               http://mygraph.com                                  NULL
virtrdf-ifp                         virtrdf-ifp                                                 NULL
virtrdf-label                       virtrdf-label                                               NULL
virtrdf-meta-entity-class           http://www.openlinksw.com/schemas/virtrdf-meta-entity-class NULL

7 Rows. -- 0 msec.
```

See Also

DB.DBA.RDF_AUDIT_METADATA()

DB.DBA.RDF_BACKUP_METADATA()

DB.DBA.RDF_LOAD_RDFXML_MT()

DB.DBA.TTLP()

DB.DBA.TTLP_MT()

DB.DBA.RDF_TRIPLES_TO_RDF_XML_TEXT()

DB.DBA.SPARQL_REXEC()

DB.DBA.SPARQL_REXEC_TO_ARRAY()

DB.DBA.SPARQL_REXEC_WITH_META()

DB.DBA.RDF_QUAD_URI_L()

DB.DBA.RDF_QUAD_URI_L_TYPED()

DB.DBA.RDF_TTL2HASH()

DB.DBA.RDF_GEO_FILL

DB.DBA.RDF_GEO_FILL — Converts geo:lat and geo:long properties properties into geometries.

Synopsis

DB.DBA.RDF_GEO_FILL (*n_threads* integer (optional, default 4)) ;

Description

Converts geo:lat and geo:long properties into geometries. This operation reads through all graphs and for each subject with at least one geo:lat and geo:long, a point geometry is made for each distinct lat/long pair where lat and long are in the same graph.

It should not happen in practice that a single subject has multiple lats or long within one graph. If this still happens, a geometry is made for each combination. The geometry is added to the subject with the lat and long as the value of the geo:geometry property.

This is performed in parallel on multiple threads and is optimized for cluster execution and done without transaction logging and is not transactional. To make the result persistent, the operator should do an explicit checkpoint by executing:

```
SQL>cl_exec ('checkpoint');
```

on any process of a cluster or single server. Otherwise the result may be lost if the server terminates abnormally before an automatic checkpoint is made.

Parameters

n_threads
: Number of threads to run. Default is to run 4 threads that will make geomerty objects (not counting the thread that searches for separate geo:lat and geo:long properties).

Examples

Example 24.282. Simple example

```
SQL> rdf_geo_fill ();

Done. -- 282 msec.
```

See Also

Creating Geometries From RDF Data

Programmatic Manipulation of Geometries in RDF

Using Geometries With Existing Databases

`st_point`

`st_x`

`st_y`

`st_distance`

`st_srid`

`st_setsrid`

st_astext

st_geomfromtext

st_intersects

st_contains

st_within

isgeometry

geo_insert

geo_delete

DB.DBA.RDF_GEO_ADD

DB.DBA.RDF_GEO_ADD

DB.DBA.RDF_GEO_ADD — Translates a geometry into a RDF box

Synopsis

DB.DBA.RDF_GEO_ADD (in *g* any);

Description

This function translates a geometry into a RDF box. It looks if an identical geometry already exists and if so assigns the existing id to it. If the geometry is new, it gets a new ID and is stored in the RDF literals table RDF_OBJ. At this time it is also automatically inserted into the RDF geometry index.

Note: In a cluster situation one should use the dpipe mechanism for inserting into RDF quad so as to get large numbers of inserts into a single message. This is essential for performance.

Parameters

g
: Geometry to be translated into RDF box.

Examples

Example 24.283. Simple example

```
INSERT INTO RDF_QUAD (g, s, p, o)
VALUES (
         "g",
         "s",
         iri_to_id ('http://www.w3.org/2003/01/geo/wgs84_pos#geometry'),
         DB.DBA.rdf_geo_add (rdf_box (st_point (lng, lat), 256, 257, 0, 1)));
```

See Also

Creating Geometries From RDF Data

Programmatic Manipulation of Geometries in RDF

Using Geometries With Existing Databases

`st_point`

`st_x`

`st_y`

`st_distance`

`st_srid`

`st_setsrid`

`st_astext`

`st_geomfromtext`

`st_intersects`

st_contains

st_within

isgeometry

geo_insert

geo_delete

DB.DBA.RDF_GEO_FILL

DB.DBA.RDF_GRAPH_COLLECT_FP_LIST

DB.DBA.RDF_GRAPH_COLLECT_FP_LIST — Gets inverse / functional properties for given graphs.

Synopsis

```
DB.DBA.RDF_GRAPH_COLLECT_FP_LIST ( in g_iid_list any ,
                                   in direct_fp integer ,
                                   inout fp_list any ) ;
```

Description

Gets inverse / functional properties for given graphs. The procedure will fetch all triples of all specified graphs during the run, so it will take some time if graphs are big and/or numerous.

Parameters

g_iid_list
: Vector of IRI_IDs of sample graphs

direct_fp
: When zero gets list of inverse functional properties; non-zero gets functional properties.

fp_list
: Variable that will be set to the vector of properties.

Return Types

any

See Also

Fast Approximate RDF Graph Diff and Patch

DB.DBA.RDF_GRAPH_DIFF

DB.DBA.RDF_SUO_DIFF_TTL

DB.DBA.RDF_SUO_APPLY_PATCH

DB.DBA.RDF_GRAPH_DIFF

DB.DBA.RDF_GRAPH_DIFF — performs the core operation of preparing graph diff.

Synopsis

```
DB.DBA.RDF_GRAPH_DIFF ( in gfrom IRI_ID ,
                        in gto IRI_ID ,
                        in only_gfrom any ,
                        in only_gto any ,
                        in accuracy integer ,
                        inout fp_list any ,
                        inout invfp_list any ,
                        inout gfrom_rules any ,
                        inout gto_rules any ,
                        in equiv_map any := null ,
                        in equiv_rev any := null ) ;
```

Description

Performs the core operation of preparing graph diff - matching between nodes of two graphs.

Typically is called from wrapper that returns the diff in an extended GUO ontology (http://webr3.org/owl/guo#) as a string output in TURTLE syntax.

Parameters

gfrom
: IRI ID of the from graph

gto
: IRI ID of the destination graph

only_gfrom
: Dictionaries of triples that present only in gfrom (but have no matches in gto)

only_gto
: Dictionaries of triples that present only in gto (but have no matches in gfrom)

accuracy
: Tells the procedure what should be done if one or both graphs contain blank nodes. If zero then different bnodes are treated as distinct. Bnodes are allocated individually for each data load, so even if one and the same resource is loaded in two different graphs then these graphs will contain no blank nodes in common, accuracy=0 would result in the diff containing all triples from both graphs that have bnode subject o object. If nonzero then the procedure will try to match blank nodes, zipper style. In this case rest of arguments comes to play.

fp_list
: Vector of IRI_IDs of predicates that are supposed to be functional properties, i.e., any given subject has no more than one value of a given predicate.

invfp_list
: Vector of IRI_IDs of (supposedly) inverse functional properties, i.e., any given object could be the value of a given predicate for no more than one subject. The best source of such vectors is a good ontology suitable for both graphs. Types owl:functionalProperty and owl:inverseFunctionalProperty will provide truly "meaningful" lists of predicates that will provide really "meaningful" patch. Meaningful patches have three important advantages:

fp_list
: If NULL but accuracy is not zero then the function will call DB.DBA.RDF_GRAPH_COLLECT_FP_LIST internally, using graphs gfrom and gto as specimens.

invfp_list
: If NULL but accuracy is not zero then the function will call DB.DBA.RDF_GRAPH_COLLECT_FP_LIST internally, using graphs gfrom and gto as specimens.

gfrom_rules
: Describes how bnodes of graphs were matched, i.e., how sliders were moved along chains of predicates.

gto_rules
: Describes how bnodes of graphs were matched, i.e., how sliders were moved along chains of predicates.

equiv_map
: Dictionary that describe the match between bnodes in gfrom and gto. In equiv_map, bnodes from gfrom are keys and matching bnodes from gto are values.

equiv_rev
: Dictionary that describes the match between bnodes in gfrom and gto. In equiv_rev, same pairs are in different roles, bnodes from gfrom are values for matching bnodes from gto that are now keys. These data can be incomplete or NULL at all, but if provided then the function will trust them blindly.

Return Types

any

See Also

Fast Approximate RDF Graph Diff and Patch

DB.DBA.RDF_SUO_DIFF_TTL

DB.DBA.RDF_SUO_APPLY_PATCH

DB.DBA.RDF_GRAPH_COLLECT_FP_LIST

DB.DBA.RDF_SUO_APPLY_PATCH

DB.DBA.RDF_SUO_APPLY_PATCH — Modifies either the target graph specified in the patch via `guo:target_graph` or the graph specified by default_target_graph argument.

Synopsis

DB.DBA.RDF_SUO_APPLY_PATCH (in *patch_graph* IRI_ID ,
in *default_target_graph* IRI_ID) ;

Description

This function tries to modify either the target graph specified in the patch via `guo:target_graph` (hence, the non-NULL target_graph argument of the diff creator, DB.DBA.RDF_SUO_DIFF_TTL()) or the graph specified by default_target_graph argument. The function returns zero in case of success, non-zero value otherwise. In addition, the function produces a result set that describes the process of matching blank node in the target graph to instructions in the patch.

The TURTLE made by DB.DBA.RDF_GRAPH_DIFF can be loaded later into an empty graph and the IRI_ID of that graph can be passed as the first argument to this patch function.

Parameters

patch_graph
: IRI ID of the patch graph

default_target_graph
: IRI ID of the default target graph

Return Types

integer

See Also

Fast Approximate RDF Graph Diff and Patch

`DB.DBA.RDF_GRAPH_DIFF()`

`DB.DBA.RDF_SUO_DIFF_TTL()`

`DB.DBA.RDF_GRAPH_COLLECT_FP_LIST()`

DB.DBA.RDF_SUO_DIFF_TTL

DB.DBA.RDF_SUO_DIFF_TTL — performs the core operation of preparing graph diff.

Synopsis

```
DB.DBA.RDF_SUO_DIFF_TTL ( in gfrom any ,
                          in gto any ,
                          in target_graph any ,
                          in accuracy integer ,
                          inout fp_list any ,
                          inout invfp_list any ,
                          in equiv_map any := null ,
                          in equiv_rev any := null ) ;
```

Description

Performs the core operation of preparing graph diff - matching between nodes of two graphs.

Typically is called from wrapper that returns the diff in an extended GUO ontology (http://webr3.org/owl/guo#) as a string output in TURTLE syntax.

Parameters

gfrom
: varchar or UNAME or IRI_ID of the from graph

gto
: varchar or UNAME or IRI_ID of the destination graph

accuracy
: Tells the procedure what should be done if one or both graphs contain blank nodes. If zero then different bnodes are treated as distinct. Bnodes are allocated individually for each data load, so even if one and the same resource is loaded in two different graphs then these graphs will contain no blank nodes in common, accuracy=0 would result in the diff containing all triples from both graphs that have bnode subject o object. If nonzero then the procedure will try to match blank nodes, zipper style. In this case rest of arguments comes to play.

target_graph
: It is used in to the output TURTLE as value of guo:target_graph property of every guo:UpdateInstruction node, NULL value means no this property at all. If the patch procedure will not get the IRI of the graph to patch as an argument then the value of this property will be used.

fp_list
: Vector of IRI_IDs of predicates that are supposed to be functional properties, i.e. any given subject has no more than one value of a given predicate.

invfp_list
: Vector of IRI_IDs of (supposedly) inverse functional properties, i.e., any given object could be the value of a given predicate for no more than one subject. The best source of such vectors is a good ontology suitable for both graphs. Types owl:functionalProperty and owl:inverseFunctionalProperty will provide truly "meaningful" lists of predicates that will provide really "meaningful" patch. Meaningful patches have three important advantages:

fp_list
: If NULL but accuracy is not zero then the function will call DB.DBA.RDF_GRAPH_COLLECT_FP_LIST internally, using graphs gfrom and gto as specimens.

invfp_list

If NULL but accuracy is not zero then the function will call DB.DBA.RDF_GRAPH_COLLECT_FP_LIST internally, using graphs gfrom and gto as specimens.

gfrom_rules
: Describes how bnodes of graphs were matched, i.e., how sliders were moved along chains of predicates.

gto_rules
: Describes how bnodes of graphs were matched, i.e., how sliders were moved along chains of predicates.

equiv_map
: Dictionary that describe the match between bnodes in gfrom and gto. In equiv_map, bnodes from gfrom are keys and matching bnodes from gto are values.

equiv_rev
: Dictionary that describes the match between bnodes in gfrom and gto. In equiv_rev, same pairs are in different roles, bnodes from gfrom are values for mathing bnodes from gto that are now keys. These data can be incomplete or NULL at all, but if provided then the function will trust them blindly.

See Also

Fast Approximate RDF Graph Diff and Patch

`DB.DBA.RDF_GRAPH_DIFF()`

`DB.DBA.RDF_SUO_APPLY_PATCH()`

`DB.DBA.RDF_GRAPH_COLLECT_FP_LIST()`

DB.DBA.SPARQL_RDB2RDF_CODEGEN

DB.DBA.SPARQL_RDB2RDF_CODEGEN — Creates an SQL text for a given table and an operation specified by an opcode

Synopsis

```
DB.DBA.SPARQL_RDB2RDF_CODEGEN ( in table_name varchar ,
                                in opcode integer ) ;
```

Description

The function creates an SQL text for a given table and an operation specified by an opcode.

Parameters

table_name
: The name of the table.

opcode
: Currently supported opcodes are:

Return Types

string session or vector

See Also

RDB2RDF Triggers

DB.DBA.SPARQL_RDB2RDF_LIST_TABLES

DB.DBA.SPARQL_RDB2RDF_LIST_TABLES

DB.DBA.SPARQL_RDB2RDF_LIST_TABLES — Returns a vector of names of tables that are used as sources for Linked Data Views.

Synopsis

DB.DBA.SPARQL_RDB2RDF_LIST_TABLES (in *mode* integer) ;

Description

Returns a vector of names of tables that are used as sources for Linked Data Views. Application developer should decide what to do with each of them - create triggers or do some application-specific workarounds.

Note that if some SQL views are used as sources for Linked Data Views and these views does not have INSTEAD triggers then workarounds become mandatory for them, not just a choice, because BEFORE or AFTER triggers on views are not allowed if there is no appropriate INSTEAD trigger. The mode argument should be zero in current version.

Parameters

mode

Return Types

vector

See Also

RDB2RDF Triggers

`DB.DBA.SPARQL_RDB2RDF_CODEGEN()`

DB.DBA.SPARQL_SELECT_KNOWN_GRAPHS

DB.DBA.SPARQL_SELECT_KNOWN_GRAPHS — returns all explicitly created graphs.

Synopsis

DB.DBA.SPARQL_SELECT_KNOWN_GRAPHS ( );

Description

This function returns URIs for all explicitly created graphs.

Parameters

Return Types

varchar

Examples

Example 24.284. Simple Use

```
# Create explicitly a graph:
CREATE GRAPH <http://example.com/mygraph>;
callret-0
Create graph <http://example.com/mygraph> -- done
...
SQL> DB.DBA.SPARQL_SELECT_KNOWN_GRAPHS();
GRAPH_IRI
VARCHAR
_______________________________________________________________________________

http://www.openlinksw.com/schemas/virtrdf#
http://example.com/mygraph
http://example.com/DAV
http://example.com/dataspace
http://example.com/dataspace/inf
http://www.w3.org/2002/07/owl#
http://www.w3.org/2002/07/owl
http://www.openlinksw.com/schemas/RDF_Mapper_Ontology/1.0/
http://bblfish.net/work/atom-owl/2006-06-06/
http://purl.org/dc/elements/1.1/
http://www.openlinksw.com/schema/attribution#
http://purl.org/ontology/bibo/
http://purl.org/dc/terms/
http://xmlns.com/foaf/0.1/
http://purl.org/goodrelations/v1
http://open.vocab.org/terms
http://www.w3.org/1999/02/22-rdf-syntax-ns#
http://www.w3.org/2000/01/rdf-schema#
http://scot-project.org/scot/ns#
http://rdfs.org/sioc/ns#

20 Rows. -- 188 msec.
```

See Also

Fast Approximate RDF Graph Diff and Patch

DB.DBA.RDF_SUO_DIFF_TTL

DB.DBA.RDF_SUO_APPLY_PATCH

DB.DBA.RDF_GRAPH_COLLECT_FP_LIST

rowvector_digit_sort

rowvector_digit_sort — Performs a stable "digit" sort of a given array of arrays.

Synopsis

rowvector_digit_sort (inout *data* vector ,
in *key_idx_in_row* integer ,
in *sort_ascending* integer) ;

Description

The function gets an vector that contains uniform arrays ("rows") as items, such as a result set produced by `exec` (). One element of each row is a key of sorting; it is identified by its zero-based position within the row, *`key_idx_in_row`* (if the *`data`* vector is a result set then N-th elements of all rows are from N-th column of the result set). The sorting procedure edits *`data`* and reorders rows in such a way that their keys become ordered ascending or descending, depending on *`sort_ascending`* flag.

The performed sorting is stable. It means that it will not permutate rows in vain: it will preserve the relative order of any two rows that have equal keys. Using this property, one may sort a result set by a "secondary sorting columns", starting with less significant and then by "primary sorting column". E.g., if each row contain elements for country code and province code then it is possible to make two-column sorting by sorting first by province and then by country; that will work even if province codes are not globally unique (say, if they're enumerated from 1 in each country).

This function supports only integer values of sorting keys.

Parameters

data
: A vector of rows to sort.

key_idx_in_row
: Zero-based position of key element in row, should be nonnegative and less than the length of each row.

sort_ascending
: Direction of sorting, nonzero for ascending sort, zero for descending.

Return Types

The function returns the length of the *`data`* vector.

See Also

`gvector_digit_sort()`

`gvector_sort()`

IsRef

IsRef — Returns true if non-blank IRI

Synopsis

IsRef (in *iri* any) ;

Description

Returns true if non-blank IRI

Parameters

iri
: IRI resource

Return Types

Returns 1 if the IRI is non-blank IRI. Otherwise returns 0.

Examples

Example 24.285. Simple example

```
SQL>SPARQL
SELECT COUNT(DISTINCT (?o))
WHERE
  {
    ?s ?p ?o .
    FILTER ( IsRef(?o))
  }
LIMIT 10

callret-0
76027
```

See Also

DB.DBA.RDF_AUDIT_METADATA

DB.DBA.RDF_BACKUP_METADATA

DB.DBA.RDF_LOAD_RDFXML

DB.DBA.RDF_LOAD_RDFXML_MT

DB.DBA.SAMPLE

DB.DBA.SAMPLE — returns an arbitrary value from the multiset passed to it.

Synopsis

DB.DBA.SAMPLE (in *token* varchar);

Description

DB.DBA.SAMPLE is aggregate function that returns an arbitrary value from the multiset passed to it.

Note: Using the "sql:" prefix is mandatory for this aggregate.

Parameters

strg
: An item from multiset.

Return Types

any

Examples

Example 24.286.

```
SQL>SPARQL
SELECT  (sql:SAMPLE(?nick)), (sql:SAMPLE(?interest))
WHERE
  {
    ?p a foaf:Person .
    ?p foaf:nick ?nick.
    ?p foaf:interest ?interest.
  }

callret-0   callret-1
VARCHAR           VARCHAR
_______________________________________________
dr. Jones         http://purl.org/rss/1.0/

No. of rows in result: 1
```

See Also

`DB.DBA.GROUP_CONCAT_DISTINCT()`

`DB.DBA.GROUP_CONCAT()`

`DB.DBA.GROUP_DIGEST()`

Differences between DB.DBA.SAMPLE, DB.DBA.GROUP_CONCAT and DB.DBA.GROUP_DIGEST functions usage.

DB.DBA.GROUP_CONCAT

DB.DBA.GROUP_CONCAT — returns an arbitrary value from the multiset passed to it.

Synopsis

DB.DBA.GROUP_CONCAT (in *token* varchar ,
in *delim* varchar) ;

Description

DB.DBA.GROUP_CONCAT is aggregate function that performs a string concatenation across the values of an expression with a group. The order of the strings is not specified. The separator character used in the concatenation may be given with the scalar argument delimiter.

Note: the "sql:" prefix is mandatory when this aggregate is used in SPARQL queries. In SQL queries the prefix is "DB.DBA".

Parameters

token
: An item that should be added to a delimited list.

delim
: The delimeter character to be used in the concatenation.

Return Types

any

Examples

Example 24.287. Simple Use

```
SQL>SPARQL
SELECT ?name, (sql:GROUP_CONCAT(?near, ' , '))
WHERE
  {
    GRAPH ?g
      {
        [] a foaf:Person ;
        foaf:name ?name ;
        foaf:based_near ?near
      }
  }
LIMIT 10

name callret-1
ANY  ANY
_____________________________________________________
Jonas Smedegaard         nodeID://b6190457 , nodeID://b6190507
Adam Harvey            nodeID://b780751
John Breslin         nodeID://b56694
John Breslin         nodeID://b56694
Chris Bizer            http://sws.geonames.org/2950159/ , http://dbpedia.org/resource/Berlin
Leo Sauermann        nodeID://b53598
Andreas Harth        http://dbpedia.org/resource/Karlsruhe , nodeID://b53559 , nodeID://b53569
Alexandre Passant        http://dbpedia.org/resource/Galway
Leon Lord                http://ods-qa.openlinksw.com/dataspace/person/t1#based_near
Leon Smith         http://ods-qa.openlinksw.com/dataspace/person/t2#based_near
No. of rows in result: 10
```

See Also

`DB.DBA.SAMPLE()`

`DB.DBA.GROUP_CONCAT_DISTINCT()`

`DB.DBA.GROUP_DIGEST()`

Differences between DB.DBA.SAMPLE, DB.DBA.GROUP_CONCAT and DB.DBA.GROUP_DIGEST functions usage.

DB.DBA.GROUP_CONCAT_DISTINCT

DB.DBA.GROUP_CONCAT_DISTINCT — returns a string that is concatenation of token strings delimited with delimiter

Synopsis

DB.DBA.GROUP_CONCAT_DISTINCT (in *token* varchar ,
in *delim* varchar) ;

Description

DB.DBA.GROUP_CONCAT_DISTINCT is aggregate function that performs a string concatenation across the values of an expression with a group. The order of the strings is not specified. The separator character used in the concatenation may be given with the scalar argument delimiter.

Note: the "sql:" prefix is mandatory when this aggregate is used in SPARQL queries. In SQL queries the prefix is "DB.DBA".

Parameters

token
: An item that should be added to a delimited list.

delim
: The delimeter character to be used in the concatenation.

Return Types

any

Examples

Example 24.288. Simple Use

```
SQL>SPARQL
SELECT ?name, (sql:GROUP_CONCAT_DISTINCT(?near, ' , '))
WHERE
  {
    GRAPH ?g
      {
        [] a foaf:Person ;
        foaf:name ?name ;
        foaf:based_near ?near
      }
  }
LIMIT 2

name callret-1
ANY  ANY
_____________________________________________________
Donny Hathaway   http://linkeddata.uriburner.com/about/id/entity/http/sws.geonames.org/3996063/ , http://
Jill Scott           http://linkeddata.uriburner.com/about/id/entity/http/sws.geonames.org/3996034/ , htt
No. of rows in result: 2
```

See Also

DB.DBA.SAMPLE()

DB.DBA.GROUP_CONCAT()

DB.DBA.GROUP_DIGEST()

Differences between DB.DBA.SAMPLE, DB.DBA.GROUP_CONCAT, DB.DBA.GROUP_CONCAT_DISTINCT and DB.DBA.GROUP_DIGEST functions usage.

DB.DBA.GROUP_DIGEST

DB.DBA.GROUP_DIGEST — returns an arbitrary value from the multiset passed to it.

Synopsis

DB.DBA.GROUP_DIGEST (in *token* varchar ,
in *delim* varchar ,
in *maxlen* integer ,
in *mode* integer) ;

Description

DB.DBA.GROUP_DIGEST is aggregate function that performs a string concatenation across the values of an expression with a group. The order of the strings is not specified. The separator character used in the concatenation may be given with the scalar argument delimiter. The function is an extended version of sql:GROUP_CONCAT(), with two more arguments: maxlen and mode.

Note: the "sql:" prefix is mandatory when this aggregate is used in SPARQL queries. In SQL queries the prefix is "DB.DBA".

Parameters

token
: An item that should be added to a delimited list.

delim
: The delimeter character to be used in the concatenation.

maxlen
: The maximal allowed length of the result, in characters. Redundant values will be ignored. If the last value does not fit in the "maxlen" entirely, then it can be truncated and "..." is placed at the end of the resulting string.

mode
: Bitmask of properties. Right now only bit 1 is used and others are reserved. If the value of the argument is 1 then duplicate values are ignored; value 0 will put duplicate values like in case of sql:GROUP_CONCAT ().

Return Types

any

Examples

Example 24.289. Get all ?assets as a list with delimiters.

```
SQL>SPARQL
SELECT ?view (sql:GROUP_DIGEST (?path, ' ', 1000, 1)) as ?path_list
    (sql:GROUP_DIGEST (?asset, ' ', 1000, 1)) as ?asset_list
FROM <http://mygraph.com>
WHERE
  {
    ?view <viewPath> ?path ;
      <viewContent> ?asset ;
      <viewType>  'phyview'.
  };
```

See Also

`DB.DBA.SAMPLE`

`DB.DBA.GROUP_CONCAT`

`DB.DBA.GROUP_CONCAT_DISTINCT`

Differences between DB.DBA.SAMPLE, DB.DBA.GROUP_CONCAT and DB.DBA.GROUP_DIGEST functions usage.

http_nt_triple

http_nt_triple — outputs next triple to ses in NT serialization.

Synopsis

http_nt_triple (in *env* any ,
in *arg1* any ,
in *arg2* any ,
in *arg3* any ,
in *ses* any) ;

Description

Outputs next triple to ses in NT serialization.

Parameters

env
: An array of special format:

arg1
: Triple subject.

arg2
: Triple predicate.

arg3
: Triple object.

ses
: Session output.

Return Types

Any

Examples

Example 24.290. Simple Use

```
create procedure dump_rdftriples_to_nt(inout triples any, inout ses any)
{
  declare env any;
  declare tcount, tctr integer;
  tcount := length (triples);
  if (0 = tcount)
    {
      http ('# Empty NT\n', ses);
      return;
    }
  env := vector (0, 0, 0);
  for (tctr := 0; tctr < tcount; tctr := tctr + 1)
    {
      http_nt_triple (env, triples[tctr][0], triples[tctr][1], triples[tctr][2], ses);
    }
}
;
```

See Also

`http_ttl_triple()`

`DB.DBA.RDF_AUDIT_METADATA()`

`DB.DBA.RDF_BACKUP_METADATA()`

`DB.DBA.RDF_LOAD_RDFXML()`

`DB.DBA.RDF_LOAD_RDFXML_MT()`

`TTLP_MT()`

`TTLP_MT_LOCAL_FILE()`

http_ttl_triple

http_ttl_triple — outputs next triple to ses in TTL serialization.

Synopsis

http_ttl_triple (in *env* any ,
in *arg1* any ,
in *arg2* any ,
in *arg3* any ,
in *ses* any) ;

Description

Outputs next triple to ses in TTL serialization. This function does not terminate the printed triple in hope that the next triple will has same "s" or pair of "s" and "p". So "intermediate" semicolon or comma can be used instead of "trailing" dot.

Parameters

env
: An array of special format:

arg1
: Triple subject.

arg2
: Triple predicate.

arg3
: Triple object.

ses
: Session output.

Return Types

Any

Examples

Example 24.291. Example 1

```
CREATE PROCEDURE dump_one_graph
  ( IN  srcgraph           VARCHAR  ,
    IN  out_file           VARCHAR  ,
    IN  file_length_limit  INTEGER  := 1000000000
  )
  {
    DECLARE  file_name  varchar;
    DECLARE  env, ses      any;
    DECLARE  ses_len,
             max_ses_len,
             file_len,
             file_idx      integer;
    SET ISOLATION = 'uncommitted';
    max_ses_len := 10000000;
    file_len := 0;
    file_idx := 1;
    file_name := sprintf ('%s%06d.ttl', out_file, file_idx);
    string_to_file ( file_name || '.graph',
```

```
                    srcgraph,
                    -2
                  );
    string_to_file ( file_name,
                    sprintf ( '# Dump of graph <%s>, as of %s\n',
                              srcgraph,
                              CAST (NOW() AS VARCHAR)
                            ),
                    -2
                  );
    env := vector (dict_new (16000), 0, '', '', '', 0, 0, 0, 0);
    ses := string_output ();
    FOR (SELECT * FROM ( SPARQL DEFINE input:storage ""
                         SELECT ?s ?p ?o { GRAPH `iri(?:srcgraph)` { ?s ?p ?o } }
                       ) AS sub OPTION (LOOP)) DO
      {
        http_ttl_triple (env, "s", "p", "o", ses);
        ses_len := length (ses);
        IF (ses_len > max_ses_len)
          {
            file_len := file_len + ses_len;
            IF (file_len > file_length_limit)
              {
                http (' .\n', ses);
                string_to_file (file_name, ses, -1);
                file_len := 0;
                file_idx := file_idx + 1;
                file_name := sprintf ('%s%06d.ttl', out_file, file_idx);
                string_to_file ( file_name,
                                 sprintf ( '# Dump of graph <%s>, as of %s (part %d)\n',
                                           srcgraph,
                                           CAST (NOW() AS VARCHAR),
                                           file_idx),
                                 -2
                               );
                 env := vector (dict_new (16000), 0, '', '', '', 0, 0, 0, 0);
              }
            ELSE
              string_to_file (file_name, ses, -1);
            ses := string_output ();
          }
      }
    IF (LENGTH (ses))
      {
        http (' .\n', ses);
        string_to_file (file_name, ses, -1);
      }
  }
;
```

Example 2

```
create procedure dump_rdftriples_to_ttl(inout triples any, inout ses any)
{
  declare env any;
  declare tcount, tctr integer;
  tcount := length (triples);
  if (0 = tcount)
    {
      http ('# Empty TURTLE\n', ses);
      return;
    }
  env := vector (dict_new (__min (tcount, 16000)), 0, '', '', '', 0, 0, 0, 0);
  { whenever sqlstate '*' goto end_pred_sort;
    rowvector_subj_sort (triples, 1, 1);
end_pred_sort: ;
  }
  { whenever sqlstate '*' goto end_subj_sort;
    rowvector_subj_sort (triples, 0, 1);
```

```
end_subj_sort: ;
  }
  for (tctr := 0; tctr < tcount; tctr := tctr + 1)
    {
      http_ttl_triple (env, triples[tctr][0], triples[tctr][1], triples[tctr][2], ses);
    }
  http (' .', ses);
}
;
```

See Also

http_nt_triple()

DB.DBA.RDF_AUDIT_METADATA()

DB.DBA.RDF_BACKUP_METADATA()

DB.DBA.RDF_LOAD_RDFXML()

DB.DBA.RDF_LOAD_RDFXML_MT()

TTLP_MT()

TTLP_MT_LOCAL_FILE()

DB.DBA.ANN_PHRASE_CLASS_ADD

DB.DBA.ANN_PHRASE_CLASS_ADD — Returns APC_ID of a phrase class.

Synopsis

DB.DBA.ANN_PHRASE_CLASS_ADD (in *_name* varchar ,
in *_owner_uid* integer ,
in *_reader_gid* integer ,
in *_callback* varchar ,
in *_app_env* any ,
in *mode* varchar ,
in *auth_uname* varchar ,
in *auth_pwd* varchar) ;

Description

The returned value is APC_ID of a class if positive or an error code if negative. To update phrase class, no special DB.DBA.ANN_PHRASE_CLASS_UPDATE() exists, use DB.DBA.ANN_PHRASE_CLASS_ADD() with parameter "mode" equal to 'replacing'.

Parameters

_name
: a name of new phrase class

_owner_uid
: owner UID, will be stored as APC_OWNER_UID, can be NULL for public

_reader_gid
: reader GID, will be stored as APC_READER_GID, can be NULL for public

_callback
: value for APC_CALLBACK field

_app_env
: value for APC_APP_ENV field

mode
: Instructs what to do on attempt of writing over existing class: 'into' will signal, 'soft' return APC_ID of (unchanged) existing class, 'replacing' will continue with update.

auth_uname
: Authentication name

auth_pwd
: Authentication password

Examples

Example 24.292. ANN_PHRASE_CLASS_ADD

```
SQL>select ANN_PHRASE_CLASS_ADD ('My check', null, null, null, 'My check visa', '', 'dba', 'dba');
callret
INTEGER
_______________________________________________________________________________

1
```

```
1 Rows. -- 0 msec.

SQL>select ANN_PHRASE_CLASS_ADD ('Paid ads', null, null, null, 'Paid ads env', 'replacing', 'dba', 'dba')
callret
VARCHAR
_______________________________________________________________________________

2

1 Rows. -- 261 msec.
```

See Also

DB.DBA.ANN_PHRASE_CLASS_DEL

AP_BUILD_MATCH_LIST

DB.DBA.ANN_PHRASE_CLASS_DEL

DB.DBA.ANN_PHRASE_CLASS_DEL — Deletes phrase set.

Synopsis

DB.DBA.ANN_PHRASE_CLASS_DEL (in *_name* varchar ,
in *auth_uname* varchar ,
in *auth_pwd* varchar) ;

Description

The returned value is APS_ID of a deleted phrase set if positive or an error code if negative. A phrase set can not be deleted while used by some advertiser (as described here).

Parameters

_name
: a name of phrase set to delete

auth_uname
: Authentication name

auth_pwd
: Authentication password

Examples

Example 24.293. ANN_PHRASE_CLASS_DEL

```
SQL>select ANN_PHRASE_CLASS_ADD ('simple test', null, null, null, '123', '', 'dba', 'dba');
callret
INTEGER
_______________________________________________________________________________

1

1 Rows. -- 0 msec.

SQL>select ANN_PHRASE_CLASS_DEL ('simple test', 'dba', 'dba');
callret
VARCHAR
_______________________________________________________________________________

1

1 Rows. -- 70 msec.
```

See Also

DB.DBA.ANN_PHRASE_CLASS_ADD

AP_BUILD_MATCH_LIST

AP_BUILD_MATCH_LIST

AP_BUILD_MATCH_LIST — Returns report of all occurrences of phrases from the specified sets in the text.

Synopsis

AP_BUILD_MATCH_LIST (in *phrase_set_ids* vector of integers ,
in *source_UTF8_text* varchar not null ,
in *lang_name* varchar not null ,
in *source_text_is_html* integer ,
in *report_flags* integer) ;

Description

Forms a report that lists all occurrences of phrases from the specified sets in the text.

The report describes "phrase hits", i.e. occurrences of annotation phrases in the text, using "arrows" that point to specific fragments in the text, such as words of found phrases or HTML tags.

The structure of the report is complicated, due to contradiction in requirements. It is compact to provide reasonable performance and scalability, so common data should not be repeated, saving memory. It is complete enough to prevent application from reading omitted data from system tables, saving time.

All objects of one type are listed as items of some vector and the whole report consists of several such vectors. An item in one vector may refer to item in other vector by its index, without storing a local copy.

Detailed description of the report structure can be found here .

Parameters

phrase_set_ids
: vector of numeric identifiers of phrase sets at work, they may belong to various phrase classes, but if language of some phrase set differs from value of lang_name argument then the phrase set is silently ignored.

source_UTF8_text
: a plain text or an HTML

lang_name
: language name

source_text_is_html
: 0 for plain text, 1 for standard-compliant HTML or 2 for "dirty" HTML

report_flags
: Report flag

Examples

Example 24.294. Simple Use

Usage example can be found here .

See Also

DB.DBA.ANN_PHRASE_CLASS_ADD

DB.DBA.ANN_PHRASE_CLASS_DEL

AP_ADD_PHRASES

AP_ADD_PHRASES — Adds phrases to given set.

Synopsis

AP_ADD_PHRASES (in *phrase_id* integer ,
in *description_phrases* vector) ;

Description

The function gets two arguments, the integer ID of annotation phrase set and a vector of $ descriptions of phrases that should be edited in that phrase set.

Every item of vector of descriptions is in turn vector of one or two values; first value is the text of the phrase, second value is associated application specific data, the absence of second value indicates that the phrase should be removed. If same text of phrase appears in the vector of description more than once, and associated data differ then any version of data can be stored for future use; it is the roll of dice because the vector is reordered for faster processing.

Parameters

phrase_id
: integer ID of the annotation phrase set

description_phrases
: vector of descriptions of phrases that should be edited in that phrase set

Examples

Example 24.295. Simple Example

```
SQL>insert replacing DB.DBA.SYS_ANN_PHRASE_SET (APS_ID, APS_NAME, APS_OWNER_UID, APS_READER_GID, APS_APC_
values (5, 'Debug aps #5', http_dav_uid(), http_admin_gid(), 3, 'x-any', 'Debug aps #5 env', 10000, 0)
;
Done. -- 0 msec.

SQL>ap_add_phrases (5,
  vector (
    vector ('Debug5', 'Debug5 env'),
    vector ('Debug5 one', 'Debug5 one env'),
    vector ('Debug5 two', 'Debug5 two env'),
    vector ('Debug5 three', 'Debug5 three env'),
    vector ('Debug5 twenty one', 'Debug5 twenty one env'),
    vector ('Debug5 twenty two', 'Debug5 twenty two env'),
    vector ('Debug5 twenty three', 'Debug5 twenty three env')
    ) )
;
Done. -- 0 msec.
```

Detailed example can be found here .

See Also

DB.DBA.ANN_PHRASE_CLASS_ADD

DB.DBA.ANN_PHRASE_CLASS_DEL

iri_split

iri_split — Splits into the namespace prefix and the local part according to the TTL rules.

Synopsis

```
iri_split ( in qname varchar ,
            out local varchar ) ;
```

Description

Iri is a qualified name. It is split into the namespace prefix and the local part according to the TTL rules. The prefix is returned. If a second argument is given, it is set to the local part.

Parameters

qname
: namespace prefix

local
: local part

__xml_get_ns_prefix

__xml_get_ns_prefix — Returns prefix by given URI.

Synopsis

```
__xml_get_ns_prefix ( in str varchar ,
                      out flags int ) ;
```

Description

If str is a globally known namespace URI, its prefix is returned, else null. Flags is a bit mask where 1 means to look in the connection, 2 in the global set of known prefixes.

Parameters

str
: uri

flags
: bitmask of flags that permit some sorts of syntax errors in resource, use 0.

See Also

`xml_get_ns_uri` `xml_ns_uname` `xml_ns_iristr` `xml_nsexpand_iristr`

__xml_get_ns_uri

__xml_get_ns_uri — Returns URI by given namespace prefix.

Synopsis

```
__xml_get_ns_uri ( in str varchar ,
                   out flags int ) ;
```

Description

If str is a globally known namespace prefix, its URI is returned, else null. Flags is a bit mask where 1 means to look in the connection, 2 in the global set of known prefixes.

Parameters

str
: namespace prefix

flags
: A bit mask where 1 means to look in the connection, 2 in the global set of known prefixes.

See Also

`xml_get_ns_prefix()` `xml_ns_uname()` `xml_ns_iristr()` `xml_nsexpand_iristr()`

__xml_ns_uname

__xml_ns_uname — Returns an UNAME that is IRI --- concatenation of the expanded namespace IRI and the local part.

Synopsis

__xml_ns_uname (in *str1* varchar ,
in *str2* varchar) ;

Description

Returns an UNAME that is IRI -- concatenation of the expanded namespace IRI and the local part.

Parameters

str1
: Namespace prefix.

str2
: Local part (varchar).

Examples

Example 24.296. Simple Example

```
SQL> SELECT __xml_ns_uname ('foaf', 'knows');
__xml_ns_uname
VARCHAR
_______________________________________________________________________________

http://xmlns.com/foaf/0.1/knows

1 Rows. -- 0 msec.
```

See Also

`xml_ns_iristr()`

`xml_nsexpand_iristr()`

`xml_get_ns_prefix()`

`xml_get_ns_uri()`

__xml_ns_iristr

__xml_ns_iristr — Returns an varchar with box flag set to BF_IRI.

Synopsis

__xml_ns_iristr (in *str1* varchar ,
in *str2* varchar) ;

Description

This function is similar to the `xml_ns_uname` function, and it returns an varchar with box flag set to BF_IRI.

Parameters

str1
: Namespace prefix

str2
: Local part (varchar)

Examples

Example 24.297. Simple Example

Base64-encode a string

```
SQL> SELECT __xml_ns_iristr ('foaf', 'knows');
__xml_ns_iristr
VARCHAR
_______________________________________________________________________________

http://xmlns.com/foaf/0.1/knows

1 Rows. -- 1 msec.
```

See Also

`xml_ns_uname`

`xml_nsexpand_iristr`

`xml_get_ns_prefix`

`xml_get_ns_uri`

__xml_nsexpand_iristr

__xml_nsexpand_iristr — Checks if the string is QName and if it is so then replaces namespace prefix to the full namespace IRI.

Synopsis

__xml_nsexpand_iristr (in *str* varchar) ;

Description

Checks if the string is QName and if it is so then replaces namespace prefix to the full namespace IRI. Otherwise signals error if the argument is local IRI without a prefix.

Parameters

str
: QName. For ex. if QName is "prefix:local", then conveniently is replaced "prefix:" with the namespace IRI

Examples

Example 24.298. Example with FOAF namespace prefix

```
SQL> SELECT __xml_nsexpand_iristr ('foaf:knows');
__xml_nsexpand_iristr
VARCHAR
_______________________________________________________________________________

http://xmlns.com/foaf/0.1/knows

1 Rows. -- 0 msec.
```

Example 24.299. Example with no namespace prefix

```
SQL> SELECT __xml_nsexpand_iristr ('no-namespace-prefix-iri');

*** Error 22023: [Virtuoso Driver][Virtuoso Server]SR649:
No XML namespace prefix in string "no-namespace-prefix-iri"
in
__xml_nsexpand_iristr:(BIF),
        __01 => 'no-namespace-prefix-iri',
<Top Level>
at line 5 of Top-Level:
SELECT __xml_nsexpand_iristr ('no-namespace-prefix-iri')
```

Example 24.300. Example with namespace prefix "local"

```
SQL> SELECT __xml_nsexpand_iristr ('no-such-prefix:local');

*** Error 22023: [Virtuoso Driver][Virtuoso Server]SR648:
Unknown XML namespace prefix in IRI "no-such-prefix:local"
in
__xml_nsexpand_iristr:(BIF),
        __01 => 'no-such-prefix:local',
<Top Level>
at line 6 of Top-Level:
SELECT __xml_nsexpand_iristr ('no-such-prefix:local')
```

Example 24.301. Example with namespace prefix ""no-such-prefix""

```
SQL> SELECT __xml_ns_iristr ('no-such-prefix', 'knows');

*** Error 22023: [Virtuoso Driver][Virtuoso Server]SR648: Unknown XML namespace prefix "no-such-prefix"
in
__xml_ns_iristr:(BIF),
        __01 => 'no-such-prefix',
        __02 => 'knows',
<Top Level>
at line 7 of Top-Level:
SELECT __xml_ns_iristr ('no-such-prefix', 'knows')
```

See Also

`xml_ns_uname()`

`xml_ns_iristr()`

`xml_get_ns_prefix()`

`xml_get_ns_uri()`

DB.DBA.SPARQL_EVAL

DB.DBA.SPARQL_EVAL — Local execution of SPARQL via SPARQL protocol, produces a result set of SQL values.

Synopsis

DB.DBA.SPARQL_EVAL (in *query* varchar ,
in *dflt_graph* varchar ,
in *maxrows* integer) ;

Description

Local execution of SPARQL via SPARQL protocol, produces a result set of SQL values.

Parameters

query
: text of SPARQL query to execute.

dflt_graph
: default graph IRI, if not NULL then this overrides what is specified in query

maxrows
: limit on numbers of rows that should be returned.

Return Types

any

Examples

Example 24.302. Simple Use

```
SQL>DB.DBA.SPARQL_EVAL('SELECT * WHERE {?s ?p ?o}','http://example/bookStore' ,10);
s                    p              o
VARCHAR              VARCHAR        VARCHAR
_______________________________________________________________________________

http://example/book3  http://purl.org/dc/elements/1.1/title  Fundamentals
http://example/book3  http://purl.org/dc/elements/1.1/date   2002-01-01T00:00:00
http://example/book2  http://purl.org/dc/elements/1.1/title  Design notes
http://example/book2  http://purl.org/dc/elements/1.1/date   2001-01-01T00:00:00

4 Rows. -- 30 msec.
```

See Also

DB.DBA.RDF_AUDIT_METADATA

DB.DBA.RDF_BACKUP_METADATA

DB.DBA.RDF_LOAD_RDFXML_MT

DB.DBA.TTLP

DB.DBA.TTLP_MT

DB.DBA.RDF_TRIPLES_TO_RDF_XML_TEXT

DB.DBA.SPARQL_EVAL_TO_ARRAY

DB.DBA.SPARQL_EVAL_TO_ARRAY

DB.DBA.SPARQL_EVAL_TO_ARRAY — Local execution of SPARQL via SPARQL protocol, produces a vector of vectors of SQL values.

Synopsis

DB.DBA.SPARQL_EVAL_TO_ARRAY (in *query* varchar ,
in *dflt_graph* varchar ,
in *maxrows* integer) ;

Description

Local execution of SPARQL via SPARQL protocol, produces a result set of SQL values.

Parameters

query
: text of SPARQL query to execute.

dflt_graph
: default graph IRI, if not NULL then this overrides what is specified in query

maxrows
: limit on numbers of rows that should be returned.

Return Types

any

Examples

Example 24.303. Simple Use

```
SQL> select DB.DBA.SPARQL_EVAL_TO_ARRAY('SELECT * WHERE {?s ?p ?o}','http://example/bookStore' ,2);
callret
VARCHAR
_______________________________________________________________________________

vector(0x00ae6874,0x00ae6844)

1 Rows. -- 10 msec.
```

See Also

DB.DBA.RDF_AUDIT_METADATA

DB.DBA.RDF_BACKUP_METADATA

DB.DBA.RDF_LOAD_RDFXML_MT

DB.DBA.TTLP

DB.DBA.TTLP_MT

DB.DBA.RDF_TRIPLES_TO_RDF_XML_TEXT

DB.DBA.SPARQL_EVAL

DB.DBA.SPARQL_REXEC

DB.DBA.SPARQL_REXEC — Remote execution of SPARQL via SPARQL protocol, produces a result set of SQL values.

Synopsis

DB.DBA.SPARQL_REXEC (in *service* varchar ,
in *query* varchar ,
in *dflt_graph* varchar ,
in *named_graphs* any ,
in *req_hdr* any ,
in *maxrows* integer ,
in *bnode_dict* any) ;

Description

Remote execution of SPARQL via SPARQL protocol, produces a result set of SQL values.

Parameters

service
: service URI to call via HTTP.

query
: text of SPARQL query to execute.

dflt_graph
: default graph IRI, if not NULL then this overrides what is specified in query

named_graphs
: vector of named graph IRIs, if not NULL then this overrides what is specified in query

req_hdr
: additional HTTP header lines that should be passed to the service; 'Host: ...' is most popular.

maxrows
: limit on numbers of rows that should be returned.

bnode_dict
: dictionary of bnode ID references.

Return Types

any

Examples

Example 24.304. Simple Use

```
create procedure SPARQL_TEST (in rquri varchar, in graph_uri varchar, in resuri varchar, in in_result int
{
  declare REPORT varchar;
  declare rqtext, sqltext varchar;
  declare app_env any;
  declare rset, row any;
  declare etalon_vars, etalon_rowids, etalon_rows any;
  declare rctr, rcount integer;
  declare copyright_filter any;
```

```
  if (not in_result)
    result_names (REPORT);
  result ('');
  declare exit handler for sqlstate '*' {
   ...
    };
  ...
  rqtext := replace (cast (XML_URI_GET ('', rquri) as varchar), '# \044Id:', '# Id:');
      declare rexec_stat, rexec_msg varchar;
      declare bnode_dict, rexec_rmeta, rexec_rrows any;
      rexec_stat := '00000';
      rexec_msg := 'OK';
      bnode_dict := dict_new ();
      rexec_rrows := null;
      exec (
        'DB.DBA.SPARQL_REXEC (?, ?, ?, ?, ?, ?, ?)',
        rexec_stat, rexec_msg,
        vector (
          WB_CFG_HTTP_URI() || '/sparql/',
          rqtext,
          graph_uri,
          vector (),
          '',
          10000,
          bnode_dict ),
        10000, rexec_rmeta, rexec_rrows );
      if (not isarray (rexec_rrows))
        rexec_rrows := null;
      result (sprintf ('Remote exec of %s', rquri));
      dump_large_text_impl (
        sprintf ('  completed with state %s msg %s and %d rows',
        rexec_stat, rexec_msg, length (rexec_rrows) ) );
  result ('PASSED');
};
```

See Also

DB.DBA.RDF_AUDIT_METADATA

DB.DBA.RDF_BACKUP_METADATA

DB.DBA.RDF_LOAD_RDFXML_MT

DB.DBA.TTLP

DB.DBA.TTLP_MT

DB.DBA.RDF_TRIPLES_TO_RDF_XML_TEXT

DB.DBA.SPARQL_REXEC_WITH_META

DB.DBA.SPARQL_REXEC_TO_ARRAY

DB.DBA.SPARQL_REXEC_TO_ARRAY

DB.DBA.SPARQL_REXEC_TO_ARRAY — Remote execution of SPARQL via SPARQL protocol, produces a vector of vectors of SQL value.

Synopsis

DB.DBA.SPARQL_REXEC_TO_ARRAY (
in *service* varchar ,
in *query* varchar ,
in *dflt_graph* varchar ,
in *named_graphs* any ,
in *req_hdr* any ,
in *maxrows* integer ,
in *bnode_dict* any) ;

Description

Remote execution of SPARQL via SPARQL protocol, produces a vector of vectors of SQL value.

Parameters

service
: service URI to call via HTTP.

query
: text of SPARQL query to execute.

dflt_graph
: default graph IRI, if not NULL then this overrides what is specified in query

named_graphs
: vector of named graph IRIs, if not NULL then this overrides what is specified in query

req_hdr
: additional HTTP header lines that should be passed to the service; 'Host: ...' is most popular.

maxrows
: limit on numbers of rows that should be returned.

bnode_dict
: dictionary of bnode ID references.

Return Types

any

See Also

`DB.DBA.RDF_AUDIT_METADATA()`

`DB.DBA.RDF_BACKUP_METADATA()`

`DB.DBA.RDF_LOAD_RDFXML_MT`

`DB.DBA.TTLP()`

`DB.DBA.TTLP_MT()`

DB.DBA.RDF_TRIPLES_TO_RDF_XML_TEXT()

DB.DBA.SPARQL_REXEC()

DB.DBA.SPARQL_REXEC_WITH_META

DB.DBA.SPARQL_REXEC_WITH_META — Remote execution of SPARQL via SPARQL protocol. Fills in output parameters with metadata (like exec metadata) and a vector of vector s of 'long valmode' values.

Synopsis

```
DB.DBA.SPARQL_REXEC_WITH_META (  in service varchar ,
                                 in query varchar ,
                                 in dflt_graph varchar ,
                                 in named_graphs any ,
                                 in req_hdr any ,
                                 in maxrows integer ,
                                 in bnode_dict any ,
                                 out metadata any ,
                                 out resultset any );
```

Description

Remote execution of SPARQL via SPARQL protocol. Fills in output parameters with metadata (like exec metadata) and a vector of vector s of 'long valmode' values.

Parameters

service
: service URI to call via HTTP.

query
: text of SPARQL query to execute.

dflt_graph
: default graph IRI, if not NULL then this overrides what is specified in query

named_graphs
: vector of named graph IRIs, if not NULL then this overrides what is specified in query

req_hdr
: additional HTTP header lines that should be passed to the service; 'Host: ...' is most popular.

maxrows
: limit on numbers of rows that should be returned.

bnode_dict
: dictionary of bnode ID references.

metadata
: metadata like exec () returns.

resultset
: results as 'long valmode' value.

Return Types

any

See Also

DB.DBA.RDF_AUDIT_METADATA()

DB.DBA.RDF_BACKUP_METADATA()

DB.DBA.RDF_LOAD_RDFXML_MT()

DB.DBA.TTLP()

DB.DBA.TTLP_MT()

DB.DBA.RDF_TRIPLES_TO_RDF_XML_TEXT()

DB.DBA.SPARQL_REXEC()

DB.DBA.SPARQL_REXEC_TO_ARRAY()

DB.DBA.RDF_REGEX

DB.DBA.RDF_REGEX — Returns 1 if string s matches pattern p, 0 otherwise.

Synopsis

DB.DBA.RDF_REGEX (in *s* varchar ,
in *p* varchar ,
in *call* varchar := null) ;

Description

Returns 1 if string s matches pattern p, 0 otherwise

Parameters

s
: source string to check.

p
: regular expression pattern string.

coll
: unused for now (modes are not yet implemented).

Return Types

integer

Examples

Example 24.305. Simple example

```
SQL>SELECT DISTINCT DB.DBA.RDF_QNAME_OF_IID ( /*retval[*/  "s-1-1-t0"."S" /* R */ /*]retval*/ ) AS /*tmpl
FROM DB.DBA.RDF_QUAD AS "s-1-1-t0"
WHERE /* filter */
    DB.DBA.RDF_REGEX ( DB.DBA.RDF_QNAME_OF_IID ( /*retval[*/  "s-1-1-t0"."S" /* R */ /*]retval*/ ), 'http
OPTION (QUIETCAST)
VARCHAR
_____________________________________________________________________________

http://example.org/ns#y3
http://example.org/things#xp2
http://example.org/ns#a
http://example.org/book/book1
http://example.org/books#book1
http://example.org/books#book2
http://example.org/books#book4
http://example.org/books#book3
http://example.org/book/book2
http://example.org/ns#x2
http://example.org/things#xp1
http://example.org/ns#x3
http://example.org/things#xt1
http://example.org/ns#y1
http://example.org/ns#y2
13 Rows. -- 30 msec.
```

See Also

DB.DBA.RDF_AUDIT_METADATA

DB.DBA.RDF_BACKUP_METADATA

DB.DBA.RDF_LOAD_RDFXML_MT

DB.DBA.TTLP

DB.DBA.TTLP_MT

DB.DBA.RDF_TRIPLES_TO_RDF_XML_TEXT

DB.DBA.SPARQL_REXEC

DB.DBA.SPARQL_REXEC_TO_ARRAY

DB.DBA.SPARQL_REXEC_WITH_META

DB.DBA.RDF_LANGMATCHES

DB.DBA.RDF_LANGMATCHES — Returns 1 if language identifier r matches lang pattern t.

Synopsis

DB.DBA.RDF_LANGMATCHES (in *r* varchar ,
in *t* varchar) ;

Description

Returns 1 if language identifier r matches lang pattern t

Parameters

r
: language identifies (string or NULL).

t
: language pattern (exact name, first two letters or '*').

Return Types

integer

Examples

Example 24.306. Simple example

```
SQL>SELECT DB.DBA.RDF_QNAME_OF_IID ( /*retval[*/  "s-4-1-t0"."P" /* p */ /*]retval*/ ) AS /*tmpl*/ "p",
    DB.DBA.RDF_SQLVAL_OF_OBJ ( /*retval[*/  "s-4-1-t0"."O" /* v */ /*]retval*/ ) AS /*tmpl*/ "v"
FROM DB.DBA.RDF_QUAD AS "s-4-1-t0"
WHERE /* field equal to URI ref */
    "s-4-1-t0"."S" = DB.DBA.RDF_MAKE_IID_OF_QNAME_SAFE ( 'http://example.org/#x' )
    AND /* filter */
    not ( DB.DBA.RDF_LANGMATCHES ( DB.DBA.RDF_LANGUAGE_OF_OBJ ( /*retval[*/  "s-4-1-t0"."O" /* v */ /*]re
OPTION (QUIETCAST)
VARCHAR
_____________________________________________________________________________

http://example.org/data/y
http://example.org/data/x
http://example.org/a
3 Rows. -- 20 msec.
```

See Also

DB.DBA.RDF_AUDIT_METADATA

DB.DBA.RDF_BACKUP_METADATA

DB.DBA.RDF_LOAD_RDFXML_MT

DB.DBA.TTLP

DB.DBA.TTLP_MT

DB.DBA.RDF_TRIPLES_TO_RDF_XML_TEXT

DB.DBA.SPARQL_REXEC

DB.DBA.SPARQL_REXEC_TO_ARRAY

DB.DBA.SPARQL_REXEC_WITH_META

DB.DBA.RDF_REGEX

DB.DBA.RDF_TTL2HASH

DB.DBA.RDF_TTL2HASH — Returns a dict of triples in 'long valmode'.

Synopsis

DB.DBA.RDF_TTL2HASH (in *strg* any ,
in *base* varchar ,
in *graph* varchar) ;

Description

Returns a dict of triples in 'long valmode'.

Parameters

strg
: text of the resource

base
: base IRI to resolve relative IRIs to absolute

graph
: target graph IRI, parsed triples will appear in that graph.

Return Types

any

Examples

Example 24.307. Simple example

See Also

DB.DBA.RDF_AUDIT_METADATA

DB.DBA.RDF_BACKUP_METADATA

DB.DBA.RDF_LOAD_RDFXML_MT

DB.DBA.TTLP

DB.DBA.TTLP_MT

DB.DBA.RDF_TRIPLES_TO_RDF_XML_TEXT

DB.DBA.SPARQL_REXEC

DB.DBA.SPARQL_REXEC_TO_ARRAY

DB.DBA.SPARQL_REXEC_WITH_META

DB.DBA.RDF_QUAD_URI

DB.DBA.RDF_QUAD_URI — Performs simple insertion of a quad where object is a node.

Synopsis

```
DB.DBA.RDF_QUAD_URI ( in g_uri varchar ,
                      in s_uri varchar ,
                      in p_uri varchar ,
                      in o_uri varchar ) ;
```

Description

Performs simple insertion of a quad where object is a node. The arguments g_uri, s_uri and p_uri should be IRI strings or IRI_IDs. All string arguments should be in UTF-8 encoding, otherwise they will be stored but are not queryable via SPARQL.

Parameters

g_uri
: graph uri IRI string or IRI_ID

s_uri
: subject IRI string or IRI_ID

p_uri
: predicate IRI string or IRI_ID

o_uri
: object IRI string or IRI_ID

Return Types

any

See Also

DB.DBA.RDF_AUDIT_METADATA()

DB.DBA.RDF_BACKUP_METADATA()

DB.DBA.RDF_LOAD_RDFXML_MT()

DB.DBA.TTLP()

DB.DBA.TTLP_MT()

DB.DBA.RDF_TRIPLES_TO_RDF_XML_TEXT()

DB.DBA.SPARQL_REXEC()

DB.DBA.SPARQL_REXEC_TO_ARRAY()

DB.DBA.SPARQL_REXEC_WITH_META()

DB.DBA.RDF_QUAD_URI_L()

DB.DBA.RDF_QUAD_URI_L_TYPED()

DB.DBA.RDF_TTL2HASH()

DB.DBA.RDF_QUAD_URI_L

DB.DBA.RDF_QUAD_URI_L — Performs simple insertion of a quad where object is a literal value in 'SQL valmode'.

Synopsis

DB.DBA.RDF_QUAD_URI_L (in *g_uri* varchar ,
in *s_uri* varchar ,
in *p_uri* varchar ,
in *o_lit* any) ;

Description

Performs simple insertion of a quad where object is a literal value in 'SQL valmode'. The arguments g_uri, s_uri and p_uri should be IRI strings or IRI_IDs. All string arguments should be in UTF-8 encoding, otherwise they will be stored but are not queryable via SPARQL.

Parameters

g_uri
: graph uri IRI string or IRI_ID

s_uri
: subject IRI string or IRI_ID

p_uri
: predicate IRI string or IRI_ID

o_uri
: string, number or datetime, NULL is not allowed

Return Types

any

Examples

Example 24.308. Simple example

```
SQL>DB.DBA.RDF_QUAD_URI_L ('g_many', 's1', 'p_some', 'z016,g_many,s1,p_some');

Done. -- 0 msec.
```

See Also

DB.DBA.RDF_AUDIT_METADATA

DB.DBA.RDF_BACKUP_METADATA

DB.DBA.RDF_LOAD_RDFXML_MT

DB.DBA.TTLP

DB.DBA.TTLP_MT

DB.DBA.RDF_TRIPLES_TO_RDF_XML_TEXT

DB.DBA.SPARQL_REXEC

DB.DBA.SPARQL_REXEC_TO_ARRAY

DB.DBA.SPARQL_REXEC_WITH_META

DB.DBA.RDF_QUAD_URI

DB.DBA.RDF_QUAD_URI_L_TYPED

DB.DBA.RDF_TTL2HASH

DB.DBA.RDF_QUAD_URI_L_TYPED

DB.DBA.RDF_QUAD_URI_L_TYPED — Performs simple insertion of a quad where object is a literal value in 'SQL valmode' and can be specified datatype and language.

Synopsis

DB.DBA.RDF_QUAD_URI_L_TYPED (in *g_uri* varchar ,
in *s_uri* varchar ,
in *p_uri* varchar ,
in *o_lit* any ,
in *dt* any ,
in *lang* varchar) ;

Description

Performs simple insertion of a quad where object is a literal value in 'SQL valmode' and can be specified datatype and language. The arguments g_uri, s_uri and p_uri should be IRI strings or IRI_IDs. All string arguments should be in UTF-8 encoding, otherwise they will be stored but are not queryable via SPARQL.

Parameters

g_uri
: graph uri IRI string or IRI_ID

s_uri
: subject IRI string or IRI_ID

p_uri
: predicate IRI string or IRI_ID

o_uri
: string value of the literal

dt
: datatype as IRI string or IRI_ID, can be NULL

lang
: language as string or NULL

Return Types

any

Examples

Example 24.309. Simple example

```
SQL>DB.DBA.RDF_QUAD_URI_L_TYPED ('g_many', 's1', 'p_some', 'z016,g_many,s1,p_some', null, 'en');

Done. -- 0 msec.
```

See Also

DB.DBA.RDF_AUDIT_METADATA

DB.DBA.RDF_BACKUP_METADATA

DB.DBA.RDF_LOAD_RDFXML_MT

DB.DBA.TTLP

DB.DBA.TTLP_MT

DB.DBA.RDF_TRIPLES_TO_RDF_XML_TEXT

DB.DBA.SPARQL_REXEC

DB.DBA.SPARQL_REXEC_TO_ARRAY

DB.DBA.SPARQL_REXEC_WITH_META

DB.DBA.RDF_QUAD_URI

DB.DBA.RDF_QUAD_URI_L

DB.DBA.RDF_TTL2HASH

regexp_replace

regexp_replace — Replaces occurrence(s) of the matching pattern in the source_string with a specified replace_string.

Synopsis

```
varchar regexp_replace ( in source_string any ,
                         in pattern any ,
                         in replace_string varchar ,
                         in position integer ,
                         in occurrence any ,
                         in match_parameter integer ) ;
```

Description

This function replaces occurrence(s) of the matching *pattern* in the *source_string* with a specified *replace_string*, allowing complex search-and-replace operations. The traditional REPLACE SQL function substitutes one string with another. Assume your data has extraneous spaces in the text and you would like to replace them with a single space. With the REPLACE function, you would need to list exactly how many spaces you want to replace. However, the number of extra spaces may not be the same everywhere in the text.

Parameters

source_string
: Source string.

pattern
: The regular expression to match.

replace_string
: By default an empty string, ''.

position
: By default this is set to 1.

occurrence
: By default this is set to 0. If you specify occurrence as DB NULL value it will replace all the occurrences of the pattern in the string.

match_parameter
: By default this is set to null.

Returns

This function returns the a substring matching the regular expression.

Examples

Example 24.310.

This example has three spaces between Joe and Smith. The REPLACE function's parameter specifies that two spaces should be replaced with one space. In this case, the result leaves an extra space where there were three spaces in the original string between Joe and Smith.

```
SELECT
  REPLACE('Joe  Smith', '  ', ' ') AS REPLACE_NORMAL,
  REGEXP_REPLACE('Joe        Smith', '( ){2,}', ' ') AS REGEXP_REPLACE;
```

```
REPLACE_NORMAL                REPLACE_REGEXP
VARCHAR                       VARCHAR
_____________________________________________

Joe Smith                     Joe Smith
```

Compatibility:

The regular expressions supported here follow version 7.9 of the Perl Compatible Regular Expression (PCRE) syntax.

1. Wikipedia
2. PCRE.org

See Also

`regexp_match()`

`regexp_like()`

`regexp_instr()`

`regexp_parse()`

`regexp_substr()`

regexp_instr

regexp_instr — Returns the first position of the occurrence of a regular-expression pattern in a given string.

Synopsis

```
integer regexp_instr ( in source_string varchar ,
                       in pattern varchar ,
                       in start_position integer ,
                       in occurrence integer ,
                       in return_option integer ,
                       in match_parameter integer ) ;
```

Description

This function looks for a pattern and returns the first position of the pattern. Optionally, you can indicate the *start_position* you want to begin the search. The *occurrence* parameter defaults to 1 unless you indicate that you are looking for a subsequent occurrence. The default value of the *return_option* is 0, which returns the starting position of the pattern; a value of 1 returns the starting position of the next character following the match.

This function returns the starting position of a pattern, so it works much like the familiar INSTR function. The main difference between the two functions is that REGEXP_INSTR lets you specify a pattern instead of a specific search string; thus providing greater versatility.

Parameters

source_string
: The string to search.

pattern
: The regular expression to match.

start_position
: The position to begin the search from.

occurrence
: Defaulting to 1, this parameter describes how many occurrences to look for.

return_option
: Default is 0, which returns the starting position of the pattern; a value of 1 returns the starting position of the next character following the match.

match_parameter
: Match parameter value.

Returns

This function returns the a substring matching the regular expression.

Examples

Example 24.311. Examples

This example uses `REGEXP_INSTR()` to return the starting position of the five-digit zip-code pattern within the string Joe Smith, 10045 Berry Lane, San Joseph, CA 91234. If the regular expression is written as :digit:?{5}, you will get the house number's starting position instead of the zip code's, because 10045 is the first occurrence of five consecutive digits. Therefore, you must anchor the expression to the end of the line, as indicated with the $ metacharacter, and the function will display the starting

position of the zip code regardless of the number of digits for the house number.

```
SELECT REGEXP_INSTR('Joe Smith, 10045 Berry Lane, San Joseph, CA 91234',
       '[[:digit:]]{5}\$')
       AS rx_instr;
  RX_INSTR
----------
        45
```

Compatibility:

The regular expressions supported here follow version 7.9 of the Perl Compatible Regular Expression (PCRE) syntax.

1. Wikipedia
2. PCRE.org

See Also

`regexp_match()`

`regexp_like()`

`regexp_parse()`

`regexp_replace()`

`regexp_substr()`

regexp_like

regexp_like — Allows a like comparison using regular-expression.

Synopsis

```
integer regexp_like ( in source_string any ,
                      in pattern any ,
                      in match_parameter integer ) ;
```

Description

The source_string supports character datatypes (CHAR, VARCHAR2, CLOB, NCHAR, NVARCHAR2, and NCLOB but not LONG). The pattern parameter is another name for the regular expression. match_parameter allows optional parameters such as handling the newline character, retaining multiline formatting, and providing control over case-sensitivity.

Parameters

source_string
: Source string

pattern
: The regular expression to match. The following special classes are supported:

match_parameter
: This is null by default.

Examples

Example 24.312. Simple Use

The following SQL query's WHERE clause shows the REGEXP_LIKE operator, which searches the ZIP column for a pattern that satisfies the regular expression [^[:digit:]]. It will retrieve those rows in the ZIPCODE table for which the ZIP column values contain any character that is not a numeric digit.

```
SELECT postalcode FROM demo.demo.customers
 WHERE REGEXP_LIKE(postalcode, '[^[:digit:]]')

PostalCode
==========
 WA1 1DP
 S-958 22
 T2F 8M4
 EC2 5NT
 05432-043
.
.
```

Compatibility:
The regular expressions supported here follow version 7.9 of the Perl Compatible Regular Expression (PCRE) syntax.

1. Wikipedia
2. PCRE.org

See Also

`regexp_match()`

`regexp_parse()`

`regexp_instr()`

`regexp_replace()`

`regexp_substr()`

registry_get

registry_get — Returns a current value of a registry setting

Synopsis

```
varchar registry_get ( in name varchar );
```

Description

registry_get is used to retrieve values stored within database registry.

Parameters

name
: Name of the registry setting

Return Values

A string value associated to the registry by registry_set. If the variable does not exist, a 0 (zero) is returned.

Examples

Example 24.313. Setting and Retrieving Registry Settings

```
SQL> select registry_get('var_demo');
callret
VARCHAR
_______________________________________________________________________

NULL

SQL> registry_set('var_demo', 'some arb data');

SQL> select registry_get('var_demo');
callret
VARCHAR
_______________________________________________________________________

some arb data
```

See Also

registry_get_all

registry_name_is_protected

registry_set

registry_remove

sequence_get_all

sequence_next

sequence_set

sequence_remove

registry_get_all

registry_get_all — Returns a vector of all registry settings

Synopsis

```
vector registry_get_all ( );
```

Description

The function returns a vector of even length that contains two elements for every registry setting that is now stored in the database: the name and the current value of the setting. The order of 'name-value' pairs in the resulting vector may vary from call to call. To search such a vector by a setting name, the vector can be passed as a second argument to the function `get_keyword ()`.

The returned vector is a full copy of the content of the registry. To change the actual registry, use `registry_set` .

Return Types

The function returns a vector of even length.

See Also

`registry_get`

`registry_set`

`registry_name_is_protected`

`registry_remove`

`sequence_get_all`

`sequence_next`

`sequence_set`

`sequence_remove`

registry_name_is_protected

registry_name_is_protected — The function checks if a given registry variable is read-only or protected from occasional changes.

Synopsis

registry_name_is_protected (in *name* varchar);

Description

Some registry variables are used solely by internal server routines so they should not be changed by any application. Some of these "protected" variables can be updated by DBA only (in built-in Virtuoso/PL routines) whereas some can not be updated by any Virtuoso/PL routine at all.

The function gets a variable name as a parameter and returns zero if such a variable variable can be created/modified/removed by any application, one if a variable can be modified by DBA but can not be removed and two if the variable can not be altered even by DBA.

The returned value does not indicate that the variable exists or not exists. The name of not yet existing variable can be protected anyway to prevent future misuse of an variable by an application.

The name of any protected variable starts with two underscores. It can be a good idea to not start names of application-specific registry variable with two underscores to ensure compatibility with future versions of Virtuoso Server.

Parameters

name
: The name of registry variable to check.

Return Types

The function returns an integer

Examples

Example 24.314. Two checks of variable names

One name can be used by any application, other is for internal use in the FTP server:

```
> select registry_name_is_protected ('my_example');
0

> select registry_name_is_protected ('__next_free_port');
1
```

See Also

`registry_get`

`registry_get_all`

`registry_set`

`registry_remove`

`sequence_get_all`

`sequence_next`

sequence_set

sequence_remove

registry_set

registry_set — Associates a value to the name in the Database registry

Synopsis

```
registry_set ( in name varchar ,
               in value varchar ,
               in force integer ) ;
```

Description

This associates a value to the name in the Database registry. The name should be a string and the value must be also string.

Registry settings are kept in Database, therefore they are persistent.

Some registry variables are used solely by internal server routines so they should not be set by any application. Some of these "protected" variables can be updated by DBA only (in built-in Virtuoso/PL routines) whereas some can not be updated by any Virtuoso/PL routine at all. If the function is called by DBA then a third argument can be specified to indicate how the function should try to update such a variable, but you will probably never use this feature.

Parameters

name
: VARCHAR name to associate the value with.

value
: value to be stored. Must be of VARCHAR type. Also if needs to be stored some other datatype it needs to be serialized (see serialize and deserialize functions).

force
: Flag to indicate how the function should try to update protected variables. The value 0 is to signal an error on such an attempt, 1 is to try to update and signal an error if failed, 2 is to try to update and silently return 0 if failed.

Return Values

The function returns 1 if success, 0 if a (protected) variable value cannot be changed.

Errors

Table 24.74. Errors signalled by registry_set

SQLState	Error Code	Error Text	Description
22023	SR014	Function registry_set needs a string as argument 2, not an arg of [datatype]	
42000	SR483	Function registry_set needs nonzero third argument to modify registry variable [name].	
42000	SR484	Function registry_set can not modify protected registry variable [name].	

Examples

Example 24.315. Setting and Retrieving Registry Setting

From SQL

```
SQL> select registry_get('var_demo');
callret
VARCHAR
_______________________________________________________________________
```

```
NULL

SQL> registry_set('var_demo', 'some arb data');

SQL> select registry_get('var_demo');
callret
VARCHAR
_______________________________________________________________________

some arb data
```

See Also

registry_get

registry_name_is_protected

registry_get_all

registry_remove

sequence_get_all

sequence_next

sequence_set

sequence_remove

registry_remove

registry_remove — Remove a variable from registry

Synopsis

registry_remove (in *name* varchar) ;

Description

The function removes a given variable from registry, so future calls of registry_get returns zero.

Some registry variables are used solely by internal server routines so they should not be changed by any application. Nobody can remove such a "protected" variable even if some of them can be updated by DBA.

Parameters

name
: Name of variable to be removed.

Return Types

The function returns the last saved value of a variable or zero if the specified variable does not exists.

Errors

Table 24.75. Errors signalled by registry_set

SQLState	Error Code	Error Text	Description
22023	SR014	Function registry_remove needs a string as argument 1, not an arg of [datatype]	
42000	SR485	Function registry_remove can not remove protected registry variable [name].	

Examples

Example 24.316. Creation and Removal of a Registry Variable

This sequence of operations demonstrates that `registry_remove` reverts the effect of `registry_set` .

```
> select registry_get ('my_variable');
0

> select registry_set ('my_variable', 'my_value');
1

> select registry_get ('my_variable');
my_value

> select registry_remove ('my_variable');
my_value

> select registry_get ('my_variable');
0
```

See Also

`registry_set`

`registry_get`

registry_name_is_protected

registry_get_all

sequence_get_all

sequence_next

sequence_set

sequence_remove

repeat

repeat — returns a new string consisting of its string argument repeated a given number of times

Synopsis

repeat (in *str* string ,
in *count* integer) ;

Description

repeat returns a new string, composed of the string str repeated count times. If count is zero, an empty string '' is returned.

```
repeat('bar',2) -> 'barbar'
```

replace

replace — This replaces every occurrence of the second argument in the first argument with the third argument.

Synopsis

```
string replace ( in string varchar ,
                 in what varchar ,
                 in repl_with varchar ) ;
```

Description

This replaces every occurrence of the second argument in the first argument with the third argument. The arguments can be narrow or wide strings.

Example 24.317. Example:

```
SQL> select replace ('12345512345', '23', '-----');
  =   1-----4551-----45
```

replay

replay — starts the roll forward of the given log

Synopsis

replay (in *log_file* varchar) ;

Description

This starts a roll forward of the given log. The log may have been produced by normal transaction logging, backup or crash dump. Logs may not be transferred between databases and thus cannot be rolled forward anywhere except on the database that generated them.

This function is for example useful after restoring a backup. It should be called for each archived transaction log produced since the backup, including and starting with the one that was current when the backup was made.

The operation blocks until the roll forward is complete. Other clients are not affected.

Errors

Parameter data type checking errors

Table 24.76. Errors signalled by

sqlstate	error code	error text
25000	SR074	replay must be run in a fresh transaction.
42000	FA002	Can't open file <fname>, error <OS errno> (<OS description>)

Parameters

log_file
: Full pathname of file containing the transactions to be replayed. The file must be produced by backup .

Return Values

Zero, if no error is signalled.

Examples

Example 24.318. Replaying A Log File

```
checkpoint 'new.log';
backup 'bak.log';
shutdown 'new2.log';
```

The above sequence of commands makes a checkpoint and starts logging subsequent transactions into new.log. The backup statement makes bak.log, which represents the state prior to starting new.log. The shutdown statement makes a new checkpoint and marks new2.log as the log file to be used for logging transactions after the database restarts. The database server exits at the completion of the SHUTDOWN statement.

```
replay ('bak.log');
replay ('new.log');
```

These statements executed on an empty database will recreate the state in effect after the last transaction to commit before the SHUTDOWN statement of the previous example.

See Also

backup

repl_disconnect

repl_disconnect — terminates communication with a replication publisher

Synopsis

repl_disconnect (in *publisher* varchar) ;

Description

This terminates any communication with the publisher. Any pending synchronization communication is disconnected and all subscribed publications are marked as 'OFF'. The effect is reversed on a subscription by subscription basis by calling repl_sync for each.

Parameters

publisher
: Publisher's name.

Example

Example 24.319. Disconnecting from publisher server 'demo-srv'.

This is to disconnect from the publisher server 'demo-srv', the all subscriptions to this server will be disconnected.

```
SQL> repl_disconnect ('demo-srv');
```

See Also

`sub_schedule()`

`repl_grant()`

`repl_init_copy()`

`repl_new_log()`

`repl_pub_add()`

`repl_pub_init_image()`

`repl_pub_remove()`

`repl_publish()`

`repl_revoke()`

`repl_sched_init()`

`repl_server()`

`repl_server_rename()`

`repl_stat()`

`repl_status()`

repl_subscribe()

repl_sync()

repl_sync_all()

repl_text()

repl_this_server()

repl_unpublish()

repl_unsubscribe()

REPL_GRANT

REPL_GRANT — grant privileges for subscription to a publication

Synopsis

REPL_GRANT (in *publication* varchar ,
in *grantee* varchar) ;

Description

This function is used to grant privilege for subscription to a SQL account for particular publication. The DBA accounts always have right for subscription to all available publications.

Parameters

publication
: A publication account name.

grantee
: A valid DB account name to be granted subscription rights.

Example

Example 24.320. Granting access to 'table_publication' for SQL user account 'demo'.

This is to grant permissions for table_publication publication account to subscribe with a SQL user account 'demo'.

```
SQL> DB.DBA.REPL_GRANT ('table_publication', 'demo');
```

See Also

`sub_schedule()`

`repl_disconnect()`

`repl_init_copy()`

`repl_new_log()`

`repl_pub_add()`

`repl_pub_init_image()`

`repl_pub_remove()`

`repl_publish()`

`repl_revoke()`

`repl_sched_init()`

`repl_server()`

`repl_server_rename()`

repl_stat()

repl_status()

repl_subscribe()

repl_sync()

repl_sync_all()

repl_text()

repl_this_server()

repl_unpublish()

repl_unsubscribe()

REPL_INIT_COPY

REPL_INIT_COPY — create initial subscription state

Synopsis

REPL_INIT_COPY (in *server_name* varchar ,
in *publication* varchar ,
in *errors_mode* integer) ;

Description

This function is called on the subscriber to copy the current state of the elements of the publication from the publishing server. Copied data can include DAV collections, tables, procedures etc. Syncing the subscription using repl_sync will synchronize the subscription once it has been initialized with this function. The state copied corresponds to the state of the published data as of the last checkpoint completed on the publisher.

Parameters

server_name
: target publisher server name.

account
: publication account name.

errors_mode
: Important: this parameter is valid only for SQL replication data (tables and WebDAV). When set to 1, in case of errors as result of the replication copy, these errors are return in vector; When set to 0, in case of error, this error is return and the replication copying is stopped.

Example

Example 24.321. Loading of the initial data on subscriber

This shows setting on-line of the initial state of the subscription to 'demo_publication' from server 'demo-srv'. The data for items will be transferred thru the VDB connection. Note that on large publications this can take a lot of time.

```
SQL> DB.DBA.REPL_INIT_COPY ('demo-srv', 'demo_publication', 1);
```

See Also

`sub_schedule()`

`repl_disconnect()`

`repl_grant()`

`repl_new_log()`

`repl_pub_add()`

`repl_pub_init_image()`

`repl_pub_remove()`

`repl_publish()`

repl_revoke()

repl_sched_init()

repl_server()

repl_server_rename()

repl_stat()

repl_status()

repl_subscribe()

repl_sync()

repl_sync_all()

repl_text()

repl_this_server()

repl_unpublish()

repl_unsubscribe()

repl_new_log

repl_new_log — create new publication log

Synopsis

repl_new_log (in *publication* varchar ,
in *file* varchar) ;

Description

This function switches to a new file for logging data for the publication. If the file is NULL a new file name will be generated based on the previous file name by appending or replacing a datetime field in the file name.

Parameters

publication
: publication account name.

file
: name of file to be used for publication's transactions logging, if size exceed 1Mb limit a new file suffixed with timestamp will be opened.

Example

Example 24.322. Opening a new replication log

This command can be issued on the publisher server to swith replication logging to the new file 'new_log_file_name.log'. In practice we make a publication with REPL_PUBLISH() and it opens the replication log file, so this is for maintenance only. Furthermore the replication server will switch to a new log suffixed with timestamp when current file is greater than 1,000,000 bytes.

```
SQL> repl_new_log ('demo-publication', 'new_log_file_name.log');
```

See Also

`sub_schedule()`

`repl_disconnect()`

`repl_grant()`

`repl_init_copy()`

`repl_new_log()`

`repl_pub_add()`

`repl_pub_init_image()`

`repl_pub_remove()`

`repl_publish()`

`repl_revoke()`

repl_sched_init()

repl_server()

repl_server_rename()

repl_stat()

repl_status()

repl_subscribe()

repl_sync()

repl_sync_all()

repl_text()

repl_this_server()

repl_unpublish()

repl_unsubscribe()

REPL_PUBLISH

REPL_PUBLISH — create publication on publisher

Synopsis

REPL_PUBLISH (in *publication* varchar ,
in *log_path* varchar ,
in *is_updateable* integer ,
in *sync_user* varchar) ;

Description

This function starts a publication and associates a file system file to it. The file will be used to log transaction records associated to the publication. The server will periodically start new files, so that replication log files do not grow indefinitely. New files will go to the same directory as the initial one and will have names suffixed with the date and time of their creation.

Parameters

publication
: Publication name. It must not contain spaces or special symbols.

log_path
: Full path and filename to the file where transactions to this account will be stored.

is_updateable
: If this parameter is specified and it is non-zero an updateable publication is created.

sync_user
: Local DB account used to replay replication feeds from subscribers (when publication is updateable).

Example

Example 24.323. Creating an updateable publication

This will create an updateable publication'demo-publication' with log 'demo-publication.log'. Replication feeds from subscribers will be replayed by used 'demo'.

```
SQL> DB.DBA.REPL_PUBLISH ('demo-publication', 'demo-publication.log', 1, 'demo');
```

See Also

`sub_schedule()`

`repl_disconnect()`

`repl_grant()`

`repl_init_copy()`

`repl_new_log()`

`repl_pub_add()`

`repl_pub_init_image()`

repl_pub_remove()

repl_revoke()

repl_sched_init()

repl_server()

repl_server_rename()

repl_stat()

repl_status()

repl_subscribe()

repl_sync()

repl_sync_all()

repl_text()

repl_this_server()

repl_unpublish()

repl_unsubscribe()

REPL_PUB_ADD

REPL_PUB_ADD — add item to a publication

Synopsis

REPL_PUB_ADD (in *publication* varchar ,
in *item* varchar ,
in *type* integer ,
in *mode* integer ,
in *procedure_replication_options* integer) ;

Description

This function is used to add items to a pre-existing publication and to set replication options for the published items. Operations concerning the added item will henceforth be logged into the publication's log. Performing this operation will copy the item and optionally its definition to existing subscribers.

Parameters

publication
: publication account name.

item
: dependent on type should be:

type
: type of item, can accept following types:

mode
: mode of remote copy:

procedure_replication_options
: valid only in case of Virtuoso/PL procedure:

Example

Example 24.324. Adding a table to the publication

This is to add a table Demo.demo.Orders (available in the demo DataBase) to the an existing publication named table_publication. The default flag for removal when publication is dropped is set to off. The parameter mode is NULL as it has effect only for procedures.

```
SQL> DB.DBA.REPL_PUB_ADD ('table_publication', 'Demo.demo.Orders', 2, 0, NULL);
```

See Also

`sub_schedule()`

`repl_disconnect()`

`repl_grant()`

`repl_init_copy()`

repl_new_log()

repl_pub_init_image()

repl_pub_remove()

repl_publish()

repl_revoke()

repl_sched_init()

repl_server()

repl_server_rename()

repl_stat()

repl_status()

repl_subscribe()

repl_sync()

repl_sync_all()

repl_text()

repl_this_server()

repl_unpublish()

repl_unsubscribe()

REPL_PUB_INIT_IMAGE

REPL_PUB_INIT_IMAGE — create initial image of publication on publisher

Synopsis

REPL_PUB_INIT_IMAGE (in *publication* varchar ,
in *image_file_path* varchar ,
in *bytes_per_file* integer) ;

Description

The image creation process forces checkpoint and runs in atomic mode.

If *image_file_path* does not contain path components (slashes or backslashes), image slices are stored in one of backup directories defined in virtuoso.ini using round-robin strategy.

This function is used to create image of publication to be replayed on a subscribed upon initial setup. This can be used when publication data is so large to be copied via repl_init_copy() which uses VDB operations. The image file(s) created by this function must be loaded on subscriber with replay() function in order of their creation (if image is split on to a several files).

Parameters

publication
: publication account name.

image_file_path
: full path to the image file where to store the initial image of publication.

bytes_per_file
: at which bytes count to split file into next slice.

Example

Example 24.325. Creating and loading of the initial image

This shows creating a image with inital data of the published items and store in the 'tbl_pub.log' file.

```
SQL> DB.DBA.REPL_PUB_INIT_IMAGE ('table_publication', 'tbl_pub.log', 1000000);
```

Furthermore the image can be loaded on a subscriber. Note that this command MUST be issued on the subscriber side. Note that REPL_PUB_INIT_IMAGE() will make more files when size of the initial file is greater than 1Mb, so in that case the additional files will be named tbl_pub.log.N where N is a sequence beginning from 1 and they also need to be loaded.

```
SQL> replay 'tbl_pub.log';
... if there a more than one image file load sequentially ...
SQL> replay 'tbl_pub.log.1';
...
```

See Also

`sub_schedule()`

`repl_disconnect()`

`repl_grant()`

repl_init_copy()

repl_new_log()

repl_pub_add()

repl_pub_remove()

repl_publish()

repl_revoke()

repl_sched_init()

repl_server()

repl_server_rename()

repl_stat()

repl_status()

repl_subscribe()

repl_sync()

repl_sync_all()

repl_text()

repl_this_server()

repl_unpublish()

repl_unsubscribe()

REPL_PUB_REMOVE

REPL_PUB_REMOVE — remove item from publication.

Synopsis

REPL_PUB_REMOVE (in *publication* varchar ,
in *item* varchar ,
in *type* integer ,
in *flag* integer) ;

Description

This function is used to remove an item from existing publication. It's action is performed on publisher side and depending of flag it can send replication message to subscribers to remove this from subscription.

Parameters

publication
: publication account name.

item
: dependent on type should be:

type
: type of item, can accept following types:

flag
: Behavior on the subscriber side

Example

Example 24.326. Removal of an item from publication

This will remove the table Demo.demo.Products from publication. Furthermore issuing of the replication messages for this item will be stopped and subscriber will remove this entry from local description.

```
SQL> DB.DBA.REPL_PUB_REMOVE ('demo-publication', 'Demo.demo.Products', 2, 1);
```

See Also

`sub_schedule()`

`repl_disconnect()`

`repl_grant()`

`repl_init_copy()`

`repl_new_log()`

`repl_pub_add()`

`repl_pub_init_image()`

`repl_publish()`

repl_revoke()

repl_sched_init()

repl_server()

repl_server_rename()

repl_stat()

repl_status()

repl_subscribe()

repl_sync()

repl_sync_all()

repl_text()

repl_this_server()

repl_unpublish()

repl_unsubscribe()

REPL_REVOKE

REPL_REVOKE — revoke privileges for subscription

Synopsis

REPL_REVOKE (in *publication* varchar ,
in *grantee* varchar) ;

Description

Revokes Privileges for Subscription. This is called on the publisher to revoke access to the publication from the user account on the publisher. The subscriber will no longer gain access to the publication with this account.

Parameters

publication
: The publication name.

grantee
: A valid DB account name to be refused subscription rights.

Example

Example 24.327. Access to 'table_publication' removal for SQL user account 'demo'.

This is to remove permissions to subscribe from SQL user account 'demo' for table_publication publication account.

```
SQL> DB.DBA.REPL_REVOKE ('table_publication', 'demo');
```

See Also

`sub_schedule()`

`repl_disconnect()`

`repl_grant()`

`repl_init_copy()`

`repl_new_log()`

`repl_pub_add()`

`repl_pub_init_image()`

`repl_pub_remove()`

`repl_publish()`

`repl_sched_init()`

`repl_server()`

`repl_server_rename()`

repl_stat()

repl_status()

repl_subscribe()

repl_sync()

repl_sync_all()

repl_text()

repl_this_server()

repl_unpublish()

repl_unsubscribe()

REPL_SCHED_INIT

REPL_SCHED_INIT — adds scheduled job to synchronize subscriptions

Synopsis

```
REPL_SCHED_INIT (   );
```

Description

Adds scheduled job to synchronize all subscriptions. The server will attempt to start synchronization of all non-synchronized subscriptions at a 1 minute interval. The action can be reversed by deleting the corresponding row from the SYS_SCHEDULED_EVENTS table.

Example

Example 24.328. Enabling a synchronization for all subscriptions

This will enable scheduled task to synchronize all defined subscriptions.

```
SQL> DB.DBA.REPL_SCHED_INIT ();
```

See Also

`sub_schedule()`

`repl_disconnect()`

`repl_grant()`

`repl_init_copy()`

`repl_new_log()`

`repl_pub_add()`

`repl_pub_init_image()`

`repl_pub_remove()`

`repl_publish()`

`repl_revoke()`

`repl_server()`

`repl_server_rename()`

`repl_stat()`

`repl_status()`

`repl_subscribe()`

`repl_sync()`

`repl_sync_all()`

repl_text()

repl_this_server()

repl_unpublish()

repl_unsubscribe()

REPL_SERVER

REPL_SERVER — defines a server that will participate in replication

Synopsis

REPL_SERVER (in *server_name* varchar ,
in *dsn* varchar ,
in *replication_address* varchar) ;

Description

This function defines a server that will participate in replication. The name is a symbolic name that should match the name specified as the ServerName configuration parameter in the replication section of the virtuoso.ini file of the server being defined. The address is the <host:port> where the server designated by the name is listening. The DSN is an ODBC data source name referring to the server, so that the subscriber can retrieve schema and other information. Note that replication communication itself does not take place through ODBC but that ODBC access to the publisher is required to initiate the subscription. In order to subscribe to publications from a server the server must first be declared with this function. If this function is called to define the local server, i.e. the given server name is the ServerName in the Replication section of the local ini file the function has no effect.

Parameters

server_name
: unique replication server name of publisher (specified in [Replication] -> [ServerName] section in ini file of publisher).

dsn
: data source name of publisher.

replication_address
: host:port pair of publishing server where this subscriber will connect to.

Example

Example 24.329. Adding a new publisher server

This is to define a new publisher server in the subscriber's Database. The DSN of publisher is named 'Virtuoso 1111', so it is started on the same machine on port 1111.

```
SQL> DB.DBA.REPL_SERVER ('demo-pub', 'Virtuoso 1111', '127.0.0.1:1111');
```

See Also

`sub_schedule()`

`repl_disconnect()`

`repl_grant()`

`repl_init_copy()`

`repl_new_log()`

`repl_pub_add()`

`repl_pub_init_image()`

repl_pub_remove()

repl_publish()

repl_revoke()

repl_sched_init()

repl_server_rename()

repl_stat()

repl_status()

repl_subscribe()

repl_sync()

repl_sync_all()

repl_text()

repl_this_server()

repl_unpublish()

repl_unsubscribe()

repl_server_rename

repl_server_rename — rename the publishing server instance

Synopsis

repl_server_rename (
in *old_name* varchar ,
in *new_name* varchar) ;

Description

This function is useful to rename the publishing servers data (that stored in to the replication tables) after renaming the server in virtuoso.ini file. The call of the procedure will associate the all data belong to the old server name to the current server name. It will also set the appropriate transaction levels. In case of duplicate publications (publications with the same name exists in old and new server definitions) it will reject the operation.

Parameters

old_name
: The old name of the publishing server.

new_name
: The new name of the publishing server (must be the same as ServerName from Replication section of INI file).

Example

Example 24.330. Renaming of the existing publisher server

An existing publisher 'demo-pub' server is renamed to the 'working-pub'. In order to update the subscriber descriptions we MUST issue the following:

```
SQL> DB.DBA.REPL_SERVER_RENAME ('demo-pub', 'working-pub');
```

See Also

`sub_schedule()`

`repl_disconnect()`

`repl_grant()`

`repl_init_copy()`

`repl_new_log()`

`repl_pub_add()`

`repl_pub_init_image()`

`repl_pub_remove()`

`repl_publish()`

`repl_revoke()`

repl_sched_init()

repl_server()

repl_stat()

repl_status()

repl_subscribe()

repl_sync()

repl_sync_all()

repl_text()

repl_this_server()

repl_unpublish()

repl_unsubscribe()

REPL_STAT

REPL_STAT — retrieve status of all subscriptions and publications

Synopsis

```
REPL_STAT (   );
```

Description

Retrieves status of all subscriptions and publications. This function is for interactive use (via ISQL tool).

Example

Example 24.331. Retrieving of the replication status

This is to show status of the subscriptions. The server shows the name of publisher, account is a name of the publication account, level is a current level of the publication and stat is a current status of the publication/subscription. Note that status for local publications are shown always as OFF, it is to show only subscription status.

```
SQL> DB.DBA.REPL_STAT ();
server                    account                   level       stat
VARCHAR                   VARCHAR                   INTEGER     INTEGER
_______________________________________________________________________________

my-rep                    my-rep                    0           OFF
demo-rep1                 demo_tb                   3           IN SYNC
```

See Also

`sub_schedule()`

`repl_disconnect()`

`repl_grant()`

`repl_init_copy()`

`repl_new_log()`

`repl_pub_add()`

`repl_pub_init_image()`

`repl_pub_remove()`

`repl_publish()`

`repl_revoke()`

`repl_sched_init()`

`repl_server()`

`repl_server_rename()`

`repl_status()`

repl_subscribe()

repl_sync()

repl_sync_all()

repl_text()

repl_this_server()

repl_unpublish()

repl_unsubscribe()

repl_status

repl_status — returns status of a published or subscribed publication

Synopsis

repl_status (in *publisher* varchar ,
in *publication* varchar ,
out *level* integer ,
out *stat* integer) ;

Description

Given a publisher and publication name this returns the status of the publication on the local server.

Parameters

publisher
: Name of the publisher.

publication
: Name of the publication.

level
: If the publisher is the name of the local server this returns the next transaction number to be assigned to a transaction as the level output parameter.

stat
: The stat output parameter reflects the current state of the subscription. If the publisher is the local server the stat is always 0. Otherwise it has the following possible values:

Example

Example 24.332. Retrieving the subscription status

This example shows an analogue of the REPL_STAT() function.

```
create procedure MY_REPL_STAT ()
{
  declare server, account, status varchar;
  declare level, stat integer;
  status := vector ('OFF', 'SYNCING', 'IN SYNC', 'REMOTE DISCONNECTED', 'DISCONNECTED', 'TO DISCONNECT');
  result_names (server, account, level, stat);
  for select SERVER, ACCOUNT from DB.DBA.SYS_REPL_ACCOUNTS do
    {
      repl_status (SERVER, ACCOUNT, level, stat);
        result (SERVER, ACCOUNT, level, aref (status, stat));
    }
};
```

See Also

`sub_schedule()`

`repl_disconnect()`

`repl_grant()`

repl_init_copy()

repl_new_log()

repl_pub_add()

repl_pub_init_image()

repl_pub_remove()

repl_publish()

repl_revoke()

repl_sched_init()

repl_server()

repl_server_rename()

repl_stat()

repl_subscribe()

repl_sync()

repl_sync_all()

repl_text()

repl_this_server()

repl_unpublish()

repl_unsubscribe()

REPL_SUBSCRIBE

REPL_SUBSCRIBE — add a subscription

Synopsis

```
REPL_SUBSCRIBE ( in server_name varchar ,
                 in publication varchar ,
                 in dav_user varchar ,
                 in dav_group varchar ,
                 in replication_user varchar ,
                 in replication_password varchar ,
                 in sync_user varchar ) ;
```

Description

This function is used to subscribe to an existing publication, and to specify the local WebDAV owner for replicated WebDAV content. Before making a subscription the `repl_server()` function must be called in order to define the publishing server. After making a subscription it becomes off-line awaiting synchronization from a scheduled task or call to the `repl_sync()` function. Also the initial data of the subscription will be not loaded until `repl_init_copy()` is called or the initial image has been loaded.

Parameters

server_name
: target publisher server name.

publication
: publication account name.

dav_user
: A valid local WebDAV user account name that will own local copy. If this is null 'REPLICATION' user account will be created that will be disabled by default.

dav_group
: A valid local WebDAV group name to own the local copy.

replication_user
: Is used for authentication on the publisher. This should be a valid DB account on publisher.

replication_password
: Is used for authentication on the publisher. This should be valid password for replication_user on publisher.

sync_user
: A local DB account used to replay replication feeds.

Example

Example 24.333. Subscribing to a publication

This example shows making a subscription to the 'table_publication' published on a 'demo-srv' server. The WebDAV items will belong to the WebDAV administrator.

```
SQL> DB.DBA.REPL_SUBSCRIBE ('demo-srv','demo-publication', 'dav', 'dav', 'demo', 'demo');
```

See Also

sub_schedule()

repl_disconnect()

repl_grant()

repl_init_copy()

repl_new_log()

repl_pub_add()

repl_pub_init_image()

repl_pub_remove()

repl_publish()

repl_revoke()

repl_sched_init()

repl_server()

repl_server_rename()

repl_stat()

repl_status()

repl_sync()

repl_sync_all()

repl_text()

repl_this_server()

repl_unpublish()

repl_unsubscribe()

repl_sync

repl_sync — starts the syncing process against an existing subscription

Synopsis

```
repl_sync ( in publisher varchar ,
            in publication varchar ,
            in uid varchar ,
            in password varchar ) ;
```

Description

This starts the syncing process against an existing subscription. The function returns as soon as the request has been successfully sent. The synchronization will take place in the background. If the syncing process is already underway or if the subscriber is already in sync and connected to the publisher this function has no effect. If there is no connection to the publisher at the time of calling this function and the connection fails and an error is immediately signalled. To initiate a synchronization a valid SQL account must be specified. Also the account must have rights to the publication unless publisher DBA's account is used to connect.

Parameters

publisher
: The server name of the publisher.

publication
: The name assigned to the publication.

uid
: SQL account with rights to the publication, or publishers DBA account credentials will be required.

password
: SQL account password.

Example

Example 24.334. Synchronizing a subscription

The following shows requesting a sync from publisher for an existing subscription. The account for authentication is a 'repl_user' SQL account.

```
SQL> repl_sync ('demo-srv', 'demo-publication', 'repl_user', 'repl_passwd');
```

See Also

`sub_schedule()`

`repl_disconnect()`

`repl_grant()`

`repl_init_copy()`

`repl_new_log()`

`repl_pub_add()`

repl_pub_init_image()

repl_pub_remove()

repl_publish()

repl_revoke()

repl_sched_init()

repl_server()

repl_server_rename()

repl_stat()

repl_status()

repl_subscribe()

repl_sync_all()

repl_text()

repl_this_server()

repl_unpublish()

repl_unsubscribe()

repl_sync_all

repl_sync_all — synchronize all subscriptions

Synopsis

```
repl_sync_all (   );
```

Description

This function is used to synchronize all subscriptions. It make a synchronization requests to the all publisher and will return immediately after that. The status of subscriptions can be tested with repl_stat() function.

Example

Example 24.335. Synchronizing all subscriptions

This will start synchronization of all subscriptions.

```
SQL> repl_sync_all ();
```

See Also

sub_schedule()

repl_disconnect()

repl_grant()

repl_init_copy()

repl_new_log()

repl_pub_add()

repl_pub_init_image()

repl_pub_remove()

repl_publish()

repl_revoke()

repl_sched_init()

repl_server()

repl_server_rename()

repl_stat()

repl_status()

repl_subscribe()

repl_sync()

repl_text()

repl_this_server()

repl_unpublish()

repl_unsubscribe()

repl_text

repl_text — adds a SQL statement to the log of the replication

Synopsis

repl_text (in *publication* varchar ,
in *sqltext* varchar ,
in *a-1* any ,
...) ;

Description

This SQL function adds the SQL statement to the log of the publication. The statement will typically be a procedure call but can be any SQL statement. There can be a parameters, which are bound to ?'s in the statement's text from left to right. There is no restriction on the number of parameters.

Parameters

publication
: The name of the target publication

sqltext
: sql expression with unnamed parameters if needed.

a-1 ... a-n
: parameter value, in order as in sqltext parameter.

Example

Example 24.336. Logging a SQL statement into a publication

Suppose on subscriber side we have a table 'foo' with two columns (integer and varchar). This statement will send a replication message to the subscribers to insert a row into the table 'foo'.

```
SQL> repl_text ('demo-publication', 'insert into foo values (?, ?)', 1, 'this is a test');
```

See Also

`sub_schedule()`

`repl_disconnect()`

`repl_grant()`

`repl_init_copy()`

`repl_new_log()`

`repl_pub_add()`

`repl_pub_init_image()`

`repl_pub_remove()`

`repl_publish()`

repl_revoke()

repl_sched_init()

repl_server()

repl_server_rename()

repl_stat()

repl_status()

repl_subscribe()

repl_sync()

repl_sync_all()

repl_this_server()

repl_unpublish()

repl_unsubscribe()

repl_this_server

repl_this_server — returns calling servers name

Synopsis

repl_this_server ( );

Description

Returns calling servers name.

Return Types

Returns the calling server's logical name, set by the ServerName entry in the Replication section of its virtuoso.ini file.

Example

Example 24.337. Local replication server name

The example statement shows name of the current replication server (defined in the INI file).

```
SQL> select repl_this_server();
callret
VARCHAR
_______________________________________________________________________________

demo-srv
```

See Also

sub_schedule()

repl_disconnect()

repl_grant()

repl_init_copy()

repl_new_log()

repl_pub_add()

repl_pub_init_image()

repl_pub_remove()

repl_publish()

repl_revoke()

repl_sched_init()

repl_server()

repl_server_rename()

repl_stat()

repl_status()

repl_subscribe()

repl_sync()

repl_sync_all()

repl_text()

repl_unpublish()

repl_unsubscribe()

REPL_UNPUBLISH

REPL_UNPUBLISH — drop publication on publisher

Synopsis

REPL_UNPUBLISH (in *publication* varchar) ;

Description

This function is used to remove from current replication set an existing publication. The replication messages for all existing items will be stopped, so the last replication message will instruct the subscribers that this publication is dropped. On subscriber side depending of the copy mode of items they can be removed or not, but the description entry for that publication will be removed explicitly. Any existing grants to the publication being dropped will be revoked.

Parameters

publication
: publication account name.

Example

Example 24.338. Dropping a publication

The following statement will remove the publication 'demo-publication' account from current replication set.

```
SQL> DB.DBA.REPL_UNPUBLISH ('demo-publication');
```

See Also

`sub_schedule()`

`repl_disconnect()`

`repl_grant()`

`repl_init_copy()`

`repl_new_log()`

`repl_pub_add()`

`repl_pub_init_image()`

`repl_pub_remove()`

`repl_publish()`

`repl_revoke()`

`repl_sched_init()`

`repl_server()`

`repl_server_rename()`

repl_stat()

repl_status()

repl_subscribe()

repl_sync()

repl_sync_all()

repl_text()

repl_this_server()

repl_unsubscribe()

REPL_UNSUBSCRIBE

REPL_UNSUBSCRIBE — drop subscription

Synopsis

REPL_UNSUBSCRIBE (in *server_name* varchar ,
in *publication* varchar ,
in *item* varchar) ;

Description

This function is used to stop receiving a replication messages from a publisher server for a item or whole subscription. It will be invoked automatically when a publication is dropped. The subscriber can also invoke this function to stop receiving replication messages. The existing items can be dropped or not depending on the copy mode flag. The description entries for that subscription will be removed. There is no "undo" ability. To temporally halt the replication messages you can use repl_disconnect().

Parameters

server_name
: Name of the target publishing server.

publication
: The name of the publication.

item
: The item to removed. NULL can be supplied to remove all items.

Example

Example 24.339. Unsubscribing from a publication.

This is to stop replication on subscription 'demo-publication' from publisher server 'demo-srv'.

```
SQL> DB.DBA.REPL_UNSUBSCRIBE ('demo-srv', 'demo-publication', NULL);
```

See Also

`sub_schedule()`

`repl_disconnect()`

`repl_grant()`

`repl_init_copy()`

`repl_new_log()`

`repl_pub_add()`

`repl_pub_init_image()`

`repl_pub_remove()`

`repl_publish()`

repl_revoke()

repl_sched_init()

repl_server()

repl_server_rename()

repl_stat()

repl_status()

repl_subscribe()

repl_sync()

repl_sync_all()

repl_text()

repl_this_server()

repl_unpublish()

repl_purge

repl_purge — purges transactional replication logs for specified account

Synopsis

repl_purge (in *server* varchar ,
in *account* varchar) ;

Description

This procedure purges transactional replication logs for specified account. Logs whose start replication level lags more than REPL_MAX_DELTA (1000000000) transactions behind current replication level of a specified account are removed. After `repl_purge()` is finished next `repl_purge()` run is scheduled using SYS_SCHEDULED_EVENT facility.

Account sync requests from subscribers are delayed while `repl_purge()` is running for this account.

Parameters

server
: The server name of the publisher.

account
: The name assigned to the publication.

Example

Example 24.340. Purging transactional replication logs

The following shows purging transactional replication logs for pushback account for updateable subscription 'demopub' from 'demoserver':

```
SQL> repl_purge ('demoserver', '!demopub');
```

See Also

`repl_publish()`

`repl_subscribe()`

`repl_unpublish()`

`repl_unsubscribe()`

REPL_CREATE_SNAPSHOT_PUB

REPL_CREATE_SNAPSHOT_PUB — Create bi-directional snapshot publication

Synopsis

```
None REPL_CREATE_SNAPSHOT_PUB ( in item varchar ,
                                in type integer ) ;
```

Description

This procedure should be used to create a bi-directional snapshot publication. If the *item* parameter is a table then this procedure creates an updateable snapshot log and generates an updating procedure.

Parameters

item
: Item is a DAV collection or table name.

type
: Type is used to denote the type of Item: 1 if item is a DAV collection, or 2 if item is a table name.

Return Types

None.

Examples

Example 24.341. Creating bi-directional snapshot publication

The following statement creates bi-directional snapshot publication of table 'Demo.demo.Shippers':

```
SQL> DB.DBA.REPL_CREATE_SNAPSHOT_PUB ('Demo.demo.Shippers', 2);
```

The following statement creates bi-directional snapshot publication of DAV collection '/DAV/doc':

```
SQL> DB.DBA.REPL_CREATE_SNAPSHOT_PUB ('/DAV/doc', 1);
```

See Also

REPL_CREATE_SNAPSHOT_SUB()

REPL_DROP_SNAPSHOT_SUB()

REPL_DROP_SNAPSHOT_PUB()

REPL_INIT_SNAPSHOT()

REPL_UPDATE_SNAPSHOT()

REPL_SNP_SERVER()

REPL_SERVER_NAME()

REPL_CREATE_SNAPSHOT_SUB

REPL_CREATE_SNAPSHOT_SUB — Create bi-directional snapshot subscription to an existing publication.

Synopsis

```
None REPL_CREATE_SNAPSHOT_SUB ( in server varchar ,
                                in item varchar ,
                                in type integer ) ;
```

Description

This procedure should be used to create a bi-directional snapshot subscription.

Note:
This function should be run on publisher.

Parameters

server
: The name of the server defined with REPL_SNP_SERVER() . The name of the server can be obtained using REPL_SERVER_NAME() function.

item
: Item is a DAV collection or table name.

type
: Type is used to denote the type of Item: 1 if item is a DAV collection, or 2 if item is a table name.

Return Types

None.

Examples

Example 24.342. Creating bi-directional snapshot subscription

The following statement creates bi-directional snapshot subscription of server with DSN 'localhost:1121' for table 'Demo.demo.Shippers':

```
SQL> DB.DBA.REPL_CREATE_SNAPSHOT_SUB (REPL_SERVER_NAME ('localhost:1121'), 'Demo.demo.Shippers', 2);
```

The following statement creates bi-directional snapshot subscription of server with DSN 'localhost:1121' for DAV collection '/DAV/doc':

```
SQL> DB.DBA.REPL_CREATE_SNAPSHOT_SUB (REPL_SERVER_NAME ('localhost:1121'), '/DAV/doc', 1);
```

See Also

`REPL_CREATE_SNAPSHOT_PUB()`

`REPL_DROP_SNAPSHOT_SUB()`

`REPL_DROP_SNAPSHOT_PUB()`

`REPL_INIT_SNAPSHOT()`

REPL_UPDATE_SNAPSHOT()

REPL_SNP_SERVER()

REPL_SERVER_NAME()

REPL_DROP_SNAPSHOT_SUB

REPL_DROP_SNAPSHOT_SUB — Drops a subscription to a bi-directional snapshot publication

Synopsis

```
None REPL_DROP_SNAPSHOT_SUB ( in server varchar ,
                              in item varchar ,
                              in type integer ) ;
```

Description

This procedure drops a subscription to a snapshot publication.

Note:
This function should be run on publisher.

Parameters

server
: The name of the server defined with REPL_SNP_SERVER() . The name of the server can be obtained using REPL_SERVER_NAME() function.

item
: Item is a DAV collection or table name.

type
: Type is used to denote the type of Item: 1 if item is a DAV collection, or 2 if item is a table name.

Return Types

None.

Examples

Example 24.343. Dropping bi-directional snapshot subscription

The following statement drops bi-directional snapshot subscription of server with DSN 'localhost:1121' for table 'Demo.demo.Shippers':

```
SQL> DB.DBA.REPL_DROP_SNAPSHOT_SUB (REPL_SERVER_NAME ('localhost:1121'), 'Demo.demo.Shippers', 2);
```

See Also

REPL_CREATE_SNAPSHOT_SUB()

REPL_CREATE_SNAPSHOT_PUB()

REPL_DROP_SNAPSHOT_PUB()

REPL_INIT_SNAPSHOT()

REPL_UPDATE_SNAPSHOT()

REPL_SNP_SERVER()

```
REPL_SERVER_NAME()
```

REPL_DROP_SNAPSHOT_PUB

REPL_DROP_SNAPSHOT_PUB — Drop bi-directional snapshot publication

Synopsis

```
None REPL_DROP_SNAPSHOT_PUB ( in item varchar ,
                              in type integer ) ;
```

Description

This procedure drops a snapshot publication.

Parameters

item
: Item is a DAV collection or table name.

type
: Type is used to denote the type of Item: 1 if item is a DAV collection, or 2 if item is a table name.

Return Types

None.

Examples

Example 24.344. Dropping bi-directional snapshot publication

The following statement drops bi-directional snapshot publication of table 'Demo.demo.Shippers':

```
SQL> DB.DBA.REPL_DROP_SNAPSHOT_PUB ('Demo.demo.Shippers', 2);
```

See Also

REPL_CREATE_SNAPSHOT_SUB()

REPL_CREATE_SNAPSHOT_PUB()

REPL_DROP_SNAPSHOT_SUB()

REPL_INIT_SNAPSHOT()

REPL_UPDATE_SNAPSHOT()

REPL_SNP_SERVER()

REPL_SERVER_NAME()

REPL_INIT_SNAPSHOT

REPL_INIT_SNAPSHOT — Initializes a bi-directional snapshot publication

Synopsis

```
None REPL_INIT_SNAPSHOT ( in server varchar ,
                          in item varchar ,
                          in type integer ,
                          in how_many integer ) ;
```

Description

This procedure should be used to initialize a bi-directional snapshot publication.

Note:
This function should be run on publisher.

Parameters

server
: The name of the server defined with REPL_SNP_SERVER() . The name of the server can be obtained using REPL_SERVER_NAME() function.

item
: Item is a DAV collection or table name.

type
: Type is used to denote the type of Item: 1 if item is a DAV collection, or 2 if item is a table name.

how_many
: This parameter specifies the number of copied rows per transaction during doing initial copy in table case (when type is 2).

Return Types

None.

Examples

Example 24.345. Initializing bi-directional snapshot subscription

The following statement does initial copy of data for bi-directional snapshot subscription of server with DSN 'localhost:1121' for table 'Demo.demo.Shippers':

```
SQL> DB.DBA.REPL_INIT_SNAPSHOT (REPL_SERVER_NAME ('localhost:1121'), 'Demo.demo.Shippers', 2);
```

See Also

REPL_CREATE_SNAPSHOT_SUB()

REPL_CREATE_SNAPSHOT_PUB()

REPL_DROP_SNAPSHOT_SUB()

```
REPL_DROP_SNAPSHOT_PUB()

REPL_UPDATE_SNAPSHOT()

REPL_DROP_SNAPSHOT_PUB()

REPL_DROP_SNAPSHOT_PUB()
```

REPL_UPDATE_SNAPSHOT

REPL_UPDATE_SNAPSHOT — Updates a bi-directional snapshot publication

Synopsis

```
None REPL_UPDATE_SNAPSHOT ( in item varchar ,
                            in type integer ) ;
```

Description

This procedure should be used to update a bi-directional snapshot publication. This procedure pulls all the updates from subscribers (with conflict resolution) and then pushes all the updates from publisher to all the subscribers.

Note:
This function should be run on the publisher.

Parameters

item
: Item is a DAV collection or table name.

type
: Type is used to denote the type of Item: 1 if item is a DAV collection, or 2 if item is a table name.

Return Types

None.

Examples

Example 24.346. Updating bi-directional snapshot

The following statement updates bi-directional snapshot publication of table 'Demo.demo.Shippers'.

```
SQL> REPL_UPDATE_SNAPSHOT ('Demo.demo.Shippers', 2);
```

See Also

REPL_CREATE_SNAPSHOT_SUB()

REPL_CREATE_SNAPSHOT_PUB()

REPL_DROP_SNAPSHOT_SUB()

REPL_DROP_SNAPSHOT_PUB()

REPL_INIT_SNAPSHOT()

REPL_SNP_SERVER()

REPL_SERVER_NAME()

REPL_SNP_SERVER

REPL_SNP_SERVER — Define bi-directional snapshot replication server name

Synopsis

```
varchar REPL_SNP_SERVER ( in dsn varchar ,
                          in uid varchar ,
                          in pwd varchar ) ;
```

Description

This function should be used to define a server for bi-directional snapshot replication.

Parameters

dsn
: Dsn is a data source name.

uid
: Uid is data source user name.

pwd
: Uid is data source password.

Return Types

REPL_SNP_SERVER() returns remote server name which can be used in calls to `REPL_CREATE_SNAPSHOT_SUB()`, `REPL_DROP_SNAPSHOT_SUB()` and `REPL_INIT_SNAPSHOT_SUB()` functions.

Examples

Example 24.347. Defining bi-directional snapshot replication server

The following statement defines bi-directional replication server for server with DSN 'localhost:1121':

```
SQL> REPL_SNP_SERVER ('localhost:1121', 'dba', 'dba');
```

See Also

`REPL_CREATE_SNAPSHOT_SUB()`

`REPL_CREATE_SNAPSHOT_PUB()`

`REPL_DROP_SNAPSHOT_SUB()`

`REPL_DROP_SNAPSHOT_PUB()`

`REPL_INIT_SNAPSHOT()`

`REPL_DROP_SNAPSHOT_PUB()`

`REPL_SERVER_NAME()`

REPL_SERVER_NAME

REPL_SERVER_NAME — Return bi-directional snapshot replication server name for specified DSN

Synopsis

```
varchar REPL_SERVER_NAME ( in dsn varchar );
```

Description

This function should be used to determine bi-directional snapshot replication server name.

Parameters

item
: Dsn is a data source name.

Return Types

REPL_SERVER_NAME() returns remote server name which can be used in calls to `REPL_CREATE_SNAPSHOT_SUB()`, `REPL_DROP_SNAPSHOT_SUB()` and `REPL_INIT_SNAPSHOT_SUB()` functions.

Examples

Example 24.348. Obtaining replication server name

The following statement demonstrates obtaining replication server name for server with DSN 'localhost:1121':

```
SQL> select REPL_SERVER_NAME ('localhost:1121');
callret
VARCHAR
_______________________________________________________________________________

demoserver2

1 Rows. -- 4 msec.
```

See Also

`REPL_CREATE_SNAPSHOT_SUB()`

`REPL_CREATE_SNAPSHOT_PUB()`

`REPL_DROP_SNAPSHOT_SUB()`

`REPL_DROP_SNAPSHOT_PUB()`

`REPL_INIT_SNAPSHOT()`

`REPL_DROP_SNAPSHOT_PUB()`

`REPL_SNP_SERVER()`

REPL_ADD_CR

REPL_ADD_CR — Creates conflict resolver for bi-directional transactional replication

Synopsis

```
None REPL_ADD_CR ( in tbl varchar ,
                   in name_suffix varchar ,
                   in type char ,
                   in order integer ,
                   in class varchar ,
                   in col varchar ) ;
```

Description

Simple conflict resolvers can be generated by calling this function.

Parameters

tbl
: Name of table for which conflict resolved is required.

name_suffix
: Conflict resolver name suffix.

type
: The resolved type, one of: ('I', 'U' OR 'D').

order
: Resolver order number.

class
: The kind of resolver. Class can be one of the following:

col
: This should specify the column name if class is not one of 'pub_wins', 'sub_wins' or 'custom'.

Return Types

None.

Examples

Example 24.349. Conflict Resolver

Suppose we have the following table:

```
create table items(
  item_id integer primary key,

  name varchar,
  count decimal
);
```

A 'U' conflict resolver that increments items.count in publisher table can be defined with the following statement:

```
REPL_ADD_CR('DB.DBA.items', 'count', 'U', 10, 'add', 'count');
```

REPL_ADD_DAV_CR

REPL_ADD_DAV_CR — Creates conflict resolver for bi-directional replication in DAV

Synopsis

```
None REPL_ADD_DAV_CR ( in col varchar ,
                       in name_suffix varchar ,
                       in order integer ,
                       in class varchar ) ;
```

Description

Simple conflict resolvers for DAV bi-directional replication can be generated by calling this function.

Parameters

col
: Collection for which to generate a conflict resolver for.

name_suffix
: Conflict resolver name suffix.

order
: Resolver order number.

class
: class can be one of the following:

Return Types

None.

Examples

Example 24.350. Conflict Resolver

A conflict resolver that chooses a resource with minimal creation time can be defined with the following statement:

```
REPL_ADD_DAV_CR('/DAV/regress', 'min_ctime', 100, 'min_ctime');
```

See Also

`REPL_ADD_SNAPSHOT_CR()`

REPL_ADD_SNAPSHOT_CR

REPL_ADD_SNAPSHOT_CR — Creates conflict resolver for bi-directional replication

Synopsis

```
None REPL_ADD_SNAPSHOT_CR ( in tbl varchar ,
                            in name_suffix varchar ,
                            in type char ,
                            in order integer ,
                            in class varchar ,
                            in col varchar ) ;
```

Description

Simple conflict resolvers can be generated by calling this function.

Parameters

tbl
: Name of table for which conflict resolved is required.

name_suffix
: Conflict resolver name suffix.

type
: The resolved type, one of: ('I', 'U' OR 'D').

order
: Resolver order number.

class
: The kind of resolver. Class can be one of the following:

col
: This should specify the column name if class is not one of 'pub_wins', 'sub_wins' or 'custom'.

Return Types

None.

Examples

Example 24.351. Conflict Resolver

Suppose we have the following table:

```
create table items(
  item_id integer primary key,

  name varchar,
  price decimal
);
```

A 'U' conflict resolver that averages items.price in publisher table can be defined with the following statement:

```
REPL_ADD_SNAPSHOT_CR('DB.DBA.items', 'count', 'U', 10, 'ave', 'price');
```

See Also

REPL_ADD_DAV_CR()

RDF_REPL_START

RDF_REPL_START — Enables RDF Replication.

Synopsis

```
RDF_REPL_START (   );
```

Description

This function is called to enable (start) RDF Publication.

Example

Example 24.352. Enable RDF Publication

```
SQL> DB.DBA.RDF_REPL_START();
```

See Also

RDF_REPL_STOP()

RDF_REPL_SYNC()

RDF_REPL_GRAPH_INS()

RDF_RDF_REPL_GRAPH_DEL()

sub_schedule()

repl_disconnect()

repl_grant()

repl_new_log()

repl_pub_add()

repl_pub_init_image()

repl_pub_remove()

repl_publish()

repl_revoke()

repl_sched_init()

repl_server()

repl_server_rename()

repl_stat()

repl_status()

repl_subscribe()

```
repl_sync()

repl_sync_all()

repl_text()

repl_this_server()

repl_unpublish()

repl_unsubscribe()
```

RDF_REPL_STOP

RDF_REPL_STOP — Stops the RDF replication at the publishing side

Synopsis

RDF_REPL_STOP (in *quiet* integer := 0) ;

Description

This function stops the RDF replication at the publishing side. It calls `repl_unpublish` but does not make empty reates replication "publication" named '__rdf_repl' and makes a log file '__rdf_repl.log' to record changes in replicated graphs.

Parameters

quiet
: Sets silent operation mode on / off i.e. 1/ 0. Default is 0 -- errors are shown. If set to 1 -- no errors are shown.

Example

Example 24.353. Enable RDF Publication

```
SQL> DB.DBA.RDF_REPL_STOP ();
```

See Also

`RDF_REPL_START()`

`RDF_REPL_SYNC()`

`RDF_REPL_GRAPH_INS()`

`RDF_REPL_GRAPH_DEL()`

`sub_schedule()`

`repl_disconnect()`

`repl_grant()`

`repl_new_log()`

`repl_pub_add()`

`repl_pub_init_image()`

`repl_pub_remove()`

`repl_publish()`

`repl_revoke()`

`repl_sched_init()`

`repl_server()`

repl_server_rename()

repl_stat()

repl_status()

repl_subscribe()

repl_sync()

repl_sync_all()

repl_text()

repl_this_server()

repl_unpublish()

repl_unsubscribe()

RDF_REPL_SYNC

RDF_REPL_SYNC — Starts the syncing process against an existing RDF subscription

Synopsis

RDF_REPL_SYNC (in *publisher* varchar ,
in *user* varchar ,
in *user_pwd* varchar) ;

Description

This function starts the syncing process against an existing RDF subscription. It will not only initial synchronisation but also wait for the end of subscription to guarantee that the total effect of INSERT and DELETE operations is correct even if these operations were made in an order that differs from the original one.

Parameters

publisher
: The server name of the publisher.

user
: SQL account with rights to the publication, or publishers DBA account credentials will be required.

user_pwd
: SQL account password.

Example

Example 24.354. Enable RDF Publication

```
SQL> DB.DBA.RDF_REPL_SYNC ('demo-rdf-srv', 'repl_user', 'repl_passwd');
```

See Also

`RDF_REPL_START()`

`RDF_REPL_STOP()`

`RDF_REPL_GRAPH_INS()`

`RDF_REPL_GRAPH_DEL()`

`sub_schedule()`

`repl_disconnect()`

`repl_grant()`

`repl_new_log()`

`repl_pub_add()`

`repl_pub_init_image()`

`repl_pub_remove()`

repl_publish()

repl_revoke()

repl_sched_init()

repl_server()

repl_server_rename()

repl_stat()

repl_status()

repl_subscribe()

repl_sync()

repl_sync_all()

repl_text()

repl_this_server()

repl_unpublish()

repl_unsubscribe()

RDF_REPL_GRAPH_INS

RDF_REPL_GRAPH_INS — Inserts graph to RDF publication.

Synopsis

RDF_REPL_GRAPH_INS (in *graph* varchar) ;

Description

This function inserts graph to enabled RDF Replication. The graphs in this replication can be more than one.

Parameters

graph
: Graph URI to be inserted.

Example

Example 24.355. Insert graph to RDF replication

```
SQL> DB.DBA.RDF_RDF_REPL_GRAPH_INS ('http://test.org');
```

Example 24.356. Replicate all graphs

To replicate all graphs (except the system graph http://www.openlinksw.com/schemas/virtrdf#), one should use http://www.openlinksw.com/schemas/virtrdf#rdf_repl_all as graph IRI:

```
SQL> DB.DBA.RDF_RDF_REPL_GRAPH_INS ('http://www.openlinksw.com/schemas/virtrdf#rdf_repl_all');
```

See Also

RDF_REPL_START()

RDF_REPL_STOP()

RDF_REPL_SYNC()

RDF_RDF_REPL_GRAPH_DEL()

sub_schedule()

repl_disconnect()

repl_grant()

repl_new_log()

repl_pub_add()

repl_pub_init_image()

repl_pub_remove()

repl_publish()

repl_revoke()

repl_sched_init()

repl_server()

repl_server_rename()

repl_stat()

repl_status()

repl_subscribe()

repl_sync()

repl_sync_all()

repl_text()

repl_this_server()

repl_unpublish()

repl_unsubscribe()

RDF_REPL_GRAPH_DEL

RDF_REPL_GRAPH_DEL — Deletes graph from RDF publication.

Synopsis

RDF_REPL_GRAPH_DEL (in *graph* varchar);

Description

This function deletes graph from RDF Replication.

Parameters

graph
: Graph URI to be deleted.

Example

Example 24.357. Delete graph from RDF replication

```
SQL> DB.DBA.RDF_RDF_REPL_GRAPH_DEL ('http://test.org');
```

See Also

RDF_REPL_START()

RDF_REPL_STOP()

RDF_REPL_SYNC()

RDF_REPL_GRAPH_INS()

sub_schedule()

repl_disconnect()

repl_grant()

repl_new_log()

repl_pub_add()

repl_pub_init_image()

repl_pub_remove()

repl_publish()

repl_revoke()

repl_sched_init()

repl_server()

repl_server_rename()

repl_stat()

repl_status()

repl_subscribe()

repl_sync()

repl_sync_all()

repl_text()

repl_this_server()

repl_unpublish()

repl_unsubscribe()

result

result — Sends one row of results to the calling client.

Synopsis

result (in *col_1* any ,
.. ,
in *col_n* any) ;

Description

The result_names() predefines variables to be used in a result set to follow. The variables must be previously declared, from which the column data types are ascertained. This assigns the meta data but does not send any results. The result() function sends its parameters as a single row of results. These parameters should be compatible with those in the previous result_names() . The end_results() function can be used to separate multiple result sets. The result_names() can then be used to alter the structure of the next result set.

The result_names() call can be omitted if the application already knows what columns and their types are to be returned.

Parameters

col_1, ..., col_n
: The data to be sent to the client.

Examples

Example 24.358.

This example demonstrates outputting multiple result sets from a stored procedure. This procedure can be entered using ISQL but ISQL does not support multiple result sets. To view the output you can use the the sample application odbc test or another application that supports multiple result sets. The RES procedure below returns two result sets -- the first has two rows and the second has four rows. The result_names function gives each result column a type and title. This can be omitted if the application knows which columns to expect. The result function sends the actual result row. The end_result function indicates that a new result set will begin. These functions can be used inside loops or subprocedures, thus allowing with one call to yield a variable number of result sets.

```
CREATE PROCEDURE RES (IN I INTEGER)
{
  DECLARE STR, STR2 VARCHAR;

  RESULT_NAMES (I, STR);
  RESULT (I, 'FIRST');
  RESULT (I + 1, 'SECOND');
  RESULT_NAMES (I, STR, STR2);

  END_RESULT ();

  RESULT (I, 'FIRST');
  RESULT (I + 1, 'SECOND');
  RESULT (I + 2, 'THIRD', 'FF');
  RESULT (I + 3, 'FOURTH', 'FF');

  RETURN 1;
}
```

See Also

`result_names()`

`end_result()`

result_names

result_names

Synopsis

```
result_names ( in var_1 ,
               ... ,
               in var_n ) ;
```

Description

The `result_names()` predefines variables to be used in a result set to follow. The variables must be previously declared, from which the column data types are ascertained. This assigns the meta data but does not send any results. The `result()` function sends its parameters as a single row of results. These parameters should be compatible with those in the previous `result_names()` . The `end_results()` function can be used to separate multiple result sets. The `result_names()` can then be used to alter the structure of the next result set.

The `result_names()` call can be omitted if the application already knows what columns and their types are to be returned.

Parameters

var_1, ..., var_n
: The column variables previously declared or otherwise.

See Also

`result()`

`end_result()`

rexecute

rexecute — execute a SQL statement on a remote DSN

Synopsis

```
rexecute ( in dsn varchar ,
           in sql_stmt varchar ,
           out sql_state varchar ,
           out error_message varchar ,
           in in_params vector ,
           out num_cols integer ,
           out stmt_meta vector ,
           out result_set vector ,
           out cursor_handle long ) ;
```

Description

This function can be used to execute SQL on a remote data source directly.

The *result_set* parameter is useful for obtaining a result-set quickly and easily. However, if the result-set is going to be large, this comes at a cost in terms of time and resources, particularly memory, since Virtuoso will have to obtain all results from the statement and build the result-set arrays in memory before returning back to the caller.

A more efficient way is to obtain a cursor handle and iterate through the result set one row at a time:

Use `rexecute()` to get the cursor handle.
Iterate over the results with `rnext()`
If `rmoreresults()` does not return an error or `SQL_NO_DATA_FOUND` continue the loop with `rnext()` , otherwise close the cursor with `rclose()`

To keep Virtuoso from obtaining the whole result set from the remote, pass NULL as the *result_set* parameter when calling rexecute.

`Rexecute()` supports IN parameters and can also support INOUT and OUT parameters. If INOUT and OUT parameters are to be used then cursors should not be used with this function (as the values of output parameters in ODBC are not guaranteed to be set before SQLMoreResults() returns SQL_NO_DATA_FOUND). `Rexecute` supports INOUT and OUT parameters by extending the *in_params* vector and requiring it to be a PL variable so values can be set back to it.

Unless explicitly granted, only the DBA group is permitted to use the `rexecute()` to maintain security. Caution is required here since any user granted use of `rexecute()` has full control of the remote data source set-up by the DBA, however limited to the overall abilities of the remote user on the remote data source. Users can be granted and denied access to this function using the following commands:

```
GRANT REXECUTE ON '<attached_dsn_name>' TO <user_name>
REVOKE REXECUTE ON '<attached_dsn_name>' FROM <user_name>
```

Note:
`rstmtexec()` provides a short-hand to this function.

Parameters

dsn

The data source where the SQL statement should be executed. You must make sure that you have already defined the data source using the vd_remote_data_source function or by attaching tables from it.

sql_stmt

the SQL statement to execute.

sql_state
: A varchar containing the SQL State returned from the remote data source.

error_message
: A varchar containing any error message returned from the remote.

in_params
: A vector of parameters to the statement if the executed statement has parameters. IN input parameters are specified as literals whereas OUT and INOUT parameters are specified as vectors of 3 elements for OUT and 4 elements for INOUT as follows:

num_cols
: Number of columns in the result set if the statement returned one.

stmt_meta
: A vector containing result metadata, etc.

result_set
: A vector of vectors containing each row in the result set.

cursor_handle
: The cursor handle (long).

Examples

Example 24.359. Remote execute example 1

Remote execute, retrieving the whole result from remote at once.

```
create procedure
test_rexecute_1 (in remote_dsn varchar)
{
  declare stmt varchar;
  declare mdta_out any;
  declare res_vec_out, param_vec any;
  declare sql_state, err_msg varchar;
  declare inx integer;
  declare num_cols_out integer;

  declare Company, Contact varchar;
  result_names (Company, Contact);

  stmt := 'SELECT CompanyName, ContactName FROM \
            Demo.Demo.Customers WHERE CompanyName BETWEEN ? and ?';
  sql_state := '00000';
  param_vec := vector ('A', 'B');

  rexecute (remote_dsn, stmt, sql_state, err_msg, param_vec,
            num_cols_out, mdta_out, res_vec_out, NULL);

  if (sql_state <> '00000')  -- See if we got an error
    {
      signal ('ZILCH',
              concat ('Remote execution returned ',
                      sql_state, ' ', err_msg));
    }

  -- now iterate through result set stored in res_vec_out

  inx := 0;
  while (inx < length (res_vec_out))
    {
      result (aref (aref (res_vec_out, inx), 0),
              aref (aref (res_vec_out, inx), 1));
      inx := inx + 1;
```

```
    }

  end_result ();
}

SQL> test_rexecute_1 ('Local Virtuoso Demo');
Company                   Contact
VARCHAR                   VARCHAR
_______________________________________________________________________________

Alfreds Futterkiste       Maria Anders
Ana Trujillo Emparedados y helados  Ana Trujillo
Antonio Moreno Taquería   Antonio Moreno
Around the Horn           Thomas Hardy

4 Rows. -- 4 msec.
```

Example 24.360. Remote execute example 2

Remote execute, retrieving the result using returned cursor handle.

```
create procedure
test_rexecute_2 (in remote_dsn varchar, in max_results integer)
{
  declare stmt varchar;
  declare mdta_out any;
  declare res_vec_out, param_vec any;
  declare sql_state, err_msg varchar;
  declare inx integer;
  declare num_cols_out integer;
  declare cursor_out any;

  declare Company, Contact varchar;
  result_names (Company, Contact);

  stmt := 'SELECT CompanyName, ContactName FROM \
            Demo.Demo.Customers WHERE CompanyName BETWEEN ? and ?';
  sql_state := '00000';
  param_vec := vector ('A', 'Z');

  rexecute (remote_dsn, stmt, sql_state, err_msg, param_vec,
            num_cols_out, mdta_out, NULL, cursor_out);

  if (sql_state <> '00000')  -- See if we got an error
    {
      signal ('ZILCH',
              concat ('Remote execution returned ',
                      sql_state, ' ', err_msg));
    }

  -- now iterate through result set with cursor

  while (0 = rnext (cursor_out, res_vec_out) and inx < max_results)
    {
      result (aref (res_vec_out, 0), aref (res_vec_out, 1));
      inx := inx + 1;
    }

  rclose (cursor_out);
  end_result ();
}

SQL> test_rexecute_2 ('Local Virtuoso Demo', 10);
Company                   Contact
VARCHAR                   VARCHAR
_______________________________________________________________________________

Alfreds Futterkiste       Maria Anders
Ana Trujillo Emparedados y helados  Ana Trujillo
Antonio Moreno Taquería   Antonio Moreno
```

```
Around the Horn           Thomas Hardy
B's Beverages             Victoria Ashworth
Berglunds snabbköp        Christina Berglund
Blauer See Delikatessen   Hanna Moos
Blondel père et fils      Frédérique Citeaux
Bon app'                  Laurence Lebihan
Bottom-Dollar Markets     Elizabeth Lincoln

10 Rows. -- 19 msec.
```

Example 24.361. Remote execute example 3

Remote DBMS native SQL execution

```
create procedure test_rexecute_3 (in remote_dsn varchar, in max_results integer)
{
    declare res_vec_out, param_vec, mdta_out, cursor_out any;
    declare url, stmt, sql_state, err_msg varchar;
    declare num_cols_out, inx, _id integer;

    result_names(_id,url);

    stmt := 'select id from mysql_pk';

    sql_state := '00000';
    param_vec := vector ('');

    rexecute (remote_dsn, stmt, sql_state, err_msg, param_vec, num_cols_out, mdta_out, NULL, cursor_out);

    _id := '';

    if (sql_state <> '00000')  -- See if we got an error
    {
      signal ('ZILCH',concat ('Remote execution returned ', sql_state, ' ', err_msg));
    }

    -- now iterate through result set stored in cursor_out
    inx := 0;
    while (0 = rnext (cursor_out, res_vec_out) and inx < max_results)
    {
      _id := aref (res_vec_out, 0);
      {
        whenever not found goto znext;
        select url into url from rdfstore_mysql where id = _id;
      }

       result(_id, url);

       if (0 = mod (inx, 5000))
       {
         log_message(sprintf ('%d rows copied, stay patient (id: %d)', inx, _id));
       }

        znext:;
        inx := inx + 1;
    }

  rclose (cursor_out);
  end_result ();
};
```

Example 24.362. Remote procedures with OUT params.

The following example demonstrates the use of OUT params using a sample remote procedure stored in another Virtuoso server, although this can be any database of course, and a local procedure that uses the `rexecute()` to execute it.

```
--On remote virtuoso:
create procedure FUNCRET (in data varchar) returns varchar { return concat (data, 'Out'); };

--On local virtuoso:
create procedure call_FUNCRET (in data varchar, in DSN varchar) returns integer
{
   declare params any;
   params := vector (
             vector ('out', 'varchar', 50), -- an OUT VARCHAR parameter for the result
                                            -- 50 here means receive no more than 50 bytes of
                                            -- the output value
             data                           -- the IN parameters are passed as usual
            );
   rexecute (DSN, '{?=call FUNCRET(?)}',
            null, null,
            params); -- params should be a PL variable, as it will hold the substituted
                     -- output values for the OUT and INOUT parameters upon return.
   -- retrieve the output value of the first parameter (i.e.
   -- the function return value) and return it:
   return params[0];
};
```

This should return as follows:

```
select call_FUNCRET ('abc', 'Local Virtuoso Demo');
---------------------------
abcOut
```

Example 24.363. Remote procedures with INOUT params.

As well as OUT parameters INOUT parameters can be used. Consider the following variation of the above example:

```
--On remote Virtuoso:
create procedure INOUT_PROC (inout DATA varchar)
{
   DATA := concat (DATA, 'InOut');
};

--On local Virtuoso:
create procedure call_INOUT_PROC (in DATA varchar, in DSN varchar) returns varchar
{
  declare params any;
  params := (
          vector ('inout', 'VARCHAR', 50, DATA) -- note we pass 4 element array for INOUT,
                                                -- as we need the input value for the parameter
             );
  rexecute (DSN, '{call INOUT_PROC (?)}',
                 null, null,
                 params);
  return params[0];
};
```

This, when called returns:

```
select call_INOUT_PROC ('abc', 'Local Virtuoso Demo');
----------------------
abcInOut
```

See Also

`rstmtexec()`, `rnext()`, `rmoreresults()`, `rclose()`

rstmtexec

rstmtexec — execute a SQL statement on a remote DSN, provides a result set where applicable.

Synopsis

rstmtexec (in *dsn* varchar ,
in *stmt* varchar ,
in *max_rows* integer ,
in *params_array* vector) ;

Description

This function can be used to execute SQL on a remote data source directly. It returns a result set where one is expected.

This function is wrapper for the `rexecute()` provided for convenience as a shortcut.

Unless explicitly granted, only the DBA group is permitted to use the `rstmtexec()` to maintain security. Caution is required here since any user granted use of `rstmtexec()` has full control of the remote data source set-up by the DBA, albeit limited to the overall abilities of the remote user on the remote data source. Users can be granted and denied access to this function using the following commands:

```
GRANT REXECUTE ON '<attached_dsn_name>' TO <user_name>
REVOKE REXECUTE ON '<attached_dsn_name>' FROM <user_name>
```

This command directly affects the grant to the `rexecute()` function, which is the underlying mechanism for providing this function.

Parameters

dsn
: The data source where the SQL statement should be executed. You must make sure that you have already defined the data source using the vd_remote_data_source function or by attaching tables from it.

stmt
: the SQL statement to execute on the remote data source dsn .

max_rows
: This controls the number of rows to be returned as follows:

in_params
: A vector of parameters to the statement if the executed statement has parameters. IN input parameters are specified as literals whereas OUT and INOUT parameters are specified as vectors of 3 elements for OUT and 4 elements for INOUT as follows:

Return Values

This function returns a result set.

See Also

`rexecute()` , `rnext()` , `rmoreresults()` , `rclose()`

right

right — get n rightmost characters of a string

Synopsis

right (in *str* string ,
in *count* integer) ;

Description

right returns the *count* rightmost characters of string str.

If count is zero an empty string '' is returned.

If length of str is less than count then a copy of the whole *str* is returned.

Examples

Example 24.364. Simple Example

```
right('AbracadabrA',4) -> 'abrA'
```

rmoreresults

rmoreresults — move to next result set of rexecute()

Synopsis

```
rmoreresults ( in cursor_handle long ,
               out sql_state varchar ,
               out sql_error varchar ,
               out num_cols integer ,
               out stmt_meta vector ) ;
```

Description

This function moves to the next result set when handling result sets returned by statement executed with rexecute.

Parameters

cursor_handle
: The cursor handle obtained from rexecute

sql_state
: Output parameter for returning SQL state.

sql_error
: Output parameter for returning an error message.

num_cols
: Output parameter for returning number of columns in a result row.

stmt_meta
: The metadata vector as described in documentation for rexecute .

See Also

`rexecute` , `rnext` , `rclose` .

rnd

rnd — returns a random number between 0 and n - 1 inclusive

Synopsis

```
number rnd ( in n integer );
```

Description

The rnd function returns a random number between zero and n - 1, inclusive.

Randomize initializes the random number generator.

The random number generator is initialized after the clock at first usage, so the produced sequences will be different each time unless specifically initialized.

rnext

rnext — Get next result from a remote result set

Synopsis

```
integer retcode rnext ( in cursor_handle long ,
                        out row_array vector ,
                        out sql_state varchar ,
                        out sql_error varchar ) ;
```

Description

Use `rnext` in combination with `rmoreresults` to iterate over a result set produced by a statement run in a remote data source with `rexecute` .

open a cursor with rexecute .
loop over the results with rnext
if rmoreresults does not return an error or SQL_NO_DATA_FOUND continue the loop with rnext, otherwise
close the cursor with rclose

Parameters

cursor_handle
: The long cursor handle as obtained from rexecute.

row_array
: An output vector that will contain the result columns.

sql_state
: Optional varchar output parameter for SQL state.

sql_error
: Optional varchar output parameter for any error message.

Return Values

All data is returned in output parameters.

See Also

`rexecute` , `rmoreresults` , `rclose` .

row_count

row_count — returns number of rows affected by the previous DML statement in a procedure body

Synopsis

```
integer row_count ( );
```

Description

This function returns the count of rows affected by the previous DML statement in a procedure body. The scope is local to the procedure. Calling this from ODBC will always return 0. This is the PL equivalent of the SQLRowCount ODBC function. The count is set after any in-line searched or positioned update, insert or delete. This is also set by the exec function. The count stays set until overwritten by the next DML operation. This is not set by rexecute.

Return Types

An integer representing the number of affected rows from a previous query.

rtrim

rtrim — trims given characters from right of given string

Synopsis

rtrim (*str* string ,
trimchars string) ;

Description

ltrim returns a copy of subsequence of string str with all the characters present in trimchars trimmed off from the beginning. If the second argument is omitted, it is a space ' ' by default.

rtrim is similar except that it trims from the right.

trim trims from both ends.

```
concat('*',trim('   SIMURG   '),'*') -> '*SIMURG*'
ltrim('AbracadabrA','bAr')  -> 'acadabrA'
rtrim('AbracadabrA','bAr')  -> 'Abracada'
trim('AbracadabrA','bAr')    -> 'acada'
```

search_excerpt

search_excerpt — Returns excerpts with hit words from text

Synopsis

search_excerpt (in *hit_words* any ,
in *text* varchar ,
in *within_first* int ,
in *max_excerpt* int ,
in *total* int ,
in *html_hit_tag* varchar) ;

Description

This function produces representative samples for use in displaying a query hit. There are two modes: html mode and text mode. In html mode everything looks like html tags is ignored and searching of excerpt begins from <body> tag. All found hit words highlighted by html_hit_tag.

In text mode text is treated as plain text, html tag detection is disabled and hit words is not highlighted.

Parameters

hit_words
: array of hit words to be found in text. Number of hit words can not be more than 10.

text
: original text where hit words are searched

within_first
: number of chars in text from the start to consider. Default value is 20000.

max_excerpt
: maximum length of single excerpt phrase. Default value is 90.

total
: maximum length of whole excerpt. Default value is 200.

html_hit_tag
: HTML tag which is used to highlight hit words in excerpt. Default value is "b". If it is NULL text mode is used

Return Types

varchar

Examples

Example 24.365. Creating search excerpt

creates a search excerpt from found DAV resource

```
for select RES_CONTENT from WS.WS.SYS_DAV_RES
 where contains (RES_CONTENT, 'place and knowledge')
do {
  http_value (search_excerpt (vector ('place', 'knowledge'),
                              cast (RES_CONTENT as varchar),
                              200000, 90, 200, 'b', 1),
              'P');
```

```
}
```

second

second — get second from a datetime

Synopsis

second (in *dt* datetime) ;

Description

second takes a datetime and returns an integer containing a number representing the second of the datetime.

Parameters

dt
: A datetime .

Return Values

An INTEGER containing the second.

Examples

Example 24.366. Simple example

Get current second.

```
SQL> select second (now ());
callret
INTEGER
_______________________________________________________________________________

28
```

See Also

dayname , dayofmonth , dayofweek , dayofyear , quarter , week , month , year , hour , minute , timezone

sequence_get_all

sequence_get_all — Returns a vector of states of all sequence objects

Synopsis

```
vector sequence_get_all ( );
```

Description

The function returns a vector of even length that contains two elements for every sequence object that is now stored in the database: the name and the current state. The order of 'name-state' pairs in the resulting vector may vary from call to call. To search such a vector by a setting name, the vector can be passed as a second argument to the function `get_keyword` ().

Changes in the returned vector will not alter sequence objects of the database. To change actual sequence objects, use `sequence_set` ().

Return Types

The function returns a vector of even length.

See Also

Sequence Objects

`sequence_next`

`sequence_set`

`sequence_remove`

`registry_get`

`registry_get_all`

`registry_set`

`registry_name_is_protected`

`registry_remove`

sequence_next

sequence_next — Returns the current state of the specified sequence and increments it by one.

Synopsis

```
integer sequence_next ( in name varchar ,
                        in increment_by integer ) ;
```

Description

This function returns the current state of the specified sequence and atomically increments it by one. The next call will thus return a number one greater than the previous. The sequence is shared between all connections and all transactions so an increment that is is made in one of connection will be seen in other connection immediately. Using a sequence never involves locking.

Parameters

name
: The server-wide name of a sequence.

increment_by
: This is to specify how much will be added to the sequence (must be greater than zero) If not specified defaults to 1.

Return Types

The function returns integer

See Also

Sequence Objects

`sequence_get_all`

`sequence_set`

`sequence_remove`

`registry_get`

`registry_get_all`

`registry_set`

`registry_name_is_protected`

`registry_remove`

sequence_remove

sequence_remove — Removes a sequence object

Synopsis

sequence_remove (in *name* varchar) ;

Description

The function removes a sequence object created before by `sequence_set` or `sequence_next` .

Parameters

name
: The name of the sequence object to be deleted.

Return Types

The function returns integer one if the sequence object is removed, zero if there was no object with the specified name.

Examples

Example 24.367. Creation and Removal of a Sequence Object

This sequence of operations demonstrates how `sequence_remove` reverts the effect of `sequence_next` .

```
> select sequence_next ('my_sequence');
0

> select sequence_next ('my_sequence');
1

> select sequence_next ('my_sequence');
2

> select sequence_remove ('my_sequence');
1

> select sequence_next ('my_sequence');
0
```

See Also

Sequence Objects

`sequence_get`

`sequence_get_all`

`sequence_next`

`registry_get`

`registry_get_all`

`registry_set`

`registry_name_is_protected`

registry_remove

sequence_set

sequence_set — Sets and returns the state of a sequence object.

Synopsis

```
integer sequence_set ( in name varchar ,
                       in state integer ,
                       in mode integer ) ;
```

Description

The function sets and returns the state of a sequence object. The *mode* specifies whether a check for order of values should be made. If *mode* equals 0, the state is set regardless of the previous state. If *mode* is non-zero, the state is changed only if the new state is greater than the previous state. This gives some (weak) protection from occasional 'rewind' the sequence back to values that are already in use.

Parameters

name
: The server-wide name of a sequence.

state
: A new state of the sequence.

mode
: Flags if a new state of the sequence should be ignored if it is less than the current one.

Return Types

The function returns an integer that is equal to what the next call to `sequence_next` () will return.

Examples

Example 24.368. Different modes of sequence_set()

The sequence of calls demonstrates various calls of `sequence_set` ().

```
select sequence_set ('sample', 5, 0);
5

1 Rows. -- 0 msec.

select sequence_next ('sample');
5

1 Rows. -- 0 msec.

select sequence_next ('sample');
6

1 Rows. -- 0 msec.

-- This has no effect because current state(6) is greater than 2.
select sequence_set ('sample', 2, 1);
7

1 Rows. -- 0 msec.

select sequence_next ('sample');
7
```

```
1 Rows. -- 0 msec.

-- But this change has effect:
select sequence_set ('sample', 2, 0);
2

1 Rows. -- 0 msec.

select sequence_next ('sample');
2

1 Rows. -- 0 msec.
```

See Also

Sequence Objects

`sequence_get_all`

`sequence_next`

`sequence_remove`

`registry_get`

`registry_get_all`

`registry_set`

`registry_name_is_protected`

`registry_remove`

serialize

serialize , deserialize — convert any heterogeneous array or tree of arrays into a binary string and back

Synopsis

```
binary string serialize ( in tree any );

binary string deserialize ( in str varchar );
```

Description

These functions will convert any heterogeneous array or tree of arrays into a binary string and back. The format is platform independent.

```
deserialize (serialize (x))
```

is the identity function.

These functions are useful for persisting heterogeneous arrays.

Note:
The serialization can be stored as a blob, so that there is no practical length limit. The string length is however limited to 16 MB.

serialize_to_UTF8_xml

serialize_to_UTF8_xml — Converts argument to its UTF-8 string representation.

Synopsis

serialize_to_UTF8_xml (in *value* any);

Description

The function converts its argument to a narrow string in UTF-8 encoding. The way conversion is done depends on the type of the argument.

If *value* is a wide (national) string or a LONG NVARCHAR then it is directly converted to UTF-8 string.

If *value* is a string or a LONG VARCHAR then it is converted from current charset of client connection to the UTF-8.

If *value* is an XML entity then it is converted to its XML text representation in UTF-8 encoding. The resulting string is a well-formed XML document if the given entity is an element entity, otherwise it may be well-formed generic entity, i.e. it can be parsed by an XML parser only as a part of some other XML document.

If *value* is NULL, the NULL is returned.

Values of other types are first converted to VARCHAR, this intermediate VARCHAR is converted from current charset of client connection to the UTF-8 and is returned.

Parameters

value
: The value of arbitrary type to be converted into its UTF-8 string representation.

Return Types

UTF-8 string or NULL.

Examples

Example 24.369. UTF-8 encoding of non-ASCII characters

A sample document is parsed and the resulting entity is serialized to UTF-8 string.

```
select serialize_to_UTF8_xml(xtree_doc('<text>0-9 a-z A-Z &#128;-&#255; &#256;-&#511; &#1024;-&#8191;</te
callret
VARCHAR
_______________________________________________________________________________

<text>0-9 a-z A-Z fl-ÿ  -   - </text>
```

See Also

charset_recode

SERV_QUEUE_TOP

SERV_QUEUE_TOP — Retrieve target website and store within Virtuoso

Synopsis

```
WS.WS. SERV_QUEUE_TOP ( in target varchar ,
                        in WebDAV_collection varchar ,
                        in update integer ,
                        in debug integer ,
                        in function_hook varchar ,
                        in data any ) ;
```

Description

Web Robot site retrieval can be performed with the WS.WS.SERV_QUEUE_TOP PL function integrated in to the Virtuoso server.

To run multiple walking robots all you simply need to do is kick them off from separate ODBC/SQL connections and all robots will walk together without overlapping.

From a VSP interface, after calling the retrieval function you may call http_flush to keep running tasks in the server and allowing the user agent to continue with other tasks.

Parameters

target
: URI to target site.

WebDAV_collection
: Local WebDAV collection to copy the content to.

update
: Flag to set updatable, can be 1 or 0 for on or off respectably.

debug
: Debug flag, must be set to 0

function_hook.
: Fully qualified PL function hook name. If not supplied or NULL then the default function will be used.

data
: application dependent data, usually an array, is passed to the PL function hook to perform next queue entry extraction. In our example we use an array with names of non-desired sites.

Examples

Example 24.370. Retrieve External Sites

```
WS.WS.SERV_QUEUE_TOP (
  'www.foo.com', 'sites/www_foo_com', 0, 0, 'DB.DBA.my_hook',
    vector ('www.skip.me','www.bar.com')
);
```

ses_connect

ses_connect — Opens a TCP session and returns its handle.

Synopsis

```
session_handle ses_connect ( in host_and_port varchar );
```

Description

Requires dba privileges.

This function is used to establish a new TCP connection to the target host. It returns a special datatype which represents the session handle.

Parameters

host_and_port
: The host and port number of the target host in the format <host>[:<port>]. If the optional port number defaults to 80 if not supplied.

Return Types

Returns a special internal datatype that represents the session handle. This session handle can be used in other session functions. Since this is an internal type it cannot be stored.

See Also

`ses_disconnect()`

`ses_read_line()`

`ses_write()`

ses_disconnect

ses_disconnect — Disconnections a TCP session.

Synopsis

ses_disconnect (in *ses* ses_handle);

Description

This function is used to disconnect from a session and destroy the session handle. After this function is called the session handle becomes invalid and hence cannot be used for reading or writing as an error will occur.

Parameters

ses
: A valid session handle.

See Also

`ses_connect()`

`ses_write()`

`ses_read_line()`

ses_read_line

ses_read_line — Read a line of character data from a TCP session.

Synopsis

```
varchar ses_read_line ( in ses any ,
                        in throw_exception_on_eof integer ,
                        in binary_mode integer ) ;
```

Description

This function is used to read a line of character data from an open TCP session. This function will read characters until it reaches an end of line up to a maximum of 1024 characters. The session can be passed as a session_handle. If the session_handle is omitted then execution is in current session/VSP context and will read from the open HTTP session. `Ses_read_line()` will suspend execution while attempting to read from the session until the timeout period of 100 seconds expires. When the timeout expires an error will be produced to indicate that the operation was unsuccessful.

Parameters

ses
: Optional session handle. If none is supplied the current TCP client session is assumed. If this is NULL then the function will read from the string session to be sent to the client upon request completion.

throw_exception_on_eof
: Controls whether to throw an error if there is a problem reading from the session or simply return null in text mode or what ever content has has been read so far in binary mode. If this parameter is set to 1, then an error will be thrown. If this parameter is set to 0 then either null or remaining content will be returned as above.

binary_mode
: Instructs the function to be in binary or text mode.

Return Types

The characters read from the session are returned from this function.

Examples

Example 24.371. Simple demonstration of reading from the HTTP session

This example reads the first line of the session and outputs the contents. This code can be run from a vsp file.

```
<p>Some text</p>
<?vsp
    declare xx any;
    xx := ses_read_line (NULL);

    http (sprintf ('length is %d, "%s"', length (xx), xx));
?>
```

See Also

`ses_connect()`

`ses_disconnect()`

`ses_write()`

ses_write

ses_write — Write character data to an open TCP session.

Synopsis

ses_write (in *buffer* varchar ,
in *ses* session_handle) ;

Description

This function is used to write character data taken from the buffer parameter to an open TCP session. Attempts to write to a close session will result in an error being returned.

Parameters

buffer
: Character data to be sent to the session.

ses
: A valid session handle. If none is supplied the current VSP context is assumed.

See Also

`ses_connect()`

`ses_disconnect()`

`ses_read_line()`

set_row_count

set_row_count — sets the affected rows counter in the current context or in the context of the caller

Synopsis

```
integer set_row_count ( in increment integer ,
                        in what integer ) ;
```

Description

The function set_row_count () is used to set the affected rows counter in the current context or in the context of the caller. Therefore it can be used to set the affected rows counter (returned by `row_count()`) in places where an instead of trigger is used. If result of decrement of the affected rows counter is an negative integer it will be set to zero.

Parameters

increment
: An integer to be added to the affected rors counter, can be negative also

what
: The what have to be 0 or 1, 0 is the default. 0 - for update in current context the counter, 1 - to update the counter in context of the caller.

Return Types

The function returns new value of the affected rows , or -1 if action is not applicable.

Examples

Example 24.372. Setting the affected rows counter in instead of trigger

The following SQL script will make two instead of triggers on atable "hid" and inside them will are called set_row_count to set the rows inserted or updated in the second table. Also there are two procedures which are used to demonstrate the return value of row_count.

```
create table hid (id integer primary key, dt varchar);
create table sho (id1 integer primary key, dt1 varchar);

create trigger i_hid instead of insert on hid
{
  insert into sho values (id, dt);
  set_row_count (row_count(), 1);
};

create trigger u_hid instead of update on hid
{
  update sho set dt1 = dt where id1 = id;
  set_row_count (row_count(), 1);
};

create procedure test_ins ()
{
  insert into hid select * from chid;
  return row_count();
};

create procedure test_upd ()
{
  update hid set dt = '';
  return row_count ();
};
```

See Also

`row_count()`

set_user_id

set_user_id — sets the current user for execution

Synopsis

```
integer set_user_id ( in user_name varchar ,
                      in mode integer ,
                      in password varchar ) ;
```

Description

This function changes the effective user and group to that of the user defined by user_name parameter. When called by a user with DBA group privileges, the optional password may be omitted. Otherwise it has to be the valid password for user_name. The mode parameter determines persistence (context) of effects of the call: If omitted or set to integer value 1, the effective user privileges will remain in effect within current stored procedure context only - upon returning, the effective user privileges will be automatically reset to state effective before set_user_id. When mode is equal to integer value 0, the effective user privileges will remain set for duration of current ODBC session, current request in web server context, or until next call of set_user_id. This function is analogous to the UNIX 'su' command.

Parameters

user_name
: the name of SQL user account to be used as effective user and group.

mode
: optional, integer 1 (default) means change of privileges remains in effect only while within current stored procedure context.

password
: this optional parameter is needed if calling the function without DBA privileges. Password of the user user_name .

Return Types

No return value

set_identity_column

set_identity_column — sets the sequence starting value for an identity column

Synopsis

```
integer set_identity_column ( in table_name varchar ,
                              in column_name varchar ,
                              in new_value integer ) ;
```

Description

This function takes the table name, the column name and the new sequence value as parameters. It checks for the existence of the identity column, and then sets the sequence value (using the sequence_set) and returns the old sequence value. The table and column names must be properly qualified to ensure the correct resource is located. The effect of calling this function is immediate.

Parameters

table_name
: the fully qualified table name in the correct case exactly as it appears in the DB.DBA.SYS_KEYS table.

column_name
: the exact column name as it appears in the DB.DBA.SYS_COLS table.

new_value
: the new sequence value.

Return Types

the previous sequence value

sign

sign — returns -1, 0, or 1 depending on the sign of its numerical

Synopsis

sign (*num* numeric) ;

Description

sign returns either -1, 0 or 1 depending whether its numeric argument is negative, zero or positive.

```
sign(-12)                       -> -1
sign(0)                         -> 0
sign(910)                       -> 1
sign(atof('-1.23456789'))       -> -1
sign(0.0)                       -> 0
```

signal

signal — Signal an exception in the calling procedure

Synopsis

signal (in *sqlstate* varchar ,
in *message* varchar) ;

Description

This signals the given SQLSTATE with the message. The calling procedure will transfer control to the most appropriate local handler. In the absence of a local handler the procedure terminates and signals the exception in the scope where it was called from, until there either is a handler or there are no more calling procedures. If there is no handler in the entire stack of call contexts the error is signalled to the client. Handlers can be declared with whenever .. goto and the declare handler for construct. See the Virtuoso/PL documentation.

```
CREATE PROCEDURE WITHDRAW (IN C_ID VARCHAR, IN DELTA NUMERIC)
{
    DECLARE BAL NUMERIC;

    DECLARE CR CURSOR FOR SELECT C_BALANCE FROM CUSTOMER WHERE C_ID = C_ID;

    WHENEVER NOT FOUND GOTO NOCUSTOMER;

    OPEN CR (EXCLUSIVE);
    FETCH CR INTO BAL;
    IF (BAL > DELTA)
        UPDATE CUSTOMER SET C_BALANCE = BAL - DELTA WHERE CURRENT OF CR;
    ELSE
        SIGNAL ('NOMONEY', 'INSUFFICIENT BALANCE.');

    RETURN;

NOCUSTOMER:
    SIGNAL ('NOCUS', 'BAD CUSTOMER ID');
}
```

sinv_create_key_mapping

sinv_create_key_mapping — Creates a key mapping function & table.

Synopsis

sinv_create_key_mapping (in *map_name* varchar ,
in *part_defs* any ,
in *do_drops* integer := 1) ;

Description

Creates a key mapping function & table (as described in the doc section SQL Inverse Functions)

Parameters

map_name
: (string) the name of the mapping. This becomes the name of the forward function (from parts to single int value) and participates in the name of the lookup table (like MAP_ <map_name>)

part_defs
: contains an array of name/type pairs describing the parts of the mapping. The types are used in the MAP table as column types and the type names are used in naming the referenced functions (one for each part) like : <map_name>_ <part_name>

do_drops
: integer (0, 1, 2 : default value 1)

See Also

`sinv_create_inverse` , `sinv_drop_inverse`

sinv_create_inverse

sinv_create_inverse — Creates inverse mapping for the mentioned functions.

Synopsis

```
sinv_create_inverse (  in _SINVM_NAME_IN varchar ,
                       in _SINV_INVERSE any ,
                       in _SINVM_FLAGS integer ) ;
```

Description

Creates inverse mapping for the mentioned functions (as described in the doc section SQL Inverse Functions)

Parameters

_SINVM_NAME_IN
: the name of the forward function

_SINV_INVERSE
: an array of the names of the inverse functions (for each part).

_SINVM_FLAGS
: bit mask with:

See Also

`sinv_create_key_mapping` , `sinv_drop_inverse`

sinv_drop_inverse

sinv_drop_inverse — Reverses the effect of `sinv_create_inverse` procedure.

Synopsis

`sinv_drop_inverse` (in *_SINVM_NAME_IN* varchar) ;

Description

Reverses the effect of `sinv_create_inverse` procedure.

Parameters

_SINVM_NAME_IN
: the name of the forward function

See Also

`sinv_create_key_mapping`, `sinv_create_inverse`

smime_sign

smime_sign — Converts a MIME message to a signed S/MIME message

Synopsis

```
varchar smime_sign ( in msg_text varchar ,
                     in signer_cert varchar ,
                     in private_key varchar ,
                     in private_key_pass varchar ,
                     in signer_CA_certs any ,
                     in flags integer ) ;
```

Description

Converts a MIME message to a signed S/MIME message.

Parameters

msg_text
: The text of the message

signer_cert
: Signer certificate.

private_key
: Private Key

private_key_pass
: Private Key Pass

signer_CA_certs
: Array of strings of CA Certificates

flags

Return Types

This function takes a message and converts it to an S/MIME signed message based on the flags value (if supplied - default PKCS7_DETACHED).

Examples

Example 24.373. Signing a MIME Message

Example (where signed.eml contains the output from smime_sign.

```
select smime_sign ('just a test', file_to_string ('thwate_pub.pem'),
  file_to_string ('thwate_pri.pem'), 'very_secret_password',
        vector (file_to_string ('thwate_ca.pem')), 4*16 + 1);
```

```
MIME-Version: 1.0
Content-Type: multipart/signed ; protocol="application/x-pkcs7-signature" ;
  micalg=sha1 ; boundary="----23F1D9057532E126962121287FDB4793"

This is an S/MIME signed message
```

```
------23F1D9057532E126962121287FDB4793
Content-Type: text/plain

just a test
------23F1D9057532E126962121287FDB4793
Content-Type: application/x-pkcs7-signature; name="smime.p7s"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7s"

MIILaAYJKoZIhvcNAQcCoIILWTCCC1UCAQExCzAJBgUrDgMCGgUAMAsGCSqGSIb3
DQEHAaCCCTswggLZMIICQqADAgECAgMD2DgwDQYJKoZIhvcNAQEEBQAwgZIxCzAJ
BgNVBAYTAlpBMRUwEwYDVQQIEwxXZXN0ZXJuIENhcGUxEjAQBgNVBAcTCUNhcGUg
VG93bjEPMA0GA1UEChMGVGhhd3RlMR0wGwYDVQQLExRDZXJ0aWZpY2F0ZSBTZXJ2
aWNlczEoMCYGA1UEAxMfUGVyc29uYWwgRnJlZW1haWwgUlNBIDIwMDAuOC4zMDAe
Fw0wMDEyMjIwODM1MjFaFw0wMTEyMjIwODM1MjFaMHkxHzAdBgNVBAMTFlRoYXd0
ZSBGcmVlbWFpbCBNZW1iZXIxLjAsBgkqhkiG9w0BCQEWH2tnZW9yZ2VAcGxvdmRp
di50ZWNobm8tbGluay5jb20xJjAkBgkqhkiG9w0BCQEWF2drb2Rpbm92QG9wZW5s
aW5rLmNvLnVrMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDTOklS3QEDt2Pm
vtNOK5Zi7PPn9U2EDmdjoCjKEHq8Zkyod+rATNBrqH24ewaoqCPWtRTb9GkwW9EM
z+2pu+liAqiA4Es+sNpfgKvO4T4/bliyp3FAJ/03s9XfU0hyJT5JVFDt0hfBou0c
wgG1spwuz/3Gtl/DQDS1AMZ1R8jacQIDAQABo1UwUzBDBgNVHREEPDA6gR9rZ2Vv
cmdlQHBsb3ZkaXYudGVjaG5vLWxpbmsuY29tgRdna29kaW5vdkBvcGVubGluay5j
by51azAMBgNVHRMBAf8EAjAAMA0GCSqGSIb3DQEBBAUAA4GBAAArP5xEnmt5goAO
2+8UE/8dd4CwfG0w/IO53psFgpjPlM/D9ePt/LPF/vhTtPHXubm54xJlNVi0MMPU
g91BU3bPkX2rrEhv1plInRc0eaKIooy+qz8KPhz5DwZ0PfidOMn3G2/HkdODVqjK
ZReIhcaU5pDamp0NCMjzi5isB2qfMIIDKTCCApKgAwIBAgIBDDANBgkqhkiG9w0B
AQQFADCB0TELMAkGA1UEBhMCWkExFTATBgNVBAgTDFdlc3Rlcm4gQ2FwZTESMBAG
A1UEBxMJQ2FwZSBUb3duMRowGAYDVQQKExFUaGF3dGUgQ29uc3VsdGluZzEoMCYG
A1UECxMfQ2VydGlmaWNhdGlvbiBTZXJ2aWNlcyBEaXZpc2lvbjEkMCIGA1UEAxMb
VGhhd3RlIFBlcnNvbmFsIEZyZWVtYWlsIENBMSswKQYJKoZIhvcNAQkBFhxwZXJz
b25hbC1mcmVlbWFpbEB0aGF3dGUuY29tMB4XDTAwMDgzMDAwMDAwMFoXDTAyMDgy
OTIzNTk1OVowgZIxCzAJBgNVBAYTAlpBMRUwEwYDVQQIEwxXZXN0ZXJuIENhcGUx
EjAQBgNVBAcTCUNhcGUgVG93bjEPMA0GA1UEChMGVGhhd3Rl
```

See Also

smime_verify

smime_encrypt

smime_decrypt

pem_certificates_to_array

smime_verify

smime_verify — Verifies signature of signed MIME message

Synopsis

```
varchar smime_verify ( in msg_text varchar ,
                       in certs any ,
                       out signer_certs any ,
                       in flags integer ) ;
```

Description

This function takes the RFC822 text of an e-mail containing an S/MIME signed message and verifies it's signature using the CA certificates in certs, which is an array of strings containing single or multiple PEM-encoded certificates.

Parameters

msg_text
: The text of the message

certs
: array of strings containing CA certificates

signer_certs
: for receipt of PEM encoded certificates

flags
: A bitmask. See table below for valid mask values. Default is 0.

Return Types

If the signer_certs is supplied, an array of PEM encoded certificates of the signers is returned.

Examples

Example 24.374. Verifying a Signed MIME Message

Example (where signed.eml contains the output from smime_sign.

```
select smime_verify (file_to_string ('signed.eml'),
  vector (file_to_string ('thwate_ca.pem')), NULL, 1)

_____________________________________________________________________________
just a test
_____________________________________________________________________________
```

See Also

smime_sign

smime_encrypt

smime_decrypt

pem_certificates_to_array

smime_encrypt

smime_encrypt — Performs encryption of a (signed) e-mail msg

Synopsis

```
varchar smime_encrypt ( in mail_message varchar ,
                        in array_of_recipient_certificates any ,
                        in cipher_name varchar ) ;
```

Description

This function performs encryption of a (signed) e-mail message.

Parameters

mail_message
: The text of the e-mail msg.

array_of_recipient_certificates
: An array of strings containing certificates (per format) of the recipient individuals. The certificates must contain RSA keys. See for reference here .

cipher_name
: Encryption algorithm 'des', 'des3', 'aes256', 'aes128' etc.

Return Types

varchar

Examples

Example 24.375. Encrypting (signed) e-mail msg

```
SQL> select smime_encrypt (file_to_string ('test.eml'),
                           pem_certificates_to_array (file_to_string ('testcerts.pem')),
                           'aes256');

_____________________________________________________________________________
MIME-Version: 1.0
Content-Disposition: attachment; filename="smime.p7m"
Content-Type: application/x-pkcs7-mime; smime-type=enveloped-data; name="smime.p7m"
Content-Transfer-Encoding: base64

MIIDwwYJKoZIhvcNAQcDoIIDtDCCA7ACAQAxggGXMIIBkwIBADB7MHYxCzAJBgNV
BAYTAkJHMRAwDgYDVQQIEwdQbG92ZGl2MRMwEQYDVQQKEwpQcm9nZW0gT09EMQ0w
CwYDVQQLEwRMYWJzMRAwDgYDVQQDEwdSb290IENBMR8wHQYJKoZIhvcNAQkBFhBp
bWl0a29AZ21haWwuY29tAgEAMA0GCSqGSIb3DQEBAQUABIIBADWF2bOMROmztQqO
ytW7JfG3vjpk4TKHKoSnjKP0bBOSypY6/0tOzIg9jQHvl/Kay0do6gzfNbWV9SQP
UWRa0lyD9rpG9WbxStOTPzDhDhbFJ0EXr1CR0uKMoUUuY3ovWkdZx0+whhCuwwVF
pSY0vvh9P+Bw0hMUsJPsDtudti220a7A0jJ5Rz2ubcnN5A3M5Odxwr59ix93RYD/
zfQ5ETqH8zx16F1cs7uS2uD7m2uVp8mQBab4onDYzpLUWvVi/dPX3h2ifAjn25At
SmWROYd+NbyNxs9uQp4pQZ6ZEJo6dPVrsM7SfR55O3ukq5V9zOUYEVN+VJpPlGBt
Gru9tlwwggIOBgkqhkiG9w0BBwEwHQYJYIZIAWUDBAEqBBCykjJ22Mk6baYYlP/w
PTb2gIIB4KyYoU7j4vREgvTO6zBauDnACAZO0oOdzGDuJKtLd8rTJO10ntp+CF6Y
5hAZnUefW0cCm6Pn9yMbll5Btthek+CCscKsPTNCMSb/dBuPTbmKQcfYwlHV/Ejz
Ko5uJYMHLh8RkjHPt6nov6YZrMA82IA4KGCWxhhsQ7flHYtLG9PEWOaQr7Y+BYBs
a3sQdiGgGBwarTX+y65ERYcE32h+xnP4RW3QD8VEmhlyesG5xoALnGM4CQ9VObh1
VOTQhPRbu1PSsa4fEjbn3ey7+89mrjpYD/GwdwBEqOM0xGUZI4UBnhDwCIE/A4xX
jEdf59vpUt+mVzGYXCwSvkdwMCpPoPH9asYXSEWLoDOIagNUbuUQ5Y/A1iKWpM8p
21IND9tI+UiXMBgrDpeR/0b/E+Waswfb9qdXDSBDFGdB1aJo/f9KsikHNwxYczxh
YF0Ra3WqJC0n3+IEgoaCy/8Shon6VDsnZtB0aMXBIUEd+tpx0CBggLmHWL7jwcwy
CRD2bYzuM3yvux/u7U0B5GmfDEmyuNoEyMuUGC1+Sk7TFBtW0I9LpBDY5m5hPl1y
```

```
VeATJx0vPbmcy/smzkVKYTwhFWF/AYT4CrNbrTYkkb39JmWHTH6GU1KEfoonTbA7
oAUG6cFvFA==
-----------------------------------------------------------------------------
```

See Also

smime_decrypt

smime_sign

smime_verify

pem_certificates_to_array

smime_decrypt

smime_decrypt — Decryption of a PKCS7 encrypted smime msg

Synopsis

```
varchar smime_decrypt ( in encrypted_mail_message varchar ,
                        in recipient_certificate varchar ,
                        in recipient_key varchar ,
                        in password_for_the_key varchar ) ;
```

Description

This function performs decryption of a PKCS7 encrypted smime msg

Parameters

encrypted_mail_message
: The text of the encrypted smime message

recipient_certificate
: The certificate of the recipient (pem format)

recipient_key
: The private key of recipient matching the recipient_certificate (pem format)

password_for_the_key
: The secret to open the recipient_key (if key is encrypted)

Return Types

varchar

Examples

Example 24.376. Decryption a PKCS7 encrypted smime msg

```
select smime_decrypt (file_to_string ('test.p7m'),
                      file_to_string ('test.pem'),
                      file_to_string ('test.key.pem'),
                      'secret');
```

```
Content-Transfer-Encoding: 7bit
Content-Type: text/plain;
 charset=us-ascii

this is a test
```

See Also

smime_sign

smime_encrypt

smime_verify

pem_certificates_to_array

smtp_send

smtp_send — send message to SMTP server

Synopsis

smtp_send (in *server* string ,
in *sender* string ,
in *recipient* string ,
in *body* string) ;

Description

Virtuoso can act as an SMTP client. This means that Virtuoso is able to send emails directly to a mail SMTP server. Virtuoso has a simple function to facilitate this. This can be called from stored procedures, VSP pages, triggers etc.

The sender and recipient email addresses must be enclosed with <..> and separated by commas i.e. string '<support@openlinksw.co.uk>,<sales@openlinksw.co.uk>'

The message Body contains headers such as Subject, From, To, Cc, Bcc and then continues with the actual message text itself. New lines can be added using '\r\n'

Example 24.377. Example:

```
'Subject: subject message\r\nFrom: sender\r\nTo: recipient\r\nCc:
copy\r\nBcc: copy\n\n body of message'
```

Virtuoso will pick up Subject and other headers from the body content. Note that the RFC insists there should be a NULL line between body headers and the message body text.

Example 24.378. Example:

```
smtp_send(
                'mail.example.com:25',
                '<sender@example.com>',
                '<receiver@example.com>',
                concat(
                  'X-Mailer: Virtuoso Universal Server\r\n',
                        'Date: ', soap_print_box (now (), '', 1), '\r\n',
                        'Message-ID: <', regexp_replace(cast(now() as varchar), '[- :.]', '', 1, null), '
                        'Subject: This is a test mail...\r\n',
                        'From: <sender@example.com>\r\n',
                        'To: <receiver@example.com>\r\n',
                        '\n',
                        'Hi Receiver, this is a test message from Virtuoso')
        );
```

A more involved example. It is the responsibility of the developer to ensure that the message is correctly formed, complete with all necessary headers. This example shows a complete use of the function.

soap_box_xml_entity

soap_box_xml_entity — Converts an XML entity to an SQL value given the desired SQL type.

Synopsis

```
any soap_box_xml_entity ( in entity any ,
                          in try_typed_as any ,
                          in soap_version integer ) ;
```

Description

Converts an XML entity to an SQL value based on the type of the entity and the desired SQL type. This function is called internally to convert a SOAP request parameter to a PL procedure parameter when a SOAP request is being processed by the SOAP server.

Parameters

entity
: The XML fragment as a vector (as returned from xml_tree() or a subpart of it).

try_typed_as
: A sample value, whose type is taken as a desired type for conversion.

soap_version
: Optional (default 1). The soap version (1 for SOAP 1.0, 11 for SOAP 1.1).

Return Types

The return type of `soap_box_xml_entity()` can vary according to the type of *`entity`* parameter, described by *`try_typed_as`* as follows:

If *`entity`* is NULL, the return value will be NULL.

If *`entity`* is not a vector() then its value will be cast to the *`try_typed_as`* type, if possible.

If *`entity`* is a vector() containing XML tree then it will be converted to an appropriate SQL type (vector(), varchar, integer etc.) depending on the SOAP RPC encoding rules (XMLSchema-datatypes and SOAP-RPC encoding schema) for in/out SOAP messages. i.e. it depends on the structure/content of the XML tree passed as the *`entity`* argument.

soap_dt_define

soap_dt_define — define re-define or erase the complex datatype definition for SOAP calls

Synopsis

soap_dt_define (in *name* varchar ,
in *schema_string* varchar) ;

Description

This defines a new complex SOAP datatype (usually array of structure) named 'name'.

The schema_string string represents definition as complexType element from XML Schema. The only complexContent, all and sequence elements can be used within the complexType. This means that optional elements in the defined datatype are not supported as a variant of the SOAP parameter datatype. If the schema descriptions contains an unsupported element , the SQL error will be signalled and error message will explain what element is wrong.

Parameters

name
: A varchar containing the expanded name of SOAP type to be defined/removed or an empty string (''). If an empty string is supplied this function will try to extract it from the given schema_string schema fragment (attribute @name'). Name cannot be an empty string for removing SOAP types.

schema_string
: XMLSchema excerpt as varchar or NULL (null is used for removal).

Return Types

This function returns a varchar of the name of the registered SOAP type.

Errors

This function can generate the following errors:

SODT1 22023 The element <element name> is not supported [<as child of complexContent>]

Examples

Example 24.379. Definition of an Array

```
<!-- file float_array.xsd -->

<complexType name="ArrayOffloat"
   xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:tns="services.wsdl">
   <complexContent>
   <restriction base="enc:Array">
   <sequence>
   <element name="item" type="float" minOccurs="0" maxOccurs="unbounded" nillable="true"/>
   </sequence>
   <attributeGroup ref="enc:commonAttributes"/>
   <attribute ref="enc:offset"/>
   <attribute ref="enc:arrayType" wsdl:arrayType="float[]"/>
   </restriction>
   </complexContent>
</complexType>
```

```
<!-- eof float_array.xsd -->

can be defined from ISQL tool or in the PL procedure
SQL> DB.DBA.soap_dt_define ('ArrayOffloat', file_to_string ('float_array.xsd'));
```

Example 24.380. Definition of an Structure

```
<!-- file struct.xsd -->

<complexType name="PERSON"
   xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:tns="services.wsdl">

   <sequence>
     <element name="firstName" type="string"/>
     <element name="lastName" type="string"/>
     <element name="ageInYears" type="int"/>
     <element name="weightInLbs" type="float"/>
     <element name="heightInInches" type="float"/>
   </sequence>
</complexType>

<!-- eof struct.xsd -->

can be defined from ISQL tool or in the PL procedure
SQL> DB.DBA.soap_dt_define ('PERSON', file_to_string ('struct.xsd'));
```

Example 24.381. Definition of composite type array of structures

```
<!-- file array_struct.xsd -->

<complexType name="ArrayOfPERSON"
   xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:tns="services.wsdl">

   <complexContent>
   <restriction base="enc:Array">
   <sequence>
   <element name="item" type="tns:PERSON" minOccurs="0" maxOccurs="unbounded" nillable="true"/>
   </sequence>
   <attributeGroup ref="enc:commonAttributes"/>
   <attribute ref="enc:offset"/>
   <attribute ref="enc:arrayType" wsdl:arrayType="tns:PERSON[]"/>
   </restriction>
   </complexContent>

</complexType>

<!-- eof array_struct.xsd -->

can be defined from ISQL tool or in the PL procedure
SQL> DB.DBA.soap_dt_define ('ArrayOfPERSON', file_to_string ('array_struct.xsd'));
```

soap_call

soap_call — calls a function from a SOAP server and returns the result. value; deprecated, use SOAP_CLIENT () instead

Synopsis

```
any soap_call ( in host string ,
                in path string ,
                in methodURI string ,
                in methodName string ,
                in params array of (ParamName, Value) ,
                in version integer ,
                in certificate varchar ,
                in password varchar ,
                in SOAPAction varchar ) ;
```

Description

This calls a function from a SOAP server and returns the result as a return value. Params is an array of (Parameter name, Value) pairs representing the parameters passed in the SOAP call. Each of these pairs become an XML sub-entity of the procedure entity. The return value of the function is the entity inside the SOAP body of the response. In debug mode the return value is an array of 3 elements; the non-debug return value (if any) as element 0, the XML text of the request as element 1 and the XML text of the server response as element 2. This function does not use any XML types when creating the XML. It represents types as a cast to varchar would, with one exception - dates and times according to ISO8061.

The Virtuoso SOAP client can work with complex datatypes, in which case the parameters array must conform to the following convention:

```
vector ( vector ([name1], [type1]), value1, vector ([name2], [type2]), value2 ...)
```

This will cause type checking and validation of the values to be encoded for SOAP request.

Parameters

host
: DNS name or IP address of the SOAP server

path
: path into the HTTP server containing the SOAP server page

methodURI
: URI of the SOAP method being called

methodName
: Name of the SOAP method being called

params
: array of parameters to the SOAP call; array of (ParamName, Value). The Virtuoso SOAP client can work with complex datatypes. In order to support this the parameters array must conform to the following convention: vector (vector ([name1], [type1]), value1, vector ([name2], [type2]), value2 ...). This will cause type checking and validation of the values to be encoded for SOAP request.

version
: the SOAP version used in call (SOAP 1.0 = 1, SOAP 1.1 = 11). Default value = SOAP 1.0. If the value is negated : i.e. -1 or -11 then the soap_call procedure enters "debug" mode

certificate

If this parameter is specified (string or null) the HTTPS operation will be performed. Path to the HTTPS client certificate in PKCS#12 format, if this parameter is set to NULL then client will do only encrypted connection.

password
: If certificate is supplied this parameter must contain password for opening the certificate file.

SOAPAction
: SOAPAction parameter can be used to make SOAPAction header value explicit. Normally this is taken from the namespace URI and SOAP method name.

Examples

Example 24.382. Calling a SQL procedure fishselect (in str varchar)

```
declare ret any;
ret := soap_call ('example.com', '/SOAP',
         'urn:com-openlinksw-virtuoso',
         'fishselect',
         vector ('str', 'G'));
```

soap_client

soap_client — Invoke a SOAP service and returns result value.

Synopsis

```
soap_client ( in url varchar ,
              in operation varchar ,
              in target_namespace varchar default null ,
              in parameters any default null ,
              in headers any default null ,
              in soap_action varchar default '' ,
              in attachments any default null ,
              in ticket any default null ,
              in passwd varchar default null ,
              in user_name varchar default null ,
              in user_password varchar default null ,
              in auth_type varchar default 'none' ,
              in security_type varchar default null ,
              in debug integer default 0 ,
              in template varchar default null ,
              in style integer default 0 ,
              in version integer default 11 ,
              in direction integer default 0 ,
              in http_header varchar default null ,
              in security_schema any default null ,
              in time_out int default 100 ) ;
```

Description

This will invoke the specified SOAP service.

Parameters

url
: The target SOAP service endpoint URL. For SSL connections this must begin with 'https:' schema.

operation
: The name of the SOAP method to be invoked.

target_namespace
: The namespace of the SOAP request.

parameters
: parameters contained inside a vector in one of two ways:

headers
: the parameters to be printed in SOAP header element. They are contained inside a vector in following way:

soap_action
: The exact value of the SOAPAction header attribute.

attachments

DIME/MIME message attachments as array of (id, type, content)

ticket
: The key instance to perform XML signature and encryption for outgoing SOAP message. Or path to PKCS#12 file containing x509 certificate to perform secure connection over SSL with server's certificate verification. To indicate to use SSL for connection encryption only, then a number as string greater than 0 must be specified (i.e. '1').

passwd
: The password for opening the x509 certificate specified in 'ticket'.

user_name
: name for basic/digest HTTP authentication.

user_password
: The password for basic/digest HTTP authentication.

auth_type
: The type of authentication to use. Valid types are:

security_type
: This parameter governs the security method of the outgoing messages. Valid values are "encrypt" or "sign", to encrypt the message or sign it only.

debug
: If this flag is set to 1 then debug mode will be enabled. When enabled the returned message will be an array of 3 elements consisting of the request, response, and error if one occurred.

template
: The content of an XML signature template. If the default template is to be used only "[key reference for signing]" (note '[]' around name). need be specified. The server will generate a default template based on the key and default rules for making a signature.

style
: This is a bit-mask parameter that is used to specify the outgoing message format. The mask relies on the following values:

version
: the SOAP version used in call (SOAP 1.0 = 1, SOAP 1.1 = 11, SOAP 1.2 = 12). Default value = SOAP 1.1.

direction
: This flag if turned on (1) then one-way messages will be sent. In that case client doesn't expect response from SOAP server, except HTTP headers.

http_header
: This parameter is used to include custom HTTP headers in the SOAP POST request. The usual purpose of it to pass special 'X-<extension name>' header to server in order to send data that can't be included in SOAP message. For example this could be used for loop-to-self detection when replicating data collection(s) via SOAP.

security_schema
: This parameter is used to designate version of WS-Security and WS-Utility protocol to be used when secure SOAP calls will be made. The value is an array containing name/value pairs for 'wsse' (WS-Security) and 'wsu' (WS-Utility) namespace URLs. An example of the value is: vector ('wsse', WSSE_OASIS_URI, 'wsu', WSSU_OASIS_URI).

time_out
: The connection timeout. Elapsing this value and having no response from server will cause client to disconnect with an connection error.

Return Types

The function will return a XML tree of the SOAP response in simple mode. Also if one-way message is sent and no body from the SOAP server, then NULL will be returned. When advanced mode is set i.e. 'style' have second bit set, then response will be as

described earlier in 'style' parameter description.

Requirements for certificate files for HTTPS operation

When users going to do operation with an HTTPS endpoint the following are requirements to the PKCS#12 file. The file specified in parameter 'ticket' MUST contain: valid x509 certificate, certificate chain and private key. The password for opening the private key must be specified in the 'passwd' parameter. How client and server certificates and private keys are created are not subject of this document. Also see Web server documentation about how to run an HTTPS listener.

Example

Example 24.383.

References: tutorial/services WS-S-2 (triple-des), WS-S-3 (RSA), WS-S-4 (X.509 signing)

The following is an excerpt from the WS-S-2 Services tutorial to make encoded SOAP message with a shared key.

```
resp :=
        SOAP_CLIENT (
          url=>'http://example.com/SecureWebServices',
          operation=>'AddInt',
          parameters=>vector (vector ('a', 'int'), 3, vector ('b', 'int'), 4),
          auth_type=>'key',
          ticket=>xenc_key_inst_create ('WSDK Sample Symmetric Key'),
          security_type=>'encrypt',
          target_namespace=>'http://temp.uri/',
          soap_action=>'"http://temp.uri/#AddInt"',
          style=>6);
```

This will produce following SOAP message:

The encoded SOAP request

```
------------ REQUEST ------------------
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/"
          encodingType="http://schemas.xmlsoap.org/soap/encoding/" >
  <Header xmlns="http://schemas.xmlsoap.org/soap/envelope/" >
    <Security xmlns="http://schemas.xmlsoap.org/ws/2002/07/secext" >
      <ReferenceList xmlns="http://www.w3.org/2001/04/xmlenc#" >
        <DataReference xmlns="http://www.w3.org/2001/04/xmlenc#"
         URI="#Id-6831bf5c-f4dc-d611-bb59-90b4c67d3be5" />
      </ReferenceList>
    </Security>
  </Header>
  <Body xmlns="http://schemas.xmlsoap.org/soap/envelope/">
      <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
                  Type="http://www.w3.org/2001/04/xmlenc#Content"

xml:id="id-6831bf5c-f4dc-d611-bb59-90b4c67d3be5" >
      <EncryptionMethod xmlns="http://www.w3.org/2001/04/xmlenc#"
                      Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
      <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#" >
         <KeyName xmlns="http://www.w3.org/2000/09/xmldsig#" >WSDK Sample Symmetric Key</KeyName>
      </KeyInfo>
      <CipherData xmlns="http://www.w3.org/2001/04/xmlenc#" >
        <CipherValue xmlns="http://www.w3.org/2001/04/xmlenc#" >s8YHzbGSxsgfslN6
        eJu7UeiTExKeqwCHrqFml24C6mY8SLxhNE0Vy6xBuS50uchjbupjE4Z8EtkSuyljR8mkpmrW
        GCfUQEvrW7iu1ji0j4XR7m5P4dgxh7RYwqxqoprwTnCZ9b6X9D/UpN0aAYVdNs+S6l2Rw56d
        s5Gf4sv2f/sTYpEHTYPReqyo+9LSm9CsbtlPXia9djkH+75PUtsKZXZvHnVfHICQGjBJPpsh
        eE7Dq7mt8AkKVQ==</CipherValue>
       </CipherData>
    </EncryptedData>
  </Body>
</Envelope>
---------------------------------------
```

Server approves the request and returns an unencrypted message. This shows one-way encoding only.

```
------------ RESPONSE -----------------
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" >
  <Body xmlns="http://schemas.xmlsoap.org/soap/envelope/" >
    <AddIntResponse xmlns="http://temp.uri/" >
      <CallReturn type="http://www.w3.org/2001/XMLSchema:int" >7</CallReturn>
    </AddIntResponse>
  </Body>
</Envelope>
---------------------------------------
```

Example 24.384. Adding a SOAP:Header

This example makes call to the interop round4 test echoVoidSoapHeader (exists in the demo DB). Request and response wire dumps are listed below PL code excerpt.

```
      ...
      resp := soap_client (url=>url, operation=>'echoVoidSoapHeader',
                           parameters=>vector(),
                           headers=>vector(
                             vector('echoMeStringRequest',
                                    'http://soapinterop.org/:echoMeStringRequest',
                                    1,
                                    'http://schemas.xmlsoap.org/soap/actor/next') ,
                                    vector('String')),
                           style=>7,
                           soap_action=>'"http://soapinterop.org"',
                           target_namespace=>'http://soapinterop.org'
                           );
      ...
```

This will produce the following SOAP message:

The SOAP request

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:ns1="http://soapinterop.org/echoheader/"
 xmlns:ns0="http://soapinterop.org/">
   <SOAP:Header>
     <ns0:echoMeStringRequest SOAP:mustUnderstand='1' SOAP:actor='http://schemas.xmlsoap.org/soap/actor/n
       <ns1:varString>String</ns1:varString>
     </ns0:echoMeStringRequest>
   </SOAP:Header>
   <SOAP:Body>
     <echoVoidSoapHeader xmlns='http://soapinterop.org' ></echoVoidSoapHeader>
   </SOAP:Body>
</SOAP:Envelope>
```

And will receive the SOAP server response:

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns2="http://soapinterop.org/echoheader/"
  xmlns:ns1="http://soapinterop.org/"
  xmlns:ns0="http://soapinterop.org">
  <SOAP:Header>
    <ns1:echoMeStringResponse>
      <ns2:varString>String</ns2:varString>
    </ns1:echoMeStringResponse>
  </SOAP:Header>
  <SOAP:Body>
    <ns1:echoVoidSoapHeaderResponse />
  </SOAP:Body>
</SOAP:Envelope>
```

Example 24.385. Performing SOAP call over secure connection

The example code below makes a simple 'upper-case' service, exposes it under secured endpoint and invoke with x509 certificate. Note that default HTTPS listener MUST be enable in the INI file. Also the cli5.p12 certificate file MUST exists in server working directory.

```
SQL> create procedure
  WS.SOAPDEMO.SOAPTEST (in par varchar)
{
  return (upper(par));
};

SQL> grant execute on WS.SOAPDEMO.SOAPTEST to SOAPDEMO;

SQL> VHOST_DEFINE (vhost=>'*sslini*',lhost=>'*sslini*',lpath=>'/secure',
 ppath=>'/SOAP/',soap_user=>'SOAPDEMO');

-- SSL connection with server's certificate verification
SQL> select xpath_eval ('//CallReturn/text()', xml_tree_doc (
        soap_client (url=>'https://localhost:4443/secure',
        operation=>'SOAPTEST', parameters=>vector('par', 'demotext'),
        ticket=>'cli5.p12', passwd=>'secret')));
-- SSL connection only, no certificate verification
SQL> select xpath_eval ('//CallReturn/text()', xml_tree_doc (
        soap_client (url=>'https://localhost:4443/secure',
        operation=>'SOAPTEST', parameters=>vector('par', 'demotext'),
        ticket=>'1')));
-- response
callret
VARCHAR
_______________________________________________________________________________

DEMOTEXT
```

See Also

Signature Templates

soap_make_error

soap_make_error — Creates a SOAP error reply XML message based on its parameters.

Synopsis

```
varchar soap_make_error ( in soap_code varchar ,
                          in sql_state varchar ,
                          in error_message varchar ,
                          in soap_version integer ,
                          in do_uddi integer ) ;
```

Description

This function creates a SOAP error reply based on the given parameters. It returns the generated XML as a varchar value.

Parameters

soap_code
: Required. The fault code according to the SOAP specification.

sql_state
: Required. The error's SQL state.

error_message
: Required. The error text.

soap_version
: Optional (default 11). The SOAP version used to encode the SOAP error reply.

do_uddi
: Optional (default 0). 1 = produce UDDI error format; 0 = SOAP error format.

Return Types

Returns generated XML as a varchar.

soap_print_box

soap_print_box — Formats an SQL value and returns it as a generated XML fragment.

Synopsis

soap_print_box (in *value* any ,
in *enclosing_tag* varchar ,
in *date_encoding_type|soap_version* integer) ;

Description

This function formats an SQL value as an XML fragment and returns it. This is used internally by the SOAP server to encode the output parameter values and return values when processing a SOAP request.

Parameters

value
: Required. Any SQL value to be represented as an XML fragment.

enclosing_tag
: Required. The XML tag to place the value into.

date_encoding_type
: Optional. Valid only if a value is a datetime. If supplied, the enclosing_tag is ignored and the date is printed out in one of the following formats:

soap_version
: Optional(Default 1). Valid only if the value type is not a datetime. The soap_version (1 for SOAP 1.0, 11 for SOAP 1.1).

Return Types

Generated XML fragment.

soap_sdl

soap_sdl — Generate SDL document for a PL module and return it as a varchar.

Synopsis

```
soap_sdl ( in module_name varchar ,
           in URL varchar ) ;
```

Description

This function generates a SDL for the procedures in a PL module the same way as /SOAP/services.xml is generated for the procedures in WS.SOAP.

Parameters

module
: Required. The name of the module to describe. This can be partial name.

URL
: Optional (default is the current VSP path if in VSP context. Otherwise error). The URL to include in the SDL file

Return Types

Returns the SDL document describing the module as an varchar value.

soap_server

soap_server — Execute SOAP request and return XML reply as a varchar.

Synopsis

```
varchar soap_server ( in req_xml any ,
                      in soap_method varchar ,
                      in lines any ,
                      in soap_version long ,
                      in procedure_mappings any ) ;
```

Description

This function executes the SOAP request in the same way as it it was directed to the `/SOAP` physical path. It returns the XML SOAP reply as a varchar value.

Parameters

req_xml
: Required. The XML entity of the SOAP request to execute.

soap_method
: Optional(default ""). The "SOAPAction" header field value

lines
: Optional(default NULL). The Request header fields (the lines parameter to the VSPs for HTTP)

soap_version
: Optional(default 11). The SOAP version (11 for SOAP 1.1 and 1 for SOAP 1.0)

procedure_mappings
: Optional(default NULL). A vector of pairs (<SOAP_method>, <PL procedure>) mapping the SOAP call request to the corresponding PL function name. If empty, then the mapping proceeds by taking the local name of the SOAP method and finding a procedure with the same name in the executing user's current qualifier and owner.

Return Types

Returns the XML SOAP reply as a varchar value.

soap_wsdl

soap_wsdl — Generate WSDL document for a PL module and return it as a varchar.

Synopsis

```
varchar soap_wsdl ( in module_name varchar ,
                    in URL varchar ) ;
```

Description

This function generates WSDL for the procedures in a PL module the same way as /SOAP/services.wsdl is generated for the procedures in WS.SOAP.

Parameters

module
: Required. The name of the module to describe. This can be partial name.

URL
: Optional(default the current VSP path if in VSP context. Otherwise error). The URL to include in the WSDL file

Return Types

Returns the WSDL document describing the module as an varchar value.

soap_wsdl_import

SOAP_WSDL_IMPORT

Synopsis

```
array DB.DBA. SOAP_WSDL_IMPORT ( in target_url varchar );
```

Description

The `soap_wsdl_import()` function is used to import operations and types definitions from an WSDL file on a remote server. The retrieved file will be parsed and PL (procedure language) wrappers will be generated for each SOAP operation that is described. The SOAP service will be represented by a PL module which will be the overall container the generated PL wrappers. Once the WSDL file has been imported the PL wrappers are automatically generated and available for use.

Parameters

target_url
: A URL to a WSDL file.

Return Types

soap_wsdl_import() returns an array that consists of the name of module, the name of PL wrapper, and a vector of parameter name/type pairs. An example of the result is: vector ([module name], [wrapper1], vector ([param1], [type1], [param2], [tyep2] ...), [wrapper2] vector (...))

Examples

Example 24.386. Importing and using a Web Services description file

```
SQL> soap_wsdl_import('http://[host:port]/chemistry.wsdl');

Done. -- 1286 msec.
SQL> select xml_tree_doc (Chemistry.getElementBySymbol('Ni'));
callret
VARCHAR
_______________________________________________________________________________

<getElementBySymbolResponse>
  <return>
    <atomicnumber>28</atomicnumber>
    <symbol>Ni</symbol>
    <name>Nickel</name>
    <mass>58.69</mass>
    <meltingPoint>1726.0</meltingPoint>
    <boilingPoint>3005.0</boilingPoint>
    <found>1751</found>
 </return>
</getElementBySymbolResponse>

Note: the namespaces from response are omitted for better reading.
```

See Also

soap_wsdl()

soap_box_structure

soap_box_structure

Synopsis

```
any soap_box_structure ( in elem1 varchar ,
                         in value1 any ,
                         in ... ,
                         in elem1 varchar ,
                         in value1 any ) ;
```

Description

This function provides a way to encapsulate a structure suitable for soap serialization. It accepts a name/value pairs which represents name and value of elements of a structure.

For example structure :

```
{
  varString 'hello',
    varInt  1234
}
```

will be represented as soap_box_structure ('varString', 'hello', 'varInt', 1234); furthermore value returned from soap_box_structure passed as an output parameter to the procedure (named structname) working as SOAP method will return :

```
    <structName>
        <varString>hello</varString>
        <varInt>1234</varInt>
</structName>
```

Please note that this is a one of possibilities to express structures for SOAP request/response in Virtuoso/PL. The other way to express structure is to use User Defined Types (see SOAP & WSDL section of the documentation).

Parameters

elem
: name which represents name of the member of a structure

value
: value which represents value of the member of a structure

Return Types

The return type of `soap_box_structure()` is an array representing a structure.

soap_current_url

soap_current_url — Returns URL used to access particular HTTP resource.

Synopsis

```
varchar soap_current_url ( );
```

Description

This function is used to get in VSP/VSPX context the URL used to access current resource (page). It check whether 'Host' header field is specified, if not will return as a host information the IP address resolved via connected socket information.

Return Types

The return type of `soap_current_url()` is an string containing the URL of the HTTP request.

space

space — returns a new string of count spaces

Synopsis

space (in *count* integer) ;

Description

`space` returns a new string, composed of count spaces. If count is zero, an empty string '' is returned.

```
space(5) -> '     '
```

split_and_decode

split_and_decode — converts escaped var=val pairs to a vector of strings

Synopsis

```
vector or string split_and_decode ( in coded_str varchar ,
                                    in case_mode integer ,
                                    in str varchar ) ;
```

Description

split_and_decode converts the escaped var=val pair inputs text to a corresponding vector of string elements. If the optional third argument is a string of less than three characters, then does only the decoding (but no splitting) and returns back a string.

Parameters

coded_str
: Input string to be converted.

case_mode
: This optional second argument, if present should be an integer either 0, 1 or 2, which tells whether "variable name"-parts (those at the left side of the fourth character given in third argument (or = if using the default URL-decoding)) are converted to UPPERCASE (1), lowercase (2) or left intact (0 or when the second argument is not given).

str
: If this argument is a string of less than three characters then this function will only decode without splitting and will return a string.

Examples

Example 24.387. Using split_and_decode

```
split_and_decode("Tulipas=Taloon+kumi=kala&Joka=haisi
            +pahalle&kuin&%E4lymystöporkkana=ilman ruuvausta",1)
produces a vector:
("TULIPAS" "Taloon kumi=kala" "JOKA" "haisi pahalle" "KUIN" NULL
"ÄLYMYSTÖPORKKANA" "ilman ruuvausta")

split_and_decode(NULL)   => NULL
split_and_decode("")     => NULL
split_and_decode("A")    => ("A" NULL)
split_and_decode("A=B")  => ("A" "B")

split_and_decode("A&B")  => ("A" NULL "B" NULL)
split_and_decode("=")    => ("" "")
split_and_decode("&")    => ("" NULL "" NULL)
split_and_decode("&=")   => ("" NULL "" "")
split_and_decode("&=&")  => ("" NULL "" "" "" NULL)
split_and_decode("%")    => ("%" NULL)
split_and_decode("%%")   => ("%" NULL)
split_and_decode("%41")  => ("A" NULL)
split_and_decode("%4")   => ("%4" NULL)
split_and_decode("%?41") => ("%?41" NULL)
```

Can also work like Perl's split function (we define the escape prefix and space escape character as NUL-characters, so that they will not be encountered at all:

```
split_and_decode('Un,dos,tres',0,'\0\0,') => ("Un" "dos" "tres")
split_and_decode("Un,dos,tres",1,'\0\0,') => ("UN" "DOS" "TRES")
split_and_decode("Un,dos,tres",2,'\0\0,') => ("un" "dos" "tres")
```

Can also be used as replace and ucase (or lcase) together, for example, here we use the comma as space-escape instead of element-separator: (not recommended, use replace and ucase instead.

```
split_and_decode("Un,dos,tres",0,'\0,')   => "Un dos tres"
split_and_decode("Un,dos,tres",1,'\0,')   => "UN DOS TRES"
```

Can be also used for decoding (some of) MIME-encoded mail-headers:

```
split_and_decode('=?ISO-8859-1?Q?Tiira_lent=E4=E4_taas?=',0,'=_')
=>  "=?ISO-8859-1?Q?Tiira lentää taas?="

split_and_decode('Message-Id: <199511141036.LAA06462@correo.unet.ar>\n
             From: "=?ISO-8859-1?Q?Jorge_Mo=F1as?=" <jorgem@unet.ar>\n
             To: "Jore Carvajal" <carvajal@wanabee.fr>\nSubject: RE: Um-pah-pah\n
             Date: Wed, 12 Nov 1997 11:28:51 +0100\n
             X-MSMail-Priority: Normal\nX-Priority: 3\n
             X-Mailer: Molosoft Internet Mail 4.70.1161\nMIME-Version: 1.0\n
             Content-Type: text/plain; charset=ISO-8859-1\n
             Content-Transfer-Encoding: 8bit\nX-Mozilla-Status: 0011',
1,'=_\n:');
=> ('MESSAGE-ID' ' <199511141036.LAA06462@correo.unet.ar>'
'FROM' ' "=?ISO-8859-1?Q?Jorge Moñas?=" <jorgem@unet.ar>'
'TO' ' "Jore Carvajal" <carvajal@wanabee.fr>'
'SUBJECT' ' RE: Um-pah-pah'
'DATE' ' Wed, 12 Nov 1997 11:28:51 +0100'
'X-MSMAIL-PRIORITY' ' Normal'
'X-PRIORITY' ' 3'
'X-MAILER' ' Molosoft Internet Mail 4.70.1161'
'MIME-VERSION' ' 1.0'
'CONTENT-TYPE' ' text/plain; charset=ISO-8859-1'
'CONTENT-TRANSFER-ENCODING' ' 8bit'
'X-MOZILLA-STATUS' ' 0011')
```

Same, but let's use space, not colon as a variable=value separator:

```
split_and_decode('Message-Id: <199511141036.LAA06462@correo.unet.ar>\n
             From: "=?ISO-8859-1?Q?Jorge_Mo=F1as?=" <jorgem@unet.ar>\n
             To: "Jore Carvajal" <carvajal@wanabee.fr>\nSubject: RE: Um-pah-pah\n
             Date: Wed, 12 Nov 1997 11:28:51 +0100\n
             X-MSMail-Priority: Normal\nX-Priority: 3\n
             X-Mailer: Molosoft Internet Mail 4.70.1161\nMIME-Version: 1.0\n
             Content-Type: text/plain; charset=ISO-8859-1\n
             Content-Transfer-Encoding: 8bit\nX-Mozilla-Status: 0011',
1,'=_\n ')
=> ('MESSAGE-ID:' '<199511141036.LAA06462@correo.unet.ar>'
'FROM:' '"=?ISO-8859-1?Q?Jorge Moñas?=" <jorgem@unet.ar>'
'TO:' '"Jore Carvajal" <carvajal@wanabee.fr>'
'SUBJECT:' 'RE: Um-pah-pah'
'DATE:' 'Wed, 12 Nov 1997 11:28:51 +0100'
'X-MSMAIL-PRIORITY:' 'Normal'
'X-PRIORITY:' '3'
'X-MAILER:' 'Molosoft Internet Mail 4.70.1161'
'MIME-VERSION:' '1.0'
'CONTENT-TYPE:' 'text/plain; charset=ISO-8859-1'
'CONTENT-TRANSFER-ENCODING:' '8bit'
'X-MOZILLA-STATUS:' '0011')
```

Of course this approach does not work with multiline headers, except somewhat kludgously. If the lines are separated by CR+LF, there is left one trailing CR at the end of each value part string.

sprintf

sprintf — returns a formatted string

Synopsis

sprintf (*format* string ,
arg_1 any ,
... ,
arg_x any) ;

Description

sprintf returns a new string formed by "printing" a variable number of arguments arg_1 - arg_x according to the format string format, that is, exactly the same way as with the sprintf function of C language. However the sprintf function enforces some additional limitations over the sprintf C function. It does not allow for single value output to take more than 2000 characters. It does support the following additional format characters:

diouxXeEfgcs - as in the C language printf

S - as 's' but escapes the single quotes by doubling them (as per SQL/92). This is suitable for constructing dynamic SQL statements with string literals inline.

```
sprintf ('insert into testit (data) values ('%S')', 'Test ''Real'' data')
    -> insert into testit (data) values ('Test ''Real'' data')
```

I - as 's' but escapes the string value to form a valid identifier name (will double the double quotes). This is suitable for constructing dynamic SQL statements with identifiers inline.

```
sprintf ('select * from "%I"', 'Big "Table" Name')
    -> select * from "Big ""Table"" Name"
```

U - as 's' but escapes the string value as an HTTP URL (same as http_url() function). Useful for making dynamic VSP content

```
sprintf ('<a href="%U">', 'day & night')
    -> <a href="day+%26+night">
```

R - use to replace leading spaces: the modifiers are one of + - # 0 and space.

```
sprintf ('%-R', ' 123')
-123
sprintf ('%-R', '  123')
--123
```

V - as 's' but escapes the string value as an HTTP Value (same as http_value). Useful for making dynamic VSP content

```
sprintf ('<INPUT name="test" value="%V">', 'day & night')
    -> <INPUT name="test" value="day &amp; night">
```

Note that the sprintf format length and precision modifiers do not apply to the extension format characters

```
sprintf('Int=%d/%o/%x, String=%s, Character=%c',
    42798,42798,42798,'la cadena',65)
 -> 'Int=42798/123456/a72e, String=la cadena, Character=A'
```

In addition, *%{varname}U* syntax extension is supported, mostly for uing connection variables in RDF IRI classes . That syntax is to print the value of client connection variable *varname* using print format *%U* . Other format characters are not supported for that feature in current version of Virtuoso but might be supported in the future.

sprintf_inverse

sprintf_inverse — returns vector on a specified pattern.

Synopsis

sprintf_inverse (*strg* string ,
format string ,
err integer) ;

Description

sprintf_inverse gets a string to parse, a format string and an integer (0,1 or 2) that indicates error recovery method. If the first argument matches the format string then it returns vector of the pattern values.

Parameters

strg
: a string to parse

format
: a format string

err
: indicates the error recovery method in case the string does not match to the format: 0 - to return NULL, 1 - to return shorter vector that consists of all values extracted before the mismatch has been detected, 2 - to return vector of expected length with NULLs instead of values that were not found due to mismatch

Examples

Example 24.388. Example 1

```
SQL> select sprintf_inverse ('qwe123asd456zxcABRACADABRA', 'qwe%dasd%dzxc%s', 2);
callret
VARCHAR
_______________________________________________________________________________

vector(123,456,0x00adfae8)

1 Rows. -- 0 msec.
SQL>
```

Example 24.389. Example 2

```
SQL> select sprintf_inverse ('---123', '%-R', 2);
callret
VARCHAR
_______________________________________________________________________________

vector(0x00b070e8)

1 Rows. -- 0 msec.
SQL>
```

sprintf_iri

sprintf_iri — returns a formatted string that is marked as an IRI string.

Synopsis

sprintf_iri (*format* string ,
arg_1 any ,
... ,
arg_x any) ;

Description

sprintf_iri is similar to sprintf and returns a new string formed by "printing" a variable number of arguments arg_1 - arg_x according to the format string format. The difference is that the returned string is marked as being IRI string so some applications and clients may distinguish between RDF reference string and RDF literal.

Note

No matter what is the default server charset or default encoding of host OS, IRI strings are supposed to be in UTF-8 encoding so string arguments to print as well as the format itself are supposed to be UTF-8. Application may use calls of `charset_recode` as arguments.

Examples

Example 24.390. Simple Use

```
create function job_history(
  in EMPLOYEE_ID integer,
  in START_DATE date) returns varchar
{
return sprintf_iri
('http://demo.openlinksw.com:8890/hr/job_history#%d_%s',
EMPLOYEE_ID, cast (START_DATE as varchar) );
};
```

sprintf_iri_or_null

sprintf_iri_or_null — returns a formatted IRI string or null if any of the arguments except the first is null.

Synopsis

sprintf_iri_or_null (
format string ,
arg_1 any ,
... ,
arg_x any
) ;

Description

sprintf_iri_or_null is similar to sprintf_iri and returns a new string formed by "printing" a variable number of arguments arg_1 - arg_x according to the format string format. The difference is that the function can return null if any of the arguments except the first one is null.

The returned string is marked as being IRI string so some applications and clients may distinguish between RDF reference string and RDF literal.

Note

No matter what is the default server charset or default encoding of host OS, IRI strings are supposed to be in UTF-8 encoding so string arguments to print as well as the format itself are supposed to be UTF-8. Application may use calls of `charset_recode` as arguments.

Examples

Example 24.391. Simple Use

```
create function job_history(
  in EMPLOYEE_ID integer,
  in START_DATE date) returns varchar
{
return sprintf_iri_or_null
('http://demo.openlinksw.com:8890/hr/job_history#%d_%s',
EMPLOYEE_ID, cast (START_DATE as varchar) );
};
```

sprintf_or_null

sprintf_or_null — returns a formatted string or null if any of the arguments except the first is null.

Synopsis

sprintf_or_null (*format* string ,
arg_1 any ,
... ,
arg_x any) ;

Description

sprintf_or_null is similar to sprintf and returns a new string formed by "printing" a variable number of arguments arg_1 - arg_x according to the format string format. The difference is that the function can return null if any of the arguments except the first one is null.

Examples

Example 24.392. Simple Use

```
create function job_history(
  in EMPLOYEE_ID integer,
  in START_DATE date) returns varchar
{
return sprintf_or_null
('http://demo.openlinksw.com:8890/hr/job_history#%d_%s',
EMPLOYEE_ID, cast (START_DATE as varchar) );
};
```

sql_columns

sql_columns — get column information from table on a remote DSN

Synopsis

```
vector sql_columns ( in dsn varchar ,
                     in qualifier varchar ,
                     in owner varchar ,
                     in table_name varchar ,
                     in column varchar ) ;
```

Description

This function corresponds to the ODBC catalog call of similar name. It and related functions are used by the virtual database to query remote data dictionaries.

The dsn argument must refer to a dsn previously defined by `vd_remote_data_source` or ATTACH TABLE.

For instance, the qualifier argument corresponds to the szTableQualifier and cbTableQualifier arguments of an ODBC catalog function. The SQL NULL value corresponds to the C NULL value. The arguments can contain % signs, which are interpreted as in LIKE.

Parameters

As defined in ODBC API for the corresponding catalog call.

Return Types

As defined in ODBC API for the corresponding catalog call.

This function returns an array, with one element for each row of the result set. Each row is represented as an array with one element for each column.

Examples

Example 24.393. Simple example

```
dbg_obj_print (sql_columns ('Local Virtuoso', 'Demo', NULL, 'Orders', NULL));
->
(
("Demo" "demo" "Orders" "OrderID" 4 "INTEGER" 10 10 <DB NULL> 2 1 <DB NULL> <DB NULL> 4 <DB NULL> 10 1 1
("Demo" "demo" "Orders" "CustomerID" 12 "VARCHAR" 5 5 <DB NULL> 2 1 <DB NULL> <DB NULL> 12 <DB NULL> 5 2
("Demo" "demo" "Orders" "EmployeeID" 4 "INTEGER" 10 10 <DB NULL> 2 1 <DB NULL> <DB NULL> 4 <DB NULL> 10 3
("Demo" "demo" "Orders" "OrderDate" 11 "DATETIME" 19 19 <DB NULL> 2 1 <DB NULL> <DB NULL> 11 3 19 4 4 )
("Demo" "demo" "Orders" "RequiredDate" 9 "DATE" 10 10 <DB NULL> 2 1 <DB NULL> <DB NULL> 9 1 10 5 5 )
("Demo" "demo" "Orders" "ShippedDate" 11 "DATETIME" 19 19 <DB NULL> 2 1 <DB NULL> <DB NULL> 11 3 19 6 6 )
("Demo" "demo" "Orders" "ShipVia" 4 "INTEGER" 10 10 <DB NULL> 2 1 <DB NULL> <DB NULL> 4 <DB NULL> 10 7 7
("Demo" "demo" "Orders" "Freight" 8 "DOUBLE PRECISION" 16 16 <DB NULL> 2 1 <DB NULL> <DB NULL> 8 <DB NULL
("Demo" "demo" "Orders" "ShipName" 12 "VARCHAR" 40 40 <DB NULL> 2 1 <DB NULL> <DB NULL> 12 <DB NULL> 40 9
("Demo" "demo" "Orders" "ShipAddress" 12 "VARCHAR" 60 60 <DB NULL> 2 1 <DB NULL> <DB NULL> 12 <DB NULL> 6
("Demo" "demo" "Orders" "ShipCity" 12 "VARCHAR" 15 15 <DB NULL> 2 1 <DB NULL> <DB NULL> 12 <DB NULL> 15 1
("Demo" "demo" "Orders" "ShipRegion" 12 "VARCHAR" 15 15 <DB NULL> 2 1 <DB NULL> <DB NULL> 12 <DB NULL> 15
("Demo" "demo" "Orders" "ShipPostalCode" 12 "VARCHAR" 10 10 <DB NULL> 2 1 <DB NULL> <DB NULL> 12 <DB NULL
("Demo" "demo" "Orders" "ShipCountry" 12 "VARCHAR" 15 15 <DB NULL> 2 1 <DB NULL> <DB NULL> 12 <DB NULL> 1
)
```

See Also

`sql_data_sources` , `sql_columns` `sql_primary_keys` , `sql_gettypeinfo` , `sql_statistics` , `sql_tables` .

sql_data_sources

sql_data_sources — get list of available DSNs

Synopsis

```
sql_data_sources (   );
```

Description

`sql_data_sources` is used to get the list of datasources available to the dsn. It returns a vector of 2 element vectors containing Data source name and type pairs.

Parameters

As defined in ODBC API for the corresponding catalog call.

Return Types

As defined in ODBC API for the corresponding catalog call.

A vector containing a 2 element vector for each defined data source with a string data source name as 1st element and a string data source type as the 2nd element.

Examples

Example 24.394. Simple Use

```
SQL> select aref (aref (sql_data_sources(), 0), 0);
callret
VARCHAR
_______________________________________________________________________________

Local Virtuoso

1 Rows. -- 4 msec.
```

See Also

`sql_columns` `sql_primary_keys` , `sql_gettypeinfo` , `sql_statistics` , `sql_tables` .

sql_gettypeinfo

sql_gettypeinfo — return type information from a remote DSN

Synopsis

```
vector sql_gettypeinfo ( in dsn varchar ,
                         in type integer ) ;
```

Description

You can use the functions described here to find out information about the remote datasources that you are using. These could be especially useful in Virtuoso PL later on if you are not able to know everything about the remote tables ahead of time for the ATTACH TABLE statement. statement

These SQL functions correspond to the ODBC catalog calls of similar name.

The dsn argument must refer to a dsn previously defined by vd_remote_data_source or ATTACH TABLE.

By default information for all the data types supported by the remote is returned. The optional type argument (defaults to SQL_ALL_TYPES) limits the information returned to cover only the ODBC type number supplied.

These functions return an array, with one element for each row of the result set. Each row is represented as an array with one element for each column.

```
dbg_obj_print (sql_gettypeinfo ('Local Virtuoso'));
->
(
 ("character" 1 2000 "'" "'" "length" 1 1 3 0 0 0 "varchar" <DB NULL> <DB NULL> )
 ("numeric" 2 40 "" "" "precision,scale" 1 1 2 0 0 0 "numeric" 0 15 )
 ("decimal" 3 40 "" "" "precision,scale" 1 1 2 0 0 0 "decimal" 0 15 )
 ("integer" 4 10 "" "" <DB NULL> 1 1 2 0 0 0 "integer" 0 10 )
 ("smallint" 5 3 "" "" <DB NULL> 1 1 2 0 0 0 "smallint" <DB NULL> <DB NULL> )
 ("smallint" -7 3 "" "" <DB NULL> 1 1 2 0 0 0 "smallint" <DB NULL> <DB NULL> )
 ("float" 6 16 "" "e0" <DB NULL> 1 1 2 0 0 0 "double precision" <DB NULL> <DB NULL> )
 ("real" 7 16 "" "e0" <DB NULL> 1 1 2 0 0 0 "real" <DB NULL> <DB NULL> )
 ("double precision" 8 16 "" "e0" <DB NULL> 1 1 2 0 0 0 "double precision" <DB NULL> <DB NULL> )
 ("varchar" 12 2000 "'" "'" "length" 1 1 3 0 0 0 "varchar" <DB NULL> <DB NULL> )
 ("long varchar" -1 2147483647 "'" "'" <DB NULL> 1 1 0 0 0 0 "long varchar" <DB NULL> <DB NULL> )
 ("long varbinary" -4 2147483647 "'" "'" <DB NULL> 1 1 0 0 0 0 "long varbinary" <DB NULL> <DB NULL> )
 ("datetime" 11 19 "{ts" "}" <DB NULL> 1 1 3 0 0 0 "datetime" <DB NULL> <DB NULL> )
 ("timestamp" -2 10 "0x" <DB NULL> <DB NULL> 0 0 2 0 0 0 "timestamp" <DB NULL> <DB NULL> )
 ("time" 10 8 "{t" "}" <DB NULL> 1 1 2 0 0 0 "time" <DB NULL> <DB NULL> )
 ("date" 9 10 "{d" "}" <DB NULL> 1 1 2 0 0 0 "date" <DB NULL> <DB NULL> )
 ("binary" -2 2000 "0x" "" "length" 1 1 2 0 0 0 "varbinary" <DB NULL> <DB NULL> )
 ("varbinary" -3 2000 "0x" "" "length" 1 1 2 0 0 0 "varbinary" <DB NULL> <DB NULL> )
 ("nchar" -8 1000 "N'" "'" "length" 1 1 3 0 0 0 "nvarchar" <DB NULL> <DB NULL> )
 ("nvarchar" -9 1000 "N'" "'" "length" 1 1 3 0 0 0 "nvarchar" <DB NULL> <DB NULL> )
 ("long nvarchar" -10 1073741823 "N'" "'" <DB NULL> 1 1 0 0 0 0 "long nvarchar" <DB NULL> <DB NULL> )
)
```

See Also

sql_data_sources , sql_columns sql_primary_keys , sql_statistics , sql_tables .

sql_primary_keys

sql_primary_keys — get primary key information about a table on a remote DSN

Synopsis

```
vector sql_primary_keys ( in dsn varchar ,
                          in qualifier varchar ,
                          in owner varchar ,
                          in table_name varchar ) ;
```

Description

You can use the functions described here to find out information about the remote datasources that you are using. These could be especially useful in Virtuoso PL later on if you are not able to know everything about the remote tables ahead of time for the ATTACH TABLE statement. statement

These SQL functions correspond to the ODBC catalog calls of similar name. These are used to access the data dictionary of remote data sources inside the ATTACH TABLE process.

The dsn argument must refer to a dsn previously defined by vd_remote_data_source or ATTACH TABLE.

For instance, the qualifier argument corresponds to the szTableQualifier and cbTableQualifier arguments of an ODBC catalog function. A SQL NULL value corresponds to the C NULL value. The arguments can contain % signs, which are interpreted as in LIKE.

These functions return an array, with one element for each row of the result set. Each row is represented as an array with one element for each column.

Parameters

dsn
: The data source name string

qualifier
: Qualifier string . May contain wildcards as in 'De%'.

owner
: Table owner string . May contain wildcard characters in 'Dem%'.

table_name
: Table name string . May contain wildcard characters in 'Cust%'.

Return Types

As defined in ODBC API for the corresponding catalog call.

An array, with one element for each row of the result set. Each row is represented as an array with one element for each column.

Examples

Example 24.395. Simple Example

```
dbg_obj_print (sql_primary_keys ('Local Virtuoso', 'Demo', NULL, 'Orders'));
->
(
 ("Demo" "demo" "Orders" "OrderID" 1 "Orders" "Demo" "demo" "Orders" )
)
```

See Also

`sql_data_sources` , `sql_columns` `sql_gettypeinfo` , `sql_statistics` , `sql_tables` .

sql_statistics

sql_statistics — retrieve statistics information on remote DSN

Synopsis

```
vector sql_statistics ( in dsn varchar ,
                        in qualifier varchar ,
                        in owner varchar ,
                        in table_name varchar ,
                        in is_unique integer ,
                        in detail integer ) ;
```

Description

This SQL function corresponds to the ODBC catalog call of similar name. It is used to access the data dictionary of remote data sources inside the ATTACH TABLE process.

The dsn argument must refer to a dsn previously defined by vd_remote_data_source or ATTACH TABLE.

The qualifier argument corresponds to the szTableQualifier and cbTableQualifier arguments of an ODBC catalog function. A SQL NULL value corresponds to the C NULL value. The arguments can contain % signs, which are interpreted as in LIKE.

These functions return an array, with one element for each row of the result set. Each row is represented as an array with one element for each column.

Parameters

As defined in ODBC API for the corresponding catalog call.

Return Types

As defined in ODBC API for the corresponding catalog call.

An array with one element for each row of the result set. Each row is represented as an array with one element for each column.

Examples

Example 24.396. Simple Example

```
dbg_obj_print (sql_statistics ('Local Virtuoso', 'Demo', NULL, 'Orders', 0, 1));
->
(
 ("Demo" "demo" "Orders" 0 "Demo" "Orders" 3 1 "OrderID" <DB NULL> <DB NULL> <DB NULL> <DB NULL> )
)
```

See Also

`sql_data_sources` , `sql_columns` `sql_primary_keys` , `sql_gettypeinfo` , `sql_tables` .

sql_tables

sql_tables — get list of tables from remote DSN

Synopsis

```
vector sql_tables ( in dsn varchar ,
                    in qualifier varchar ,
                    in owner varchar ,
                    in table_name varchar ,
                    in tabletype varchar ) ;
```

Description

This function corresponds to the ODBC catalog call of similar name. It and related functions are used by the virtual database to query remote data dictionaries.

The dsn argument must refer to a dsn previously defined by vd_remote_data_source or ATTACH TABLE.

The qualifier argument corresponds to the szTableQualifier and cbTableQualifier arguments of an ODBC catalog function. A SQL NULL value corresponds to the C NULL value. The arguments can contain % signs, which are interpreted as in LIKE.

Parameters

As defined in ODBC API for the corresponding catalog call.

Return Types

As defined in ODBC API for the corresponding catalog call.

This function returns an array, with one element for each row of the result set. Each row is represented as an array with one element for each column.

Examples

Example 24.397. Simple example

```
dbg_obj_print (sql_tables ('Local Virtuoso', 'Demo', NULL, NULL, NULL));
->
(
 ("Demo" "demo" "Categories" "TABLE" <DB NULL> )
 ("Demo" "demo" "Customers" "TABLE" <DB NULL> )
 ("Demo" "demo" "Employees" "TABLE" <DB NULL> )
 ("Demo" "demo" "Order_Details" "TABLE" <DB NULL> )
 ("Demo" "demo" "Orders" "TABLE" <DB NULL> )
 ("Demo" "demo" "Products" "TABLE" <DB NULL> )
 ("Demo" "demo" "Shippers" "TABLE" <DB NULL> )
 ("Demo" "demo" "Suppliers" "TABLE" <DB NULL> )
)
```

See Also

sql_data_sources , sql_columns sql_primary_keys , sql_gettypeinfo , sql_statistics .

sql_special_columns

sql_special_columns — get special column information from table on a remote DSN

Synopsis

```
vector sql_special_columns ( in dsn varchar ,
                             in qualifier varchar ,
                             in owner varchar ,
                             in table_name varchar ,
                             in scope integer ,
                             in nullable integer ) ;
```

Description

This function corresponds to the ODBC catalog call of similar name. It and related functions are used by the virtual database to query remote data dictionaries.

First argument of the ODBC SQLSpecialColumns is always SQL_BEST_ROWID.

The dsn argument must refer to a dsn previously defined by `vd_remote_data_source()` or ATTACH TABLE.

For instance, the qualifier argument corresponds to the szTableQualifier and cbTableQualifier arguments of an ODBC catalog function. The SQL NULL value corresponds to the C NULL value. The arguments can contain % signs, which are interpreted as in LIKE.

Parameters

As defined in ODBC API for the corresponding catalog call.

Return Types

As defined in ODBC API for the corresponding catalog call.

This function returns an array, with one element for each row of the result set. Each row is represented as an array with one element for each column.

Examples

Example 24.398. Simple example

```
dbg_obj_print (sql_special_columns ('Local Virtuoso', 'Demo', 'demo', 'Orders', 0, 0));
->
(
 (0ld "OrderID" 4 "INTEGER" 10 10 <DB NULL> 1 )
)
```

See Also

`sql_data_sources` , `sql_columns` `sql_primary_keys` , `sql_gettypeinfo` , `sql_statistics` , `sql_tables` .

sql_procedures

sql_procedures — get procedures information for a remote DSN

Synopsis

```
vector sql_procedures ( in dsn varchar ,
                        in qualifier varchar ,
                        in owner varchar ,
                        in name varchar ) ;
```

Description

This function corresponds to the ODBC catalog call of similar name. It and related functions are used by the virtual database to query remote data dictionaries.

The dsn argument must refer to a dsn previously defined by `vd_remote_data_source()` or ATTACH TABLE.

For instance, the qualifier argument corresponds to the szTableQualifier and cbTableQualifier arguments of an ODBC catalog function. The SQL NULL value corresponds to the C NULL value. The arguments can contain % signs, which are interpreted as in LIKE.

Parameters

As defined in ODBC API for the corresponding catalog call.

Return Types

As defined in ODBC API for the corresponding catalog call.

This function returns an array, with one element for each row of the result set. Each row is represented as an array with one element for each column.

Examples

Example 24.399. Simple example

```
dbg_obj_print (sql_procedures ('Local Virtuoso', 'DB', 'DBA', 'VT_INC_INDEX_DB_DBA_MAIL_MESSAGE'));
->
(
 ("DB" "DBA" "VT_INC_INDEX_DB_DBA_MAIL_MESSAGE" <DB NULL> <DB NULL> <DB NULL> <DB NULL> 0ld )
)
```

See Also

`sql_data_sources` , `sql_columns` `sql_primary_keys` , `sql_gettypeinfo` , `sql_statistics` , `sql_tables` .

sql_write_private_profile_string

sql_write_private_profile_string — Sets a DSN's attribute value

Synopsis

```
integer sql_write_private_profile_string ( in dsn varchar ,
                                           in type varchar ,
                                           in name varchar ,
                                           in value varchar ) ;
```

Description

This function corresponds to the ODBC catalog call of similar name. It and related functions are used by the virtual database to query remote data dictionaries.

The type argument must be either 'system' or 'user'.

Sets a data source attribute by calling SQLWritePrivateProfileString from the ODBC Installer API.

Parameters

type denotes the DSN type.

others are passed to the SQLWritePrivateProfileString

Return Types

Returns the SQL state.

See Also

sql_data_sources , sql_columns sql_primary_keys , sql_gettypeinfo , sql_statistics , sql_tables .

sql_get_private_profile_string

sql_get_private_profile_string — Gets the DSN's attributes list

Synopsis

```
vector sql_get_private_profile_string ( in dsn varchar ,
                                        in type varchar ) ;
```

Description

This function corresponds to the ODBC catalog call of similar name. It and related functions are used by the virtual database to query remote data dictionaries.

The type argument must be either 'system' or 'user'.

Gets the data source attributes by calling SQLGetPrivateProfileString from the ODBC Installer API.

Parameters

dsn
: is the first argument passed to SQLGetPrivateProfileString.

type
: denotes the DSN type.

Return Types

Returns an array of 2-element arrays representing the name/value pairs.

See Also

`sql_data_sources` , `sql_columns` `sql_primary_keys` , `sql_gettypeinfo` , `sql_statistics` , `sql_tables` .

sql_config_data_sources

sql_config_data_sources — Configures a remote DSN as in the DSN attribute string

Synopsis

```
vector sql_config_data_sources ( in driver varchar ,
                                 in type varchar ,
                                 in attrib varchar ) ;
```

Description

This function creates or modifies a user or system data source based on the semicolon separated list of DSN attributes.

The type argument must be either 'system' or 'user'.

Configures the data source by calling SQLWriteDSNToIni/SQLWritePrivateProfileString/SQLSetConfigMode from the ODBC Installer API.

Parameters

driver is the Driver name passed to SQLWriteDSNToIni.

Return Types

Returns the ODBC status code.

See Also

sql_data_sources , sql_columns sql_primary_keys , sql_gettypeinfo , sql_statistics , sql_tables .

sql_get_installed_drivers

sql_get_installed_drivers — get column information from table on a remote DSN

Synopsis

```
vector sql_get_installed_drivers ( );
```

Description

This function corresponds to the ODBC Installer call of similar name. It and related functions are used by the virtual database to query remote data sources.

Parameters

None

Return Types

As defined in ODBC API for the corresponding catalog call.

This function returns an array of varchars.

Examples

Example 24.400. Simple example

```
dbg_obj_print (sql_get_installed_drivers ());
->
(
 "OpenLink Generic 32 Bit Driver v4.0"
)
```

See Also

`sql_data_sources` , `sql_columns` `sql_primary_keys` , `sql_gettypeinfo` , `sql_statistics` , `sql_tables` .

sql_remove_dsn_from_ini

sql_remove_dsn_from_ini — removes a DSN from the ODBC registry

Synopsis

```
vector sql_remove_dsn_from_ini ( in dsn varchar ,
                                 in dsn_type varchar ) ;
```

Description

This function corresponds to the ODBC installer call of similar name. It and related functions are used by the virtual database to handle remote data sources.

Parameters

As defined in ODBC Installer API for the corresponding catalog call.

Return Types

None.

See Also

`sql_data_sources` , `sql_columns` `sql_primary_keys` , `sql_gettypeinfo` , `sql_statistics` , `sql_tables` .

sql_transact

sql_transact — get list of available DSNs

Synopsis

sql_transact (in *dsn_name* varchar ,
in *rollback* integer) ;

Description

This procedure can be used to control the commit/rollback behavior of a particular remote data source during a Virtuoso transaction such as in a stored procedure.

Under normal circumstances Virtuoso will correctly commit or rollback all associated work as expected, however it may be desirable intervene. When issued without the second parameter a commit will be forced upon the current transactions of the dsn_name above the call to `sql_transact` regardless of overall outcome. When rollback = 1 is set then a rollback will be forced likewise, hence this will not rollback work on the remote dsn_name prior to `sql_transact` .

Parameters

dsn_name
: The name of the data source being used on the remote.

rollback
: Activate the desired behavior. Set to 1 for rollback, ignore to commit.

Return Types

None.

Examples

Example 24.401. Control remote rollback/commit behavior

This code fragment can be used to demonstrate the effects of directly controlling the rollback/commit behavior of remote data sources connected to Virtuoso.

```
create procedure TEST_ROLLBACK ()
{
  commit work;
  insert into mydsn..rb_test values (1);
  sql_transact('mydsn');
  insert into mydsn..rb_test values (2);
  rollback work;
};

create procedure TEST_ROLLBACK ()
{
  commit work;
  insert into mydsn..rb_test values (1);
  sql_transact('mydsn', 1);
  insert into mydsn..rb_test values (2);
  commit work;
};

delete from mydsn..rb_test;

commit work;
TEST_COMMIT();
select MIN (ID) from mydsn..rb_test;
```

```
-- returns 1

delete from mydsn..rb_test;

commit work;
TEST_ROLLBACK();
select MIN (ID) from mydsn..rb_test;

-- returns 2
```

See Also

sql_columns sql_primary_keys , sql_gettypeinfo , sql_statistics , sql_tables .

sql_write_file_dsn

sql_write_file_dsn — Creates a new file DSN

Synopsis

```
integer sql_write_file_dsn ( in dsn varchar ,
                             in string varchar ,
                             in value varchar ) ;
```

Description

This function calls the SQLWriteFileDSN as follows :

SQLWriteFileDSN (dsn, 'ODBC', string, value)

Parameters

As described in the SQLWriteFileDSN

Return Types

Returns the SQL state.

See Also

`sql_data_sources` , `sql_columns` `sql_primary_keys` , `sql_gettypeinfo` , `sql_statistics` , `sql_tables` .

sql_driver_connect

sql_driver_connect — Tries connecting using supplied connection string

Synopsis

```
integer sql_driver_connect ( in conn_str varchar );
```

Description

This function calls SQLDriverConnect with the supplied connection string and immediately disconnects after the call.

Parameters

As described in the SQLDriverConnect in connection string

Return Types

Returns the SQL return code (or exception on an ODBC error).

See Also

`sql_data_sources` , `sql_columns` `sql_primary_keys` , `sql_gettypeinfo` , `sql_statistics` , `sql_tables` .

sqrt

sqrt — calculate square root

Synopsis

sqrt (in *x* double precision);

Description

`sqrt` calculates the square root of its argument and returns it as a IEEE 64-bit float.

Parameters

x
: double precision

Return Values

`sqrt` returns a IEEE 64-bit float.

Examples

Example 24.402. Stored Procedure Example

Calculate square roots of integers between 2 numbers.

```
SQL> create procedure
calc_sqrts (in _from integer, in _to integer)
{
  declare i integer;
  declare result double precision;

  result_names (i, result);

  i := _from;
  while (i < _to)
   {
     result (i, sqrt (i));
     i := i + 1;
   }
}
;
(0) (0) (1) (1) (1) (1) (1) (1) (1) (1) (2) (2) (2) (1)
Done. -- 7 msec.
SQL> calc_sqrts(1, 10);
i                 result
INTEGER NOT NULL  DOUBLE PRECISION NOT NULL
_____________________________________________________________________________

1                                 1.000000
2                                 1.414214
3                                 1.732051
4                                 2.000000
5                                 2.236068
6                                 2.449490
7                                 2.645751
8                                 2.828427
9                                 3.000000

9 Rows. -- 3 msec.
```

See Also

`exp` , `sqrt` , `log10` , `power`

status

status — Returns statistics for a running server as a result set

Synopsis

status (in *option* varchar);

Description

Depending of input parameter displays a statistics for running server.

Note that counters reported by status will be reset after server startup.

Parameters

option
: This option controls the level of detail of the output returned from this procedure. Possible options are:

Return Types

Status returns a result set with one varchar column, one row per line of text in the status report.

Example

Example 24.403. Retrieving Server Status

```
SQL> status('rhck');

VARCHAR
_______________________________________________________________________________

OpenLink Virtuoso Server
Version 03.00.23NN-pthreads for Linux as of Nov 27 2002
<License information>

Database Status:
  File size 55705600, 6800 pages, 1345 free.
  2000 buffers, 180 used, 2 dirty 0 wired down, repl age 0 .
  Disk Usage: 188 reads avg 0 msec, 0% read last  80 s, 25 writes,
    7 read ahead, batch = 9.
Gate:  0 2nd in reads, 0 gate write waits, 0 in while read 0 busy scrap.
Log = demo.trx, 1597 bytes
5454 pages have been changed since last backup (in checkpoint state)
Current backup timestamp: 0x0000-0x00-0x00
Last backup date: unknown
Clients: 4 connects, max 11 concurrent, 25 licensed
RPC: 55 calls, -2 pending, 2 max until now, 0 queued
Checkpoint Remap 74 pages, 0 mapped back. 0 s atomic time.
    DB master 6800 total 1345 free 74 remap 0 mapped back
   temp  200 total 197 free

Lock Status: 0 deadlocks of which 0 2r1w, 0 waits,
   Currently 1 threads running 0 threads waiting 0 threads in vdb.
Pending:

Client 1111:5:  1944 bytes in, 29516 bytes out, 1 stmts.
Transaction status: PENDING, 1 threads.
Locks:

Running Statements:
 Time (msec) Text

Replication Status: Server  repl.
   anonymous            anonymous                    0 OFF.
```

```
   rep1                 rep1                            0 OFF.

Index Usage:
Table          Index             Touches   Reads %Miss   Locks   Waits   %W n-dead
WS.WS.SYS_DAV_RES_TYPES SYS_DAV_RES_TYPES       254       1    0%     132       0   0% 0
VAD.DBA.VAD_REGISTRY VAD_REGISTRY              1       1   50%       1       0   0% 0
VAD.DBA.VAD_HELP    VAD_HELP                  20       1    4%      20       0   0% 0

Hash indexes

47 Rows. -- 9 msec.
```

key_estimate

key_estimate — Get an estimate of row count for a given set of leading index key parts

Synopsis

key_estimate (in *table_name* varchar ,
in *index_name* varchar ,
in *k1* any ,
...) ;

Description

Given a table and an index, this function takes a random sample of the index, with the first key part equal to the third argument, the second equal to the fourth argument and so on. If only the table and index are given, the returned value is the approximate count of the entire index.

Normal cast rules are applied to convert the arguments to the types of the corresponding key parts. If the cast fails, -1 is returned.

The estimates are typically within 10% of the real count. If there have been random deletions or inserts leading to uneven page filling or if index entries are of greatly varying $ length, the estimates may be less precise.

Parameters

table_name
: The name of the table as appears in SYS_KEYS, case sensitive.

index_name
: The name of the index as appears in sys_keys, case sensitive.

k1
: The value of the first key part. Any number of arguments, up to the number of key parts in the index in question may follow.

Return

Key_estimate returns the approximate count of entries in the index with the specified values in the leading key parts. -1 if cannot be determined.

Example

Example 24.404. Returns the approximate count of entries in index STR1 where the first key part is the string '123'.

```
select key_estimate ('DB.DBA.T1', 'STR1', '123');
```

strcasestr

strcasestr — case-insensitive substring search

Synopsis

strcasestr (in *str* string ,
in *sub* string) ;

Description

`strcasestr` performs a case-insensitive substring search, returning a zero-based index pointing to beginning of first occurrence of *`sub`* or NULL if not found.

Parameters

str
: String to search from.

sub
: Substring to search for.

Return Types

An integer zero-based index to first occurrence of *`sub`* .

Examples

Example 24.405. Sample example

```
strstr('AbracadabrA','abrA')
    -> 7 (Found from the eighth character onwards)
strcasestr('AbracadabrA','abrA')
    -> 0 (Found from the beginning of string)
```

See Also

`strstr` .

strchr

strchr — strchr returns a zero-based index to the first occurrence of the character.

Synopsis

strchr (*str* string ,
char string or integer) ;

Description

strchr returns a zero-based index to the first occurrence of char. If char is not found NULL is returned. char can be given either as an integer ASCII value or a string, in which case the first character of that string is searched fo.

```
strchr('AbracadabrA','A')
                -> 0 (Found as the first character).
strrchr('AbracadabrA',65)
                -> 10 (Found as the eleventh character)
```

stringdate

stringdate — Convert a string to a datetime

Synopsis

stringdate (in *str* varchar) ;

Description

stringdate converts dates and timestamps from text to the internal DATETIME type.

The external format is: YYYY.MM.DD hh:mm.ss uuuuuu where uuuuuu represents number of microseconds.

If trailing parts are omitted from the string given to stringdate, they are assumed to be zero. The three first parts are mandatory. Note that use of

```
cast (x as datetime)
```

is preferred over this function.

Parameters

str
: A varchar date in human-readable (external) format.

Errors

Table 24.81. Errors signalled by `datestring` and `stringdate`

SQLState	Error Code	Error Text	Description
22007	DT006	Cannot convert <offending string> to datetime	
22002	DT007	Nulls not allowed as parameters to stringdate	

Examples

Example 24.406. Stringdate example with datestring_GMT

We are at central european time zone CET(GMT+1).

```
SQL> use Demo;

 Done. -- 3 msec.
 SQL> select (datestring_GMT (stringdate ('2000-01-01 22:00')));
 callret
 VARCHAR
 _______________________________________________________________________________

 2000-01-01 21:00:00.000000

 1 Rows. -- 45 msec.
 SQL> select left (datestring(ShippedDate), 10) from Orders
    where ShippedDate > stringdate ('1996.6.3');
 callret
 VARCHAR
 _______________________________________________________________________________

 1996-06-04
 1996-06-04
 1996-06-05
 1996-06-05
```

```
1996-06-05

5 Rows. -- 3 msec.
```

See Also

`datestring`

The discussion of TIMESTAMP datatype in section Database Concepts of the documentation.

stringtime

stringtime — converts string to a time

Synopsis

```
time stringtime ( in str varchar );
```

Description

Converts the argument to a time. Same as

```
cast (x as time)
```

string_output

string_output — make a string output stream

Synopsis

string_output (in *threshold* integer) ;

Description

A string output stream is a special object that may be used to buffer arbitrarily long streams of data. They are useful for handling data that would not otherwise fit within normal varchar size limitations. The HTTP output functions optionally take a string output stream handle as a parameter and then output to said stream instead of the HTTP client. A string output stream can be assigned to a database column in insert or update, causing the characters written to the stream to be assigned to the column as a narrow string.

The function `string_output_string` can be used to produce a varchar out of a string output stream. It may be called repeatedly to obtain several copies of the data. `http_rewrite` can be used to flush a string output stream.

If a string output stream is passed to the function `result` the data stored in it is sent to the client.

The string output object cannot be copied. It cannot therefore be assigned between two variables or passed by value (as an IN parameter.) It can be passed by reference (OUT, INOUT parameter.)

Parameters

threshold
: Optional size threshold, exceeding this the object will be stored in a temp directory specified by 'TempSesDir' INI parameter.

Examples

Example 24.407. Handling string output streams

This example takes a string as an argument, creates a new string output stream, writes the string into the stream and inserts stream contents to a DB table.

```
create table
foo_table (
a integer identity,
b long varchar,
primary key (a));

create procedure
foo_out (in x varchar)
{
  declare str_out any;
  declare str varchar;

-- Pass correct result metadata to client
  result_names (str);

-- Get a new string output stream
  str_out := string_output();

  http (x, str_out);

-- These produce the same result
  result (string_output_string (str_out));
  result (str_out);

-- insert string output contents
  insert into foo_table (b) values (str_out);
```

```
-- Write it again to the string output
  http (concat (' ', x), str_out);

  result (str_out);
}
;

SQL> foo_out ('Ceterum censeo, Carthaginem esse delendum!');
str
VARCHAR NOT NULL
_____________________________________________________________________________

Ceterum censeo, Carthaginem esse delendum!
Ceterum censeo, Carthaginem esse delendum!
Ceterum censeo, Carthaginem esse delendum! Ceterum censeo, Carthaginem esse delendum!

2 Rows. -- 2 msec.

SQL> select * from foo_table;
a                 b
INTEGER NOT NULL  LONG VARCHAR
_____________________________________________________________________________

1                 Ceterum censeo, Carthaginem esse delendum!

1 Rows. -- 2 msec.
```

See Also

string_output_stream, http, http_value

string_output_flush

string_output_flush — resets the state of the string_output object

Synopsis

string_output_flush (inout *stream* any) ;

Description

This function resets the state of the string output object. The string associated with the string output is dropped and is of 0 characters after this call.

Parameters

stream
: stream to clear, must have been created by the string_output function.

Examples

Example 24.408. Flush the String Output Stream

```
<?vsp
  declare ses nay;
  ses := string_output ();
  http ('this text never be displayed' , ses);
  string_output_flush (ses);
  http ('stream test' , ses);
  http (string_output_string (ses));
?>
```

string_output_gz_compress

string_output_gz_compress — compress a string_output with gzip algorithm

Synopsis

string_output_gz_compress (in *str_out_in* string_session ,
out *str_out_out* string_session) ;

Description

The string_output_gz_compress compresses its string_output argument using the gzip algorithm and writes the result to another string_output given as an argument. When successful, the number of bytes written to *str_out_out* is returned.

Parameters

str_out_in
: A string session as returned by string_output function.

str_out_out
: A string session as returned by string_output function.

Return Types

INTEGER number of bytes written to *str_out_out*

Examples

Example 24.409. GZIP test

Test if the gzip implementation works.

```
create procedure
gz_test_1 (in fname varchar)
{
  declare dta, dta_2, res any;
  declare len integer;

  res := string_output ();
  dta_2 := string_output ();
  dta := file_to_string_output (fname);

  result_names (len);

  -- send length of input to client

  result (length (dta));

  -- compress and return compressed size.

  result (string_output_gz_compress (dta, res));
  gz_uncompress (string_output_string (res), dta_2);

  if (md5 (dta) <> md5 (dta_2))
    signal ('SNAFU', 'GZIP algorithm implementation is faulty.');

  result (length (dta));
  end_result();
}
;

SQL> gz_test_1 ('tmp/test.dat');
len
```

```
INTEGER
_______________________________________________________________________________

11689
2477
11689

3 Rows. -- 11 msec.
```

See Also

string_output

string_output_string

string_output_string — produce a string out of a string output stream

Synopsis

string_output_string (in *string_out* any) ;

Description

This function is used to produce a string from contents of a string output stream. See `string_output` for more information about string output streams.

Parameters

string_out
: The string output stream

Return Values

A varchar with the contents of the string output stream.

Examples

Example 24.410. Convert string output stream into a string

Create a string output, write 'foo' and ' bar' into it, convert to string and produce a result.

```
SQL> create procedure
ex_string_output_string ()
{
  declare s_out any; s_out := string_output();
  declare s varchar;

  result_names (s);
  http('foo', s_out);
  http(' bar', s_out);
  s := string_output_string (s_out);
  result (s);
}
;

SQL> ex_string_output_string ();
s
VARCHAR NOT NULL
_____________________________________________________________________________

foo bar

1 Rows. -- 5 msec.
```

See Also

`string_output` , `http` , `http_value` , `http_rewrite` .

string_to_file

string_to_file — writes a varchar to a file

Synopsis

string_to_file (in *path* varchar ,
in *string* varchar ,
in *mode* integer) ;

Description

This function requires dba privileges.

`string_to_file` writes a varchar value or string session to a file. The path is relative to the server's working directory. The mode is an integer value interpreted as a position. A mode of 0 writes the content starting at offset 0. A mode of -1 appends to the end of the file. The append option is probably the most useful for producing application level logs, etc.

The string argument can also be a string output object. In this case the content is used as the string.

If the mode is -2, the new content supersedes the old. This is different from 0 in that the file will be truncated if the new content is shorter than the old.

The DirsAllowed and DirsDenied lists in Parameters section of the virtuoso configuration file (virtuoso.ini by default) are used to control disk access. An error 42000/FA024 is signalled if an attempt is made to write to a file in a directory to which disk access is not explicitly allowed.

Parameters

path
: varchar relative path.

string
: varchar or string session to write to the file.

mode
: integer mode.

Examples

Example 24.411. Simple example

Write an entry to an application log

```
SQL>string_to_file ('logs/applog.txt',
  concat (datestring(now()),'\t','Application started.\n'), -1);

Done. -- 4 msec.
SQL> quit;
bash$ cat logs/applog.txt
2001-03-19 15:15:12.000000      Application started.
bash$
```

See Also

`file_to_string`

strrchr

strrchr — returns a zero-based index to the last occurrence of the char in str.

Synopsis

strrchr (*str* string ,
char string or integer) ;

Description

strchr returns a zero-based index to the last occurrence of char in string. If char is not found NULL is returned. char can be given either as an integer ASCII value or a string, in which case the first character of that string is searched for in str.

```
strchr('AbracadabrA','A')
                -> 0 (Found as the first character).
strrchr('AbracadabrA',65)
                -> 10 (Found as the eleventh character)
```

strstr

strstr — substring search

Synopsis

strstr (in *str* string ,
in *sub* string) ;

Description

`strstr` performs a substring search, returning a zero-based index pointing to beginning of first occurrence of *`sub`* or NULL if not found.

Parameters

str
: String to search from.

sub
: Substring to search for.

Return Types

An integer zero-based index to first occurrence of *`sub`* .

Examples

Example 24.412. Sample example

```
strstr('AbracadabrA','abrA')
    -> 7 (Found from the eighth character onwards)
strcasestr('AbracadabrA','abrA')
    -> 0 (Found from the beginning of string)
```

See Also

`strcasestr` .

subseq

subseq — returns substring of a string or sub-vector of a vector

Synopsis

subseq (*str* string or vector ,
from integer ,
to integer or NULL) ;

Description

subseq returns a copy of subsequence of string or vector str using zero-based indices from (inclusive) and to (exclusive) to delimit the substring or the vector extracted.

If to is omitted or is NULL, then it equals by default to the length of str, i.e. everything from from to the end of str is returned.

If to and from are equal, an empty string "(empty vector) is returned.

If from is greater than to or length of str an error is signalled.

If str is NULL then NULL is returned.

```
subseq('AbracadabrA',0,4) -> 'Abra'
subseq('AbracadabrA',4,8) -> 'cada'
subseq('AbracadabrA',7)    -> 'abrA'
subseq(string,0,strchr(string,'/'))
subseq(vector (1, 2, 3), 2) -> vector (3)
subseq(vector (1, 2, 3), 0, 2) -> vector (1, 2)
```

The last one with string argument returns a copy of the string cut from the first slash, leaving it and everything following out, and in the case where there are no slashes present, returns a copy of the whole string.

substring

substring — returns a substring of a string

Synopsis

substring (*str* string ,
from integer ,
sublen integer) ;

Description

Returns a substring of string str. The start index is 1 based. The substring is sublen characters long.

This function follows SQL 92.

```
substring('AbracadabrA',1,4) -> 'Abra'
substring('AbracadabrA',5,4) -> 'cada'
substring('AbracadabrA',8,4) -> 'abrA'
```

sub_schedule

sub_schedule — add scheduled job for periodic synchronization of a subscription

Synopsis

sub_schedule (in *server_name* varchar ,
in *publication* varchar ,
in *interval* integer) ;

Description

Add scheduled job for periodically synchronizing a subscription.

Parameters

server_name
: target publisher server name.

publication
: publication name.

interval
: interval between synchronization attempts (in minutes).

See Also

`repl_disconnect()`

`repl_grant()`

`repl_init_copy()`

`repl_new_log()`

`repl_pub_add()`

`repl_pub_init_image()`

`repl_pub_remove()`

`repl_publish()`

`repl_revoke()`

`repl_sched_init()`

`repl_server()`

`repl_server_rename()`

`repl_stat()`

`repl_status()`

`repl_subscribe()`

`repl_sync()`

repl_sync_all()

repl_text()

repl_this_server()

repl_unpublish()

repl_unsubscribe()

system

system — runs a shell command from SQL

Synopsis

system (in *command* varchar) ;

Description

The system function will run a shell command from SQL. The shell command is executed in the server's current directory as the user that owns the database server process.

This function is available to dba users only. Since this is a security risk this feature is normally disabled. It can be enabled by setting the AllowOSCalls setting in virtuoso.ini to 1.

uptime

uptime — returns a string result set with how much time server have been up.

Synopsis

```
uptime (   );
```

Description

The uptime function will run a string result set with how much time server have been up.

Examples

Example 24.413. Simple Use

```
SQL> uptime();
VARCHAR
_______________________________________________________________________________

13 hour(s), 12 minute(s)

1 Rows. -- 59 msec.
```

See Also

`system()`

SYS_DB_STAT

SYS_DB_STAT — gathers common statistical information about the database.

Synopsis

```
SYS_DB_STAT ( in pcnt integer ,
              in ignore_vdb integer ) ;
```

Description

The SYS_DB_STAT procedure gathers common information about the whole Virtuoso database, including attached remote tables. It traverses each table and populates the SYS_COL_STAT table with data collected. SYS_DB_STAT can be used in random mode when not all rows of the tables are used to gather statistics of the database.

SYS_DB_STAT gathers the following information for each column, COL, traversed in the database:

SYS_COL_STAT - SQL command
CS_N_DISTINCT - count (distinct XXX)
CS_MIN - min (XXX)
CS_MAX - max (XXX)
CS_AVG_LEN - avg (raw_length (XXX))
CS_N_VALUES - count (XXX)
CS_N_ROWS - count (XXX)

Parameters

pcnt
: Rate that affects percentage of database coverage. Zero means that the whole database is read. The default is 5, for a random sample of approximately 5 percent of each table.

ignore_vdb
: By default SYS_DB_STAT does not analyze the remote tables. To allow statistics gathering over linked tables, set this parameter to zero. Remote tables can be very large or access to them can be very slow or both. If this parameter is set to zero and pcnt is zero, SYS_DB_STAT will read the remote tables from start to end which may take very long. If pcnt is non-zero, this function will use its knowledge of the remote DBMS to get statistics in a DBMS specific manner if the DBMS is supported. See sys_stat_vdb_mappers table.

Return Types

None.

See Also

sys_stat_histogram

sys_stat_analyze

sys_lockdown

sys_lockdown — Set virtuoso in lockdown state and back.

Synopsis

```
sys_lockdown ( in lock_action integer ,
               in listeners_mode integer ) ;
```

Description

You must have DBA privileges to run that function.

Calling this with lock_mode = 1 causes the server not to accept any new client connections, except ones coming from localhost (127.0.0.1). This will also shut down any other listeners and terminate any possibly pending processing, rolling back all open transactions and disconnecting all clients, except ones from localhost. Calling thi with lock_mode 0 reverses the effect.

Parameters

lock_action
: (0|1) - 0 = unlock the server; 1 = lock the server

listeners
: (0|1 default 0) - 0 = don't serve the listener sockets; 1 = close the listener sockets

Return Types

Lock mode (lock_action = 1) if already in that mode returns 0 otherwise returns 1

Unlock mode (lock_action = 0) if already in that mode returns 0 otherwise returns 2

sys_stat

sys_stat — Return statistical information about the Virtuoso server.

Synopsis

any **sys_stat** (in *stat_name* varchar);

Description

This function returns the statistic related to the requested statistic name give as a parameter to the function.

Parameters

stat_name
: The name of the statistic to look-up and return. This can be one of:

Return Types

Statistic to be returned. Can be a value such as the server version, or a string such as the database name.

Examples

Example 24.414. Returning simple statistics.

```
SQL> select concat(sys_stat('st_dbms_name'), ' - ', sys_stat('st_dbms_ver'));
callret
VARCHAR
_______________________________________________________________________________

OpenLink Virtuoso PHP4 Server - 03.00.2402
```

sys_stat_analyze

sys_stat_analyze — Collects statistics on a table and its columns for use in SQL optimization

Synopsis

```
sys_stat_analyze ( in table_name varchar ,
                   in pcnt integer ,
                   in ignore_vdb integer ) ;
```

Description

Collects (or updates) column statistics for the table columns. It collects minimum, maximum average and distinct values for a column and a row count for the table and inserts the data into the DB.DBA.SYS_COL_STAT table. It does not make histograms for the columns.

The statistics are then used by the Optimized SQL compiler. All the cached compilations are discarded, because some of them may compile differently in the light of the new data.

This function will normally consider the entire database with the exception of remote tables. Since you may be concerned about time or remote tables this operation is configurable using the optional parameters, `pcnt` and `ignore_vdb`.

Parameters

table_name
: The full name of the table exactly as in the KEY_TABLE column of SYS_KEYS.

pcnt
: Determines the sample size for statistics gathering. A value of 0 will read the whole table. The default is 5, for an approximately five percentsample of the table.

ignore_vdb
: Determines whether remote tables are considered. By default remote tables (ignore_vdb = 1) are ignored. Setting this value to 0 will cause this function to gather statistical information for remote tables also. A percentage of 0 will read the remote table from beginning to end. A non-zero percentage will access the statistics tables of the remote DBMS if it is of a supported type, see the sys_stat_vdb_mappers table.

See Also

sys_stat_histogram

SYS_DB_STAT()

sys_stat_histogram

sys_stat_histogram — Collects a column value histogram for use in SQL optimization

Synopsis

```
sys_stat_histogram (
    in table_name varchar ,
    in column_name varchar ,
    in n_buckets integer ,
    in prec integer ) ;
```

Description

This function collects (or updates) values distribution data for a given column.

It splits the sorted column values in n_buckets intervals and collects the last value of each interval. The values are then inserted into the SYS_COL_HIST table.

If the table in question has not been analyzed, then it calls SYS_STAT_ANALYZE for the table.

The histograms are then used by the Optimized SQL compiler. All the cached compilations are discarded, because some of them may compile differently in the light of the new data.

Parameters

table_name
: The full name of the table exactly as in the KEY_TABLE column of SYS_KEYS.

column_name
: The full name of the column exactly as in the COLUMN column in SYS_COLS.

n_buckets
: How much intervals to form. If more intervals are available, the estimation of column predicates costs is more precise.

prec
: The density of the rows examined. Defaults to 1 (all the rows)

See Also

`SYS_DB_STAT()`

sys_stat_analyze

table_set_policy

table_set_policy — Sets policy function to table.

Synopsis

table_set_policy (in *tablename* varchar ,
in *policypl* varchar ,
in *policy_scope* varchar) ;

Description

Sets policy function to table.

Parameters

tablename
: is the table to which the policy applies

policypl
: is the actual Virtuoso Stored Procedure based policy

policy_scope
: defines the scope of the policy in relation to SQL statement processing using one of more of the following values; I (for Inserts), D (for Deletions), U (for Updates), S (for Selects).

Return Types

varchar

Note:

Note that the policy_pl procedure should have the following signature (this is how it's called by the Virtuoso SQL compiler to get the policy) :

```
<procedure_name> (in table varchar, in op varchar) returns varchar
```

See Also

`table_drop_policy()`

table_drop_policy

table_drop_policy — Drops policy function from table.

Synopsis

```
table_drop_policy ( in _tb varchar ,
                    in _type varchar ) ;
```

Description

Drops policy function from table.

Parameters

_tb
: is the table to which the policy applies

_type
: defines the scope of the policy in relation to SQL statement processing using one of more of the following values; I (for Inserts), D (for Deletions), U (for Updates), S (for Selects).

Return Types

None.

Note:
Note that if either the table or the procedure gets dropped the policy is also autodropped.

See Also

`table_set_policy()`

strcontains

strcontains — Performs substring search

Synopsis

strcontains (in *str* string ,
in *sub* string) ;

Description

`strcontains` performs a substring search, returning 1 or 0.

Parameters

str
: String to search from.

sub
: Substring to search for.

Return Types

1 if found, 0 if not found.

Examples

Example 24.415. Simple example

```
SQL> select strcontains('AbracadabrA','abrA');
strcontains
INTEGER
_______________________________________________________________________________

1

1 Rows. -- 0 msec.

SQL> select strcontains('AbracadabrA','XZ');
strcontains
INTEGER
_______________________________________________________________________________

0

1 Rows. -- 0 msec.
```

See Also

`strcasestr`.

`starts_with`.

`end_with`.

starts_with

starts_with — Checks whether string X begins with Y

Synopsis

starts_with (in *str* string ,
in *sub* string) ;

Description

`starts_with` checks whether string X begins with Y, ignoring occurencies of Y in other places. Returns 1 or 0.

Parameters

str
: String to search from.

sub
: Substring to search for.

Return Types

1 if found, 0 if not found.

Examples

Example 24.416. Simple example

```
SQL> select starts_with('AbracadabrA','Abr');
starts_with
INTEGER
_______________________________________________________________________________

1

1 Rows. -- 0 msec.
```

See Also

`strcasestr`.

`strcontains`.

`end_with`.

ends_with

ends_with — Checks whether string X ends with Y

Synopsis

ends_with (in *str* string ,
in *sub* string) ;

Description

`ends_with` checks whether string X ends with Y, ignoring occurencies of Y in other places. Returns 1 or 0.

Parameters

str
: String to search from.

sub
: Substring to search for.

Return Types

1 if found, 0 if not found.

Examples

Example 24.417. Simple example

```
SQL> select ends_with('AbracadabrA','rA');
ends_with
INTEGER
_______________________________________________________________________________

1

1 Rows. -- 0 msec.
```

See Also

`strcasestr`.

`strcontains`.

`starts_with`.

tcpip_gethostbyname

tcpip_gethostbyname — Returns IP address by host domain name

Synopsis

```
varchar tcpip_gethostbyname ( in name varchar );
```

Description

This calls the system function gethostbyname() and returns h_addr from the hostent structure or empty string if no hostent structure returned (host not exists).

Parameters

name
: DNS name of the host

Return Types

A string containing IP address of the host or empty string if not exists

tcpip_gethostbyaddr

tcpip_gethostbyaddr — Returns fully qualified DNS name of the host associated with given IP address

Synopsis

```
varchar tcpip_gethostbyaddr ( in address varchar );
```

Description

This calls the system function gethostbyaddr() and returns h_name from the hostent structure returned by it. If no hostent structure returned, then it returns it's argument.

Parameters

address
: A string containing an IP address

Return Types

Returns fully qualified DNS name of the host associated with 'address'. If no such address the return will contain same value as 'address' parameter.

tmp_file_name

tmp_file_name — returns the unique file name within temporary directory of the operating system

Synopsis

```
varchar tmp_file_name ( in prefix varchar ,
                        in suffix varchar ) ;
```

Description

The following function is used to obtain unique name of a file, which is located in temporary directory on file system. The two optional parameters can be supplied: prefix of the file and extension for it. These will be prepended and appended (dot MUST be omitted) to the unique string. The directory where this file is located depends of $TMPDIR or %TMP% environment settings for UNIX's and Windows platforms respectively. If these environment settings are not available or empty, the defaults will be used for the operation system. (in practice for most UNIX's it's /tmp or /var/tmp directory). Note that this function do NOT open the file, it only give us a name.

Parameters

prefix
: prefix to the file name to be added. MUST be at most five characters.

suffix
: extension to the file to be added, the leading dot MUST be omitted.

Errors

Parameter data type checking errors

Examples

Example 24.418. Getting a unique file name

```
SQL> select tmp_file_name('FOO', 'EXT');
callret
VARCHAR
_______________________________________

/tmp/FOOFoCnq5.EXT

1 Rows. -- 7 msec.
```

tidy_html

tidy_html — Invoke built-in version of HTML Tidy utility to fix typical errors in HTML text

Synopsis

tidy_html (in *raw_html* varchar ,
in *tidy_config* varchar) ;

Description

This function improves the given source HTML text, by invoking a custom version of HTML Tidy utility. To learn more about Tidy see http://www.w3.org/People/Raggett/tidy/ . Some particular combinations of errors in source HTML may cause Tidy to misinterpret the source so the output may be incomplete or corrupted. This is an unavoidable problem, due to heuristic nature of the procedure. On the other hand, Tidy will process almost any HTML suitable for some "popular" browser, e.g. Internet Explorer or Netscape Navigator.

Parameters

raw_html
: Source HTML text to process. Note that the encoding of this text must be specified in tidy_config string, and default encoding of session will not be mentioned by Tidy.

tidy_config
: Configuration string is a list of options, delimited by newlines, with exactly the same syntax as original Tidy's configuration file. Not all options of original Tidy will work, due to obvious reasons, unsupported options will be silently ignored, so you may read your favorite Tidy's configuration file by file_to_string function and pass it to tidy_html.

Return Types

Errors

Table 24.82. Errors signalled by tidy_html

SQLState	Error Code	Error Text	Description
XTID2		HTML Tidy failed	The given HTML text contains serious errors. To get detailed list of them, call tidy_list_errors with same parameters as tidy_html.

Examples

Example 24.419.

```
fine_html := tidy_html (ugly_html, '
indent: auto
indent-spaces: 2
wrap: 72
markup: yes
output-xml: no
input-xml: no
show-warnings: yes
numeric-entities: yes
quote-marks: yes
quote-nbsp: yes
quote-ampersand: no
break-before-br: no
uppercase-tags: no
uppercase-attributes: no
char-encoding: latin1
new-inline-tags: cfif, cfelse, math, mroot,
```

```
    mrow, mi, mn, mo, msqrt, mfrac, msubsup, munderover,
    munder, mover, mmultiscripts, msup, msub, mtext,
    mprescripts, mtable, mtr, mtd, mth
  new-blocklevel-tags: cfoutput, cfquery
  new-empty-tags: cfelse');
```

See Also

tidy_list_errors

tidy_list_errors — Invoke built-in version of HTML Tidy utility to get list of errors in given input HTML text

Synopsis

tidy_list_errors (in *raw_html* varchar ,
in *tidy_config* varchar) ;

Description

This function lists errors in given source HTML text, by invoking some custom version of HTML Tidy utility. To learn more about Tidy see http://www.w3.org/People/Raggett/tidy/ .

Parameters

raw_html
: Source HTML text to validate. Note that the encoding of this text must be specified in tidy_config string, and default encoding of session will not be mentioned by Tidy.

tidy_config
: Configuration string, space-delimited list of options, exactly as in original Tidy's command-line or in Tidy's configuration file. Not all options of original Tidy will work, due to obvious reasons, unsupported options will be silently ignored, so you may read your favorite Tidy's configuration file by file_to_string function and pass it to tidy_list_errors. For more details, see tidy_html .

Return Types

Text with the list of found errors and warnings, as string of type varchar.

Errors

This function should not signal errors. Unlike `tidy_html` , it will log errors into the resulting string without signalling them.

See Also

timezone

timezone — get timezone difference from a datetime

Synopsis

timezone (in *dt* datetime ,
in *ignore_timezone* integer) ;

Description

The function returns timezone offset of its first argument, as an integer value in minutes. If the first argument is timezoneless and second argument is missing or zero then the returned value is NULL. If the first argument is timezoneless and second argument is nonzero then the returned value is 0.

Parameters

dt
: A datetime .

ignore_timezone
: Flag

Examples

Example 24.420. Simple example

Get current timezone.

```
SQL> select timezone (now ());
callret
INTEGER
_______________________________________________________________________________

120
```

See Also

now

forget_timezone

is_timezoneless

adjust_timezone

rdf_now_impl

current_timestamp

curdatetime

curdatetimeoffset

curutcdatetime

sysutcdatetime

dayname

dayofmonth

dayofweek

dayofyear

quarter

week

month

year

hour

minute

second

trace_off

trace_off — Disable extra logging for Virtuoso server

Synopsis

```
integer trace_off ( in parameter varchar );
```

Description

You must have DBA privileges to run that function.

This function is used to disable logging of various information enabled by default with the TraceOn ini file option or with the trace_on() function.

Return Types

Upon success zero will be returned, otherwise an error is.

Errors

Table 24.83. Errors signalled by `trace_off`

SQLState	Error Code	Error Text
22005	SR323	"option" is not valid trace_off option

Examples

Example 24.421. Simple example

To show users logs to the server and failed user logs

```
SQL> trace_off ('user_log', 'failed_log');

Done. -- 0 msec.
SQL>
```

See Also

`trace_on()`

`trace_status()`

trace_on

trace_on — Enable extra debug logging

Synopsis

```
integer trace_on ( in parameter varchar );
```

Description

This function requires dba privileges.

This function enables logging specified server operations for debugging purposes. The log entries will be shown at the server console (if started with foreground option) and will be written into the server message log file. The traceable events are divided into several groups: user activity, transactions, compilation of the SQL statements, DDL statements, statements execution and VDB actions.

Parameters

The following options are available for logging:

user_names - include the full user name, otherwise user ID will be logged.

user_log - log the connects/disconnects for users.

failed_log - log incorrect logins.

compile - log the names of procedures / triggers name compiled.

ddl_log - log the DDL statements execution.

client_sql - log the compilation of the client's SQL statements (first 500 chars).

errors - log all server errors.

dsn - log the connection/disconnection to DSNs, registration and removal of the DSNs, compilation of the SQL statements executed thru the VDB.

sql_send - log the compilation of SQL statements executed thru the VDB.

transact - log the transactions.

remote_transact - log the remote transactions.

exec - log SQL statement execution.

soap - log SOAP server requests and responses.

thread - log THRD_1 %ld OS threads freed. This is when OS threads are freed due to inactivity (being idle for more than ThreadCleanupInterval time).

cursor - log CURS_[0-9] - various VDB statements actions.

Return Types

Upon success zero will be returned, otherwise an error is signalled.

The message log file and/or server debug screen will list details for activated log options. The formats are as follows:

```
USER_0 (user) (IP) (peer) logout
USER_1 (user) (IP) (peer) login

FAIL_0 (user) (IP) (peer)

COMP_0 (user) (IP) (peer) trigger (name)
COMP_1 (user) (IP) (peer) procedure (name)

DDLC_0 (user) (IP) (peer) Create table (name)
DDLC_1 (user) (IP) (peer) Drop table (name)
DDLC_2 (user) (IP) (peer) Create procedure (name)
DDLC_3 (user) Drop procedure (name)
```

```
DDLC_4 (user) Create view (name)
DDLC_5 (user) Create index (name) or (table name)
DDLC_6 (user) Drop index (name) or (table name)
DDLC_7 (user) Rename table (new name) or (old name)
DDLC_8 (user) Create trigger (name) or (table name)
DDLC_9 (user) drop trigger (name) or (table name)

CSLQ_0 (user) (IP) (peer) (sql)

ERRS_0 (code) (server code) (error text)

DSNL_0 (dsn) (sql)
DSNL_1 Disconnecting DSN (name)
DSNL_2 (user) (IP) Registration remote data source (name)
DSNL_3 (user) Disconnect remote data source (name)

DSNS_0 (dsn) (sql) (prepare)
DSNS_1 (dsn) (sql) (execute)

LTRS_0 (user) (from) (peer) Begin transact tnx
LTRS_1 (user) (from) (peer) (Commit / Rollback) transact txn
LTRS_2 (user) (from) (peer) Restart transact txn

RTRS_0 (user) (from) (peer) (dsn) Begin transact tnx autocommit: (on/off)
RTRS_1 (user) (from) (peer) (dsn) (Commit / Rollback) transact tnx autocommit: (on/off)

EXEC_0 (user) (from) (peer) Exec cursor (stmt)
EXEC_1 (user) (from) (peer) (stmt) Exec (n) time(s)

SOAP_0 (request)
SOAP_1 (response)
```

Errors

Table 24.84. Errors signalled by `trace_on`

SQLState	Error Code	Error Text
22005	SR322	"option" is not valid trace_on option

Examples

Example 24.422. Simple example

To show users logging in to the server and failed user logins

```
SQL> trace_on ('user_log', 'failed_log');

Done. -- 0 msec.
SQL>

The server console  and log file may thus contain lines such as:
...
17:17:24 Server online at 1111 (pid 2173)
17:17:36 USER_0 0 xxx.xx.xx.xxx 1111:2 logout
17:17:40 USER_1 0 xxx.xx.xx.xxx 1111:3 login
17:17:45 USER_0 0 xxx.xx.xx.xxx 1111:3 logout
17:18:04 FAIL_0 dba xxx.xx.xx.xxx
...
```

See Also

`trace_off()`

`trace_status()`

trace_status

trace_status — show current trace settings

Synopsis

```
void trace_status ( );
```

Description

You must have DBA privileges to run that function.

This function returns an array of all available trace options and current status of the traces.

Examples

Example 24.423. Example

This is to show the current state of all trace log options.

```
SQl> create procedure trace_status_show ()
   {
     declare i, l integer;
     declare opt, status varchar;
     declare arr any;
     arr := trace_status();
     i := 0; l := length (arr);
     result_names (opt, status);
     while (i < l)
       {
         result (arr[i], arr[i+1]);
         i := i + 2;
       }
   };

SQL> trace_status_show();
opt                       status
VARCHAR                   VARCHAR
_______________________________________________________________________________

user_log                  off
failed_log                off
compile                   on
ddl_log                   off
client_sql                off
errors                    off
dsn                       off
sql_send                  off
transact                  on
remote_transact           off
exec                      off
soap                      off
```

See Also

`trace_on()`

`trace_off()`

tree_md5

tree_md5 — returns MD5 checksum of array argument

Synopsis

```
string tree_md5 ( in tree any );
```

Description

Returns a string of 16 characters representing the binary MD5 checksum of the argument. The argument can be any array or scalar.

Parameters

tree
: String or string_session to be processed with MD5 algorithm.

Return Values

A string of 32 lower case hex digits that are the result of the MD5 algorithm used on the data argument.

Errors

Parameter data type checking errors

HS_Resolve

HS_Resolve — Returns URL that represents the given DOI

Synopsis

```
string HS_Resolve (     in doi_val
                        string );
```

Description

Returns URL that represents the given DOI. The function is installed from the hslookup plugin which uses http://www.handle.net/.

Note that you need to have in your Virtuoso database ini file in section Plugins added the hslookup.dll file, which location should be in the plugins folder under your Virtuoso server installation. For ex:

```
[Plugins]
LoadPath = ./plugin
...
Load6    = plain,hslookup
```

Parameters

doi_val
: DOI value

Return Values

A string of URL that represents the given DOI.

Examples

Example 24.424. Simple example

```
SQL> select HS_Resolve('10.1038/35057062');
callret
VARCHAR
_______________________________________________________________________________

http://www.nature.com/doifinder/10.1038/35057062

1 Rows. -- 22 msec.
```

trigonometric

acos , asin , atan , cos , sin , tan , cot , degrees , radians , atan2 , pi — trigonometric functions

Synopsis

acos (in *x* double precision) ;

asin (in *x* double precision) ;

atan (in *x* double precision) ;

cos (in *x* double precision) ;

sin (in *x* double precision) ;

tan (in *x* double precision) ;

cot (in *x* double precision) ;

degrees (in *x* double precision) ;

radians (in *x* double precision) ;

atan2 (in *x* double precision ,
in *y* double precision) ;

pi ( ) ;

Description

All these functions work with double precision floating point numbers. They convert their argument to an IEEE 64-bit float and return a result of that type.

```
:  acos arc cosine
:  asin arc sine
:  atan arc tangent
:  cos  cosine
:  sin  sine
:  tan  tangent
:  cot  cotangent
:  degrees  convert an angle in radians to degrees.
:  radians  convert an angle in radians to degrees.
:  atan2  arc tangent with x and y coordinates, can return an angle in any quadrant.
:  pi   return pi.
```

trim

trim — removes characters from both ends of string argument

Synopsis

trim (*str* string ,
trimchars string) ;

Description

`trim` returns a copy of subsequence of string str with all the characters present in trimchars trimmed off from the beginning. If the second argument is omitted, it is a space ' ' by default.

`rtrim` is similar except that it trims from the right.

trim trims from both ends.

```
concat('*',trim('   SIMURG   '),'*') -> '*SIMURG*'
ltrim('AbracadabrA','bAr')  -> 'acadabrA'
rtrim('AbracadabrA','bAr')  -> 'Abracada'
trim('AbracadabrA','bAr')    -> 'acada'
```

txn_error

txn_error — poison current transaction forcing rollback

Synopsis

txn_error (in *code* integer) ;

Description

Calling this function will poison the current transaction. This means that it is forced to roll back at when committed. The code can be in integer that selects the error message generated when trying to commit. This is useful before signalling application errors from SQL code that runs in manual commit mode. This can ensure that even if the client attempts a commit after getting the error signalled by the application the transaction will not commit.

In most cases the code should be the constant 6, resulting the in the 'transaction rolled back due to previous SQL Error'.

Table 24.85. Transaction error codes

code	SQL state	Error ID	SQL message	Description
1	S1T00	SR171	Transaction timed out	
2	40001	SR172	Transaction deadlocked	
3	40003	SR173	Transaction out of disk	
4	40004	SR174	Log out of disk	
5	23000	SR175	Uniqueness violation. Transaction killed	
6	4000X	SR176	Transaction rolled back due to previous SQL error	The safest way of poisoning a transaction
7	4000X	SR177		Reserved for system use
8	08U01	SR324	Remote server has disconnected making the transaction uncommittable. Transaction has been rolled back	
9	40001	SR325	Transaction aborted due to a database checkpoint or database-wide atomic operation. Please retry transaction	
10	40005	SR325	Transaction aborted because its log after image size went above the limit	
11	40006	SR337	Transaction aborted because the server is out of memory	
12	4000X	SR177		Reserved for system use
13	08C02	SR337	Transaction aborted due to cluster connection failure	
14	40001	SR337	Transaction aborted due to async rollback in cluster	
15	40007	CLTSY	Transaction not committable because async update branch not synced before commit. Commit has overtaken the branch message or the branch message was lost by the network	
16	40007	CLPNC	Transaction prepared but not committed. Probably dropped commit message. The branch will automatically query coordinator for the final status. The situation will reset itself in a few seconds	
17	4000X	SR177		Reserved for system use
18	42000	RPERM	No permission to delete from given graph	
anything else	4000X	SR177	Misc Transaction Error	

txn_killall

txn_killall — kill all pending transactions

Synopsis

txn_killall (in *code* integer) ;

Description

This function will terminate all pending transactions. This can be used for resetting infinite loops in stored procedures etc.

The code determines the error reported to the client. Number 6 is preferable, corresponding to the 'transaction rolled back due to previous SQL error'. See `txn_error` for all supported codes.

Once any SQL statement or procedure notices that its transaction is dead, e.g. deadlocked, it signals the error and takes appropriate action, which is typically to signal the error to the caller and ultimately to the client.

Examples

```
txn_killall (1);
```

-- kills all transactions with the S1T00 'timed out' error.

ucase

ucase — returns upper case version of string argument

Synopsis

ucase (*str* string) ;

Description

`ucase` returns a copy of string str with all the lowercase alphabetical characters converted to corresponding uppercase letters. This includes also the diacritic letters present in the ISO 8859/1 standard in range 224 - 254 decimal, excluding the character 255, y diaeresis, which is not converted to a German double-s.

upper is just an alias for ucase.

```
ucase('AbracadabrA')  -> 'ABRACADABRA'
```

Parameters

str
: String to convert to upper case.

uddi_delete_binding

uddi_delete_binding — Causes one or more *bindingTemplate* structures to be deleted.

Syntax

```
<uddi_delete_binding
  generic="1.0"
  xmlns="urn:uddi-org:api" >
        <authInfo/>
        <bindingKey/>
        [ <bindingKey/> ...]
</uddi_delete_binding>
```

Attributes & Children

authInfo
: This required argument is an element that contains an authentication token obtained using the get_authToken call.

bindingKey
: One or more uuid_key values that represent specific instances of known bindingTemplate data.

Return Types

Upon successful completion, a *dispositionReport* is returned with a single success indicator.

Errors

If an error occurs in processing this message, a *dispositionReport* structure will be returned to the caller in a SOAP Fault. The following error information will be relevant:

Table 24.86. Errors signalled by uddi_delete_binding

Error Code	Description
E_invalidKeyPassed	signifies that one of the *uuid_key* values passed did not match with any known *bindingKey* values. No partial results will be returned; if any *bindingKey* values passed are not valid, this error will be returned.
E_authTokenExpired	signifies that the authentication token value passed in the authInfo argument is no longer valid because the token has expired.
E_authTokenRequired	signifies that the authentication token value passed in the authInfo argument is either missing or is not valid.
E_userMismatch	signifies that one or more of the *bindingKey* values passed refers to data not controlled by the entity the authentication token represents.
E_operatorMismatch	signifies that one or more of the *bindingKey* values passed refers to data not controlled by the server that received the request for processing.

uddi_delete_business

uddi_delete_business — Remove one or more *businessEntity* structures.

Syntax

```
<uddi_delete_business
  generic="1.0"
  xmlns="urn:uddi-org:api" >
        <authInfo/>
        <businessKey/>
        [ <businessKey/> ...]
</uddi_delete_business>
```

Attributes & Children

authInfo
: This required argument is an element that contains an authentication token obtained using the get_authToken call.

businessKey
: One or more uuid_key values that represent specific instances of known businessEntity data.

Return Types

Upon successful completion, a *dispositionReport* structure is returned with a single success indicator.

Errors

If an error occurs in processing this message, a *dispositionReport* structure will be returned to the caller in a SOAP Fault. The following error information will be relevant:

Table 24.87. Errors signalled by uddi_delete_business

Error Code	Description
E_invalidKeyPassed	signifies that one of the *uuid_key* values passed did not match with any known *businessKey* values. No partial results will be returned; if any *businessKey* values passed are not valid, this error will be returned.
E_authTokenExpired	signifies that the authentication token value passed in the authInfo argument is no longer valid because the token has expired.
E_authTokenRequired	signifies that the authentication token value passed in the authInfo argument is either missing or is not valid.
E_userMismatch	signifies that one or more of the *businessKey* values passed refers to data not controlled by the entity the authentication token represents.
E_operatorMismatch	signifies that one or more of the *businessKey* values passed refers to data that is not controlled by the server that received the request for processing.

uddi_delete_service

uddi_delete_service — Remove one or more *businessService* structures.

Syntax

```
<uddi_delete_service
  generic="1.0"
  xmlns="urn:uddi-org:api" >
        <authInfo/>
        <serviceKey/>
        [ <serviceKey/> ...]
</uddi_delete_service>
```

Attributes & Children

authInfo
: This required argument is an element that contains an authentication token obtained using the get_authToken call.

serviceKey
: One or more uuid_key values that represent specific instances of known businessService data.

Return Types

Upon successful completion, a *dispositionReport* message is returned with a single success indicator.

Errors

If an error occurs in processing this message, a *dispositionReport* structure will be returned to the caller in a SOAP Fault. The following error information will be relevant:

Table 24.88. Errors signalled by uddi_delete_service

Error Code	Description
E_invalidKeyPassed	signifies that one of the *uuid_key* values passed did not match with any known *serviceKey* values. No partial results will be returned; if any *serviceKey* values passed are not valid, this error will be returned.
E_authTokenExpired	signifies that the authentication token value passed in the *authInfo* argument is no longer valid because the token has expired.
E_authTokenRequired	signifies that the authentication token value passed in the *authInfo* argument is either missing or is not valid.
E_userMismatch	signifies that one or more of the *serviceKey* values passed refers to data not controlled by the entity the authentication token represents.
E_operatorMismatch	signifies that one or more of the *serviceKey* values passed refers to data not controlled by the server that received the request for processing.

uddi_delete_tModel

uddi_delete_tModel — Remove or retire one or more *tModel* structures.

Syntax

```
<uddi_delete_tModel
  generic="1.0"
  xmlns="urn:uddi-org:api" >
        <authInfo/>
        <tModelKey/>
        [ <tModelKey/> ...]
</uddi_delete_tModel>
```

Attributes & Children

authInfo
: This required argument is an element that contains an authentication token obtained using the get_authToken call.

tModelKey
: One or more URN-qualified uuid_key values that represent specific instances of known tModel data. All tModelKey values begin with a uuid URN qualifier (e.g. "uuid:" followed by a known tModel UUID value.)

Return Types

Upon successful completion, a *dispositionReport* structure is returned with a single success indicator.

Behavior
: If a tModel is deleted and any other managed data references that tModel by uuid_key (e.g. within a categoryBag , identifierBag or within a tModelInstanceInfo structure) then the tModel will not be physically deleted as a result of this call. Instead it will be marked as hidden. tModels hidden in this way are still accessible to their owner, via the get_registeredInfo call; however, they will be omitted from any results returned by calls to find_tModel .

Errors

If an error occurs in processing this message, a *dispositionReport* structure will be returned to the caller in a SOAP Fault. The following error information will be relevant:

Table 24.89. Errors signalled by uddi_delete_tModel

Error Code	Description
E_invalidKeyPassed	signifies that one of the URN-qualified *uuid_key* values passed did not match with any known *tModelKey* values. No partial results will be returned; if any *tModelKey* values passed are not valid, this error will be returned. Any *tModelKey* values passed without a uuid URN qualifier will be considered invalid.
E_authTokenExpired	signifies that the authentication token value passed in the *authInfo* argument is no longer valid because the token has expired.
E_authTokenRequired	signifies that the authentication token value passed in the *authInfo* argument is either missing or is not valid.
E_userMismatch	signifies that one or more of the *tModelKey* values passed refers to data not controlled by the entity the authentication token represents.
E_operatorMismatch	signifies that one or more of the *tModelKey* values passed refers to data that is not controlled by the server that received the request for processing.

uddi_discard_authToken

uddi_discard_authToken — Inform a UDDI server that the authentication token can be discarded.

Syntax

```
<uddi_discard_authToken
  generic="1.0"
  xmlns="urn:uddi-org:api" >
        <authInfo/>
</uddi_discard_authToken>
```

Description

The *uddi_discard_authToken* message is used to tell a UDDI-enabled server that the authentication token can be discarded. Subsequent calls that use the same *authToken* may be rejected. This message is optional for UDDI-enabled servers that do not manage session state or that do not support the *get_authToken* message.

Attributes & Children

authInfo
: This required argument is an element that contains an authentication token. Authentication tokens are obtained using the get_authToken message.

Return Types

Upon successful completion, the server returns a *dispositionReport* is returned with a single success indicator. Discarding an expired *authToken* will be reported as a success condition.

Errors

If an error occurs in processing this message, a *dispositionReport* structure will be returned to the caller in a SOAP Fault. The following error information will be relevant:

Table 24.90. Errors signalled by get_authToken

Error Code	Description
E_authTokenRequired	signifies that the authentication token value passed in the *authInfo* argument is either missing or is not valid.

uddi_find_binding

uddi_find_binding — Retrieves matching bindings

Syntax

```
<uddi_find_binding
  serviceKey="uuid_key"
  generic="1.0"
  [ maxRows="nn" ]
  xmlns="urn:uddi-org:api" >
        [<findQualifiers/>]
        <tModelBag/>
</uddi_find_binding>
```

Description

The *uddi_find_binding* message returns a *bindingDetail* message that contains a *bindingTemplates* structure with zero or more *bindingTemplate* structures matching the criteria specified in the argument list.

Attributes & Children

maxRows
: This optional integer value allows the requesting program to limit the number of results returned.

serviceKey
: This uuid_key is used to specify a particular instance of a businessService element in the registered data. Only bindings in the specific businessService data identified by the serviceKey passed will be searched.

findQualifiers
: This collection of findQualifier elements can be used to alter the default behavior of search functionality.

tModelBag
: This is a list of tModel uuid_key values that represent the technical fingerprint the server should locate in a bindingTemplate structure contained within the businessService instance specified by the serviceKey value. If more than one tModel key is specified in this structure, only bindingTemplate information that exactly matches all of the tModel keys specified will be returned (logical AND). The order of the keys in the tModel bag is not relevant. All tModelKey values begin with a uuid URN qualifier (e.g. "uuid:" followed by a known tModel uuid value.

Return Types

This function returns a *bindingDetail* message on success. In the event that no matches were located for the specified criteria, the *bindingDetail* structure returned in the response will be empty; that is, it will contain no *bindingTemplate* data.

In the event of a large number of matches, a UDDI-enabled server may truncate the result set. If this occurs, the response message will contain the *truncated* attribute with the value of this attribute set to true.

Searching using *tModelBag* will also return any *bindingTemplate* information that matches due to *hostingRedirector* references. The resolved *bindingTemplate* structure will be returned, even if that *bindingTemplate* is owned by a different *businessService* structure.

Errors

If an error occurs in processing this message, a *dispositionReport* structure will be returned to the caller in a SOAP Fault. The following error information will be relevant:

Table 24.91. Errors signalled by uddi_find_binding

Error Code	Description
E_invalidKeyPassed	signifies that the *uuid* _key value passed did not match with any known *serviceKey* or *tModel* key values. The error structure will signal which condition occurred first.
E_tooManyOptions	signifies that more than one mutually exclusive argument was passed.
E_unsupported	signifies that one of the *findQualifier* values passed was invalid.

uddi_find_business

uddi_find_business — Retrieves a *businessList* message matching supplied criteria.

Syntax

```
<uddi_find_business
  generic="1.0"
  [ maxRows ="nn" ]
  xmlns="urn:uddi-org:api" >
        [<findQualifiers/>]
        <name/> | <identifierBag/> | <categoryBag/> | <tModelBag/> | <discoveryURLs>
</uddi_find_business>
```

Attributes & Children

All arguments listed here are mutually exclusive except *findQualifiers* .

maxRows
: This optional integer value allows the requesting program to limit the number of results returned.

findQualifiers
: This collection of findQualifier elements can be used to alter the default behavior of search functionality.

name
: This string value is a partial name. The businessList return message contains businessInfo structures for businesses whose name matches the value passed (leftmost match).

identifierBag
: This is a list of business identifier references. The returned businessList contains businessInfo structures matching any of the identifiers passed (logical OR).

categoryBag
: This is a list of category references. The returned businessList contains businessInfo structures matching all of the categories passed (logical AND).

tModelBag
: The registered businessEntity data contains bindingTemplates that in turn contain specific tModel references. The tModelBag argument lets you search for businesses that have bindings that are compatible with a specific tModel pattern. The returned businessList contains businessInfo structures that match all of the tModel keys passed (logical AND). tModelKey values must be formatted as URN qualified uuid values (e.g. prefixed with "uuid:")

discoveryURLs
: This is a list of URLs to be matched against the data associated with the discoveryURLs contents of registered businessEntity information. To search for a URL without regard to useType attribute values, pass the useType component of the discoveryURL elements as empty attributes. If useType values are included, then the match will be made only on registered information that matches both the useType and URL value. The returned businessList contains businessInfo structures matching any of the URLs passed (logical OR).

Return Types

This function returns a *businessList* on success. In the event that no matches were located for the specified criteria, a *businessList* structure with zero *businessInfo* structures is returned.

In the event of a large number of matches, a UDDI-enabled server may truncate the result set. If this occurs, the *businessList* will contain the *truncated* attribute with the value set to true.

Searching using *tModelBag* will also return any *businessEntity* that contains *bindingTemplate* information that matches due to *hostingRedirector* references. In other words, the *businessEntity* that contains a

bindingTemplate with a *hostingRedirector* value referencing a *bindingTemplate* that matches the *tModel* search requirements will be returned.

Errors

If an error occurs in processing this message, a *dispositionReport* structure will be returned to the caller in a SOAP Fault. The following error information will be relevant:

Table 24.92. Errors signalled by get_authToken

Error Code	Description
E_nameTooLong	signifies that the partial name value passed exceeds the maximum name length designated by the UDDI-enabled server.
E_tooManyOptions	signifies that more than one search argument was passed.
E_unsupported	signifies that one of the *findQualifier* values passed was invalid.

uddi_find_service

uddi_find_service — Retrieves *serviceList* message matching search criteria

Syntax

```
<uddi_find_service
  businessKey="uuid_key"
  generic="1.0"
  [ maxRows ="nn" ]
  xmlns="urn:uddi-org:api" >
        [<findQualifiers/>]
        <name/> | <categoryBag/> | <tModelBag/>
</uddi_find_service>
```

Attributes & Children

The *name* , *categoryBag* , and *tModelBag* arguments are mutually exclusive.

maxRows
: This optional integer value allows the requesting program to limit the number of results returned.

businessKey
: This uuid_key is used to specify a particular BusinessEntity instance.

findQualifiers
: This collection of findQualifier elements can be used to alter the default behavior of search functionality.

name
: This string value represents a partial name. Any businessService data contained in the specified businessEntity with a matching partial name value gets returned.

categoryBag
: This is a list of category references. The returned serviceList contains businessInfo structures matching all of the categories passed (logical AND).

tModelBag
: This is a list of tModel uuid_key values that represent the technical fingerprint the server should locate within a bindingTemplate structure contained within any businessService contained by the businessEntity specified. If more than one tModel key is specified in this structure, only businessServices that contain bindingTemplate information that matches all of the tModel keys specified will be returned (logical AND).

Return Types

This function returns a *serviceList* on success. In the event that no matches were located for the specified criteria, the *serviceList* structure returned will contain an empty *businessServices* structure. This signifies zero matches.

In the event of a large number of matches, a UDDI-enabled server may truncate the result set. If this occurs, the *serviceList* will contain the *truncated* attribute with the value of this attribute set to true.

Searching using *tModelBag* will return serviceInfo structures for all qualifying *businessService* data, including matches due to *hostingRedirector* references. In other words, if the *businessEntity* whose *businessKey* is passed as an argument contains a *bindingTemplate* with a *hostingRedirector* value, and that value references a *bindingTemplate* that matches the *tModel* search requirements, then the *serviceInfo* for the *businessService* containing the *hostingRedirector* will be returned.

Errors

Table 24.93. Errors signalled by uddi_find_service

Error Code	Description
E_invalidKeyPassed	signifies that the *uuid_key* value passed did not match with any known *businessKey* or *tModel* key values. The error structure will signal which condition occurred first.
E_nameTooLong	signifies that the partial name value passed exceeds the maximum name length designated by the server.
E_tooManyOptions	signifies that more than one mutually exclusive argument was passed.
E_unsupported	signifies that one of the *findQualifier* values passed was invalid.

uddi_find_tModel

uddi_find_tModel — locate list of tModel entries matching supplied criteria

Syntax

```
<uddi_find_tModel
  generic="1.0"
  [ maxRows ="nn" ]
  xmlns="urn:uddi-org:api" >
        [<findQualifiers/>]
        <name/> | <identifierBag/> | <categoryBag/>
</uddi_find_tModel>
```

Description

This uddi_find_tModel message is for locating a list of tModel entries that match a set of specific criteria. The response will be a list of abbreviated information about tModels that match the criteria (tModelList).

Attributes & Children

The arguments to this call are mutually exclusive except findQualifiers

maxRows
: This optional integer value allows the requesting program to limit the number of results returned.

findQualifiers
: This collection of findQualifier elements can be used to alter the default behavior of search functionality.

name
: This string value represents a partial name. The returned tModelList contains tModelInfo structures for businesses whose name matches the value passed (leftmost match).

IdentifierBag
: This is a list of business identifier references. The returned tModelList contains tModelInfo structures matching any of the identifiers passed (logical OR).

categoryBag
: This is a list of category references. The returned tModelList contains tModelInfo structures matching all of the categories passed (logical AND).

Return Types

This function returns a tModelList on success. In the event that no matches were located for the specified criteria, an empty tModelList structure will be returned (e.g. will contain zero tModelInfo structures). This signifies zero matches.

In the event of a large number of matches, a UDDI-enabled server may truncate the result set. If this occurs, the tModelList will contain the *truncated* attribute with the value of this attribute set to true.

Errors

If an error occurs in processing this message, a *dispositionReport* structure will be returned to the caller in a SOAP Fault. The following error information will be relevant:

Table 24.94. Errors signalled by uddi_find_tModel

Error Code	Description
E_nameTooLong	

Error Code	Description
	signifies that the partial name value passed exceeds the maximum name length designated by the UDDI-enabled server.
E_tooManyOptions	signifies that more than one mutually exclusive argument was passed.
E_unsupported	signifies that one of the findQualifier values passed was invalid.

uddi_get_authToken

uddi_get_authToken — Obtain authentication token.

Syntax

```
<uddi_get_authToken
  generic="1.0"
  xmlns="urn:uddi-org:api"
  userID="someLoginName"
  cred="someCredential">
</uddi_get_authToken>
```

Description

The *uddi_get_authToken* message is used to obtain an authentication token. Authentication tokens are opaque values that are required for all other publisher API calls. This message is not required for UDDI-enabled servers that have an external mechanism defined for users to get an authentication token. This API is provided for implementations that do not have some other method of obtaining an authentication token or certificate, or that choose to use password-based authentication.

Attributes & Children

userID
: This required attribute argument is the identifier that an authorized entity was assigned by a UDDI-enabled server. Each UDDI-enabled server provides a way for entities to obtain userids and passwords that are valid only at that server.

cred
: This required attribute argument is the password or credential that is associated with the entity.

Return Types

uddi_get_authToken returns an *authToken* message containing a valid *authInfo* element usable in subsequent calls requiring an *authInfo* value.

Errors

If an error occurs in processing this message, the server returns a *dispositionReport* structure in a SOAP Fault. The following error information will be relevant:

Table 24.95. Errors signalled by uddi_get_authToken

Error Code	Description
E_unknownUser	signifies that the UDDI-enabled server that received the request does not recognize the userID and/or password argument values passed as credentials

uddi_get_bindingDetail

uddi_get_bindingDetail — Request run-time *bindingTemplate* location information.

Syntax

```
<uddi_get_bindingDetail
  generic="1.0"
  xmlns="urn:uddi-org:api" >
        <bindingKey/>
        [ <bindingKey/> ...]
</uddi_get_bindingDetail>
```

Description

The *uddi_get_bindingDetail* message requests the run-time *bindingTemplate* information for the purpose of invoking a registered business API.

Attributes & Children

bindingKey
: One or more uuid _key values that represent specific instances of known bindingTemplate data.

Behavior
: In general, it is recommended that bindingTemplate information be cached locally by applications so that repeated calls to a service described by a bindingTemplate can be made without having to make repeated round trips to a registry. In the event that a call made with cached data fails, the uddi _get_bindingDetail message can be used to get fresh bindingTemplate data. This is useful in cases such as when a service you are using relocates to another server or is restored at a disaster recovery site.

Return Types

This function returns a *bindingDetail* message on successful match of one or more *bindingKey* values. If multiple *bindingKey* values were passed, the results will be returned in the same order as the keys passed.

In the event of a large number of matches, a server may truncate the result set. If this occurs, the *bindingDetail* result will contain the *truncated* attribute with the value of this attribute set to true.

Errors

If an error occurs in processing this message, a *dispositionReport* structure will be returned to the caller in a SOAP Fault. The following error information will be relevant:

Table 24.96. Errors signalled by uddi_get_bindingDetail

Error Code	Description
E_invalidKeyPassed	signifies that one of the *uuid_key* values passed did not match with any known *bindingKey* key values. No partial results will be returned; if any *bindingKey* values passed are not valid *bindingKey* values, this error will be returned.

uddi_get_businessDetail

uddi_get_businessDetail — returns complete *businessEntity* information for one or more specified *businessEntities*

Syntax

```
<uddi_get_businessDetail
  generic="1.0"
  xmlns="urn:uddi-org:api" >
        <businessKey/>
        [ <businessKey/> ...]
</uddi_get_businessDetail>
```

Attributes & Children

businessKey
: one or more uuid_key values that represent specific instances of known businessEntity data.

Return Types

This function returns a *businessDetail* message on successful match of one or more *businessKey* values. If multiple *businessKey* values were passed, the results will be returned in the same order as the keys passed.

In the event of a large number of matches, a server may truncate the result set. If this occurs, the *businessDetail* response message will contain the *truncated* attribute with the value of this attribute set to true.

Errors

If an error occurs in processing this message, a *dispositionReport* structure will be returned to the caller in a SOAP Fault. The following error information will be relevant:

Table 24.97. Errors signalled by uddi_get_businessDetail

Error Code	Description
E_invalidKeyPassed	signifies that one of the *uuid_key* values passed did not match with any known *businessKey* values. No partial results will be returned; if any *businessKey* values passed are not valid, this error will be returned.

uddi_get_businessDetailExt

uddi_get_businessDetailExt — Returns extended *businessEntity* information for one or more specified *businessEntities*.

Syntax

```
<uddi_get_businessDetailExt
  generic="1.0"
  xmlns="urn:uddi-org:api" >
        <businessKey/>
        [ <businessKey/> ...]
</uddi_get_businessDetailExt>
```

Description

The *uddi_get_businessDetailExt* message returns extended *businessEntity* information for one or more specified *businessEntities*. This message returns exactly the same information as the *get_businessDetail* message, but may contain additional attributes if the source is an external registry that is compatible with this API specification, rather than a UDDI-enabled server.

Attributes & Children

businessKey
: One or more uuid_key values that represent specific instances of known businessEntity data.

Return Types

This message returns a *businessDetailExt* message on successful match of one or more *businessKey* values. If multiple *businessKey* values were passed, the results will be returned in the same order as the keys passed.

In the event of a large number of matches, a server may truncate the result set. If this occurs, the *businessDetailExt* response message will contain the *truncated* attribute with the value of this attribute set to true.

Errors

If an error occurs in processing this message, a *dispositionReport* structure will be returned to the caller in a SOAP Fault. The following error information will be relevant:

Table 24.98. Errors signalled by uddi_get_businessDetailExt

Error Code	Description
E_invalidKeyPassed	signifies that one of the *uuid_key* values passed did not match with any known *businessKey* values. No partial results will be returned; if any *businessKey* values passed are not valid, this error will be returned.
E_unsupported	signifies that the implementation queried does not support the extended detail function. If this occurs, *businessDetail* information should be queried via the *get_businessDetail* message. UDDI-enabled servers will not return this code, but will instead return a *businessDetailExt* result with full *businessDetail* information embedded.

uddi_get_registeredInfo

uddi_get_registeredInfo — Retrieve an abbreviated list of all *businessEntity* keys.

Syntax

```
<uddi_get_registeredInfo
  generic="1.0"
  xmlns="urn:uddi-org:api" >
        <authInfo/>
</uddi_get_registeredInfo>
```

Description

The *uddi_get_registeredInfo* message is used to get an abbreviated list of all *businessEntity* keys and *tModel* keys controlled by the entity associated with the credentials passed.

Attributes & Children

authInfo
: This required argument is an element that contains an authentication token. Authentication tokens are obtained using the get_authToken API call.

Return Types

Upon successful completion, the server returns a *registeredInfo* structure listing abbreviated business information in one or more *businessInfo* structures, and *tModel* information in one or more *tModelInfo* structures. This message is useful for determining the extent of registered information controlled by a single entity.

Errors

If an error occurs in processing this message, a *dispositionReport* structure will be returned to the caller in a SOAP Fault. The following error information will be relevant:

Table 24.99. Errors signalled by uddi_get_registeredInfo

Error Code	Description
E_authTokenExpired	signifies that the authentication token value passed in the *authInfo* argument is no longer valid because the token has expired.
E_authTokenRequired	signifies that the authentication token value passed in the *authInfo* argument is either missing or is not valid.

uddi_get_serviceDetail

uddi_get_serviceDetail — request full information about a known *businessService* structure

Syntax

```
<uddi_get_serviceDetail
  generic="1.0"
  xmlns="urn:uddi-org:api" >
        <serviceKey/>
        [ <serviceKey/> ...]
</uddi_get_serviceDetail>
```

Attributes & Children

serviceKey
: one or more uuid_key values that represent specific instances of known businessService data.

Return Types

This message returns a *serviceDetail* message on successful match of one or more *serviceKey* values. If multiple *serviceKey* values were passed, the results will be returned in the same order as the keys passed.

In the event of a large number of matches, a server may truncate the result set. If this occurs, the response will contain a *truncated* attribute with the value of this attribute set to true.

Errors

If an error occurs in processing this message, a *dispositionReport* structure will be returned to the caller in a SOAP Fault. The following error information will be relevant:

Table 24.100. Errors signalled by uddi_get_serviceDetail

Error Code	Description
E_invalidKeyPassed	signifies that one of the *uuid_key* values passed did not match with any known *serviceKey* values. No partial results will be returned; if any *serviceKey* values passed are not valid, this error will be returned.

uddi_get_tModelDetail

uddi_get_tModelDetail — Request full information about a known *tModel* structure.

Syntax

```
<uddi_get_tModelDetail
  generic="1.0"
  xmlns="urn:uddi-org:api" >
        <tModelKey/>
        [ <tModelKey/> ...]
</uddi_get_tModelDetail>
```

Attributes & Children

tModelKey
: one or more URN qualified uuid_key values that represent specific instances of known tModel data. All tModelKey values begin with a uuid URN qualifier such as "uuid:" followed by a known tModel uuid value.

Return Types

This message returns a *tModelDetail* message on successful match of one or more *tModelKey* values. If multiple *tModelKey* values were passed, the results will be returned in the same order as the keys passed.

In the event of a large number of matches, a server may truncate the result set. If this occurs, the response will contain a *truncated* attribute with the value of this attribute set to true.

Errors

If an error occurs in processing this message, a *dispositionReport* structure will be returned to the caller in a SOAP Fault. The following error information will be relevant:

Table 24.101. Errors signalled by uddi_get_tModelDetail

Error Code	Description
E_invalidKeyPassed	signifies that one of the URN-qualified *uuid_key* values passed did not match with any known *tModelKey* values. No partial results will be returned; if any *tModelKey* values passed are not valid, this error will be returned. Any *tModelKey* values passed without a uuid URN qualifier will be considered invalid.
E_keyRetired	signifies that the request cannot be satisfied because the owner has retired the *tModel* information. The *tModel* reference may still be valid and used as intended, but the information defining the *tModel* behind the key is unavailable.

uddi_save_binding

uddi_save_binding — save or update a complete *bindingTemplate* structure

Syntax

```
<uddi_save_binding
  generic="1.0"
  xmlns="urn:uddi-org:api" >
        <authInfo/>
        <bindingTemplate/>
        [<bindingTemplate/>...]
</uddi_save_binding>
```

Description

The *uddi_save_binding* message is used to save or update a complete *bindingTemplate* structure. This message can be used to add or update one or more *bindingTemplate* structures to one or more existing *businessService* structures.

Attributes & Children

authInfo
: This required argument is an element that contains an authentication token, obtained using the get_authToken call.

bindingTemplate
: One or more complete bindingTemplate structures. The order in which these are processed is not defined. To save a new bindingTemplate , pass a bindingTemplate structure with an empty bindingKey attribute value.

Behavior
: Each bindingTemplate structure passed must contain a serviceKey value that corresponds to a registered businessService controlled by the same identity saving the bindingTemplate data. The effect of this call is to establish the parent businessService relationship for each bindingTemplate affected by this call. If the same bindingTemplate (determined by matching bindingKey value) is listed more than once, any relationship to the containing businessService will be determined by processing order, which is determined by the position of the bindingTemplate data in first-to-last order.

Return Types

This message returns a *bindingDetail* message that reflects the newly registered information for the affected *bindingTemplate* structures.

Errors

If an error occurs in processing this message, a *dispositionReport* structure will be returned to the caller in a SOAP Fault. The following error information will be relevant:

Table 24.102. Errors signalled by uddi_save_binding

Error Code	Description
E_invalidKeyPassed	signifies that the request cannot be satisfied because one or more *uuid_key* values specified is not a valid key value, or that a *hostingRedirector* value references a *bindingTemplate* that itself contains a *hostingRedirector* value.
E_authTokenExpired	signifies that the authentication token value passed in the *authInfo* argument is no longer valid because the token has expired.
E_authTokenRequired	signifies that the authentication token value passed in the *authInfo* argument is either missing or is not valid.
E_userMismatch	signifies that one or more of the *uuid_key* values passed refers to data that is not controlled by the entity represented by the authentication token.

Error Code	Description
E_operatorMismatch	signifies that one or more of the *uuid_key* values passed refers to data that is not controlled by the server that received the request for processing.
E_keyRetired	signifies that the request cannot be satisfied because one or more *uuid_key* values specified has previously been hidden or removed by the requester. This specifically applies to the *tModelKey* values passed.
E_invalidURLPassed	an error occurred with one of the *uploadRegister* URL values.
E_accountLimitExceeded	signifies that user account limits have been exceeded.

uddi_save_business

uddi_save_business — Save or update information about a complete *businessEntity* structure.

Syntax

```
<uddi_save_business
  generic="1.0"
  xmlns="urn:uddi-org:api" >
        <authInfo/>
        <businessEntity/>
        [<businessEntity/>...] | <uploadRegister/> [<uploadRegister/>...]
</uddi_save_business>
```

Description

The *uddi_save_business* message is used to save or update information about a complete *businessEntity* structure. This message has the broadest scope of all of the save calls in the publisher's API, and can be used to make sweeping changes to the published information for one or more *businessEntity* structures controlled by an identity.

Attributes & Children

Either *businessEntity* arguments or *uploadRegister* arguments may be passed in a given *uddi_save_business* message, but not both. Any number of *businessEntity* or *uploadRegister* values can be passed in a single save (up to a server-imposed limit), but the two types of parameters may not be mixed.

authInfo
: This required argument is an element that contains an authentication token obtained using the get_authToken call.

businessEntity
: One or more complete businessEntity structures can be passed. These structures can be obtained in advance by using the get_businessDetail call or by any other means.

uploadRegister
: one or more resolvable HTTP URL addresses that each point to a single and valid businessEntity or businessEntityExt structure. This variant argument allows a registry to be updated to reflect the contents of an XML document that is URL-addressable. The URL must return a pure XML document that only contains a businessEntity structure as its top-level element, and be accessible using the standard HTTP-GET protocol.

Behavior
: If any of the uuid_key values within in a businessEntity structure (e.g. any data with a key value regulated by a businessKey , serviceKey , bindingKey , or tModelKey) is passed with a blank value, this is a signal that the data that is so keyed is being inserted. This does not apply to structures that reference other keyed data, such as tModelKey references within bindingTemplate or keyedReference structures, since these are references.

Return Types

This message returns a *businessDetail* message that reflects the new registered information for the *businessEntity* information provided.

Errors

If an error occurs in processing this message, a *dispositionReport* structure will be returned to the caller in a SOAP Fault. The following error information will be relevant:

Table 24.103. Errors signalled by uddi_save_business

Error Code	Description

Error Code	Description
E_invalidKeyPassed	signifies that one of the *uuid_key* values passed did not match with any known *serviceKey* values. No partial results will be returned; if any *serviceKey* values passed are not valid, this error will be returned.
E_authTokenExpired	signifies that the authentication token value passed in the authInfo argument is no longer valid because the token has expired.
E_authTokenRequired	signifies that the authentication token value passed in the authInfo argument is either missing or is not valid.
E_userMismatch	signifies that one or more of the *uuid_key* values passed refers to data that is not controlled by the individual who is represented by the authentication token.
E_operatorMismatch	signifies that one or more of the *businessKey* values passed refers to data that is not controlled by the UDDI-enabled server that received the request for processing.
E_keyRetired	signifies that the request cannot be satisfied because one or more *uuid_key* values specified has previously been hidden or removed by the requester. This specifically applies to the *tModelKey* values passed.
E_invalidURLPassed	an error occurred with one of the *uploadRegister* URL values.
E_accountLimitExceeded	signifies that user account limits have been exceeded.

uddi_save_service

uddi_save_service — Adds or updates one or more *businessService* structures.

Syntax

```
<uddi_save_service
  generic="1.0"
  xmlns="urn:uddi-org:api" >
        <authInfo/>
        <businessService/> [<businessService/>...]
</uddi_save_service>
```

Attributes & Children

authInfo
: This required argument is an element that contains an authentication token obtained using the get_authToken call.

businessService
: One or more complete businessService structures can be passed. These structures can be obtained in advance by using the get_serviceDetail call or by any other means.

Return Types

This message returns a *serviceDetail* message that reflects the newly registered information for the affected *businessService* structures.

Errors

If an error occurs in processing this message, a *dispositionReport* structure will be returned to the caller in a SOAP Fault. The following error information will be relevant:

Table 24.104. Errors signalled by uddi_save_service

Error Code	Description
E_invalidKeyPassed	signifies that the request cannot be satisfied because one or more *uuid_key* values specified is not a valid key value. This includes any *tModelKey* references that are unknown.
E_authTokenExpired	signifies that the authentication token value passed in the *authInfo* argument is no longer valid because the token has expired.
E_authTokenRequired	signifies that the authentication token value passed in the *authInfo* argument is either missing or is not valid.
E_userMismatch	signifies that one or more of the *uuid_key* values passed refers to data not controlled by the entity that the authentication token represents.
E_operatorMismatch	signifies that one or more of the *uuid_key* values passed refers to data not controlled by the server that received the request for processing.
E_keyRetired	signifies that the request cannot be satisfied because one or more *uuid_key* values specified has previously been hidden or removed by the requester. This specifically applies to the *tModelKey* values passed.

uddi_save_tModel

uddi_save_tModel — Adds or updates one or more *tModel* structures.

Syntax

```
<uddi_save_tModel
  generic="1.0"
  xmlns="urn:uddi-org:api" >
        <authInfo/>
        <tModel/> [<tModel/>...] | <uploadRegister/> [<uploadRegister/>...]
</uddi_save_tModel>
```

Attributes & Children

authInfo
: This required argument is an element that contains an authentication token obtained using the get_authToken call.

tModel
: One or more complete tModel structures can be passed. If adding a new tModel , the tModelKey value should be passed as an empty element.

uploadRegister
: One or more resolvable HTTP URL addresses that each point to a single and valid tModel structure. This variant argument allows a registry to be updated to reflect the contents of an XML document that is URL-addressable. The URL must return a pure XML document that only contains a tModel structure as its top-level element, and must be accessible using the standard HTTP-GET protocol.

Return Types

This message returns a *tModelDetail* message containing the new registered information for the effected *tModel* structures.

Errors

If an error occurs in processing this message, a *dispositionReport* structure will be returned to the caller in a SOAP Fault. The following error information will be relevant:

Table 24.105. Errors signalled by uddi_save_tModel

Error Code	Description
E_invalidKeyPassed	signifies that the request cannot be satisfied because one or more *uuid_key* values specified is not a valid key value. This will occur if a *uuid_key* value is passed in a *tModel* that does not match with any known *tModel* key.
E_authTokenExpired	signifies that the authentication token value passed in the *authInfo* argument is no longer valid because the token has expired.
E_authTokenRequired	signifies that the authentication token value passed in the *authInfo* argument is either missing or is not valid.
E_userMismatch	signifies that one or more of the *uuid_key* values passed refers to data not controlled by the entity represented by the authentication token.
E_operatorMismatch	signifies that one or more of the *uuid_key* values passed refers to data not controlled by the server that received the request for processing.
E_keyRetired	signifies that the request cannot be satisfied because one or more *uuid_key* values specified has previously been hidden or removed by the requester. This specifically applies to the *tModelKey* values passed.
E_invalidURLPassed	an error occurred with one of the *uploadRegister* URL values.
E_accountLimitExceeded	signifies that user account limits have been exceeded.

udt_defines_field

udt_defines_field — Determines whether a user defined type contains a specified member.

Synopsis

```
integer udt_defines_field ( in udt any ,
                            in member_name varchar ) ;
```

Description

This function is used to determine whether the supplied member_name is a member contained by the supplied udt.

Parameters

udt
: A user defined type name as varchar or type instance.

member_name
: The requested member name.

Return Types

This function returns either 1 (true) or 0 (false). 1 (true) is returned if the udt contains a member whose name is equal to the value of member_name, or 0 otherwise.

Examples

Example 24.425. Simple Use

```
select udt_defines_field (new SER_UDT(), 'A');
```

returns 1

```
select udt_defines_field (new SER_UDT_SUB(), 'A');
```

returns 1

```
select udt_defines_field (new SER_UDT(), 'B');
```

returns 0;

See Also

`udt_get()`

`udt_implements_method()`

`udt_instance_of()`

`udt_set()`

udt_get

udt_get — Retrieves a copy of the requested member from a user defined type instance

Synopsis

```
any udt_get ( in udt_inst any ,
              in member_name varchar ) ;
```

Description

This function returns a copy of the member named member_name, if any, for type instance udt_inst. It is the functional equivalent of member observer.

Parameters

udt_inst
: a user defined type instance.

member_name
: the name of the requested member.

Return Types

This function returns a copy of the requested member, if any.

Examples

Example 24.426. Observing members

This example simply fetches a member from a user defined type.

```
....
declare inst SER_UDT;

inst := new SER_UDT ();
return udt_get (inst, 'A');
....
```

See Also

`udt_defines_field()`

`udt_implements_method()`

`udt_instance_of()`

`udt_set()`

udt_implements_method

udt_implements_method — provides a handle to the first method with specific name in a user defined type.

Synopsis

```
any udt_implements_method ( in udt any ,
                            in method_name varchar ) ;
```

Description

This returns a non-zero value (true) if the udt has an instance method with a name equal to the value of method_name. For methods with the same name, but with different signatures this function will return the handle of the first method in order of definition. If a method is not found, 0 will be returned. The return value can be used to call the method using the indirect call statement. In which case an instance should be passed as a first argument to the indirect call statement.

Parameters

udt
: Type name as varchar or a type instance.

method_name
: The requested method name as a varchar.

Return Types

An integer will be returned. If the named method is not contained within the user defined type 0 (false) will be returned. Otherwise an integer representing the handle to first definition of a contained method with that name will be returned.

Examples

Example 24.427. Finding methods within a UDT

This example demonstrates how one might go about utilizing the handle of a method if found within a user defined type.

```
....
declare mtd any;
declare inst SER_UDT;

inst := new SER_UDT ();
mtd := udt_implements_method (inst, 'NEGATE');

if (mtd <> 0)
  return call (mtd) (inst);
else
  signal ('42000', 'No method NEGATE');
....
```

See Also

`udt_get()`

`udt_defines_field()`

`udt_instance_of()`

`udt_set()`

udt_instance_of

udt_instance_of — returns the type name of supplied type or compares two input types.

Synopsis

```
varchar udt_instance_of ( in udt any );

integer udt_instance_of ( in udt1 any ,
                          in udt2 any );
```

Description

This function returns information about the supplied type(s). There are two versions of this function, one returns the name of the type of the supplied argument, and the other than compares two supplied arguments for matching types. An error will be signalled if either of the types is not defined.

Parameters

udt, udt1, udt2
: Varchar or type instance.

Return Types

`udt_instance_of(udt)` returns the fully qualified name of the type represented by the udt argument.

`udt_instance_of(udt1, udt2)` returns 1 (true) if udt1 is of the same type or a subtype of udt2. Otherwise it returns o (false).

Examples

Example 24.428. Using udt_instance_of()

```
select udt_instance_of ('SER_UDT')
```

returns `DB.DBA.SER_UDT` (if the `SER_UDT` type was created by DBA in the DB database).

```
select udt_instance_of (new SER_UDT())
```

returns `DB.DBA.SER_UDT` (if the `SER_UDT` type was created by DBA in the DB database).

```
select udt_instance_of ('SER_UDT_SUB', 'SER_UDT');
```

Returns 1

```
select udt_instance_of (new SER_UDT(), 'SER_UDT_SUB');
```

Returns 0;

See Also

`udt_get()`

`udt_defines_field()`

`udt_implements_method()`

udt_set()

udt_set

udt_set

Synopsis

udt_set (in *udt_inst* any ,
in *member_name* varchar ,
in *new_value* any) ;

Description

This copies the instance udt_inst, sets new_value to the member with a name equal to the value of member_name (if any) and returns the modified instance copy. This is a functional equivalent of a member mutator.

Parameters

udt_inst
: A user defined type instance

member_name
: The name of the requested member

new_value
: The new value to be set

Return Types

A modified copy of udt_inst with the_value set for member_name.

Examples

Example 24.429. Setting user defined type member values

This sample code block shows a user defined type member's value being set.

```
....
declare inst SER_UDT;

inst := new SER_UDT ();
inst := udt_set (inst, 'A', 30);
return inst.A;
....
```

See Also

`udt_get()`

`udt_defines_field()`

`udt_instance_of()`

`udt_implements_method()`

unimport_clr

unimport_clr — This function automatically drops the SQL Type wrappers based on the CLR Reflection API.

Synopsis

```
any unimport_clr ( in assemblies_vector any ,
                   in classes_vector any ) ;
```

Description

This function automatically drops the SQL Type wrappers based on the CLR Reflection API. If there is compiled Virtuoso/PL code that references the type it will fail to execute (with a compilation error) when executed or compiled.

Parameters

assemblies_vector
: a vector of assembly names (as VARCHAR) to look into (or null).

classes_vector
: a vector of type names to create SQL type wrappers for (or null to mark all the types in the assemblies specified by assemblies_vector. In that case the assemblies_vector cannot be NULL).

Examples

Example 24.430. Dropping an imported Class

Now this library must be introduced to the Virtuoso Server. In ISQL use the following commands to test the CLR:

```
SQL> DB..unimport_clr (vector ('sanity'), vector ('sanity'));

Done. -- 300 msec.
SQL> select sanity::test('Rob');

*** Error 37000: [Virtuoso Driver][Virtuoso Server]UD041: No user defined type DB.DBA.sanity
at line 2 of Top-Level:
select sanity::test('Rob')
```

See Also

import_clr

The Runtime Hosting Chapter

unimport_jar

unimport_jar — Drops SQL wrapper types of selected Java classes

Synopsis

```
void unimport_jar (     in files any ,
                        in classes
                        any );
```

Description

This function calls the `jvm_ref_import()` function to produce the XML, then transforms it to a set of DROP TYPE statements and executes them.

Parameters

files
: Null, string or vector of strings. This can contain one or many Java .class files, .zip or .jar files to pick classes from. Null means only the classes specified in the classes argument regardless of their physical location.

classes
: Wildcard string or a vector of strings specifying which classes to describe in the resulting XML.

Examples

Example 24.431. Simple Use

```
SQL> import_jar (null, vector ('java.lang.System'));

Done. -- 126 msec.
SQL> select java_lang_System::getProperty('java.vm.name');
callret
NVARCHAR
_______________________________________________________________________________

Java HotSpot(TM) Client VM
SQL> unimport_jar (null, vector ('java.lang.System'));

Done. -- 54 msec.
SQL> select java_lang_System::getProperty('java.vm.name');

*** Error 37000: [Virtuoso Driver][Virtuoso Server]UD041: No user defined type DB.DBA.java_lang_System
at line 8 of Top-Level:
select java_lang_System::getProperty('java.vm.name')
```

See Also

`import_jar()`

`jvm_ref_import()`

updateXML

updateXML — Create a changed copy of given document by replacing some nodes.

Synopsis

```
any updateXML ( in source any ,
                in path1 varchar ,
                in replacement1 any ,
                in path2 varchar ,
                in replacement2 any ,
                ... ,
                in pathN varchar ,
                in replacementN any ) ;
```

Description

This is a synonym name for XMLUpdate() function. This name is added for compatibility with Oracle 9i.

upper

upper — returns upper case version of string argument

Synopsis

upper (*str* string);

Description

`ucase` returns a copy of string str with all the lowercase alphabetical characters converted to corresponding uppercase letters. This includes also the diacritic letters present in the ISO 8859/1 standard in range 224 - 254 decimal, excluding the character 255, y diaeresis, which is not converted to a German double-s.

upper is just an alias for ucase.

```
ucase('AbracadabrA')  -> 'ABRACADABRA'
```

USER_CHANGE_PASSWORD

USER_CHANGE_PASSWORD — Change the password of an existing user account.

Synopsis

```
USER_CHANGE_PASSWORD ( in name varchar ,
                       in old_pwd varchar ,
                       in new_pwd varchar ) ;
```

Description

This function is used to change the password of an existing user account.

Parameters

name
: A string containing the name of the user whose password is to be changed.

old_pwd
: A string containing the old password of the user.

new_pwd
: A string containing the new password to be used.

Return Types

None

See Also

User Options Section

`USER_CREATE()`

`USER_ROLE_CREATE()`

`USER_DROP()`

`USER_ROLE_DROP()`

`USER_SET_QUALIFIER()`

`USER_GRANT_ROLE()`

`USER_REVOKE_ROLE()`

`USER_SET_OPTIONS()`

`USER_GET_OPTIONS()`

USER_CREATE

USER_CREATE — create a new user account

Synopsis

```
USER_CREATE ( in name varchar ,
              in password varchar ,
              in options vector ) ;
```

Description

This function creates a new user account. The account is valid for SQL and/or DAV, depending n options.

Parameters

name
: The name of the user account as a string.

password
: The clear text password for the user account.

options
: A vector of name-value pairs for user options, even elements for option names, odd ones for values. Arbitrary options can be supplied. The Virtuoso reserved options can be found in the User Options sub-section of the User Administration Section.

Return Types

None

See Also

User Options Section

`USER_ROLE_CREATE()`

`USER_DROP()`

`USER_ROLE_DROP()`

`USER_CHANGE_PASSWORD()`

`USER_SET_QUALIFIER()`

`USER_GRANT_ROLE()`

`USER_REVOKE_ROLE()`

`USER_SET_OPTIONS()`

`USER_GET_OPTIONS()`

USER_DROP

USER_DROP — This deletes a user ccount, optionally including schema objects, DAV resources and other possible resources owned by the user.

Synopsis

USER_DROP (in *name* varchar ,
in *cascade* integer) ;

Description

This is used to remove an existing user account from local security schema.

Parameters

name
: A string containing the name of the user account to be deleted.

cascade
: This flag (0/1) specifies whether to delete the stored procedures/functions, PL modules, tables and user defined types owned by the user. It cannot drop assembly definitions (created with CREATE ASSEMBLY), because CREATE ASSEMBLY is a database wide operation (done by the DBA group) so the assembly has no owner (since you can not drop DBA). The default value is zero. Sometimes it is not possible to drop the resources, for example if a table owned by the user is referenced by foreign key in a table that would not be dropped. The drop user will detect that and will print the appropriate error message (giving up the operation).

Return Types

None

See Also

User Options Section

`USER_CREATE()`

`USER_ROLE_CREATE()`

`USER_ROLE_DROP()`

`USER_CHANGE_PASSWORD()`

`USER_SET_QUALIFIER()`

`USER_GRANT_ROLE()`

`USER_REVOKE_ROLE()`

`USER_SET_OPTIONS()`

`USER_GET_OPTIONS()`

`DROP USER [CASCADE]()` statement

USER_GET_OPTION

USER_GET_OPTION — Retrieve a user option for a given user account.

Synopsis

USER_GET_OPTION (in *name* varchar ,
in *opt* varchar) ;

Description

This function is used to retrieve an existing user option.

Parameters

name
: The name of the user account.

opt
: Option name.

Return Types

any, depending on option

See Also

User Options Section

`USER_CREATE()`

`USER_ROLE_CREATE()`

`USER_DROP()`

`USER_ROLE_DROP()`

`USER_CHANGE_PASSWORD()`

`USER_SET_QUALIFIER()`

`USER_GRANT_ROLE()`

`USER_REVOKE_ROLE()`

`USER_SET_OPTIONS()`

USER_GRANT_ROLE

USER_GRANT_ROLE — Assign an existing role to an existing security object.

Synopsis

```
USER_GRANT_ROLE ( in name varchar ,
                  in role varchar ,
                  in grant_opt int ) ;
```

Description

This function is used to assign an existing role to an existing security object. This is the same as the `GRANT <role>` statement.

All roles assigned to the role object will be inherited.

Parameters

name
: The name of the security object to affect.

role
: The name of the role (group) to be assigned to the security object.

grant_option
: A flag (0 or 1) indicating whether this object can grant this role to other security objects.

Return Types

None

See Also

User Options Section

`USER_CREATE()`

`USER_ROLE_CREATE()`

`USER_DROP()`

`USER_ROLE_DROP()`

`USER_CHANGE_PASSWORD()`

`USER_SET_QUALIFIER()`

`USER_REVOKE_ROLE()`

`USER_SET_OPTIONS()`

`USER_GET_OPTIONS()`

USER_REVOKE_ROLE

USER_REVOKE_ROLE — Removes a role from an existing security object.

Synopsis

```
USER_REVOKE_ROLE ( in name varchar ,
                   in role varchar ) ;
```

Description

This function is used to remove a role from the specified security object. This is equivalent to using the `REVOKE <role>` statement.

Parameters

name
: The name of the security object to affect.

role
: The name of the role (group) to be removed from the security object.

Return Types

None

See Also

User Options Section

`USER_CREATE()`

`USER_ROLE_CREATE()`

`USER_DROP()`

`USER_ROLE_DROP()`

`USER_CHANGE_PASSWORD()`

`USER_SET_QUALIFIER()`

`USER_GRANT_ROLE()`

`USER_SET_OPTIONS()`

`USER_GET_OPTIONS()`

USER_ROLE_CREATE

USER_ROLE_CREATE — Creates a new SQL role. Same as the create role statement.

Synopsis

USER_ROLE_CREATE (in *name* varchar) ;

Description

Creates a new SQL role.

Parameters

name
: A string containing the name of the new role.

Return Types

None

See Also

User Options Section

`USER_CREATE()`

`USER_DROP()`

`USER_ROLE_DROP()`

`USER_CHANGE_PASSWORD()`

`USER_SET_QUALIFIER()`

`USER_GRANT_ROLE()`

`USER_REVOKE_ROLE()`

`USER_SET_OPTIONS()`

`USER_GET_OPTIONS()`

USER_ROLE_DROP

USER_ROLE_DROP — Remove an existing role (group).

Synopsis

USER_ROLE_DROP (in *name* varchar) ;

Description

This is used to remove an existing role (group) from the local security schema.

Parameters

name
: A string containing the name of the group to be deleted.

Return Types

None

See Also

User Options Section

`USER_CREATE()`

`USER_ROLE_CREATE()`

`USER_ROLE_DROP()`

`USER_CHANGE_PASSWORD()`

`USER_SET_QUALIFIER()`

`USER_GRANT_ROLE()`

`USER_REVOKE_ROLE()`

`USER_SET_OPTIONS()`

`USER_GET_OPTIONS()`

USER_SET_QUALIFIER

USER_SET_QUALIFIER — Change the default catalog (qualifier/database) of a user account.

Synopsis

```
USER_SET_QUALIFIER ( in name varchar ,
                     in qual varchar ) ;
```

Description

This function is used to change the default catalog (qualifier/database) of a given user account.

Parameters

name
: A string containing the name of the user account.

qual
: A string containing the initial default qualifier of the session after a SQL client login.

Return Types

None

See Also

User Options Section

`USER_CREATE()`

`USER_ROLE_CREATE()`

`USER_DROP()`

`USER_ROLE_DROP()`

`USER_CHANGE_PASSWORD()`

`USER_GRANT_ROLE()`

`USER_REVOKE_ROLE()`

`USER_SET_OPTIONS()`

`USER_GET_OPTIONS()`

USER_SET_OPTION

USER_SET_OPTION — Set a user option for a given user account.

Synopsis

```
USER_SET_OPTION ( in name varchar ,
                  in opt varchar ,
                  in value any ) ;
```

Description

This function is used to set a User Option. The options can be any one of those described in User Options section or any other option. Note that existing option will be replaced.

Parameters

name
: The name of the user account to affect.

opt
: Option name.

value
: The new value of the named option.

Return Types

None

See Also

User Options Section

`USER_CREATE()`

`USER_ROLE_CREATE()`

`USER_DROP()`

`USER_ROLE_DROP()`

`USER_CHANGE_PASSWORD()`

`USER_SET_QUALIFIER()`

`USER_GRANT_ROLE()`

`USER_REVOKE_ROLE()`

`USER_GET_OPTIONS()`

user_set_password

user_set_password — Allows dba to change a user's password.

Synopsis

```
user_set_password ( in user_name string ,
                    in new_password string ) ;
```

Description

Explicitly sets a new password for the SQL account *user_name* to *new_password*. Only users in the dba group may execute this function. It allows the database administrator to reset lost passwords of SQL accounts. The new password will be set without further confirmation, so the DBA must be sure of the new password.

Parameters

user_name
: SQL user account name to change.

new_password
: New password for the user as plain text.

Errors

Parameter data type checking errors

Table 24.106. Errors signalled by `user_set_password()`

SQL Code	Error Message	Virtuoso Code
42000	Function user_set_password restricted to dba group	SR285
42000	The user <user_name> does not exist	SR286
42000	The new password for <user_name> cannot be empty	SR287

username

username — returns the login name of the current user

Synopsis

```
username (   );
```

Description

Returns the login name of the user of the connection. Selecting `user` is equivalent.

Parameters

None

Return Values

A string containing the login name of current user

Examples

Example 24.432. Get current DB user

```
SQL> select username();
callret
VARCHAR
_______________________________________________________________________________

dba

1 Rows. -- 5 msec.
SQL> select user;
callret
VARCHAR
_______________________________________________________________________________

dba

1 Rows. -- 4 msec.
```

uudecode

uudecode — Decodes a string previously encoded by uuencode

Synopsis

uudecode (in *input* string ,
in *mode* integer) ;

Description

Uudecode transforms uuencoded data into original form. Uuencode may return a number of sections as a vector of them, each of these sections should be decoded by separate call and results should be concatenated in order to compose original text. The mode of decoding should match to the mode used for encoding, of course.

RFC 2045, (N. Borenstein, N. Freed. MIME (Multipurpose Internet Mail Extensions) Part One: The Format of Internet Message Bodies), contains detailed description of most important encodings used by mail systems. Older RFC 1521 is now obsoleted.

Currently, eight conventions are used for mail attachments. In Virtuoso, they are enumerated by integer IDs.

Table 24.107. Unicode-like standards, supported by Virtuoso

ID	Standard	Description
1	"Native" UUencode	Optional header is one line started by "begin " or "<pre>begin " keyword, with some system-specific data after it. Optional footer is one line started by "end" keyword.
2	Base-64, UNIX version	There are no agreements about headers or footers.
3	Base-64, MIME version	This standard is very similar to "Base-64, UNIX version". Virtuoso, like all modern mail clients, feel no difference between them when decode sections, because the only difference is in the number of data bytes per line.
4	XXencode	This standard is an obsolete standard, similar to Unicode, but not compatible. There are no agreements about headers or footers.
5	BinHex	There are no header or footer lines, but first line of every section should be prefixed by colon (':') character.
6-9	reserved	More standards may be used in the future. It is unsafe to assume that all existing standards are listed here. Real application will often receive messages with unknown or misspelled encodings' names, syntax errors in data etc.
10	MIME Plain-Text	This "encoding" is suitable only for texts. They are stored "as is", but line ends will not be preserved.
11	MIME Quoted-Printable (for texts)	This encoding is suitable for any sort of textual data, because ASCII printable characters are stored "as is" and only nonprintable characters are encoded. UNIX linefeeds (LF chars) will be encoded as "hard breaks", so decoding side may convert them into its own system-specific "line end" chars, e.g. in CRLF sequence.
12	MIME Quoted-Printable (for binaries)	This encoding is suitable for any sort of data but it is especially useful for textual data, because ASCII printable characters are stored "as is" and only nonprintable characters are encoded. No "hard breaks" will be used for UNIX linefeeds (LF chars) of source file, so the file will be decoded exactly as it was encoded, no matter which character sequence is used for "line end" in the decoder's OS. When in trouble, whether the data encoded are text or binary, use this variant ("for binaries"): text may be easily recovered by recipient if linefeeds are wrong whereas binary data will become unusable if encoded as text.

If there's no information about the encoding used in the message, zero may be passed to the uudecode() function instead of proper ID. uudecode() will try to guess the proper algorithm. In any case, decoder feels no difference between modes 2 and 3 (two slightly different "Base64" encodings) and between modes 11 and 12 (two "Quoted-Printable" methods which are different only encoding side). Application may try all methods in turn if automatic guess will fail.

Parameters

input
: String or string-output session with data to be encoded.

mode
: Integer ID of encoding to be used.

Return Types

Vector of strings, where every string contains all data lines of a section.

Errors

Table 24.108. Errors signalled by

SQLState	Error Code	Error Text	Description
22003	UUD01	Unsupported type of UU-encoding (...)	
22003	UUD02	Data string contains errors [...]	
22003	UUD03	Encoded data ended prematurely	
22003	UUD04	No data found to be decoded	

Examples

Example 24.433. Encoding with auto-detection of the encoding type

The function gets a string 'Hello' in BinHex encoding and zero to let it guess that the encoding is BinHex. The call returns 'Hello'.

```
-- note doubled single quotes inside string literal
select uudecode (':5''9XE''m:', 0);
Hello
```

See Also

uuencode , uuvalidate

uuencode

uuencode — Encodes string or string session into sequence of printable characters, suitable for transfer via "ASCII-only" data channels

Synopsis

uuencode (in *input* string or string session ,
in *mode* integer ,
in *maxlines* integer) ;

Description

There are many protocols, like classic UNIX uuencode, which are used to transmit binary files over transmission mediums that do not support other than simple ASCII data. The epoch of physical lines of such sort is in past but file attachments in most popular mail systems still follow old regulations.

Encoded data are transmitted as a sequence of one or more "sections". They may be stored or sent as independent documents. Every section contains some range of original document's data. They may be decoded one after another, and original document may be composed by concatenation of decoded fragments. If the document is small (or if there's no limit on the size of message), it may be sent as single section.

Every section has some header and footer and a set of lines with data between them. Headers and especially footers are usually optional and may vary from system to system whereas data lines are described by standards. Data lines of any two consequent sections may be concatenated together, if needed, to create longer section.

`uuencode` creates a vector of strings, where every string contains some number of data lines, without headers or footers. A Virtuoso/PL stored procedure may be used to create some output stream(s) and put there sections of appropriate format with data lines from vector. Every item of the created vector will contain up to *`maxlines`* lines of data, usually 60 to 80 bytes per line; *`maxlines`* may vary from 10 to 120000 so section may be 0.8Kb to 10Mb long depending on your choice. Last section may be shorter than other, if only partially filled. 10Mb limit may be bypassed by sending of sections one after another without intermediate footers or headers, but please keep in mind that you cannot concatenate two strings in memory if the sum of their lengths exceeds system-wide 10Mb.

RFC 2045, (N. Borenstein, N. Freed. MIME (Multipurpose Internet Mail Extensions) Part One: The Format of Internet Message Bodies), contains detailed description of most important encodings used by mail systems. Older RFC 1521 is now obsoleted.

Currently, eight conventions are used for mail attachments. In Virtuoso, they are enumerated by integer IDs.

Table 24.109. Unicode-like standards, supported by Virtuoso

ID	Standard	Description
1	"Native" UUencode	Optional header is one line started by "begin " or "<pre>begin " keyword, with some system-specific data after it. Optional footer is one line started by "end" keyword.
2	Base-64, UNIX version	There are no agreements about headers or footers.
3	Base-64, MIME version	This standard is very similar to "Base-64, UNIX version". Virtuoso, like all modern mail clients, feel no difference between them when decode sections, because the only difference is in the number of data bytes per line.
4	XXencode	This standard is an obsolete standard, similar to Unicode, but not compatible. There are no agreements about headers or footers.
5	BinHex	There are no header or footer lines, but first line of every section should be prefixed by colon (':') character. Application should place colon before data lines of every section. (this function will prepare only data lines without this prefix).
6-9	reserved	More standards may be used in the future. It is unsafe to assume that all existing standards are listed here. Real application will often receive messages with unknown or misspelled encodings' names,

ID	Standard	Description
		syntax errors in data etc.
10	MIME Plain-Text	This "encoding" is suitable only for texts. They are stored "as is", but line ends will not be preserved.
11	MIME Quoted-Printable (for texts)	This encoding is suitable for any sort of textual data, because ASCII printable characters are stored "as is" and only nonprintable characters are encoded. UNIX linefeeds (LF chars) will be encoded as "hard breaks", so decoding side may convert them into its own system-specific "line end" chars, e.g. in CRLF sequence.
12	MIME Quoted-Printable (for binaries)	This encoding is suitable for any sort of data but it is especially useful for textual data, because ASCII printable characters are stored "as is" and only nonprintable characters are encoded. No "hard breaks" will be used for UNIX linefeeds (LF chars) of source file, so the file will be decoded exactly as it was encoded, no matter which character sequence is used for "line end" in the decoder's OS. When in trouble, whether the data encoded are text or binary, use this variant ("for binaries"): text may be easily recovered by recipient if linefeeds are wrong whereas binary data will become unusable if encoded as text.

Parameters

input
: String or string-output session with data to be encoded.

mode
: Integer ID of encoding to be used.

maxlines
: Number of data lines per section. Should be in range 10 to 120000, otherwise nearest suitable value will be used without reporting any error.

Return Types

Vector of strings, where every string contains all data lines of a section.

Errors

Table 24.110. Errors signalled by

SQLState	Error Code	Error Text	Description
22023	UUE01	Function uuencode needs a string output or a string as argument 1, not an arg of type ... (...)	
22023	UUE02	Unsupported type of UU-encoding (...)	

Examples

Example 24.434. BinHex

This function displays BinHex result of uuencode as a result-set of sections.

```
-- Line 4:
create procedure BINHEX_ENCODE (in txt varchar, in lines_per_section integer)
{
  declare SECTION varchar;
  declare sections any;
  result_names (SECTION);
  sections := uuencode (txt, 5, lines_per_section);
  foreach (varchar sect in sections) do
    result (':' || sect);
}

Done. -- 00000 msec.

BINHEX_ENCODE ('Hello', 10)
SECTION
VARCHAR
```

```
_______________________________________________________________________________

:5'9XE'm:

1 Rows. -- 00000 msec.

BINHEX_ENCODE (repeat ('Hello', 100), 10)
SECTION
VARCHAR
_______________________________________________________________________________

:5'9XE'p)C@aXEdKPE'a[5'9XE'p)C@aXEdKPE'a[5'9XE'p)C@aXEdKPE'a[
5'9XE'p)C@aXEdKPE'a[5'9XE'p)C@aXEdKPE'a[5'9XE'p)C@aXEdKPE'a[
5'9XE'p)C@aXEdKPE'a[5'9XE'p)C@aXEdKPE'a[5'9XE'p)C@aXEdKPE'a[
5'9XE'p)C@aXEdKPE'a[5'9XE'p)C@aXEdKPE'a[5'9XE'p)C@aXEdKPE'a[
5'9XE'p)C@aXEdKPE'a[5'9XE'p)C@aXEdKPE'a[5'9XE'p)C@aXEdKPE'a[
5'9XE'p)C@aXEdKPE'a[5'9XE'p)C@aXEdKPE'a[5'9XE'p)C@aXEdKPE'a[
5'9XE'p)C@aXEdKPE'a[5'9XE'p)C@aXEdKPE'a[5'9XE'p)C@aXEdKPE'a[
5'9XE'p)C@aXEdKPE'a[5'9XE'p)C@aXEdKPE'a[5'9XE'p)C@aXEdKPE'a[
5'9XE'p)C@aXEdKPE'a[5'9XE'p)C@aXEdKPE'a[5'9XE'p)C@aXEdKPE'a[
5'9XE'p)C@aXEdKPE'a[5'9XE'p)C@aXEdKPE'a[5'9XE'p)C@aXEdKPE'a[

:5'9XE'p)C@aXEdKPE'a[5'9XE'p)C@aXEdKPE'a[5'9XE'p)C@aXEdKPE'a[
5'9XE'm:

2 Rows. -- 00000 msec.
```

See Also

uudecode , uuvalidate

uuvalidate

uuvalidate — Encodes string or string session into sequence of printable characters, suitable for transfer via "ASCII-only" data channels

Synopsis

uuvalidate (in *input* string or string session ,
in *mode* integer) ;

Description

This function tries to ensure what applied data have a pointed encoding mode. If mode parameter is 0 (ie unknown) or if the validation fails, it will try to determine which mode was used in fact.

RFC 1521, (N. Borenstein, N. Freed. MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies), contains detailed description of most important encodings used by mail systems. RFC 2045, (N. Borenstein, N. Freed. MIME (Multipurpose Internet Mail Extensions) Part One: The Format of Internet Message Bodies).

Currently, eight conventions are used for mail attachments. In Virtuoso, they are enumerated by integer IDs.

Table 24.111. Unicode-like standards, supported by Virtuoso

ID	Standard	Description
1	"Native" UUencode	Optional header is one line started by "begin " or "<pre>begin " keyword, with some system-specific data after it. Optional footer is one line started by "end" keyword.
2	Base-64, UNIX version	There are no agreements about headers or footers.
3	Base-64, MIME version	This standard is very similar to "Base-64, UNIX version". Virtuoso, like all modern mail clients, feel no difference between them when decode sections, because the only difference is in the number of data bytes per line.
4	XXencode	This standard is an obsolete standard, similar to Unicode, but not compatible. There are no agreements about headers or footers.
5	BinHex	There are no header or footer lines, but first line of every section should be prefixed by colon (':') character. Application should place colon before data lines of every section. (this function will prepare only data lines without this prefix).
6-9	reserved	More standards may be used in the future. It is unsafe to assume that all existing standards are listed here. Real application will often receive messages with unknown or misspelled encodings' names, syntax errors in data etc.
10	MIME Plain-Text	This "encoding" is suitable only for texts. They are stored "as is", but line ends will not be preserved.
11	MIME Quoted-Printable (for texts)	This encoding is suitable for any sort of textual data, because ASCII printable characters are stored "as is" and only nonprintable characters are encoded. UNIX linefeeds (LF chars) will be encoded as "hard breaks", so decoding side may convert them into its own system-specific "line end" chars, e.g. in CRLF sequence.
12	MIME Quoted-Printable (for binaries)	This encoding is suitable for any sort of data but it is especially useful for textual data, because ASCII printable characters are stored "as is" and only nonprintable characters are encoded. No "hard breaks" will be used for UNIX linefeeds (LF chars) of source file, so the file will be decoded exactly as it was encoded, no matter which character sequence is used for "line end" in the decoder's OS. When in trouble, whether the data encoded are text or binary, use this variant ("for binaries"): text may be easily recovered by recipient if linefeeds are wrong whereas binary data will become unusable if encoded as text.

Parameters

input
: String or string-output session with data to be encoded.

mode
: Integer ID of encoding to be used.

Return Types

Vector of strings, where every string contains all data lines of a section.

Errors

Table 24.112. Errors signalled by

SQLState	Error Code	Error Text	Description
22003	UUV01	Unsupported type of UU-encoding (..)	

Examples

Example 24.435. Detection of encoding type

The function gets a string 'Hello' in BinHex encoding and returns 5 indicating that the string is probably in BinHex encoding.

```
-- note doubled single quotes inside string literal
select uuvalidate (':5''9XE''m:', 0);
5
```

See Also

uuencode , uudecode

USER_KEY_STORE

USER_KEY_STORE — Saves a key in the database and in case the database is restarted, the key will be loaded again.

Synopsis

```
USER_KEY_STORE (  in username varchar ,
                  in key_name varchar ,
                  in key_type varchar ,
                  in key_format varchar ,
                  in key_pwd varchar ,
                  in key_value varchar := NULL ) ;
```

Description

This function saves a key in the database and in case the database is restarted, the key will be loaded again.

Parameters

username
: User name

key_name
: Key name

key_type
: Key type

key_format
: Key format

key_pwd
: Key password

key_value
: Key value

Examples

Example 24.436. Persisting a key & certificate

The example below persists a key named 'id_rsa' and certificate created for it into user's key space. Thus after server restart the key will be available again.

```
SQL> USER_KEY_STORE (user, 'id_rsa', 'X.509', 2, '', xenc_pkcs12_export ('id_rsa', 'CA Certificate', ''))
```

See Also

`xenc_pkcs12_export()`

VAD_CHECK

VAD_CHECK — Checks the package has not been altered since installation

Synopsis

```
array DB.DBA. VAD_CHECK ( in package_name varchar );
```

Description

This checks to see if the elements of the package are as they are defined in the original distribution. A list of differing elements is returned. This does not always indicate a corruption since a later version or another package may add columns to tables, and some resources may be customized after installation.

Parameters

package_name
: name of package '/' version e.g: 'virtodp/1.0'

Return Types

A list of differing elements is returned as an array of 3-item structures: key, initial value, final value.

See Also

`vad_check_uninstallability()`

`vad_check_installability()`

`vad_install()`

`vad_pack()`

`vad_safe_exec()`

`vad_uninstall()`

`vad_fail_check()`

`vad_load_file()`

VAD_CHECK_INSTALLABILITY

VAD_CHECK_INSTALLABILITY — Checks the presence and correct versions of required packages and of the Virtuoso platform

Synopsis

```
varchar DB.DBA. VAD_CHECK_INSTALLABILITY ( in package_uri varchar );
```

Description

Checks the presence and correct versions of required packages and of the Virtuoso platform. It does not execute any pre-install Virtuoso/PL code from the package, so there is no guarantee that installation will be successful if the check found no error.

Parameters

package_uri
: URI of VAD file

See Also

`vad_check_uninstallability()`

`vad_fail_check()`

`vad_install()`

`vad_pack()`

`vad_safe_exec()`

`vad_uninstall()`

`vad_load_file()`

`vad_check()`

VAD_CHECK_UNINSTALLABILITY

VAD_CHECK_UNINSTALLABILITY — Checks if the package can be uninstalled.

Synopsis

```
varchar DB.DBA. VAD_CHECK_UNINSTALLABILITY ( in package_name varchar );
```

Description

Checks if the package can be uninstalled. It does not executes any pre-uninstall Virtuoso/PL code from the package, so there is no guarantee that uninstallation will be successful if the check found no error.

Parameters

package_name
: name of package '/' version e.g: 'virtodp/1.0'

See Also

`vad_fail_check()`

`vad_check_installability()`

`vad_install()`

`vad_pack()`

`vad_safe_exec()`

`vad_uninstall()`

`vad_load_file()`

`vad_check()`

VAD_FAIL_CHECK

VAD_FAIL_CHECK — Signals package check failure

Synopsis

```
DB.DBA. VAD_FAIL_CHECK ( in msg varchar );
```

Description

makes "rollback work", exits from atomic mode and fails server with raw_exit(-1)

Parameters

msg
: text of message

See Also

`vad_check_uninstallability()`

`vad_check_installability()`

`vad_install()`

`vad_pack()`

`vad_safe_exec()`

`vad_uninstall()`

`vad_load_file()`

`vad_check()`

VAD_INSTALL

VAD_INSTALL — Invoke VAD installation process

Synopsis

```
varchar DB.DBA. VAD_INSTALL ( in package_uri varchar ,
                              in source_type integer ) ;
```

Description

Invoke the install operation from interactive SQL or from the web user interface. This will:

- enter into atomic mode
- Check for version and prerequisite package compatibility
- Run the pre-install SQL script
- Load all SQL code and other resources in the order given by the developer
- Run any post install SQL code
- If DB.DBA.VAD_INSTALL() succeeded the server comes back on-line.

If there was a failure in mid-install, such as running out of disk or some other serious unrecoverable database error, the server exits. The installation can be undone manually by halting the server, deleting the transaction log file and restarting. The server will start from the checkpoint as if the installation was never attempted.

Parameters

package_uri
: URI of VAD file

source_type
: The second parameter to VAD_INSTALL controls whether the given path to the package is taken as a filesystem path or a DAV path. Value of 1 means the path is interpreted as a DAV path. The default value is 0 which means the VAD package is read from filesystem path.

See Also

`vad_check_uninstallability()`

`vad_check_installability()`

`vad_fail_check()`

`vad_pack()`

`vad_safe_exec()`

`vad_uninstall()`

`vad_load_file()`

`vad_check()`

VAD_LOAD_FILE

VAD_LOAD_FILE — executes statements of a SQL file

Synopsis

```
DB.DBA. VAD_LOAD_FILE (     in fname varchar
);
```

Description

This splits a plain sql file into single statements and executes them one by one. The root directory for this procedure is the 'code' root of VAD's repository.

Parameters

fname
: path to file to exec

See Also

`vad_load_sql_file()`

`vad_check_uninstallability()`

`vad_check_installability()`

`vad_install()`

`vad_pack()`

`vad_safe_exec()`

`vad_uninstall()`

`vad_fail_check()`

`vad_check()`

VAD_LOAD_SQL_FILE

VAD_LOAD_SQL_FILE — Loads SQL file and executes its content's statements.

Synopsis

```
DB.DBA. VAD_LOAD_SQL_FILE ( in sql_file_name varchar ,
                            in grouping integer ,
                            in report_errors varchar ,
                            in is_dav integer ) ;
```

Description

Loads SQL file (can be regular file or DAV resource) and then splits its content into single statements and executes them one by one. Along the process, the function makes needed reports and changes 'VAD_errcount' registry variable.

Parameters

sql_file_name
: For files, this parameter is either absolute or relative to server's working directory. The directory should be readable according to the virtuoso.ini settings. For DAV resources, the sql_file_name is always absolute: ('/DAV/<folder-name>...').

grouping
: Always set this parameter to 1.

report_errors
: The supported values for this parameter are:

is_dav
: This parameter is 0 for plain file and 1 for resource loaded in DAV.

Examples

Example 24.437. Simple Use

```
DB.DBA.VAD_LOAD_SQL_FILE ('./conductor/vdir_helper.sql', 1, 'report', 0);
```

See Also

vad_load_file()

vad_check_uninstallability()

vad_check_installability()

vad_install()

vad_pack()

vad_safe_exec()

vad_uninstall()

vad_fail_check()

vad_check()

VAD_PACK

VAD_PACK — get VAD resource

Synopsis

```
varchar DB.DBA. VAD_PACK ( in sticker_uri varchar ,
                           in base_uri_of_resources varchar ,
                           in package_uri varchar ) ;
```

Description

This function gets the resource identified by the sticker_uri, which contains the vad:package root element. The URIs present there are interpreted in the context of the base_uri_of_resources and the individual resources are fetched. These are parsed to make sure that they are syntactically correct and the resources are appended to the generated package resource, which is stored into the result_uri. vad_pack() returns a human-readable log of error and warning messages, vad_pack() will signal errors if some resources or database objects will be unavailable. By convention, VAD package files have the extension '.vad'.

Parameters

sticker_url
: stickers file URI

base_uri_of_resources
: inlined resources root

package_uri
: path of output VAD file

Return Types

vad_pack() returns a human-readable log of error and warning messages, vad_pack() will signal errors if some resources or database objects will be unavailable.

See Also

`vad_check_uninstallability()`

`vad_check_installability()`

`vad_fail_check()`

`vad_install()`

`vad_safe_exec()`

`vad_uninstall()`

`vad_load_file()`

`vad_check()`

VAD_SAFE_EXEC

VAD_SAFE_EXEC — execute without signalling errors

Synopsis

```
DB.DBA. VAD_SAFE_EXEC (
    in expr varchar
);
```

Description

Safe way to do something without generating an exception, e.g.: when it is necessary to drop a table without insurance of it's existence.

Parameters

expr
: text of expression

See Also

`vad_check_uninstallability()`

`vad_check_installability()`

`vad_fail_check()`

`vad_pack()`

`vad_install()`

`vad_uninstall()`

`vad_load_file()`

`vad_check()`

VAD_UNINSTALL

VAD_UNINSTALL — Vad package uninstallation

Synopsis

```
varchar DB.DBA. VAD_UNINSTALL ( in package_name varchar );
```

Description

Invokes the uninstall operation from interactive SQL or from the web user interface. This will:

- enter into atomic mode
- Check if no other package is using the subject of uninstall
- Run the pre-uninstall SQL script
- Run any post-uninstall SQL code
- Remove all VSP and XSLT and other web resources from their designated place in DAV or the Virtuoso Web root where permitted.
- If DB.DBA.VAD_UNINSTALL() succeeded the server comes back on-line. If there was a failure in mid-uninstall, such as running out of disk or some other serious unrecoverable database error, the server exits. The failed uninstallation can be undone manually. As usual, halting the server, deleting the transaction log file and restarting will force the server to start from the checkpoint as if the uninstallation was never attempted.

Parameters

package_name
: name of package '/' version e.g: 'virtodp/1.0'

See Also

`vad_check_uninstallability()`

`vad_check_installability()`

`vad_fail_check()`

`vad_install()`

`vad_safe_exec()`

`vad_pack()`

`vad_load_file()`

`vad_check()`

vd_remote_data_source

vd_remote_data_source — prepares a remote DSN for use

Synopsis

```
vd_remote_data_source ( in dsn varchar ,
                        in connstr varchar ,
                        in user varchar ,
                        in password varchar ) ;
```

Description

A remote data source is uniquely identified by its DSN, the dsn argument to this function. The connstr argument is presently ignored. The user and password are the login name and password to use when communicating with the remote data source. All Virtuoso users dealing with the remote data source will appear as this user to the remote data source. Virtuoso will make as many connections as there are concurrent users of the data source. Connections are cached by Virtuoso.

The default qualifier of the user of the remote data source is usually not relevant. This function connects to the DSN in order to retrieve various meta data, which it stores locally. The DSN should be defined in the server's environment and the DSN's database should be on line.

Parameters

dsn
: The name of the remote datasource to prepare.

connstr
: Currently ignored parameter.

user
: username for the connection.

password
: password for the user.

Return Types

Status of the connection attempt.

Errors

Table 24.113. Errors signalled by

SQLState	Error Code	Error Text	Description

See Also

vd_remote_table

vdd_disconnect_data_source

Manually Setting Up A Remote Data Source

vd_remote_proc_wrapper

vd_remote_proc_wrapper — Creating a PL wrapper for remote procedure execution

Synopsis

```
varchar vd_remote_proc_wrapper ( in dsn varchar ,
                                 in remote_name varchar ,
                                 in local_name varchar ,
                                 in metadata any ,
                                 out status_code varchar ,
                                 out error_message varchar ,
                                 in make_resultset integer ,
                                 in description varchar ) ;
```

Description

This is to create a PL stored procedure to execute a Remote Stored Procedures. It returns results as a SQL result set as well as an array(vector) depending of the 'make_resultset' flag.

Parameters

dsn
: the remote datasource name

remote_name
: name of the remote procedure

local_name
: name of the PL wrapper to be created locally

metadata
: A vector of vectors containing a parameters information each of vectors need to have following structure:

status_code
: returns SQL status code of the PL wrapper generation

error_message
: returns the error message if status code is different than 00000

make_resultset
: A flag 0/1 default 0. If is true (1) the wrapper will be generated to return result set. Otherwise will be generated to return an array.

description
: The text of a comment to added to the wrapper. It then will be shown as a method description if the PL wrapper is published as a SOAP method.

Examples

Example 24.438. Creating a PL wrapper which returns resultset

The following example will create a PL wrapper to invoke the MS SqlServer Stored Procedure 'Northwind.dbo.CustOrderHist'

```
-- the wrapper creation (fragment of Virtuoso/PL)
declare state, msg varchar;
vd_remote_proc_wrapper ('Northwind.dbo.CustOrderHist', 'MS.SQL.CustOrderHist',
```

```
'sql_lite', vector (vector ('IN', '_CustomerID', 'VARCHAR', '')), state, msg , 1, 'This is a test');
if (state <> '00000')
  signal (state, msg);

-- PL wrapper source that will be created
create procedure "MS"."SQL"."CustOrderHist" (IN "_CustomerID" VARCHAR) returns any array array
{
--PL Wrapper for remote procedure
--##This is a test
--"DSN:sql_lite PROCEDURE:Northwind.dbo.CustOrderHist"
declare dta, mdta any;
declare params any;
params := vector ("_CustomerID");
set_user_id ('dba');
rexecute ('sql_lite', '{call "Northwind"."dbo"."CustOrderHist" (?)}', NULL, NULL, params, 0, mdta, dta);
exec_result_names(mdta[0]);
declare i, l integer;
i := 0; l := length (dta);
 while(i<l) {
   exec_result(dta[i]);
   i:=i+1;
 }
}

-- result from execution
SQL> "MS"."SQL"."CustOrderHist" ('ALFKI');
ProductName                              Total
VARCHAR NOT NULL                         INTEGER
_____________________________________________________________________________

Aniseed Syrup                            6
Chartreuse verte                         21
Escargots de Bourgogne                   40
Flotemysost                              20
Grandma's Boysenberry Spread             16
Lakkalikri                               15
Original Frankfurter grne Soe            2
Raclette Courdavault                     15
Rssle Sauerkraut                         17
Spegesild                                2
Vegie-spread                             20

11 Rows. -- 18 msec.
```

See Also

rexecute vd_remote_data_source

vd_remote_table

vd_remote_table — declares an existing table as resident on a DSN previously declared with vd_remote_data_source

Synopsis

```
vd_remote_table ( in dsn varchar ,
                  in local_name varchar ,
                  in remote_name varchar ) ;
```

Description

Declares an existing table as resident on a data source previously declared with vd_remote_data_source().

This function declares the table local_name as table remote_name on the dsn. The tables names should be full, names with qualifier and owner. The names are case sensitive and must be in the exact case where they appear in the local and remote schemas.

If remote_name is NULL, the effect of a possible previous vd_remote_table is reversed. The table is thereafter treated as a local table, except in procedures and statements compiled when the remote declaration was in effect.

Parameters

dsn
: The name of the remote datasource previously connected using vd_remote_data_source()

local_name
: Fully qualified name of a local table.

remote_name
: Fully qualified name of the remote table.

See Also

vd_remote_data_source()

Manually Setting Up A Remote Data Source

vd_statistics

VD_STATISTICS — Update VDB RPC cost statistics for given DSN

Synopsis

```
DB.DBA. VD_STATISTICS ( in _dsn varchar ,
                        in vd_table_mask varchar ) ;
```

Description

This procedure will update the RPC round-trip statistics for selected data sources.

Parameters

_dsn
: This parameter is a LIKE mask for the name of the data source, as stored in DS_DSN column of SYS_DATA_SOURCE system table. Its default value of '%' means update the "round-trip" times for all remote data sources.

vd_table_mask
: This parameter is a LIKE mask for the name of the table , as stored in RT_NAME column of SYS_DATA_SOURCE system table. Its default value of '%' means update all tables.

See Also

`vdd_measure_rpc_time()`

vdd_disconnect_data_source

vdd_disconnect_data_source — Disconnects a data source if no active transactions are using resources from it.

Synopsis

vdd_disconnect_data_source (in *dsn* varchar) ;

Description

You must have DBA privileges to run that function.

This function disconnects all the idle opened connections to a VDB datasource. If there are active transactions server-side, using connections to that datasource, they are not closed. After they finish, this function can be called again to disconnect the new idle connections.

The datasource continues to be valid and any subsequent transactions using this datasource will open a new connection to it.

Parameters

dsn
: The name of the remote datasource to disconnect.

See Also

vd_remote_table

vd_remote_data_source

Manually Setting Up A Remote Data Source

vdd_measure_rpc_time

vdd_measure_rpc_time — Estimate VDB RPC round-trip for a given DSN in milliseconds

Synopsis

```
float vdd_measure_rpc_time ( in _dsn varchar );
```

Description

This function will return the estimated number of milliseconds to perform an RPC round trip on the DSN supplied.

Parameters

_dsn
: The name of the data source to estimate round-trip time for.

Return Types

The length of time taken in milliseconds as a float type.

See Also

`vd_statistics()`

vector

vector — make a vector

Synopsis

vector (*elem1* any ,
elem2 any ,
... ,
elem-n any) ;

Description

`vector` returns a new vector (one-dimensional array) constructed from the given arguments.

Parameters

elem1..n
: Values of any types (not necessarily of one and the same type).

Return Values

A vector (heterogeneous array) of as many elements as there were arguments containing copies of the arguments.

Examples

Example 24.439. Inspecting a vector with dbg_obj_print

SQL clients can not process vectors directly so the simplest way to look at the content of a vector is to print it no server's console.

```
dbg_obj_print (vector (1, 2.34, 'A string', atof('3.14')))
```

Example 24.440. Pretty-print function for vectors

The following function gets a heterogeneous vector of strings, nubers and other vectors and returns a string that is an SQL expression that will return a copy of a given vector.

```
create procedure DUMP_VEC_IMPL (inout _vec any, inout _ses any)
{
  declare _len, _ctr integer;
  if (193 <> __tag (_vec))
    {
      if (isstring (_vec))
        http (WS.WS.STR_SQL_APOS (_vec), _ses);
      else
        http (cast (_vec as varchar), _ses);
      return;
    }
  _len := length (_vec);
  _ctr := 0;
  http ('\nvector (', _ses);
  while (_ctr < _len)
    {
      if (_ctr > 0)
        http (', ', _ses);
      DUMP_VEC_IMPL (_vec[_ctr], _ses);
      _ctr := _ctr+1;
    }
  http (')', _ses);
```

```
}

create function DUMP_VEC (in _vec any)
{
  declare _ses any;
  _ses := string_output();
  DUMP_VEC_IMPL (_vec, _ses);
  return string_output_string (_ses);
}

select DUMP_VEC (vector ('abc', 1, vector (3.1415), vector ()));
callret
VARCHAR
_______________________________________________________________________________

vector ('abc', 1,
vector (3.1415),
vector ())

1 Rows.
```

See Also

aset aref vector_concat

vector_concat

vector_concat — concatenate vectors

Synopsis

```
vector vector_concat ( vec1 vector ,
                       vec2 vector ,
                       ... ,
                       vec-n vector ) ;
```

Description

`vector_concat` takes a variable number of vectors (heterogeneous arrays) and constructs a new vector containing copies of each (top level) element in the arguments.

Parameters

vec1..n
: vector .

Return Values

A vector made of copies of elements contained in vec1..n.

Examples

Example 24.441. Simple Use

```
dbg_obj_print (vector_concat (vector (now(), 'black', 'crow'),
                              vector ('said', 'nevermore')));

dbg_obj_print (vector_concat (vector ('a','b'),
                              vector (vector ('c', 'd'), 'e')));
```

Would output something like this on the Virtuoso console:

```
({ts 2001-08-07 16:57:12.000000}  "black" "crow" "said" "nevermore" )
("a" "b" ("c" "d" ) "e" )
```

See Also

`aset()` `aref()` `vector()`

VHOST_DEFINE

VHOST_DEFINE — define a virtual host or virtual directory

Synopsis

```
Void DB.DBA. VHOST_DEFINE ( in vhost varchar ,
                            in lhost varchar ,
                            in lpath varchar ,
                            in ppath varchar ,
                            in is_dav boolean ,
                            in is_brws boolean ,
                            in def_page varchar ,
                            in auth_fn varchar ,
                            in realm varchar ,
                            in ppr_fn varchar ,
                            in vsp_user varchar ,
                            in soap_user varchar ,
                            in sec varchar ,
                            in ses_vars boolean ,
                            in soap_opts any ,
                            in auth_opts any ,
                            in opts any ,
                            in is_default_host integer ) ;
```

Description

VHOST_DEFINE is used to define virtual hosts and virtual paths on the Virtuoso HTTP server. Effectively this procedure inserts a row in table DB.DBA.HTTP_PATH Virtuoso supports both flavours of virtual hosting: IP-based and name-based.

Parameters

vhost
: A string containing the virtual host name that the browser presents as Host: entry in the request headers. i.e. Name-based virtual hosting. The default value is taken from the Virtuoso INI file.

lhost
: A string containing the address of the network interface the Virtuoso HTTP server uses to listen and accept connections. The default value is taken from the Virtuoso INI file.

lpath
: A string containing the path component of the URI for the logical path.

ppath
: A string containing the physical path that the logical path points to. i.e. a directory or a path to dav collection on server.

is_dav
: An integer . If non-zero, it indicates that the physical_path points to a collection in DAV repository. Default value is 0.

is_brws
: An integer . If non-zero, it indicates that the server will generate a directory listing in case a default page is absent. Default value is 0.

def_page

A string containing the file name of the default page. Default value is NULL.

auth_fn
: A string that contains the fully qualified Virtuoso/PL procedure name of authentication hook function that will check and perform authentication for this virtual host or directory. If NULL, Virtuoso will not attempt authentication. The default value is NULL.

realm
: A string with the realm to be passed to the authentication function auth_func . The default value is NULL.

ppr_fn
: A string containing the fully qualified name of the Virtuoso/PL stored procedure used for post-processing of the page. The default values is NULL.

vsp_user
: A string containing a valid DB user name. The VSP pages contained in the virtual directory shall be run with the grants effective for this user at time of execution. The default values is NULL.

soap_user
: A string containing a valid SOAP user for SOAP calls. The default values is NULL.

sec
: Security restrictions (SSL, Digest). The default values is NULL.

ses_vars
: An integer . If non-zero, indicates that session variables are persistent. The default values is 0.

soap_opts
: Options for SOAP service. The default values is NULL. The SOAP options are name-value pairs contained in a vector: i.e. vector ('name1', 'value1', ...). The SOAP server accepts the optional parameters as described in Optional parameters to the SOAP endpoint .

auth_opts
: Options for the authentication hook and HTTPS listen hosts. The default values is NULL. If the sec_method (security method) defined as 'SSL' the following auth_options must be supplied: https_cert - HTTPS server certificate file path, https_key - HTTPS server private key file path. In addition to check X509 certificate of clients, the https_cv option with path to the file containing trusted certificate authorities must be supplied and https_cv_depth - integer to set depth of client certificate checking.

opts
: Extension options to the virtual directory itself. The default is NULL. When it's used the value MUST be a vector of name and value pairs. The supported extension options are: "noinherit" - denotes that all paths requested and matched this directory will be treated only as physical location (no expansion, no inheritance). It may used with a single page to the directory mapping. "xml_templates" - This denotes that .xml files under that directory and indirect children will be treated as a XML Template files (i.e. they will be executed). "executable" - this flag is used to override the execution permissions of the active pages stored within WebDAV repository (i.e. if it's set the active pages will be executed no matter what are their execution permission flag)

is_default_host

Return Types

The Return is VOID.

Examples

Example 24.442. Define Virtual Hosts

The following command will:

listen on port 8889 of the interface corresponding with host.foo.com
maps /appsimple -> /samples/appsimple/
can get a directory listing
SQL user which will perform VSP execution is WS
the persistent session variables flag is on, so can be used in VSPs

```
SQL> VHOST_DEFINE (vhost=>'www.foo.com:8889',
  lhost=>'host.foo.com:8889',
  lpath=>'/appsimple',
  ppath=>'/samples/appsimple/',
  is_brws=>1,
  vsp_user=>'WS',
  ses_vars=>1);
```

listen on standard HTTP port (80)
maps /appurl -> /samples/appurl/
can get a directory listing
SQL user which will perform VSP execution is WS
the persistent session variables flag is on , can be used in VSPs

```
SQL> VHOST_DEFINE (vhost=>'www.foo.com',
  lhost=>'host.foo.com',
  lpath=>'/appurl',
  ppath=>'/samples/appurl/',
  is_brws=>1,
  vsp_user=>'WS',
  ses_vars=>1);
```

HTTPS server listen on 443 port (will accept HTTP connections via SSL)
maps /secure -> /samples/appurl/
SQL user which will perform VSP execution is DBA
The server certificate is in the ./server.cert file
and server private key is in the ./server.key file

```
SQL> VHOST_DEFINE (vhost=>'www.foo.com:443',
  lhost=>'host.foo.com:443',
  lpath=>'/secure',
  ppath=>'/samples/appurl/',
  sec=>'SSL',
  vsp_user=>'dba',
  auth_opts=>vector ('https_key','./server.key','https_cert','./server.cert'));
```

If a browser points to the URL http://www.foo.com/ it will get a directory listing of /samples/appurl/, or if a browser points to http://www.foo.com:8889/ it will will get a directory listing of /samples/appsimple/.

The following example demonstrates the use of SOAP options used in Interop testing.

```
  VHOST_DEFINE (lpath=>'/Interop',
                ppath=>'/SOAP/',
                soap_user=>'INTEROP',
                soap_opts=> vector(
                              'SchemaNS','http://soapinterop.org/xsd',
                              'Namespace','http://soapinterop.org/',
                              'MethodInSoapAction','no',
                              'ServiceName', 'InteropTests',
                              'HeaderNS', 'http://soapinterop.org/echoheader/',
                              'CR-escape', 'yes'));
```

Example that covers TLS mutual authentication and custom re-write rules handlers for 403 and 401 HTTP Status responses:

```
DB.DBA.VHOST_DEFINE (
      lhost=>':443',
```

```
      vhost=>'example.com',
      lpath=>'/DAV',
      ppath=>'/DAV/',
      is_dav=>1,
      vsp_user=>'dba',
      ses_vars=>0,
      opts=>vector ('url_rewrite', 'rdf_sink_rule_list', '401_page',
'demo/authenticate.vsp', '403_page', 'demo/authenticate.vsp'),
      sec=>'SSL',
      auth_opts=>vector ('https_cert', 'db:john_smith_CA2', 'https_key',
'db:john_smith_CA2', 'https_verify', 3, 'https_cv_depth', 10),
      is_default_host=>0
```

See Also

Chapter Virtuoso Web and XML Support in the Virtuoso Manual for discussion on this topic. vhost_remove .

VHOST_REMOVE

VHOST_REMOVE — remove a virtual host or virtual directory

Synopsis

```
Void DB.DBA. VHOST_REMOVE ( in vhost varchar ,
                            in lhost varchar ,
                            in lpath varchar ,
                            in del_vsps integer ) ;
```

Description

`vhost_remove` is used to remove virtual hosts and virtual paths on the Virtuoso HTTP server. Effectively this procedure deletes a row in the table `DB.DBA.HTTP_PATH` . Virtuoso supports both flavours of virtual hosting: IP-based and name-based.

Parameters

vhost
: A string containing the virtual host name that the browser presents as Host: entry in the request headers. i.e. Name-based virtual hosting. Default value as defined in the Virtuoso INI file.

lhost
: A string containing the address of the network interface the Virtuoso HTTP server uses to listen and accept connections. Default value as defined in the Virtuoso INI file.

lpath
: A string containing the path component of the URI for the logical path.

del_vsps
: if a positive number will indicate to the server to drop all compilations of VSP files in this domain. Default value is 0.

Return Types

The return is VOID.

Examples

Example 24.443. Removing Virtual Host Definitions

```
SQL> VHOST_REMOVE (vhost=>'www.foo.com:8889',
  lhost=>'host.foo.com:8889',
  lpath=>'/appsimple');
```

Now, attempts to retrieve http://www.foo.com:8889/ will be rejected.

See Also

The Virtuoso Web Server Chapter in the Virtuoso Manual for discussion on this topic.

vhost_remove .

virtuoso_ini_path

virtuoso_ini_path — Return full name of configuration INI file

Synopsis

```
virtuoso_ini_path (   );
```

Description

This function returns the complete path to the configuration INI file. It is typically used by the cfg_ functions that modify or read the contents of the INI file.

Examples

Example 24.444. Simple examples

```
SQL> select virtuoso_ini_path();
callret
VARCHAR
_______________________________________________

D:\Program Files\OpenLink\Virtuoso 2.5\demo\demo.ini

1 Rows. -- 60 msec.
```

See Also

`cfg_section_count cfg_section_name cfg_item_count cfg_item_name cfg_item_value cfg_write`

server_root

server_root — Returns server working directory.

Synopsis

```
server_root (   );
```

Description

This function returns the complete physical path to server working directory.

Examples

Example 24.445. Simple examples

Show in the result the server working directory and INI file name:

```
SQL> select server_root(), virtuoso_ini_path ();
callret                                  callret
VARCHAR                                  VARCHAR
_________________________________________

C:\Program Files\OpenLink\Virtuoso 5.0\   C:\Program Files\OpenLink\Virtuoso 5.0\demo\demo.ini

1 Rows. -- 20 msec.
```

See Also

cfg_section_count cfg_section_name cfg_item_count cfg_item_name cfg_item_value cfg_write

vsp_calculate_digest

vsp_calculate_digest — calculate on server-side a digest to perform a HTTP digest authentication

Synopsis

vsp_calculate_digest (in *username* varchar ,
in *password* varchar ,
in *credentials* any) ;

Description

The vsp_calculate_digest() function is used to calculate on server-side a digest to perform a HTTP digest authentication. When the authentication type is 'digest' the function will return a md5 checksum based on credentials, user name and password. The checksum calculation will be made as required for HTTP Digest authentication to compare against 'response' element of credentials. If the authentication is basic a NULL will be returned.

Parameters

username
: A string with name of the user account name

password
: A string with a plain text password from the users table (can be from external LDAP server etc.)

credentials
: A vector of name/value pairs (the keyword and value are strings) of:

Return Types

A string containing md5 digest, if 'authtype' option is 'digest'. Otherwise returns null.

Examples

Example 24.446. A VSP page performing digest authentication

The following page check for digest authentication and returns OK if authentication succeeds, otherwise it asks for authentication again.

```
<?vsp
    declare auth any;
    declare cs varchar;
    auth := app_auth_vec (lines);
    if (not isarray (auth))
      app_get_auth ();
    cs := vsp_calculate_digest ('u1', 'secret', auth);
    if (cs is not null and cs = get_keyword ('response',auth))
       http ('OK');
    else
      app_get_auth ();
?>

-- making the HTTP Digest authentication header
create procedure app_get_auth ()
 {
   http_request_status ('HTTP/1.1 401 Unauthorized');
   http_header (sprintf ('WWW-Authenticate: Digest realm="%s", domain="%s", nonce="%s", opaque="%s"
 };

-- returns an array suitable for vsp_calculate_digest
create procedure app_auth_vec (in lines any)
 {
```

```
  declare ahdr, arr, authvec any;
  ahdr := http_request_header (lines, 'Authorization');
  if (isstring (ahdr) and ahdr like 'Digest%')
    {
      ahdr := subseq (ahdr, 6, length (ahdr));
      arr := split_and_decode (ahdr, 0, '\0\0,=');
      authvec := vector ('authtype', 'Digest', 'method', http_request_get ('REQUEST_METHOD'));
      foreach (varchar elm in arr) do
        {
          declare elm1 varchar;
          elm1 := trim (elm, '" ');
          authvec := vector_concat (authvec, vector (elm1));
        }
     }
   return authvec;
}
```

See Also

vhost_define

vt_batch

vt_batch — Returns a vt batch object.

Synopsis

```
batch object vt_batch ( );
```

Description

This object can be used to update a free text index by feeding document information into it using vt_batch_d_id to set the free text document ID and vt_batch_feed to feed actual words.

This object may not be assigned to other variables and may only be passed as an inout parameter.

The batch is applied to the index by calling the VT_BATCH_PROCESS_<table>_<column> function generated by CREATE TEXT INDEX .

See Also

`vt_drop_ftt()`

`vt_batch_d_id()`

`vt_batch_update()`

`vt_create_text_index()`

vt_batch_d_id

vt_batch_d_id — Specify a document to update in a vt batch.

Synopsis

vt_batch_d_id (in *d_id* any);

Description

Multiple documents may be indexed or unindexed with a single batch. In this case this function will be called for each document id, in ascending order of ID.

Parameters

d_id
: the free text document ID of the row whose index entry is to be updated.

Errors

Table 24.115. Errors signalled by

SQLState	Error Code	Error Text	Description

See Also

vt_drop_ftt()

vt_batch()

vt_batch_update()

vt_create_text_index()

vt_batch_feed

vt_batch_feed — Add words to a free text update batch.

Synopsis

vt_batch_feed (in *vt_batch* any ,
in *text_or_xml* any ,
in *is_del* integer ,
in *is_xml* integer) ;

Description

This function allows you to add words to a free text update batch. It can be called in sequence to feed group of documents that share a common document ID and the result is similar to the single call whose *text_or_xml* parameter is concatenation of documents of that group. It is even allowed to mix XML and non-XML documents by feeding an XML document and some text documents: the first document to feed may be an XML document, other documents should be only texts.

If one wishes to mix XML and non-XML documents, knowledge of details of the indexing method is needed. vt_batch_feed assigns a sequence number to every word of the provided document using an internal counter. The call of vt_batch_d_id not only sets document id but also resets this counter to 0. The first element of an XML document should have number 0 so it is impossible to feed an XML document if there were other calls of vt_batch_feed after the last vt_batch_d_id . Moreover, xcontains will ignore words from text documents that were fed after the first XML document, only contains will use all data.

Parameters

vt_batch
: must be an object returned by vt_batch on which vt_batch_d_id has been called; e.g. it can be called inside the index hook function

text_or_xml
: must be a blob, wide blob, varchar, nvarchar or XML entity object.

is_del
: if 0 means that the data is to be added, 1 means the data is to be deleted.

is_xml
: if 1, means that the text_or_xml must be a well formed XML fragment and that it will be indexed for use with xcontains . The default is 0 that means "plain text".

Return Types

None.

Examples

Please refer to examples for function vt_batch_feed_offband() .

See Also

vt_drop_ftt()

vt_batch()

vt_batch_feed_offband()

vt_batch_update()

vt_create_text_index

vt_batch_feed_offband

vt_batch_feed_offband — Add offband information to a free text update batch.

Synopsis

vt_batch_feed_offband (in *vt_batch* any ,
in *clustered_values* string ,
in *is_del* integer) ;

Description

This function allows you to add offband information to a free text update batch. This should be done by index and unindex hook functions of the free text index if and only if the index is created with both "CLUSTERED WITH (...)" and "USING FUNCTION" options and the hook function returns non-zero value, i.e. disables standard indexing of the document.

This function is needed only for very unusual free text indexes.

Parameters

vt_batch
: must be an object returned by vt_batch on which vt_batch_d_id has been called.

clustered_values
: must be a string that is made by serialization of a vector of offband values: the length of vector is equal to number of columns listed in "CLUSTERED WITH (...)" option of "CREATE TEXT INDEX" statement and elements of the vector should be values of the listed fields for the current document.

is_del
: if 0 means that the data is to be added, 1 means the data is to be deleted.

Return Types

The returned value may vary from version to version and should not be user by application code

Examples

Example 24.447. Free text indexing of composed text clustered with offband columns

Table LEADS contains descriptions of articles of some set of newspapers. Hook functions compose text that contains not only subject of article, but also title and first name of the author. The hook function should return non-zero value to disable default processing of text of SUBJECT field but it also disables the default processing of offband values. If the call of `vt_batch_feed_offband` is missing then free text search of documents is still OK but the call like SELECT ORG_ID FROM LEADS_TEST WHERE CONTAINS (SUBJECT, 'sample lead', OFFBAND, ORG_ID) will return NULLs instead of correct values from ORG_ID column because they will not be retrieved from free text index by "OFFBAND, ORG_ID".

```
create table LEADS (
  ORG_ID      integer not null,     -- ID of a newspaper.
  LEAD_ID     integer not null,     -- Internal ID of an article in a newspaper.
  FREETEXT_ID integer not null,     -- Global document ID of an article.
  SUBJECT     varchar (255),        -- Brief content of an article or a lead.
  NAME_FIRST  varchar (30),         -- First name of the author.
  NAME_LAST   varchar (30),         -- Last name of the author.
  TITLE       varchar (255),        -- Title of an article.
  primary key (ORG_ID, LEAD_ID));

create procedure LEADS_SUBJECT_INDEX_HOOK (inout vtb any, inout pkeyid integer)
{
  declare text_to_index varchar;
  declare oid any;
```

```
  text_to_index := coalesce (
    (select concat (
      coalesce (SUBJECT, ''), ' ',
      coalesce (TITLE, ''), ' ',
      coalesce (NAME_FIRST, '') ) from LEADS
    where FREETEXT_ID = pkeyid), null);
  if (text_to_index is null)
    return 0;
  oid := coalesce ((select ORG_ID from LEADS where FREETEXT_ID = pkeyid), null);
  vt_batch_feed (vtb, text_to_index, 0);
  vt_batch_feed_offband ( vtb, serialize (vector (oid)), 0);
  return 1;
}

create procedure
LEADS_SUBJECT_UNINDEX_HOOK (inout vtb any, inout pkeyid integer)
{
  declare text_to_index varchar;
  declare oid any;
  text_to_index := coalesce (
    (select concat (
      coalesce (SUBJECT, ''), ' ',
      coalesce (TITLE, ''),' ',
      coalesce (NAME_FIRST, '') ) from LEADS
  where FREETEXT_ID = pkeyid), null);
  if (text_to_index is null)
    return 0;
  oid := coalesce ((select ORG_ID from LEADS where FREETEXT_ID = pkeyid), null);
  vt_batch_feed (vtb, text_to_index, 1);
  vt_batch_feed_offband ( vtb, serialize (vector (oid)), 1);
  return 1;
}

create text index on LEADS (SUBJECT) with key FREETEXT_ID
clustered with (ORG_ID) using function;
```

See Also

vt_drop_ftt

vt_batch

vt_batch_feed

vt_batch_update

vt_create_text_index

VT_BATCH_UPDATE

VT_BATCH_UPDATE — Set batch mode update of free text indexing.

Synopsis

```
integer DB.DBA. VT_BATCH_UPDATE ( in table varchar ,
                                  in flag varchar ,
                                  in interval_minutes integer ) ;
```

Description

This function controls the time of update of a text index. If flag is ON, changes are accumulated into a change tracking table and applied as a batch. If flag is OFF, the text index is updated in the same transaction as the indexed data itself. The change tracking table is automatically created and is named VTLOG_<q>_<o>_<table>, in the qualifier and owner of the indexed table, where q, o and table are the qualifier, owner and name of the table. The changes accumulated into that table can be explicitly applied to the index using the VT_INC_INDEX_<q>_<o>_<table> function.

Parameters

table
: the name of the table to perform batch updating of.

flag
: If ON, updates are in batch mode, if OFF, they are synchronous.

interval_minutes
: the update interval. NULL means that updates are not scheduled.

Errors

Table 24.116. Errors signalled by

SQLState	Error Code	Error Text	Description

See Also

vt_drop_ftt()

vt_batch()

vt_create_text_index()

vt_batch_feed()

vt_create_text_index

vt_create_text_index — Add text index to an existing table

Synopsis

```
vt_create_text_index ( in table varchar ,
                       in col varchar ,
                       in id_col varchar ,
                       in is_xml integer ,
                       in defer_generation integer ,
                       in clustered_columns any ,
                       in use_hook_function integer ,
                       in language_name varchar ,
                       in encoding_name integer ,
                       in silent integer ) ;
```

Description

The vt_create_text_index procedure adds a text index to an existing table. There can at most be one text index per table, including super tables and subtables.

The table argument is a string naming the table. The column is the name of the column to index. The id_col should be the name of a unique integer row identifier column. If null, the system will either add such a column or use an existing integer primary key if one is available. The is_xml argument, if non-0, specifies that the values of the indexed column should be checked for XML well formedness and that the XML structure should be taken into account in indexing the values.

Use the CREATE TEXT INDEX statement as an alternative to the vt_create_text_index function.

In order for a table to be referenced in a text index it must have a uniquely identifying integer key. If the table in question has such a key this can be used as the id column. If there is no such column this procedure makes one. Using a previously existing identifier column saves space and if that is the primary key of the table this also saves in retrieval time.

If the table being indexed has a single part integer primary key vt_create_text_index will automatically use this as the identifier. Note that the zero and negative numbers may not be used as identifier values.

Creating the index will read through the table's contents and generate the index. When the table is changed the index can either be updated after each change or periodically, depending on the application needs. The rationale for background maintenance of the text index is that it is up to several times more efficient to maintain a text index in batches of several changed documents than after each single document change. The default maintenance mode is synchronous, meaning that each insert, delete or update of the indexed column will be immediately reflected in the index. This mode can be set using the `vt_batch_update()` procedure. The mode should be set to batch if there are any massive operations on the table.

It will create two additional tables:

```
<datatable>_<datacolumn>_WORDS
```

and

```
VTLOG_<datatable_qualifieri>_<data_table_owner>_<data_table_name>
```

and two procedures:

```
vt_index_<datatable_qualifier>_<data_table_owner>_<data_table_name>
                (in to_delete integer)
```

and

```
vt_inc_index<datatable_qualifieri>_<data_table_owner>_<data_table_name> ().
```

The table <datatable>_<datafield>_WORDS contains the full text index data.

The table VTLOG_<datatable>_<datafield> is an update tracking table, similar to the snapshot log table but using the key column instead of the primary key.

Note:
in the transaction semantics section that there is the sync mode for the purpose of creating a text index.

Parameters

table
: the table containing the data to index.

col
: the column in the data table containing the data to index (a long varchar column).

id_col
: an integer unique indexed column used by the free text index as a key. If Virtuoso tries to choose such a column among the existing in the table and if it doesn't find a suitable column it adds such a column with the name <datacolumn>_ID, fills that up and makes an unique index on it.

is_xml
: if greater than 0 installs two additional triggers before insert and before update on the data table to ensure the data being inserted into it are valid XML documents.

defer_generation
: if nonzero then the free-text index will not be filled by actual data immediately after the creation. It will remain empty until explicit request for "incremental indexing".

clustered_columns
: a vector of names of "clustered columns" or NULL to not store such data in the index.

use_hook_function
: if nonzero, two user-defined Virtuoso/PL functions will be called when free-text data are updated, not the default system routine. These functions are recognized by their special names: <datatable>_<datafield>_INDEX_HOOK will be called to index new documents and <datatable>_<datafield>_UNINDEX_HOOK will be called to remove obsolete index information about deleted documents.

language_name
: the name of the language that is used for building the index. If the parameter is omitted or is equal to '*ini*' string, indexing routines will use the language specified in server's configuration.

encoding_name
: the name of the encoding that is used by default to index source texts. If the parameter is omitted or is equal to '*ini*' string, indexing routines will use the encoding specified by charset of the RDBMS connection that is in use when the index is created.

silent
: Flag with values 1/0. Default is 0. When set to 0, the function signals error if the text index exists. If set to 1, the function signals no error if the text index exists.

Examples

Example 24.448. Creating A Text Index Using FT functions

```
create table xml_text (xt_id integer, xt_file varchar,
                       xt_text long varchar, primary key (xt_id));
```

```
create index xt_file on xml_text (xt_file);

vt_create_text_index ('xml_text',
                        'xt_text', null, 1);
```

-- create a text index on xt_text with XML well formedness checking on

```
VT_INDEX_DB_DBA_XML_TEXT (0);
```

-- populate the text index based in the data in the table.

```
VT_inc_INDEX_DB_DBA_XML_TEXT ();
```

-- refresh the index to reflect changes to the data since the last call

-- to this function or the initial call to VT_INDEX_DB_DBA_XML_TEXT.

```
vt_batch_update ('DB.DBA.XML_TEXT', 'on', 10);
```

-- sets the index maintenance to be asynchronous, refreshed at maximum intervals of 10 minutes or at every scheduler cycle if the latter is longer than 10 minutes.

```
vt_batch_update ('DB.DBA.XML_TEXT', 'off', null);
```

-- sets the index maintenance to be synchronous, within the same transaction

-- as the operation changing the indexed data.

VT_DROP_FTT

VT_DROP_FTT — drop free text trigger

Synopsis

```
DB.DBA. VT_DROP_FTT ( in target_table_name varchar ,
                      in opt_data_column or NULL varchar ) ;
```

Description

Drops free text trigger.

Parameters

DATA_COLUMN
: column where content is stored

target_table_name
: the table containing the trigger

See Also

`vt_create_text_index()`

`vt_batch()`

`vt_batch_update()`

`vt_batch_feed()`

vt_is_noise

vt_is_noise — determines whether input is a noise word

Synopsis

vt_is_noise (in *word* varchar ,
in *encoding* varchar ,
in *language* varchar) ;

Description

Determines whether input is a noise word.

Parameters

word
: Narrow string of the word to be checked

encoding
: valid encoding string

language
: valid language string

Return Values

1 or 0. This function will return 1 if the encoded word in the specified language is considered a noise word, or 0 if not.

Examples

Example 24.449. Determining if a word is a Noise Word

```
select vt_is_noise ('a', 'UTF-8', 'X-ANY');
```

will return 1

```
select vt_is_noise ('effective', 'UTF-8', 'X-ANY');
```

will return 0

week

week — get number of week from a datetime

Synopsis

week (in *dt* datetime) ;

Description

week takes a datetime and returns an integer containing a number representing the week of year of the datetime.

Parameters

dt
: A datetime .

Return Values

An INTEGER containing number of the week of year.

Examples

Example 24.450. Simple example

Get current week of year.

```
SQL> select week (now ());
callret
INTEGER
_______________________________________________________________________________

42
```

See Also

dayname , monthname , dayofmonth , dayofweek , dayofyear , quarter , month , year , hour , minute , second , timezone

wsdl_import_udt

WSDL_IMPORT_UDT — Returns a string containing a UDT definition

Synopsis

```
varchar WSDL_IMPORT_UDT ( in url varchar ,
                          in f varchar ,
                          in exec any ) ;
```

Description

This function is used to create a user defined type (UDT) definition automatically based on a WSDL file. The source WSDL is supplied via a URL.

The returned definition can be saved to a file and/or executed automatically to provide instant access to the new UDT.

Note:
To save the file you must have an appropriate ACL defined.

Parameters

url
: The URI of the target WSDL file to build a UDT definition of.

f
: The name of the file if the definition is to be saved. This will produce a file on the file system under the server root directory.

exec
: This flag indicates whether the definition should be immediately executed upon import. This takes the form of 0 or 1. The default value of 0 will not execute the definition.

Return Types

This function returns the text of the UDT definition as a varchar.

Examples

Example 24.451. Simple example of a WSDL driven UDT

Web Service proxy wrapper for Redcoal SMS SOAP interface:

```
SQL> WSDL_IMPORT_UDT ('http://xml.redcoal.com/soapserver.dll/wsdl/ISoapServer', 'redcoal.sql');
```

This will create the following file on file system called `redcoal.sql` :

```
-- Automatically generated code
-- imported from WSDL URI: "http://xml.redcoal.com/soapserver.dll/wsdl/ISoapServer"
-- UDT class
drop type "ISOAPServerservice"
;

create type "ISOAPServerservice"
  as
    (
      debug int default 0,
      url varchar default 'http://xml.redcoal.com/soapserver.dll/soap/ISoapServer',
      request varchar,
      response varchar
    )
```

```
-- Binding: "http://tempuri.org/:ISOAPServerbinding"

method "SendTextSMS"
       (
        "strInSerialNo" any,
        "strInSMSKey" any,
        "strInRecipients" any,
        "strInMessageText" any,
        "strInReplyEmail" any,
        "strInOriginator" any,
        "iInType" any,
        "strOutMessageIDs" any,
        "return" any
       ) returns any,

.... more methods follows ; do not supplied for brevity

method "RedWebServiceVersion"
       (
        "return" any
       ) returns any,

method style () returns any
;

-- Methods

create method "SendTextSMS"
       (
        in "strInSerialNo" any,
        in "strInSMSKey" any,
        in "strInRecipients" any,
        in "strInMessageText" any,
        in "strInReplyEmail" any,
        in "strInOriginator" any,
        in "iInType" any,
        inout "strOutMessageIDs" any,
        out "return" any
       ) for "ISOAPServerservice"
{
  declare action, namespace, enc varchar;
  declare style int;
  declare _result, _body, xe any;
  action := 'urn:SOAPServerImpl-ISOAPServer#SendTextSMS';

  namespace := 'urn:SOAPServerImpl-ISOAPServer';

  style := 0;

  if (self.debug)
    style := style + 2;
  _result := SOAP_CLIENT (
               url=>self.url,
               operation=>'SendTextSMS',
               soap_action=>action,
               target_namespace=>namespace,
               parameters=>vector
    (
     vector('strInSerialNo', 'http://www.w3.org/2001/XMLSchema:string'), "strInSerialNo" ,
     vector('strInSMSKey', 'http://www.w3.org/2001/XMLSchema:string'), "strInSMSKey" ,
     vector('strInRecipients', 'http://www.w3.org/2001/XMLSchema:string'), "strInRecipients" ,
     vector('strInMessageText', 'http://www.w3.org/2001/XMLSchema:string'), "strInMessageText" ,
     vector('strInReplyEmail', 'http://www.w3.org/2001/XMLSchema:string'), "strInReplyEmail" ,
     vector('strInOriginator', 'http://www.w3.org/2001/XMLSchema:string'), "strInOriginator" ,
     vector('iInType', 'http://www.w3.org/2001/XMLSchema:int'), "iInType" ,
     vector('strOutMessageIDs', 'http://www.w3.org/2001/XMLSchema:string'), "strOutMessageIDs"
    ),
               style=>style
              );
  if (self.debug)
    {
      _body := _result[0];
      self.request := _result[1];
```

```
      self.response := _result[2];
    }
  else
    _body := _result;
  xe := xml_cut (xml_tree_doc (_body));
  "strOutMessageIDs" := xml_cut (xpath_eval ('//strOutMessageIDs', xe, 1));
  "strOutMessageIDs" := soap_box_xml_entity_validating ("strOutMessageIDs",
    'http://www.w3.org/2001/XMLSchema:string', 0);
  "return" := xml_cut (xpath_eval ('//return', xe, 1));
  "return" := soap_box_xml_entity_validating ("return", 'http://www.w3.org/2001/XMLSchema:int', 0);

  return _result;
}
;

.... more method definitions follows ; not supplied for brevity

create method "RedWebServiceVersion"
       (
        out "return" any
       ) for "ISOAPServerservice"
{
  declare action, namespace, enc varchar;
  declare style int;
  declare _result, _body, xe any;
  action := 'urn:SOAPServerImpl-ISOAPServer#RedWebServiceVersion';

  namespace := 'urn:SOAPServerImpl-ISOAPServer';

  style := 0;

  if (self.debug)
    style := style + 2;
  _result := SOAP_CLIENT (
               url=>self.url,
               operation=>'RedWebServiceVersion',
               soap_action=>action,
               target_namespace=>namespace,
               parameters=>vector
                       (

                       ),
               style=>style
              );
  if (self.debug)
    {
      _body := _result[0];
      self.request := _result[1];
      self.response := _result[2];
    }
  else
    _body := _result;
  xe := xml_cut (xml_tree_doc (_body));
  "return" := xml_cut (xpath_eval ('//return', xe, 1));
  "return" := soap_box_xml_entity_validating ("return",
    'http://www.w3.org/2001/XMLSchema:string', 0);

  return _result;
}
;
```

This script, executed via the ISQL tool, or automatically if the 'exec' flag is set to 'true', will define the desired UDT in database. Developing a simple application for checking the Redcoal SMS SOAP interface could be done as follows using a stored procedure:

```
create procedure
redcoaltest ()
{
  declare svc "ISOAPServerservice";
  declare res nvarchar;
  svc := new "ISOAPServerservice" ();
```

```
  svc."RedWebServiceVersion" (res);
  result_names (res);
  result (res);
}
;
```

Which could have the following output:

```
SQL> redcoaltest();
res
NVARCHAR
_______________________________________________

5.0

1 Rows. -- 1974 msec.
```

See Also

The ??? section.

wst_cli

wst_cli — Request a security token from WST endpoint

Synopsis

```
token wst_cli ( in req soap_client_req ,
                in policy any ) ;
```

Description

This function is for use with a SOAP client contacting a WS-Trust endpoint for a security token.

Parameters

req
: A UDT containing soap_client parameters as described in WS-RM.

policy
: contains policy to be applied, an example:

Return Types

A security token.

See Also

WS Reliable Messaging

USER_KEY_LOAD

USER_KEY_LOAD — Register and existing key.

Synopsis

```
USER_KEY_LOAD (   in key_name varchar ,
                  in key_value any ,
                  in key_type varchar ,
                  in key_format varchar ,
                  in key_passwd varchar ,
                  in key_pkey any );
```

Description

This function is used to register and persist an existing key in the Virtuoso server. This is needs to be made once per key/certificate, and does not need to be repeated after server restart, unless the key is erased and has to be redefined.

Parameters

key_name
: The key reference, as a 'Sample Key' or 'file:keys/srv.pem'

key_value
: When the key is an internal reference i.e. non 'file:', the content of a key. The default value is NULL.

key_type
: The key algorithm, possible values are "3DES", "RSA", "DSA", "X.509", "AES".

key_format
: The format of the key content storage, possible values are "DER", "PEM", "PKCS12".

key_passwd
: The password for opening the key if required, this can be NULL to indicate no password required.

key_pkey
: When an X.509 certificate is imported, this can be supplied to load the corresponding private key also. This is optional. Note that when the PKCS#12 format is used to import an X.509 certificate, the private key is in the 'key_value', hence this parameter in such case must be omitted.

Return Types

None.

Example

Example 24.452. Simple Use

To load and persist an X.509 certificate contained in the file wss.pfx in the server working directory one can use:

```
USER_KEY_LOAD ('file:wss.pfx', NULL, 'X.509', 'PKCS12', 'virt');
```

To load and persist a shared secret:

```
USER_KEY_LOAD ('WSDK Sample Symmetric Key', 'EE/uaFF5N3ZNJWUTR8DYe+OEbwaKQnso', '3DES', 'DER', null);
```

To load a DSA key contained in a file dsa.der:

```
USER_KEY_LOAD ('file:dsa.der', NULL, 'DSA', 'DER', null);
```

dsig_template_ext

dsig_template_ext — Generates a Digital signature template based on a XML document.

Synopsis

```
varchar dsig_template_ext ( in xdoc any ,
                            in tmpl varchar ,
                            in wss_ver any ,
                            in ns-n varchar ,
                            in elm-n varchar ,
                            in ... varchar ) ;
```

Description

The function is used to generate dynamically a digital signature template containing references to be signed.

Parameters

xdoc
: input XML document

tmpl
: a string containing base XML template

wss_ver
: vector containing WS-Security and WS-Utility version URIs

ns-n
: namespace to match

elm-n
: element name to match

Return Types

On success the function will return a string containing a XML template suitable to pass to the `xenc_encrypt` . The elements matching listed in *`elm-n`* from namespace URIs in *`ns-n`* and having Id attribute will be included in the XML signature reference list.

Examples

Example 24.453. Making a XML signature template

```
create procedure XENC_TEMPLATE (in body varchar, in key_name varchar)
{
  declare tmpl, algo varchar;
  declare ns any;

  -- OASIS WS-Security extensions
  ns := vector (
        'wsse', 'http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd',
        'wsu', 'http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd'
        );

  algo := xenc_get_key_algo (key_name);
  if (algo is null)
    return NULL;

  -- base XML template
```

```
  tmpl := sprintf ('<?xml version="1.0" encoding="UTF-8"?>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#" >
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
    <SignatureMethod Algorithm="%s" />
  </SignedInfo>
  <SignatureValue></SignatureValue>
  <KeyInfo>
    <KeyName>%s</KeyName>
  </KeyInfo>
</Signature>', algo, key_name);

  return dsig_template_ext (xtree_doc (body), tmpl, ns,
      'http://schemas.xmlsoap.org/soap/envelope/', 'Body',
       -- WS-Addressing
      'http://schemas.xmlsoap.org/ws/2004/03/addressing', 'Action',
      'http://schemas.xmlsoap.org/ws/2004/03/addressing', 'From',
      'http://schemas.xmlsoap.org/ws/2004/03/addressing', 'To',
      'http://schemas.xmlsoap.org/ws/2004/03/addressing', 'MessageID',
      'http://schemas.xmlsoap.org/ws/2004/03/addressing', 'ReplyTo',
      'http://schemas.xmlsoap.org/ws/2004/03/addressing', 'FaultTo',
      'http://schemas.xmlsoap.org/ws/2004/03/addressing', 'RelatesTo'
      );

}
;
```

x509_certificate_verify

x509_certificate_verify — Verifies X.509 certificate

Synopsis

```
varchar x509_certificate_verify ( in cert varchar ,
                                  in cacerts any ,
                                  in flags varchar ) ;
```

Description

This function takes a X.509 certificate and verifies it against list of CA certificates. It checks for various certificate attributes such as self signed, expiration date etc. If an error is detected it will be signalled.

The certificates are passed as a strings containing X.509 certificate binary data in DER (raw) format.

Parameters

cert
: The X.509 certificate to be verified

cacerts
: array of strings containing CA certificates

flags
: A string containing comma separated list of verification options. See table below for valid values.

Return Types

None

Errors

Table 24.118. Errors signalled by `x509_certificate_verify`

SQLState	Error Code	Error Text	Description
22023	CR014	Invalid certificate	The input can't be decoded as a X.509 certificate
22023	CR016	Can not allocate a X509 store	
22023	CR019	Invalid CA certificate	Some of CA certificates can not be loaded due to bad format
22023	CR017	Can not allocate X509 verification context	
22023	CR018	Can not initialize X509 verification context	
22023	CR015	X509 error: [the verification error text]	

Examples

Example 24.454. Verification of a X.509 certificate

```
SQL> x509_certificate_verify (file_to_string ('keys/srv/cert.cer'), vector (file_to_string ('keys/srv

Done. -- 29 msec.
```

See Also

get_certificate_info()

xenc_X509_certificate_serialize

xenc_X509_certificate_serialize — Exports a X.509 certificate from user's repository

Synopsis

```
varchar xenc_X509_certificate_serialize ( in name varchar );
```

Description

This function is used to export X.509 from user's space.

Parameters

name
: The name of the key associated with X.509 certificate.

Return Types

The function returns base64 encoded string containing X.509 certificate in DER (raw) format. Also the certificate will contain the key associated with it.

Examples

Example 24.455. Exporting a X.509 certificate

The example exports a X.509 certificate associated with 'dsa-test' key from user's space.

```
...
declare x509 varchar;
x509 := xenc_X509_certificate_serialize ('dsa-test');
...
```

xenc_decrypt_soap

xenc_decrypt_soap — Decrypt and verify a SOAP message

Synopsis

```
varchar xenc_decrypt_soap ( in xml_text varchar ,
                            in soap_version int ,
                            in validate_flag int ,
                            in encoding varchar ,
                            in lang varchar ,
                            in opts any ,
                            out keys any ) ;
```

Description

The function is used to decrypt and optionally verify signature (depends of a `validate_flag` parameter) of a SOAP message.

Parameters

xml_text
: A string containing SOAP message

soap_version
: An integer indication SOAP version (11 for v1.1, 10 for 1.0 etc)

validate_flag
: Bit mask flag. See below for details.

encoding
: The message (xml_text) character encoding

lang
: The message (xml_text) language

opts
: A vector containing options for User Name token profile. For example : vector ('UsernameToken', vector ('label', 'lab1', 'keyAlgorithm' , '[3des algo uri]'))

keys
: if supplied the parameter will be set with keys names used for decryption and signature verification. The structure is as follows : vector (vector ([enc-key1],[enc-key2],...), vector ('[signing token name]', '[matching token]')). Where enc-key is a key used to decrypt the message fragment; '[signing token name]' is the temporary key used to validate the signature and '[matching token]' is the token (key) from user's space that matches the signing token.

Return Types

On success the function returns decrypted SOAP message.

Examples

Example 24.456. Decrypting SOAP message

```
declare ekeys, opts, decoded any;
opts := vector ('UsernameToken',
                vector ('label',
                        'Application-Label',
                             'keyAlgorithm',
```

```
                                    'http://www.w3.org/2001/04/xmlenc#tripledes-cbc'));
      decoded := xenc_decrypt_soap (body_str, 11, 6, 'UTF-8', 'x-any', opts, ekeys);
```

See Also

`xenc_encrypt()`

xenc_delete_temp_keys

xenc_delete_temp_keys — Remove the temporary keys from user's space

Synopsis

```
int xenc_delete_temp_keys ( );
```

Description

When signing an XML document or doing the reverse work like verification etc. number of temporary keys are created in user's space. These keys at some point may not needed anymore, so they can be removed from memory.

Return Types

The function returns number of temp keys removed.

Examples

Example 24.457. Delete temporary keys

```
SQL> select xenc_delete_temp_keys ();
callret
VARCHAR
_______________________________________________________________________________

1
```

See Also

`xenc_decrypt_soap()`

`xenc_key_3des_rand_create()`

`xenc_key_aes_rand_create()`

xenc_encrypt

xenc_encrypt — Encrypt and optionally sign a SOAP message

Synopsis

```
varchar xenc_encrypt ( in message varchar ,
                       in soap_ver int ,
                       in template varchar ,
                       in ns any ,
                       in xp-n varchar ,
                       in ki-n any ,
                       in what-n varchar ,
                       ... ) ;
```

Description

Encrypt SOAP message and optionally attach an XML signature. The keys are retrieved from the key store of the calling user account.

Parameters

message
: A string containing SOAP message

soap_ver
: SOAP version (11 = 1.1, 10 = 1.0 etc.)

template
: a string containing Digital signature template.

ns
: an array of strings containing WS-Security extension and WS-Utility version information.

xp-n
: XPath expression to locate n-th part of the message to be encrypted

ki-n
: Key instance to encrypt the part

what-n
: 'Content' - the content under located element will be encrypted; 'Element' the located element and content will be encrypted.

Return Types

On success the function will return a string containing encrypted and optionally signed SOAP message.

Examples

Example 24.458. Encrypting and signing SOAP message

```
...
ns := vector ();
template := dsig_template_ext (...);
keyi := xenc_key_inst_create ('myAES', xenc_key_inst_create ('rsa-test'));
resp := xenc_encrypt (request, 11, template, ns, '/Envelope/Body', keyi, 'Content');
```

```
...
```

See Also

`encrypt()`

`decrypt()`

`dsig_template_ext()`

`xenc_key_inst_create()`

encrypt

encrypt — Encrypt content in base64 encoding

Synopsis

```
varchar encrypt ( in string_session any ,
                  in key_name varchar ) ;
```

Description

Returns encrypted content in base64 encoding. The algorithm to encrypt depends of the key, so if the key is AES the AES will be used, if it is 3DES it is 3DES etc. The AES key can be created with xenc_key_AES_rand_create (key_name, 256). Alternatively 3DES key can be generated with xenc_key_3DES_rand_create ().

Parameters

string_session
: A content to be encrypted.

key_name
: The key name keeping secret.

Return Types

On success the function will return encrypted content in base64 encoding.

Examples

Example 24.459. Simple Use

```
-- The 'ses' is a function which print the string into a string session:
xenc_key_AES_rand_create ('kaes', 256).
decrypt (encrypt (ses ('data to encrypt'), 'kaes'), 'kaes');
```

See Also

`decrypt()`

`xenc_encrypt()`

`dsig_template_ext()`

`xenc_key_inst_create()`

decrypt

decrypt — Encrypt content in base64 encoding

Synopsis

```
varchar decrypt ( in base64_string_session any ,
                  in key_name varchar ) ;
```

Description

Returns decrypted content as a string_session. The algorithm to encrypt depends of the key, so if the key is AES the AES will be used, if it is 3DES it is 3DES etc. The AES key can be created with xenc_key_AES_rand_create (key_name, 256). Alternatively 3DES key can be generated with xenc_key_3DES_rand_create ().

Parameters

base64_string_session
: A base64 string content to be decrypted.

key_name
: The key name keeping secret.

Return Types

On success the function will return decrypted content as a string_session.

Examples

Example 24.460. Simple Use

```
-- The 'ses' is a function which print the string into a string session:
xenc_key_AES_rand_create ('kaes', 256).
decrypt (encrypt (ses ('data to encrypt'), 'kaes'), 'kaes');
```

See Also

`encrypt()`

`xenc_encrypt()`

`dsig_template_ext()`

`xenc_key_inst_create()`

TOTP_generate

TOTP_generate — Returns TOTP string Time-based One-time Password Generator (rfc6238)

Synopsis

```
varchar TOTP_generate ( in start_date datetime ,
                        in date datetime ,
                        in step integer ,
                        in return_digits integer ,
                        in key_name varchar ,
                        in digest_algo varchar ) ;
```

Description

Returns TOTP string Time-based One-time Password Generator (rfc6238).

Parameters

start_date
: T0 (zero time, usually 1970-1-1) .

date
: The current time ex. now () .

step
: Time step in secconds, usually 30.

return_digits
: How much digits to return, from 1 to 8.

key_name
: The key name keeping secret. Note: when setting up the key, should use 20 bytes for hmac-sha1, 32 bytes for hmac-sha256. Also the systems that are used for TOTP should have time synchronized.

digest_algo
: string, one of hmac-sha1 or hmac-sha256 .

Return Types

On success the function will return TOTP string Time-based One-time Password Generator (rfc6238).

Examples

Example 24.461.

```
xenc_key_RAW_read ('k2', encode_base64 (cast (hex2bin ('313233343536373839303132333435363738393031323334
select TOTP_generate (stringdate ('1970-01-01Z'), stringdate ('2603-10-11 11:33:20Z'), 30, 8, 'k2', 'hma
```

See Also

`encrypt()`

`xenc_encrypt()`

`dsig_template_ext()`

xenc_key_inst_create()

xenc_get_key_algo

xenc_get_key_algo — Determine XML signature algorithm of a key

Synopsis

varchar **xenc_get_key_algo** (in *key_name* varchar);

Description

The function is used to determine signing algorithm supported by given key.

Parameters

key_name
: Name of a key from user's space.

Return Types

The function returns string containing algorithm identifier value as described in http://www.w3.org/TR/xmldsig-core .

Examples

Example 24.462. Determine signing algorithm of a key

```
SQL> select xenc_get_key_algo ('myKey');
callret
VARCHAR
_______________________________________________________________________________

http://www.w3.org/2000/09/xmldsig#dsa-sha1
```

See Also

`dsig_template_ext()`

`xenc_get_key_identifier()`

xenc_get_key_identifier

xenc_get_key_identifier — Determine a key identifier

Synopsis

```
varchar xenc_get_key_identifier ( in key_name varchar );
```

Description

The function returns key identifier.

Parameters

key_name
: Name of the key

Return Types

The function returns a base64 encoded string containing the key identifier. If key identifier is not available (the key have no identifier or key does not exists a NULL will be returned).

Examples

Example 24.463. Determine the key identifier

```
SQL> select xenc_get_key_identifier ('myKey');
callret
VARCHAR
_______________________________________________________________________________

LtVJHvVXJFFvzRnfrCaKmdxQaGQ=
```

See Also

`xenc_get_key_algo()`

`get_certificate_info()`

xenc_key_3DES_read

xenc_key_3DES_read — Importing a triple-des key into user's repository

Synopsis

xenc_key_3DES_read (in *name* varchar ,
in *key_data* varchar) ;

Description

This function is used to import a triple-des serialized key into user's repository and register it with a name supplied.

Note that key will not be persisted. It is loaded in the memory only.

Parameters

name
: Name of the key to register

key_data
: The base64 encoded binary data with key material

Return Types

No return value.

Examples

Example 24.464. Loading a shared secret

```
xenc_key_3DES_read ('Sample Symmetric Key', 'EE/uaFF5N3ZNJWUTR8DYe+OEbwaKQnso');
```

See Also

`xenc_key_3DES_create`

`xenc_key_3des_rand_create`

`xenc_key_aes_create`

`xenc_key_aes_rand_create`

`xenc_key_DSA_read`

`xenc_key_dsa_create`

`xenc_key_RSA_read`

`xenc_key_create_cert`

`xenc_key_serialize`

`xenc_key_remove`

xenc_key_3DES_create

xenc_key_3DES_create — Used to make a symmetric key.

Synopsis

xenc_key_3DES_create (in *name* varchar ,
in *secret* varchar) ;

Description

Used to make a symmetric session key for triple-des algorithm.

Parameters

name
: The name of the key.

secret
: The shared secret for key generation.

Return Types

None.

Example

Example 24.465. Simple Use

```
xenc_key_3DES_create ('ses_key', 'mysecret');
```

See Also

`xenc_key_3DES_read()`

`xenc_key_3des_rand_create()`

`xenc_key_aes_create()`

`xenc_key_aes_rand_create()`

`xenc_key_DSA_read()`

`xenc_key_dsa_create()`

`xenc_key_RSA_read()`

`xenc_key_create_cert()`

`xenc_key_serialize()`

`xenc_key_remove()`

xenc_key_3DES_rand_create

xenc_key_3DES_rand_create — Used to make a temporary session key.

Synopsis

xenc_key_3DES_rand_create (in *name* varchar) ;

Description

Used to make a temporary session key.

Parameters

name
: The name of the key.

Return Types

None.

Example

Example 24.466. Simple Use

```
xenc_key_3DES_rand_create ('ses_key');
```

See Also

`xenc_key_3DES_create()`

`xenc_key_AES_rand_create()`

`xenc_key_aes_create()`

`xenc_key_3DES_read()`

`xenc_key_DSA_read()`

`xenc_key_dsa_create()`

`xenc_key_RSA_read()`

`xenc_key_create_cert()`

`xenc_key_serialize()`

`xenc_key_remove()`

xenc_key_DSA_read

xenc_key_DSA_read — Importing a DSA key into user's repository

Synopsis

xenc_key_DSA_read (in *name* varchar ,
in *key_data* varchar) ;

Description

This function is used to import a DSA serialized key into user's repository and register it with a name supplied.

Note that key will not be persisted. It is loaded in the memory only.

Parameters

name
: Name of the key to register

key_data
: The base64 encoded binary data with DSA key material

Return Types

No return value.

Examples

Example 24.467. Loading a shared secret

```
xenc_key_DSA_read ('myDSAkey', 'MII..skipped..Klmu+tlaA==');
```

See Also

`xenc_key_DSA_create()`

xenc_key_RSA_read

xenc_key_RSA_read — Importing a RSA key into user's repository

Synopsis

xenc_key_RSA_read (in *name* varchar ,
in *key_data* varchar) ;

Description

This function is used to import an RSA serialized key into user's repository and register it with a name supplied.

Note that key will not be persisted. It is loaded in the memory only.

Parameters

name
: Name of the key to register

key_data
: The base64 encoded binary data with RSA key material

Return Types

No return value.

Examples

Example 24.468. Loading a shared secret

```
xenc_key_RSA_read ('MyRSAkey', 'MIG..skipped..MBAAE=');
```

See Also

`xenc_key_RSA_create()`

xenc_key_AES_create

xenc_key_AES_create — Used to make a symmetric key.

Synopsis

xenc_key_AES_create (in *name* varchar ,
in *bits* int ,
in *secret* varchar) ;

Description

Used to make a symmetric session key for AES encryption algorithm.

Parameters

name
: The name of the key.

bits
: Number of bits in generated key. The values supported are 128, 192 and 256 bits.

secret
: The shared secret for key generation.

Return Types

None.

Example

Example 24.469. Simple Use

```
xenc_key_AES_create ('ses_key', 128, 'mysecret');
```

See Also

`xenc_key_3DES_create()`

`xenc_key_3des_rand_create()`

`xenc_key_3des_read()`

`xenc_key_aes_rand_create()`

`xenc_key_DSA_read()`

`xenc_key_dsa_create()`

`xenc_key_RSA_read()`

`xenc_key_create_cert()`

`xenc_key_serialize()`

`xenc_key_remove()`

xenc_key_AES_rand_create

xenc_key_AES_rand_create — Used to make a temporary session key.

Synopsis

xenc_key_AES_rand_create (in *name* varchar ,
in *bits* int) ;

Description

Used to make a temporary session key for AES encryption.

Parameters

name
: The name of the key.

bits
: Number of bits in generated key. The values supported are 128, 192 and 256 bits.

Return Types

None.

Example

Example 24.470. Simple Use

```
xenc_key_AES_rand_create ('ses_key', 192);
```

See Also

`xenc_key_3DES_create()`

`xenc_key_3des_rand_create()`

`xenc_key_aes_create()`

`xenc_key_3DES_read()`

`xenc_key_DSA_read()`

`xenc_key_DSA_create()`

`xenc_key_RSA_read()`

`xenc_key_create_cert()`

`xenc_key_serialize()`

`xenc_key_remove()`

xenc_key_create_cert

xenc_key_create_cert — Import a key from a certificate

Synopsis

xenc_key_create_cert (in *name* varchar ,
in *cert* varchar ,
in *type* varchar ,
in *fmt* int ,
in *pkey* varchar ,
in *pass* varchar) ;

Description

The function is used to import a key (usually an asymmetric key like RSA or DSA) into user's space from a certificate.

Parameters

name
: The name of the key to be registered

cert
: String containing the certificate

type
: 'X.509' is only supported value

fmt
: The input format of the certificate: 3 - "DER", 1 - "PEM" or 2 - "PKCS#12".

pkey
: When an X.509 certificate is imported, this can be supplied to load the corresponding private key also. This is optional. Note that when the PKCS#12 format is used to import an X.509 certificate, the private key is in the 'cert', hence this parameter in such case must be omitted.

pass
: The password to unlock the private key. If not required this can be NULL to indicate no password is needed.

Examples

Example 24.471. Importing a key from X.509 certificate

```
...
xenc_key_create_cert ('myRSA', file_to_string ('keys/mycert.pfx'), 'X.509', 2, NULL, 'secret');
xenc_set_primary_key ('myRSA');
...
```

See Also

`xenc_set_primary_key()`

`xenc_key_3DES_create()`

`xenc_key_3des_rand_create()`

`xenc_key_aes_create()`

xenc_key_3DES_read()

xenc_key_DSA_read()

xenc_key_dsa_create()

xenc_key_RSA_read()

xenc_key_serialize()

xenc_key_remove()

xenc_key_DSA_create

xenc_key_DSA_create — Used to make asymmetric DSA key.

Synopsis

xenc_key_DSA_create (in *name* varchar);

Description

Used to make asymmetric DSA key for digital signatures. The generated key will contain private and public keys.

Parameters

name
: The name of the key.

Return Types

None.

Example

Example 24.472. Simple Use

```
xenc_key_DSA_create ('myDSAkey');
```

See Also

`xenc_key_3DES_create()`

`xenc_key_3des_rand_create()`

`xenc_key_aes_create()`

`xenc_key_3DES_read()`

`xenc_key_DSA_read()`

`xenc_key_RSA_read()`

`xenc_key_create_cert()`

`xenc_key_serialize()`

`xenc_key_remove()`

xenc_key_exists

xenc_key_exists — Check if named encryption key is in the user's repository

Synopsis

```
int xenc_key_exists ( in name varchar );
```

Description

The function checks if key with given name exists in the user's keys.

Parameters

name
: The name of the key

Return Types

The function returns integer 1 - true or 0 - false (zero) if key exists or not

Examples

Example 24.473. Simple Use

```
SQL> select xenc_key_exists ('MyKey');
VARCHAR
_______________________________________________________________________________

0

1 Rows. -- 1 msec.
```

See Also

`xenc_key_remove()`

xenc_key_inst_create

xenc_key_inst_create — Key instance generation

Synopsis

```
any xenc_key_inst_create ( in name varchar ,
                           in super any ) ;
```

Description

The function is used to make a key reference used in encryption functions.

Parameters

name
: Name of the key to be instantiated.

super
: Optional parameter, is specified must contains a key reference to the super key. In other words the key used to encrypt key specified by name .

Return Types

On success the function returns a non-null value containing key reference data.

Examples

Example 24.474. Key instance creation

The example specifies to use an AES key to be used as a session key, also AES will be encrypted with a RSA one.

```
...
keyi := xenc_key_inst_create ('myAES', xenc_key_inst_create ('rsa-test'))
...
```

See Also

`xenc_key_exists()`

xenc_key_remove

xenc_key_remove — This will delete a key from current user's space.

Synopsis

xenc_key_remove (in *key_name* varchar) ;

Description

This will delete a key from current user's space.

Parameters

key_name
: The name of the key.

Example

Example 24.475. Simple Use

```
xenc_key_remove ('file:dsa.der');
```

See Also

`xenc_key_exists()`

xenc_key_serialize

xenc_key_serialize — Extracts a key from user's repository

Synopsis

```
varchar xenc_key_serialize (
                                in name
                                varchar ,
                                in public int
                                );
```

Description

The function returns a string containing base64 encoded binary key data. It is used to extract symmetric or asymmetric keys. So if key is asymmetric (RSA or DSA) the second parameter designate which part to extract private or public.

Parameters

name
: The key name.

public
: 1 - export public part, 0 - export private key if exists.

Return Types

The function returns serialized key material.

Examples

Example 24.476. Exporting a RSA public key.

The 'ServerPrivate.pfx' is a own RSA key so we exporting public part to supply to some party to encrypt data for us.

```
SQL> select xenc_key_serialize ('ServerPrivate.pfx', 1);
callret
VARCHAR
_______________________________________________________________________________

MIGJAoGBAMuSqCUCsie6RGUPBdtyM3fPF+yN5ot34i9/IPmjfmlSOlOQ/A9eYClJyvxRVDEHkkNFJWUzPWVRDoIEsUsYgBA6ls8qfai6X

1 Rows. -- 1 msec.
```

xenc_set_primary_key

xenc_set_primary_key — Associate a X.509 certificate with a key

Synopsis

xenc_set_primary_key (in *name* varchar) ;

Description

This function is used to associate a key (to be used as primary) with a X.509 certificate. Usually this function is called after key import from a X.509 certificate.

Parameters

name
: Name of the key to be processed.

Return Types

No return value.

Examples

Example 24.477. Importing a key from X.509 certificate

```
...
xenc_key_create_cert (key_name, key_value, key_type, fmt, NULL, key_passwd);
xenc_set_primary_key (key_name);
...
```

See Also

`xenc_key_create_cert()`

xenc_x509_ss_generate

xenc_x509_ss_generate — Creates a self-signed X.509v3 certificate

Synopsis

xenc_x509_ss_generate (in *key_name* varchar ,
in *serial* integer ,
in *validity* integer ,
in *subject* vector ,
in *extensions* vector) ;

Description

This function is used to create a self-signed X.509 certificate by given private key

Parameters

key_name
: The name of the key used to create and sign the certificate

serial
: The serial number

validity
: Certificate's validity in days

subject
: An array of name/value pairs representing the subject aka name enties

extensions
: An array of name/value pairs to add as X.509v3 extensions to certificate

Examples

Example 24.478. Self-signed certificate generation

The example below shows how could be created RSA private key, then self-signed certificate and finally to be stored in the user's key store.

```
...
xenc_key_RSA_create ('id_rsa', atoi (get_keyword ('num', params, '1024')));
xenc_x509_ss_generate ('id_rsa', sequence_next ('ca_id_rsa'), 365,
         vector ('C', get_keyword ('c', params),
                 'O',  get_keyword ('o', params),
                 'CN',  get_keyword ('name', params),
                 'emailAddress',  get_keyword ('email', params)),
         vector ('authorityKeyIdentifier', 'keyid,issuer:always'));
USER_KEY_STORE (user, 'id_rsa', 'X.509', 2, '', xenc_pkcs12_export ('id_rsa', 'CA Certificate', ''
...
```

See Also

`user_key_store()`

`xenc_x509_generate()`

`xenc_key_RSA_create()`

xenc_pkcs12_export()

xenc_x509_generate

xenc_x509_generate — Generates a X.509v3 certificate using client's public key

Synopsis

xenc_x509_generate (in *ca_key_name* varchar ,
in *public_key_name* varchar ,
in *serial* integer ,
in *validity* integer ,
in *subject* vector ,
in *extensions* vector ,
in *hours* integer) ;

Description

This function creates a X.509v3 certificate from a public key and sign the certificate with CA private key

Parameters

ca_key_name
: The name of CA key, the key must have corresponding certificate.

public_key_name
: The name of the public key used to create certificate, it is given by certificate owner to CA. Note that public key must be imported already in CA'a key space.

serial
: The serial number

validity
: Certificate's validity in days

subject
: An array of name/value pairs representing the subject aka name enties

extensions
: An array of name/value pairs to add as X.509v3 extensions to certificate

hours
: If specified it is added to the days which allows a more fine-grained control over the expiration date.

Examples

Example 24.479. Issuing a certificate

The example below decodes a public key sent from client and then issue a certificate and sign it with CA's key 'id_rsa'. The client's key is create as temp key.

```
declare kname, cvalue varchar;
kname := xenc_SPKI_read (null, replace (get_keyword ('key', params), '\r\n', ''));
xenc_x509_generate ('id_rsa', kname, sequence_next ('ca_id_rsa'), 365,
                vector (
                        'CN', get_keyword ('name', params, name),
                        'C',  get_keyword ('c', params, name),
                        'O',  get_keyword ('o', params, name),
                        'OU', get_keyword ('ou', params, name),
                        'emailAddress', get_keyword ('email', params)
                        ),
```

```
                    vector (
                      'subjectAltName',
                      'URI:'||webid,
                      'nsComment',
                      'Virtuoso Generated Certificate',
                            'authorityKeyIdentifier',
                            'keyid,issuer:always'));
```

See Also

`xenc_SPKI_read()`

`xenc_x509_ss_generate()`

xenc_pkcs12_export

xenc_pkcs12_export — Generates a secure Public-Key Cryptography Standards (PKCS#12) file from a PEM (base64 encoded DER) file comprised of an X.509 certificate and an associated Private Key.

Synopsis

```
xenc_pkcs12_export ( in key_name varchar ,
                     in name varchar ,
                     in pass varchar ,
                     in export_chain integer := 0 ,
                     in additional_certs varchar ) ;
```

Description

Generates a secure Public-Key Cryptography Standards (PKCS#12) file from a PEM (base64 encoded DER) file comprised of an X.509 certificate and an associated Private Key.

Parameters

key_name
: Name that identifies private key associated with X.509 certificate.

name
: Name to used to identify each pkcs#12 object bundle.

pass
: A password or pass phrase used to encrypt the pkcs#12 file

export_chain
: 1/0 - to export the certificate chain if there is a such

additional_certs
: Certificates in PEM format to be added to the bundle even if they are not in the certificate chain stored in the memory object of the certificate

Examples

Example 24.480. Persisting a key & certificate

The example below persists a key named 'id_rsa' and certificate created for it into user's key space. Thus after server restart the key will be available again.

```
USER_KEY_STORE (user, 'id_rsa', 'X.509', 2, '', xenc_pkcs12_export ('id_rsa', 'CA Certificate', ''));
```

See Also

`user_key_store()`

`xenc_x509_generate()`

`xenc_x509_ss_generate()`

xenc_pem_export

xenc_pem_export — Export of a certificate and optionally private key in PEM format

Synopsis

xenc_pem_export (in *key_name* varchar ,
in *pk_flag* integer) ;

Description

The function is used to export certificate in PEM format, optionally it can export also private key (if present)

Parameters

key_name
: The key name which keeps the certificate

pk_flag
: A flag : 1 or 0, by default 0. It denotes to export or to skip exporting private key.

Examples

Example 24.481. Creating a certificate and exporting it in PEM format

The example below imports a client public key, then it creates a certificate and sign it with CA's private key and finally it exports the generated certificate in PEM format.

```
kname := xenc_SPKI_read (null, replace (get_keyword ('key', params), '\r\n', ''));
xenc_x509_generate (
    'id_rsa',
    kname,
    sequence_next ('ca_id_rsa'),
    365,
                vector (
                        'CN', get_keyword ('name', params, name),
                        'C',  get_keyword ('c', params, name),
                        'O',  get_keyword ('o', params, name),
                        'OU', get_keyword ('ou', params, name),
                        'emailAddress', get_keyword ('email', params)
                        ),
                vector (
                  'subjectAltName',
                  'URI:'||webid,
                  'nsComment',
                  'Virtuoso Generated Certificate',
                        'authorityKeyIdentifier',
                        'keyid,issuer:always'));
cvalue := xenc_pem_export (kname);
```

Example 24.482. Export Web Server certificate for Virtuoso HTTPS endpoints

This example exports Web Server certificate generated for Virtuoso HTTPS Endpoint:

1. Configure an HTTPS Listener for handling HTTPS requests from HTTP user agents (clients).
2. Suppose the generate certificate from the previous step has name "https_key_localhost"
3. To export the certificate, execute from iSQL:

```
SQL> SELECT xenc_pem_export ('https_key_localhost');
```

Figure 24.1. iSQL Export Web Server Sertificate Output

See Also

`xenc_SPKI_read`

`xenc_xenc_x509_generate`

xenc_SPKI_read

xenc_SPKI_read — Imports a public key from simple public key infrastructure (SPKI)

Synopsis

xenc_SPKI_read (in *key_name* varchar ,
in *key_data* varchar) ;

Description

This function is used to read an RSA public key from SPKI content.

Parameters

key_name
: The name of the key to be created, if null is given a temp key will be created

key_data
: The SPKI content

Examples

Example 24.483. SPKI reading

The example below creates a temp key.

```
kname := xenc_SPKI_read (null, replace (get_keyword ('key', params), '\r\n', ''));
```

See Also

`xenc_x509_generate`

xenc_bn2dec

xenc_bn2dec — Prints as a string a big number serialized as binary using base64 encoding.

Synopsis

xenc_bn2dec (in *num* varchar) ;

Description

Prints as a string a big number serialized as binary using base64 encoding

Parameters

num
: The bas64 encoded string containing a binary representation of a big number

xenc_key_RSA_create

xenc_key_RSA_create — Used to make RSA key.

Synopsis

xenc_key_RSA_create (in *name* varchar ,
in *bits* int) ;

Description

Used to make RSA key.

Parameters

name
: The name of the key.

bits
: Number of bits in generated key. The values supported are from 512 to 2048.

Return Types

None.

Example

Example 24.484. Simple Use

```
xenc_key_RSA_create ('myRSAkey', 1024);
```

See Also

`xenc_key_3DES_create()`

`xenc_key_3des_rand_create()`

`xenc_key_aes_create()`

`xenc_key_3DES_read()`

`xenc_key_DSA_read()`

`xenc_key_RSA_read()`

`xenc_key_create_cert()`

`xenc_key_serialize()`

`xenc_key_remove()`

xenc_x509_csr_generate

xenc_x509_csr_generate — Generate Certificate Signing Request (CSR)

Synopsis

```
varchar xenc_x509_csr_generate ( in cli_pk_name varchar ,
                                 in subject array ,
                                 in extensions array ) ;
```

Description

This function generates Certificate Signing Request (CSR)

Parameters

cli_pk_name
: The user's private key name.

subject
: An array of name/value pairs representing the subject aka name enties, for ex:

extensions
: An array of name/value pairs to add as X.509v3 extensions to certificate.

Return Types

String

Errors

Table 24.119. Errors signalled by `xenc_x509_csr_generate`

SQLState	Error Code	Error Text	Description
22023	XECXX	Missing key	
22023	XECXX	Key is not DSA nor RSA	
22023	XECXX	Missing private key	
22023	XECXX	Subject array must be name/value pairs	
22023	XECXX	Extension array must be name/value pairs	
42000	XECXX	Can not create pkey	
42000	XECXX	Can not assign primary key	
42000	XECXX	Can not create pkey	
42000	XECXX	Can not create x.509 structure	
42000	XECXX	Can not sign certificate : [the sign error text]	

Examples

Example 24.485. Create a new xenc_x509 CSR

```
SQL> create procedure csr_gen_demo()
{
declare kname, cvalue varchar;
       xenc_key_RSA_create ('myRSAkey', 1024);
       xenc_x509_csr_generate ('myRSAkey',
                vector (
                        'CN', 'Demo user',
```

```
                        'C',  'US',
                        'O',  'OpenLink',
                        'OU', 'Accounts',
                        'emailAddress', 'demo@openlinksw.com'),
                vector ('subjectAltName', 'URI: http://www.openlinksw.com/dataspace/person/demo#this', 'n
                        'authorityKeyIdentifier', 'keyid,issuer:always'));
};

Done. -- 0 msec.
SQL> select csr_gen_demo();
temp
VARCHAR
_______________________________________________________________________________

-----BEGIN CERTIFICATE REQUEST-----
MIICLDCCAZUCADBrMRIwEAYDVQQDEwlEZW1vIHVzZXIxCzAJBgNVBAYTAlVTMREw
DwYDVQQKEwhPcGVuTGluazERMA8GA1UECxMIQWNjb3VudHMxIjAgBgkqhkiG9w0B
CQEWE2RlbW9Ab3Blbmxpbmtzdy5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJ
AoGBAM+e21xeQIJX5ySd6Juc3GtCnyT+JvDKgoxkmovFdqkCm1Ks7Pys0x59ksSZ
DZoEATEt/sQsEB7AnRAkH761WyG0zMtvxdkFH8Cepaaq4bcdqSgmE12DRwViR95a
ZNA8vhonr5UdTbtKkIGD67IrkGH81C8trwF+8LnYKdtf1bMdAgMBAAGggYEwfwYJ
KoZIhvcNAQkOMXIwcDA/BgNVHREEODA2hjRodHRwOi8vd3d3Lm9wZW5saW5rc3cu
Y29tL2RhdGFzcGFjZS9wZXJzb24vZGVtbyN0aGlzMC0GCWCGSAGG+EIBDQQgFh5W
aXJ0dW9zbyBHZW5lcmF0ZWQgQ2VydGlmaWNhdGUwDQYJKoZIhvcNAQEEBQADgYEA
CQ+UeQGWLnHn5X9tKumvgP2p4LgxHsNq0uFZffVBRUAV6w8WRwor5ZKOeNhHrldp
ZvxHc9ESFBTA772W01VmQNfpjkFcpH5yd81dFkSsF3lKemQIHnlz4bORU4av2TMM
T9OIp5FNOec13jXqVesymd+Kl+Vb9sRBaC1Xb1OSryU=
-----END CERTIFICATE REQUEST-----

1 Rows. -- 390 msec.
```

See Also

get_certificate_info()

xenc_x509_from_csr()

xenc_x509_generate()

xenc_x509_from_csr

xenc_x509_from_csr — Generate x509 certificate from CSR.

Synopsis

```
varchar xenc_x509_from_csr ( in ca_key_name varchar ,
                             in cli_key_name varchar ,
                             in csr_str varchar ,
                             in serial_no varchar ,
                             in days_validity varchar ,
                             in hours_validity varchar ) ;
```

Description

This function generates Certificate Signing Request (CSR). The function return 1 upon success, sql error on failure. The new certificate can be exported in PEM format with `xenc_pem_export(cli_key_name)` .

Parameters

ca_key_name
: The name of CA private key which will be used to sign certificate.

cli_key_name
: The name of a key which will be created and will contains the client certificate.

csr_str
: pem encoded CSR.

serial_no
: Serial number.

days_validity
: How many days will be valid the certificate.

hours_validity
: How many hours will be valid the certificate.

Return Types

the function returns 1 upon success, sql error on failure.

Errors

Table 24.120. Errors signalled by `xenc_x509_from_csr`

SQLState	Error Code	Error Text	Description
22023	XECXX	The key [name of the key] already exists	
22023	XECXX	Missing or invalid signer certificate	
22023	XECXX	Invalid certificate request	
22023	XECXX	Invalid certificate request public key	
22023	XECXX	Signature did not match the certificate request	
22023	XECXX	Invalid certificate request subject name	
42000	XECXX	Can not create x.509 structure	
42000	XECXX	Can not sign certificate	

SQLState	Error Code	Error Text	Description
42000	XECXX	The type of public key is not supported mus tbe RSA or DSA	
42000	XECXX	Can not create a key	
42000	XECXX	Can not sign certificate : [the sign error text]	

Example

Example 24.486. Generating a Certificate Signing Request (CSR)

The following example demonstrates how to generate Certificate Signing Request (CSR).

- Using Conductor, for user dba import a certificate with primary key and with name id_rsa.
- Execute:

```
SQL>create procedure csr_demo()
{
declare kname, cvalue varchar;
declare _key any;
declare _output int;

       xenc_key_RSA_create ('RSAKey1', 1024);

    _key := xenc_x509_csr_generate ('RSAKey1',
                vector (
                        'CN', 'Demo user',
                        'C',  'US',
                        'O',  'OpenLink',
                        'OU', 'Accounts',
                        'emailAddress', 'demo@openlinksw.com'),
                vector ('subjectAltName', 'URI: http://www.openlinksw.com/dataspace/person/demo#this', 'n
                        'authorityKeyIdentifier', 'keyid,issuer:always'));

   xenc_x509_from_csr ('id_rsa', 'MyKey1', _key , sequence_next ('ca_id_rsa'), 365, 100);
   return xenc_pem_export ('MyKey1');

}
;

Done. -- 0 msec.

SQL> select csr_demo();
temp2
VARCHAR
_______________________________________________________________________________

-----BEGIN CERTIFICATE-----
MIIDSjCCAjKgAwIBAgIBEzANBgkqhkiG9w0BAQQFADBVMQswCQYDVQQIEwJvbDEL
MAkGA1UEChMCb2wxCzAJBgNVBAsTAm9sMQ0wCwYDVQQDEwRqb2huMR0wGwYJKoZI
hvcNAQkBFg5qb2huQGdtYWlsLmNvbTAeFw0xMTA2MDcxNTAxNDhaFw0xMjA2MTAx
OTAxNDhaMGsxEjAQBgNVBAMTCURlbW8gdXNlcjELMAkGA1UEBhMCVVMxETAPBgNV
BAoTCE9wZW5MaW5rMREwDwYDVQQLEwhBY2NvdW50czEiMCAGCSqGSIb3DQEJARYT
ZGVtb0BvcGVubGlua3N3LmNvbTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA
3Ma/MlMrGruwpDsW2D3iYA6sqFqldPsgx837dNJel8ZQu5/0Nyr5DCtAQNq6nWZo
0bezK9UIfAXEQXWt4S7IMPfTF6oCT85YDsQEEE/o1spsZ9Q7kXhKu3R3LNNiTxYr
TR4FSl361pUqyYngSMTxVWJxKnPW30p94i5QuQjLF1sCAwEAAaOBkjCBjzAdBgNV
HQ4EFgQUF5wTSXH98IqiaaxfVTNcui8p8SowPwYDVR0RBDgwNoY0aHR0cDovL3d3
dy5vcGVubGlua3N3LmNvbS9kYXRhc3BhY2UvcGVyc29uL2RlbW8jdGhpczAtBglg
hkgBhvhCAQ0EIBYeVmlydHVvc28gR2VuZXJhdGVkIENlcnRpZmljYXRlMA0GCSqG
SIb3DQEBBAUAA4IBAQCwSN3y6yeCNe+/izo5GwM+16cjmZkyMUYmAO62I6T62jmI
p0nYaVhJ9WV0ntVnx1H8/LKwrgyLlhXacVw4jyXwFMSo+YuONj+kKpobNH2cl+u1
+c0kJGbY/eS99S2D3JhL6n+QukvQIqhYniZ21wT1ugwpN2A7NtY+g925+vQBO0UH
0wQm3eQk8NADEjcqrmGmJcrK22jfaBNov+O2wvcZQM7WIKm98f/7So7kBN0BoRX8
7LRf3zIhp4f9fk1QDwlm9NgwgxARqNOfRuJU2YU1ICz88LbwM4XDeb+Mdr0YMNdU
6eYkCB4vKsVH+s1E8m67QZ8TGxpNZLYXLZZBdt86
-----END CERTIFICATE-----

1 Rows. -- 172 msec.
```

See Also

get_certificate_info()

xenc_x509_csr_generate()

xenc_x509_generate()

xte_head

xte_head — Returns the vector corresponding to a head of a XML element

Synopsis

xte_head (in *tagname* varchar ,
in *attr1name* varchar ,
in *attr1val* varchar ,
... ,
in *attrNname* varchar ,
in *attrNval* varchar) ;

Description

This function takes odd number of parameters and creates vector which corresponds to a head of a XML element. The first parameter is a tag name of the head. The remaining parameters are optional. Each even parameter is a name of an attribute, each next odd parameter is a value of this attribute. If two or more attributes have the same name, the head would have only the last pair.

Parameters

tagname
: The tag name of the head

attrIname
: The name of the I-th attribute

attrIval
: The value of the I-th attribute

Errors

Table 24.121. Errors signalled by `xte_head`

SQLState	Error Code	Error Text	Description
42000	XTE01	Function xte_head should have an odd number of parameters	

Examples

Example 24.487. Simple Use

The xte_head() call below takes three strings

```
xte_head ('supplier', 'CompanyName', 'Seller' )
```

and produces the result vector which corresponds the following head of a XML element:

<supplier CompanyName="Seller">

See Also

xte_node()

xte_node_from_nodebld()

xte_nodebld_acc()

xte_nodebld_final()

xte_nodebld_init()

xte_node

xte_node — Returns the vector corresponding to a XML element

Synopsis

xte_node (in *head* any ,
in *arg1* any ,
... ,
in *argN* any) ;

Description

This function returns the vector which corresponds to a XML element. The first parameter is a head of the element. The remaining parameters are optional. Each optional parameter either is a XML element or a string. Two or more successive strings are concatenated.

Parameters

argI
: A XML element or a string

Errors

Table 24.122. Errors signalled by xte_node

SQLState	Error Code	Error Text	Description
42000	XTE02	Function xte_node should have at least one parameter	

Examples

Example 24.488. Simple Use

The xte_node() call

```
xte_node(xte_head ('supplier', 'CompanyName', 'Seller'), 'this string will be ', 'concatenated with it')
```

produces a vector corresponding the following XML element:

<supplier CompanyName="Seller">this string will be concatenated with it </supplier>

See Also

xte_head

xte_node_from_nodebld

xte_nodebld_acc

xte_nodebld_final

xte_nodebld_init

xte_node_from_nodebld

xte_node_from_nodebld — Returns the vector corresponding to a XML element

Synopsis

xte_node_from_nodebld (in *head* any ,
inout *element* any) ;

Description

This function replaces the first item of the second argument by the first argument and returns a vector corresponding to an XML element.

Parameters

head
: This argument is a vector returned by xte_head() function

element
: This argument is a vector created by xte_nodebld _int() function and then filled by xte_nodebld_acc and xte_nodebld_final() functions

Errors

Table 24.123. Errors signalled by xte_node_from_nodebld

SQLState	Error Code	Error Text	Description
22003	SR349	Too few arguments for xte_node_from_nodebld	
22003	SR350	The second argument of xte_node_from_nodebld is not made by xte_nodebld_init() function	

Examples

Example 24.489. Simple Use

The vector res below is corresponding to the following XML element:

<product><supplier CompanyName="Seller"></supplier></product>

```
create procedure test_nodebld()
              {
                declare acc any;
                declare res any;
                xte_nodebld_init (acc);
                xte_nodebld_acc(acc,xte_node(xte_head ('supplier', 'CompanyName','Seller')));
                acc:=xte_nodebld_final(acc);
                res:=xte_node_from_nodebld(xte_head('product'), acc);
                result (length(res), xml_tree_doc (res));
              };
test_nodebld()
ResLen      Res
INTEGER     VARCHAR
_______________________________________________________________________________

2           <product><supplier CompanyName="Seller" /></product>
```

See Also

xte_head()

xte_node()

xte_nodebld_acc()

xte_nodebld_final()

xte_nodebld_init()

xte_nodebld_acc

xte_nodebld_acc — Adds to the first arguments all remaining arguments

Synopsis

```
xte_nodebld_acc ( in acc any ,
                  in arg1 any ,
                  ... ,
                  in argN any ) ;
```

Description

The xte_nodebld_acc() function successively adds the remaining arguments to the first one. If the last items of the first argument and some following arguments are strings, they are concatenated. All successive strings are concatenated. The value of the first item is the number of the non-empty items in the returned vector. The length of the returned vector is the sum of the length of the first argument and number of the remaining arguments.

Parameters

acc
: A vector initiated by xte_nodebld_init() function and filled by xte_nodebld_acc() functions

argI
: A string or a vector returned by xte_node_from_nodebld or xte_node functions

Errors

Table 24.124. Errors signalled by `xte_nodebld_acc`

SQLState	Error Code	Error Text	Description
22003	SR345	Too few arguments for xte_nodebld_acc	
22003	SR346	The first argument of xte_nodebld_acc is not made by xte_nodebld_init() function	

Examples

Example 24.490. Simple Use

```
create procedure test_nodebld()
              {
                declare acc any;
                xte_nodebld_init (acc);
                xte_nodebld_acc (acc,'string1', string2');
                xte_nodebld_acc(acc, 'string3',xte_node(xte_head ('supplier', 'CompanyName','Seller')));
                . . .
              }
```

The length of the vector acc as the result of the two xte_nodebld_acc() calls below is equal to 4, but only three items are not empty:

'3' (number of not empty elements of the vector including this item)

'string1string2string3' (the result of concatenation)

vector corresponding the following XML element: <supplier CompanyName="Seller"> </supplier>

See Also

xte_head()

xte_node()

xte_node_from_nodebld()

xte_nodebld_final()

xte_nodebld_init()

xte_nodebld_final

xte_nodebld_final — Corrects input vector

Synopsis

xte_nodebld_final (inout *arg* any ,
in *head* any) ;

Description

By default this function reduces all last empty items from the first argument and returns it, but supplied a second argument it also replaces the first item of the first argument by the second argument, but does not commit the return.

Parameters

arg
: The vector created by the xte_nodebld_init() and filled by xte_nodebld_acc() function

head
: A vector returned by xte_head() function

Errors

Table 24.125. Errors signalled by `xte_nodebld_final`

SQLState	Error Code	Error Text	Description
22003	SR347	Too few arguments for xte_nodebld_final	
22003	SR348	The first argument of xte_node_nodebld_final is not made by xte_nodebld_init() function	

Examples

Example 24.491. xte_nodebld_final() with a single argument

Before the xte_nodebld_final() call the length of the vector acc is equal to 16, after the call the length is 3. The vector acc does not correspond to a xml document - there is no a root tag. The result of the xte_nodebld_final() may be used as argument for xte_node_from_nodebld() to get a vector corresponding to complete xml document.

```
create procedure test1_nodebld()
              {
                declare acc any;
                xte_nodebld_init (acc);
                xte_nodebld_acc (acc,'string1', string2');
                xte_nodebld_acc(acc, 'string3',xte_node(xte_head ('supplier', 'CompanyName','Seller')));
                acc:=xte_nodebld_final(acc);
                            . . .
              }
```

Example 24.492. xte_nodebld_final() with two arguments.

The vector acc below is corresponding to the following XML element:

<product>string1string2string3

<supplier CompanyName="Seller"></supplier>

</product>

```
create procedure test2_nodebld()
              {
                declare acc any;
                declare "Res" any;
                declare "ResLen" integer;
                result_names ("ResLen", "Res");
                xte_nodebld_init (acc);
                xte_nodebld_acc (acc,'string1', 'string2');
                xte_nodebld_acc(acc, 'string3',xte_node(xte_head ('supplier', 'CompanyName','Seller')));
                xte_nodebld_final(acc, xte_head('product'));
                result (length(acc), xml_tree_doc (acc));
              }
test_nodebld()
ResLen      Res
INTEGER     VARCHAR
_______________________________________________________________________________

3           <product>string1string2string3<supplier CompanyName="Seller" /></product>
```

See Also

xte_head()

xte_node()

xte_node_from_nodebld()

xte_nodebld_acc()

xte_nodebld_init()

xte_nodebld_init

xte_nodebld_init — Creates the empty vector

Synopsis

xte_nodebld_init (inout *arg* any);

Description

This function creates the empty vector and assign it to the argument.

Parameters

arg
: Variable of any type

Errors

Table 24.126. Errors signalled by xte_nodebld_init

SQLState	Error Code	Error Text	Description
22003	SR344	Too few arguments for xte_nodebld_init	

Examples

Example 24.493. Simple Use

After calling xte_nodebld_init() the vector acc of type any is empty and may be used by xte_nodebld_acc function

```
xte_nodebld_init(acc);
```

See Also

xte_head()

xte_node()

xte_node_from_nodebld()

xte_nodebld_acc()

xte_nodebld_final()

XMLAGG

XMLAGG — Produces a forest of elements from a collection of XML values

Synopsis

`vector` **`XMLAGG`** `(` *value_expression* any `);`

Description

`XMLAGG` is aggregate function that produces a forest of XML elements from the given list of xml elements. It concatenates the values returned from one column of multiple rows, unlike `XMLCONCAT` , which concatenates the values returned from multiple columns in the same row.

The order of element in the result of `XMLAGG` is defined by the order of retrieval of the source data rows. It is important to remember that the order of rows in an SQL resultset defined only if there's an explicit ORDER BY clause. Hence if the order of elements in the resulting forest is important then XMLAGG should be applied to data that comes from inner SELECT statement that has an ORDER BY clause, not e.g. from a table reference.

Note that `XMLAGG` is actually declared as `DB.DBA.XMLAGG` but it is not important for plain use: for compatibility with other systems, any call of `XMLAGG` in any SQl statement is always replaced with the call of `DB.DBA.XMLAGG` , no matter which qualifier and user name are in use.

Parameters

value_expression
: the result of one of the following functions XMLAGG , XMLCONCAT , XMLELEMENT , or XMLFOREST .

Return Types

The aggregate returns a vector that is a suitable input for functions `XMLELEMENT` , `XMLCONCAT` and `xml_tree_doc` .

Examples

Example 24.494. XMLAGG() enclosed in XMLELEMENT()

The following example produces an 'Emp' element with attribute 'Title' and a list of all employees having the title 'Sales Representative' as element content.

```
select XMLELEMENT ('Emp', XMLATTRIBUTES ('Sales Representative' as "Title"),
       XMLAGG (XMLELEMENT ('Name', "FirstName", ' ', "LastName")))
  from "Demo"."demo"."Employees"
 where "Title" = 'Sales Representative';

callret
VARCHAR
_______________________________________________________________________________

<Emp Title="Sales Representative">
  <Name>Nancy Davolio</Name>
  <Name>Janet Leverling</Name>
  <Name>Margaret Peacock</Name>
  <Name>Michael Suyama</Name>
  <Name>Robert King</Name>
  <Name>Anne Dodsworth</Name>
</Emp>

1 Rows.
```

Example 24.495. XMLAGG() that produces a sorted document

The result of the previous example contains an unsorted list of names. This is because data rows were retrieved from "Demo"."demo"."Employees" in primary key order, and this order has nothing common with fields "FirstName" and "LastName". To produce the sorted result, the query should contain ORDER BY in a subquery. These two variants will work identically if "FirstName" never contains whitespaces or nonprintable control characters, but the last one is formally more correct.

```
select XMLELEMENT ('Emp', XMLATTRIBUTES ('Sales Representative' as "Title"),
       XMLAGG (XMLELEMENT ('Name', "FirstName", ' ', "LastName")))
 from
    (select "FirstName", "LastName"
     from "Demo"."demo"."Employees"
     where "Title"= 'Sales Representative'
     order by 1, 2) as subq;

callret
VARCHAR
_______________________________________________________________________________

<Emp Title="Sales Representative">
  <Name>Anne Dodsworth</Name>
  <Name>Janet Leverling</Name>
  <Name>Margaret Peacock</Name>
  <Name>Michael Suyama</Name>
  <Name>Nancy Davolio</Name>
  <Name>Robert King</Name>
</Emp>

1 Rows.

select XMLELEMENT ('Emp', XMLATTRIBUTES ('Sales Representative' as "Title"),
       XMLAGG (XMLELEMENT ('Name', full_name)))
  from
    (select concat ("FirstName", ' ', "LastName") as full_name
     from "Demo"."demo"."Employees"
     where "Title"= 'Sales Representative'
     order by 1) as subq;

callret
VARCHAR
_______________________________________________________________________________

<Emp Title="Sales Representative">
  <Name>Anne Dodsworth</Name>
  <Name>Janet Leverling</Name>
  <Name>Margaret Peacock</Name>
  <Name>Michael Suyama</Name>
  <Name>Nancy Davolio</Name>
  <Name>Robert King</Name>
</Emp>

1 Rows.
```

See Also

XMLELEMENT()

XMLATTRIBUTES()

XMLFOREST()

XMLCONCAT()

XMLATTRIBUTES

XMLATTRIBUTES — Creates a list of attributes and their values

Synopsis

XMLATTRIBUTES (*string_expr1 [AS alias1]* varchar ,
string_expr2 [AS alias2] varchar ,
... ,
string_exprN [AS aliasN] varchar) ;

Description

This function creates a vector that may be used only as argument of XMLELEMENT function. The vector has an even number of elements, each odd element is a name of an attribute, an even element is its value. If the attribute value is NULL, then no attribute and no value is created. If none of the attribute is created, then the function returns NULL. If string_expr is a column name, then you can omit the AS clause, and Virtuoso uses the partially escaped form of the column name as the attribute name.

Parameters

String_exprI [AS aliasI]
: string value; AS clause is mandatory if it is not a column name but an expression of some other sort.

Examples

Example 24.496. XMLATTRIBUTES() with two parameters

The following example produces an 'EmpName' elements with two attributes (if value of the column 'Region' is not NULL) or with one attribute (if value of the column 'Region' is NULL)

```
select XMLELEMENT ("EmpName",
                   XMLATTRIBUTES ("FirstName" || ' ' || "LastName" as "Name", "Region" ))
  from "Demo"."demo"."Employees";
callret
VARCHAR
_______________________________________________________________________________
<EmpName Name="Nancy Davolio" Region="WA" />
<EmpName Name="Andrew Fuller" Region="WA" />
<EmpName Name="Janet Leverling" Region="WA" />
<EmpName Name="Margaret Peacock" Region="WA" />
<EmpName Name="Steven Buchanan" />
<EmpName Name="Michael Suyama" />
<EmpName Name="Robert King" />
<EmpName Name="Laura Callahan" Region="WA" />
<EmpName Name="Anne Dodsworth" />

9 Rows.
```

See Also

XMLELEMENT()

XMLFOREST()

XMLAGG()

XMLCONCAT()

XMLAddAttribute

XMLAddAttribute — Add an attribute to the given node of an XML tree document

Synopsis

```
integer XMLAddAttribute ( inout source any ,
                          in mode integer ,
                          in attr_name any ,
                          in attr_value any ) ;
```

Description

The function modifies the `source` document by adding an attribute to the current node of `source` entity. The `source` should be an XML tree entity, not "persistent XML" one, should be an element entity (not attribute, text etc.) and should not be a root entity. Parameters `attr_name` and `attr_value` can be of any types that can be casted to string. `attr_name` can be an attribute entity, in this case both the name and the value of `attr_name` attribute is used and `attr_value` code must be omitted.

The `mode` specifies how to resolve duplicate attribute names, if an `source` entity already has an attribute whose name is equal to `attr_name` . Mode 0 is similar to "insert into": the function signals an error if an attribute already exists. Mode 1 is similar to "insert soft": the function do nothing if an attribute already exists and not signalling an error. Mode 2 is similar to "insert replacing": the function either adds a new attribute or replacing the value of existing attribute.

Parameters

source
: The XML tree entity to change.

mode
: The mode of resolving duplicate attribute names.

attr_name
: The name of a new attribute (or an attribute XML entity that specifies both name and value of an attribute.

attr_value
: The value of a new attribute

Return Types

The function returns integer that indicate the actual operation performed. The zero value means that there was no change ("insert soft" when an attribute already exists). The value one means that a new attribute was added, the value two means that an existing attribute was replaced.

Examples

Example 24.497. Different modes of calling XMLAddAttribute

The sample procedure contains three calls of `XMLAddAttribute` with different values of `mode` , for "insert into", "insert soft" and "insert replacing" behaviour.

```
create procedure XMLAddAttribute_demo ()
{
  declare DESCRIPTION varchar (33);
  declare XMLENTITY, ent any;
  result_names (DESCRIPTION, XMLENTITY);
  result ('EXAMPLE 1', 'Adding a new attribute');
  ent := xpath_eval ('//b', xtree_doc('<a><b attrX="x" attrY="y"/></a>'));
  result ('The document to modify:', xpath_eval('/', ent));
```

```
  result ('An element node to change:', ent);
  XMLAddAttribute (ent, 0, 'attrZ', 'z');
  result ('The document after modification:', xpath_eval('/', ent));
  result ('EXAMPLE 2', 'Failed adding of an (existing) attribute');
  ent := xpath_eval ('//b', xtree_doc('<a><b attrX="x" attrY="y"/></a>'));
  result ('The document to modify:', xpath_eval('/', ent));
  result ('An element node to change:', ent);
  XMLAddAttribute (ent, 1, 'attrX', 'xNEW');
  result ('The document after modification:', xpath_eval('/', ent));
  result ('EXAMPLE 3', 'Successful replacing of an existing attribute');
  ent := xpath_eval ('//b', xtree_doc('<a><b attrX="x" attrY="y"/></a>'));
  result ('The document to modify:', xpath_eval('/', ent));
  result ('An element node to change:', ent);
  XMLAddAttribute (ent, 2, 'attrX', 'xNEW');
  result ('The document after modification:', xpath_eval('/', ent));
}
Done. -- 00000 msec.

XMLAddAttribute_demo()
DESCRIPTION                                 XMLENTITY
VARCHAR                                     VARCHAR
_______________________________________________________________________________

EXAMPLE 1                           Adding a new attribute
The document to modify:             <a><b attrX="x" attrY="y" /></a>
An element node to change:          <b attrX="x" attrY="y" />
The document after modification:    <a><b attrX="x" attrY="y" attrZ="z" /></a>
EXAMPLE 2                           Failed adding of an (existing) attribute
The document to modify:             <a><b attrX="x" attrY="y" /></a>
An element node to change:          <b attrX="x" attrY="y" />
The document after modification:    <a><b attrX="x" attrY="y" /></a>
EXAMPLE 3                           Successful replacing of an existing attribute
The document to modify:             <a><b attrX="x" attrY="y" /></a>
An element node to change:          <b attrX="x" attrY="y" />
The document after modification:    <a><b attrX="xNEW" attrY="y" /></a>

12 Rows. -- 00000 msec.
```

See Also

XMLReplace()

XMLAppendChildren()

XMLInsertBefore()

XMLInsertAfter()

XMLAppendChildren

XMLAppendChildren — Modify an XML document by adding new children to the given entity.

Synopsis

XMLAppendChildren (inout *source* any ,
in *insertion1* any ,
in *insertion2* any ,
... ,
in *insertionN* any) ;

Description

The function modifies the XML document of the given *source* XML tree entity by adding new children to the node specified by the entity. The *source* entity should be XML tree entity, not "persistent XML" entity. The value of *source* can be either an element entity or a root entity; *source* can not be an attribute entity or a leaf entity like text or processing instruction.

The values passed in parameters *insertion1* ... *insertionN* will be converted into XML nodes according to rules described in section Composing Document Fragments From DOM Function Arguments.

After calling the function, parameter *source* is still a valid XML entity that points to the modified node. The value passed as *source* can be used in the rest of caller procedure.

Parameters

source
: The XML tree entity whose document should be modified. This document should not be locked (see Changing XML Entities in DOM Style for details).

parameterI
: The value to be added as as child node of source .

Return Types

The function returns NULL.

Examples

Example 24.498. XMLAppendChildren in a Virtuoso/PL procedure

The sample procedure contains two calls of `XMLAppendChildren` . First call append three children to a node; second call demonstrates how text nodes can be merged.

```
create procedure XMLAppendChildren_demo()
{
  declare DESCRIPTION varchar (40);
  declare ENTITY, ent any;
  result_names (DESCRIPTION, ENTITY);
  result ('EXAMPLE1', 'Plain appending of some children');
  ent := xpath_eval ('//b', xtree_doc ('<a><b>Hello</b></a>'));
  result ('The document to modify', xpath_eval ('/', ent));
  result ('The node to be extended', ent);
  XMLAppendChildren (ent, xtree_doc ('<comma/>'), ' world', xtree_doc ('<excl/>'));
  result ('The changed document', xpath_eval ('/', ent));
  result ('The original entity is updated', ent);
  result ('EXAMPLE2', 'Appending that cause concatenation of text nodes');
  result_names (DESCRIPTION, ENTITY);
  ent := xpath_eval ('//b', xtree_doc ('<a><b>Hello</b></a>'));
  result ('The document to modify', xpath_eval ('/', ent));
```

```
  result ('The node to be extended', ent);
  XMLAppendChildren (ent, ', ', ' world', '!');
  result ('The changed document', xpath_eval ('/', ent));
  result ('The original entity is updated', ent);
}

Done. -- 00000 msec.

XMLAppendChildren_demo()
DESCRIPTION                              ENTITY
VARCHAR                                  VARCHAR
_______________________________________________________________________________

EXAMPLE1                                 Plain appending of some children
The document to modify                   <a><b>Hello</b></a>
The node to be extended                  <b>Hello</b>
The changed document                     <a><b>Hello<comma /> world<excl /></b></a>
The original entity is updated           <b>Hello<comma /> world<excl /></b>
EXAMPLE2                                 Appending that cause concatenation of text nodes
The document to modify                   <a><b>Hello</b></a>
The node to be extended                  <b>Hello</b>
The changed document                     <a><b>Hello,  world!</b></a>
The original entity is updated           <b>Hello,  world!</b>

10 Rows. -- 00000 msec.
```

See Also

XMLReplace

XMLAddAttribute

XMLInsertBefore

XMLInsertAfter

XMLCONCAT

XMLCONCAT — Creates a forest of elements by concatenating a list of XML values

Synopsis

XMLCONCAT (*value_expr1* varchar ,
value_expr2 varchar ,
... ,
value_exprN varchar) ;

Description

XMLCONCAT accepts a list of XML value expressions as its arguments, and produces a forest of elements by concatenating the XML values that are returned from the same row to make one value. XMLCONCAT works like XMLFOREST , except that XMLCONCAT parameters is a list of XML elements. Null expressions are dropped from the result. If all the value expressions are null, then the function returns NULL.

Parameters

value_expr1
: a vector returned by XMLELEMENT , XMLFOREST , or XMLCONCAT functions, or an entity object returned by corresponding functions (e.g. xtree_doc , xquery_eval or path_eval). In the latter case the entity object must not be an attribute.

Errors

Table 24.127. Errors signalled by XMLCONCAT

SQLState	Error Code	Error Text	Description
22003	SR355	Too few arguments for XMLCONCAT	There must be at least one argument
22003	SR359	Invalid argument type NVARCHAR (225) for arg N to XMLCONCAT	The entity object returned by xpath_eval must not be an attribute
22003	SR360	XMLCONCAT doesn't concatenate attributes	The entity object returned by xquery_eval must not be an attribute

Examples

Example 24.499. XMLCONCAT() with two parameters

The following example produces an XML elements for the 'FName', 'LName', 'country' and 'nationality', concatenates the result, and creates a one-column result set:

```
select XMLCONCAT (
        XMLELEMENT ('FName', "FirstName"),
        XMLELEMENT ('LName', "LastName"),
        xquery_eval('//country', xtree_doc('<a><country>USA</country></a>')),
        xpath_eval('//nationality', xtree_doc('<a><nationality>RUSSIAN</nationality></a>')))
from "Demo"."demo"."Employees";

callret
VARCHAR
_____________________________________________________________________________
<FName>Nancy</FName><LName>Davolio</LName><country>USA</country><nationality>RUSSIAN</nationality>
<FName>Andrew</FName><LName>Fuller</LName><country>USA</country><nationality>RUSSIAN</nationality>

. . .
```

```
<FName>Anne</FName><LName>Dodsworth</LName><country>USA</country><nationality>RUSSIAN</nationality>
9 Rows.
```

See Also

XMLAGG()

XMLATTRIBUTES()

XMLELEMENT()

XMLFOREST()

XMLELEMENT

XMLELEMENT — Creates XML element

Synopsis

XMLELEMENT (*tag_name* varchar ,
list_of_attributes sequence ,
child_or_attribute1 any ,
child_or_attribute2 any ,
... ,
child_or_attributeN any) ;

Description

XMLELEMENT takes an element name for identifier, an optional collection of attributes for the element, and arguments that make up the element's content. It returns a XML element. The second parameter may be omitted and at that time the rest parameters may be present. If one of the arguments is a call of the xpath_eval returning an attribute value, then this value would be added to element's content (not to element's attributes).

Parameters

tag_name
: name of the element, it must be valid XML element name

list_of_attributes
: a vector returned by XMLATTRIBUTES function. If the vector is NULL, then no attribute is created.

child_or_attributeI
: a string, or name of a column, or concatenation of the names and/or strings, or a vector returned by XMLELEMENT , XMLFOREST , XMLCONCAT , or XMLAGG functions, or an entity object returned by corresponding functions (e.g. xtree_doc or xquery_eval). If the entity object is an attribute entity, then it is joined to the list of the element's attributes. If a parameter is NULL, then no child element or attribute is created for that parameter.

Errors

Table 24.128. Errors signalled by XMLELEMENT

SQLState	Error Code	Error Text	Description
22003	SR354	Too few arguments for XMLELEMENT	There must be at least one argument

Examples

Example 24.500. XMLELEMENT() with a single argument

XMLELEMENT creates an 'Title' element without content.

```
SQL> select XMLELEMENT ('Title') from "Demo"."demo"."Employees";
callret
VARCHAR
_______________________________________________________________________________

<Title />
<Title />
. . .
9 Rows. -- 2 msec.
```

Example 24.501. XMLELEMENT() with content

The following example produces an 'Emp' element with three attributes (the 'region' attribute is calculated by `xquery_eval`) and five nested subelements

```
select XMLELEMENT ('Emp',
                    XMLATTRIBUTES ( "EmployeeID" AS "EmpID", "Title"),
                    XMLELEMENT ('Name', "FirstName" || ' ' || "LastName" ),
                    xquery_eval('//@region', xtree_doc ('<a region="WA"></a>')),
                    XMLFOREST ("PostalCode", "City" as "city"),
                    XMLCONCAT (XMLELEMENT ('HomePhone', "HomePhone"),
                               XMLELEMENT ('BirthDate', "BirthDate")))
  from "Demo"."demo"."Employees"
  where "EmployeeID"=1;

callret
VARCHAR
_______________________________________________________________________________

<Emp EmpID="1" Title="Sales Representative" region="WA">
  <Name>Nancy Davolio</Name>
  <city>Seattle</city>
  <PostalCode>98122</PostalCode>
  <HomePhone>(206) 555-9857</HomePhone>
  <BirthDate>1948-12-08</BirthDate>
</Emp>
```

Example 24.502. XMLELEMENT() with the aggregate function XMLAGG()

This example produces 'Emp' elements, with the list of the 'Name' of all employees.

```
select XMLELEMENT ('Emp',
                    XMLAGG (XMLELEMENT('Name', "FirstName", ' ', "LastName")))
  from "Demo"."demo"."Employees";
callret
VARCHAR
_______________________________________________________________________________

<Emp>
   <Name>Nancy Davolio</Name>
   <Name>Andrew Fuller</Name>
   <Name>Janet Leverling</Name>
   <Name>Margaret Peacock</Name>
   <Name>Steven Buchanan</Name>
   <Name>Michael Suyama</Name>
   <Name>Robert King</Name>
   <Name>Laura Callahan</Name>
   <Name>Anne Dodsworth</Name>
</Emp>
```

See Also

XMLAGG()

XMLATTRIBUTE()

XMLCONCAT()

XMLFOREST()

XMLFOREST

XMLFOREST — Produces a forest of XML elements

Synopsis

XMLFOREST (*string_expr1 [AS alias1]* varchar , *string_expr2 [AS alias2]* varchar , *...* , *string_exprN [AS aliasN]* varchar) ;

Description

XMLFOREST produces a forest of XML elements from the given list of arguments. The arguments may be string expressions with optional aliases. If string expression is a column name, then you can omit the AS clause, and Virtuoso uses the partially escaped form of the column name as the name of the enclosing tag. If the expression evaluates to NULL, then no element is created for that expression. If none of the element is created, then the function returns NULL.

Parameters

string_exprI [AS aliasI]
: string value; AS clause is mandatory if the argument is not a column name but an expression of some other sort.

Errors

Table 24.129. Errors signalled by XMLFOREST

SQLState	Error Code	Error Text	Description
37000	SQ074	The special SQL function has invalid argument	An argument is neither a column name nor an expression with an alias.

Examples

Example 24.503. XMLFOREST() with five parameters.

The following example produces five (or four) elements ('FName', 'LName', 'str', 'Title', and 'Region' - if there is a value) from the string expressions 'FirstName', 'LastName', 'string', 'Title', and 'Region', concatenates the elements produced for each employee, and produces one row for each employee in the result set.

```
select XMLFOREST (
  "FirstName" as "FName", "LastName" as "LName",
  'string' as "str", "Title", "Region" )
from "Demo"."demo"."Employees";
callret
VARCHAR
_____________________________________________________________________________

<FName>Nancy</FName>
<LName>Davolio</LName>
<str>string</str>
<Title>Sales Representative</Title>
<Region>WA</Region>

. . .
```

```
<FName>Anne</FName>
<LName>Dodsworth</LName>
<str>string</str>
<Title>Sales Representative</Title>

9 Rows.
```

The following example always produces five elements: empty 'Region' element is created for NULL values. Note the difference in the last rows of this and the previous query results:

```
select XMLFOREST (
  "FirstName" as "FName", "LastName" as "LName",
  'string' as "str", "Title", coalesce ("Region", '') as "Region")
from "Demo"."demo"."Employees";
callret
VARCHAR
_______________________________________________________________________________

<FName>Nancy</FName>
<LName>Davolio</LName>
<str>string</str>
<Title>Sales Representative</Title>
<Region>WA</Region>

. . .

<FName>Anne</FName>
<LName>Dodsworth</LName>
<str>string</str>
<Title>Sales Representative</Title>
<Region></Region>

9 Rows.
```

See Also

XMLELEMENT()

XMLATTRIBUTES()

XMLAGG()

XMLCONCAT()

XMLInsertAfter

XMLInsertAfter — Modify an XML document by inserting new children after the node specified by given entity.

Synopsis

XMLInsertAfter (inout *source* any ,
in *insertion1* any ,
in *insertion2* any ,
... ,
in *insertionN* any) ;

Description

The function modifies the XML document of the given `source` XML tree entity by adding new siblings to the node specified by the entity. Siblings will be added right after the node. The `source` entity should be XML tree entity, not "persistent XML" entity. The value of `source` should be a node entity; `source` can not be an attribute entity or a root entity.

The values passed in parameters `insertion1` ... `insertionN` will be converted into XML nodes according to rules described in section Composing Document Fragments From DOM Function Arguments.

After calling the function, parameter `source` is still a valid XML entity that points to the same node. The value passed as `source` can be used in the rest of caller procedure.

Parameters

source
: The XML tree entity whose document should be modified. This document should not be locked (see Changing XML Entities in DOM Style for details).

parameterI
: The value to be added as as child node of source .

Examples

Example 24.504. XMLInsertBefore in a Virtuoso/PL procedure

The sample procedure contains two calls of `XMLInsertAfter` . First call insert two new element nodes just after the given node; second call demonstrates how text nodes can be merged.

```
create procedure XMLInsertAfter_demo()
{
  declare DESCRIPTION varchar (40);
  declare ENTITY, ent any;
  result_names (DESCRIPTION, ENTITY);
  result ('EXAMPLE1', 'Plain inserting of some children');
  ent := xpath_eval ('//child0', xtree_doc ('<a><child0></child0></a>'));
  result ('The document to modify', xpath_eval ('/', ent));
  result ('The place of insertion', ent);
  XMLInsertAfter (ent, xtree_doc ('<child1/>'), xtree_doc ('<child2/>'));
  result ('The changed document', xpath_eval ('/', ent));
  result ('The original node is updated', ent);
  result ('EXAMPLE2', 'Appending that cause concatenation of text nodes');
  ent := xpath_eval ('//b/text()', xtree_doc ('<a><b>Hello</b></a>'));
  result ('The document to modify', xpath_eval ('/', ent));
  result ('The place of insertion', ent);
  XMLInsertAfter (ent, ', world!');
  result ('The changed document', xpath_eval ('/', ent));
  result ('The original node is updated', ent);
}
```

```
Done. -- 00000 msec.

XMLInsertAfter_demo()
DESCRIPTION                                 ENTITY
VARCHAR                                     VARCHAR
_______________________________________________________________________________

EXAMPLE1                                    Plain inserting of some children
The document to modify                      <a><child0 /></a>
The place of insertion                      <child0 />
The changed document                        <a><child0 /><child1 /><child2 /></a>
The original node is updated                <child0 />
EXAMPLE2                                    Appending that cause concatenation of text nodes
The document to modify                      <a><b>Hello</b></a>
The place of insertion                      Hello
The changed document                        <a><b>Hello, world!</b></a>
The original node is updated                Hello, world!

10 Rows. -- 00000 msec.
```

See Also

XMLReplace

XMLAddAttribute

XMLAppendChildren

XMLInsertBefore

XMLInsertBefore

XMLInsertBefore — Modify an XML document by inserting new children before the node specified by given entity.

Synopsis

XMLInsertBefore (inout *source* any ,
in *insertion1* any ,
in *insertion2* any ,
... ,
in *insertionN* any) ;

Description

The function modifies the XML document of the given *source* XML tree entity by adding new siblings to the node specified by the entity. Siblings will be added right before the node. The *source* entity should be XML tree entity, not "persistent XML" entity. The value of *source* should be a node entity; *source* can not be an attribute entity or a root entity.

The values passed in parameters *insertion1* ... *insertionN* will be converted into XML nodes according to rules described in section Composing Document Fragments From DOM Function Arguments.

After calling the function, parameter *source* is still a valid XML entity that points to the same node. The value passed as *source* can be used in the rest of caller procedure.

Parameters

source
: The XML tree entity whose document should be modified. This document should not be locked (see Changing XML Entities in DOM Style for details).

parameterI
: The value to be added as as child node of source .

Return Types

The function returns NULL.

Examples

Example 24.505. XMLInsertBefore in a Virtuoso/PL procedure

The sample procedure contains two calls of `XMLInsertBefore` . First call insert two new element nodes just before the given node; second call demonstrates how text nodes can be merged.

```
create procedure XMLInsertBefore_demo()
{
  declare DESCRIPTION varchar (40);
  declare ENTITY, ent any;
  result_names (DESCRIPTION, ENTITY);
  result ('EXAMPLE1', 'Plain inserting of some children');
  ent := xpath_eval ('//child0', xtree_doc ('<a><child0></child0></a>'));
  result ('The document to modify', xpath_eval ('/', ent));
  result ('The place of insertion', ent);
  XMLInsertBefore (ent, xtree_doc ('<child1/>'), xtree_doc ('<child2/>'));
  result ('The changed document', xpath_eval ('/', ent));
  result ('The original node is updated', ent);
  result ('EXAMPLE2', 'Insertion that cause concatenation of text nodes');
  ent := xpath_eval ('//b/text()', xtree_doc ('<a><b>world!</b></a>'));
  result ('The document to modify', xpath_eval ('/', ent));
  result ('The place of insertion', ent);
```

```
  XMLInsertBefore (ent, 'Hello, ');
  result ('The changed document', xpath_eval ('/', ent));
  result ('The original node is updated', ent);
}

Done. -- 00000 msec.

XMLInsertBefore_demo()
DESCRIPTION                              ENTITY
VARCHAR                                  VARCHAR
_______________________________________________________________________________

EXAMPLE1                                 Plain inserting of some children
The document to modify                   <a><child0 /></a>
The place of insertion                   <child0 />
The changed document                     <a><child1 /><child2 /><child0 /></a>
The original node is updated             <child0 />
EXAMPLE2                                 Insertion that cause concatenation of text nodes
The document to modify                   <a><b>world!</b></a>
The place of insertion                   world!
The changed document                     <a><b>Hello, world!</b></a>
The original node is updated             Hello, world!

10 Rows. -- 00000 msec.
```

See Also

XMLReplace

XMLAddAttribute

XMLAppendChildren

XMLInsertAfter

XMLReplace

XMLReplace — Modify a given XML document by replacing some nodes.

Synopsis

```
any XMLReplace ( inout source any ,
                 in location1 any ,
                 in replacement1 any ,
                 in location2 any ,
                 in replacement2 any ,
                 in ... ,
                 in locationN any ,
                 in replacementN any ) ;
```

Description

The function takes of the XML document referenced by `source` XML tree entity and modifies it by replacing nodes specified by `location1` , `location2` , ..., `locationN` with values specified by `replacement1` , `replacement2` , ...,`replacementN` . At the end of function call, the `source` points to the root of a modified entity.

The `source` parameter should be an XML tree entity whose document is not locked (see Changing XML Entities in DOM Style for details).

Every `replacementI` may be an XML tree entity, a NULL or a value of some other type that will be converted to varchar before use.

For simplicity, consider the case when there is only one `location1` XML entity and only one `replacement1` . If the `location1` is an XML entity that is not in the source document then no modifications is made in `source` . If `replacement1` is an entity that is in the source document then a temporary "cut" of `replacement1` is created, like `xml_cut` is called).

If the `location1` is an attribute entity then the value of the attribute is changed. If `replacement1` is NULL then the attribute is removed at all, otherwise its value is set to the string value of the `replacement1` .

If the `location1` is not an attribute entity but a entity then `location1` is replaced with the value of `replacement1` . The method of replacement depends on the type of the replacement value. If `replacement1` is NULL then the `location1` node is simply removed from the result; if the node is the root or a single child of the root then function immediately returns NULL (because XML document can not be totally empty), otherwise the parent of the `location1` will simply have one child less. If `replacement1` is an non-empty string then `location1` is replaced with a new text node whose string value is equal to `replacement1` . If `replacement1` is an empty string then the effect is exactly the same as in case of NULL because the string value of a text node can not be empty. If `replacement1` is an XML entity that is a root entity of some document then the highlighted node is replaced with a sequence of copies of all children of that root. If `replacement1` is an XML entity of some other sort (XML element, text, comment etc.) then the that is not a root entity of some document then the highlighted node is replaced with a sequence of copies of all children of that root.

After changing the `source` , some normalization may happen. The modified document may contain adjacent text nodes; every sequence of such nodes is replaced with one text node whose string value is a concatenation of string values of that nodes. E.g. if the `location1` is an element <a/> that resides between two text nodes "text1" and "text2" and it is removed (by replacing with NULL) then these text nodes become neighbors; normalization will replace them with a single text node "text1text2"

Now consider the case of more than one replacement made in one function call. If more than one pair of location and replacement is given then the function first finds all locations, then it performs all replacements, then it performs an normalization. It is very important to understand that the order of making replacements has nothing to do with the order of pairs of parameters in the function call. If some location node is an ancestor of some other location node then the effect of the replacement of the descendant location node is void: the descendant is replaced first; then the ancestor is replaced as a whole, no matter if some of its descendants are modified. The order of parameters in the function call is used to resolve ambiguity If two locations are equal: the

last pair of parameters will have a higher priority.

If the *source* is an instance of XMLType then its inner XML document is changed and "is validated" flag is reset to 0 indicating that there is no warranty that the modified version of document matches the declared schema even if the original document matched.

Parameters

source
: A source XML tree entity. It is an error to pass persistent XML entity as an argument.

locationI
: An XML entity that points to the node to replace. For compatibility with other implementations, it can be NULL, a pair of arguments is silently ignored in this case.

replacementI
: A replacement value for XML node pointed by locationI ; it may be NULL or XML tree entity or a string or anything else but not an persistent XML entity.

Return Types

The function returns NULL.

Examples

Example 24.506. XMLReplace in a Virtuoso/PL procedure

The sample procedure contains two calls of `XMLReplace` . First call replaces element 'c' with a copy of element 'replacement'. Second call demonstrates two replacements in parallel: elements titled 'c1' and 'c3' are replaced with text nodes.

```
create procedure XMLReplace_demo ()
{
  declare DESCRIPTION varchar(42);
  declare XMLENTITY, ent, ent2, ent_to_corrupt any;
  result_names (DESCRIPTION, XMLENTITY);
  ent := xtree_doc('<a><b><q/>-<c>Hello</c>-</b></a>');
  result ('EXAMPLE 1', 'A simple replacement of one element with other');
  result ('The document to modify:', ent);
  ent_to_corrupt := xquery_eval ('//c', ent);
  result ('Node to be replaced:', ent_to_corrupt);
  XMLReplace (ent, ent_to_corrupt, xtree_doc('<replacement/>'));
  result ('The document after modification:', ent);
  result ('Node to be replaced is outdated:', ent_to_corrupt);
  result ('...e.g. it'' not a descendant of its root:', xpath_eval('/', ent_to_corrupt));
  result ('EXAMPLE 2', 'Two replacements in parallel');
  ent2 := xtree_doc('<a><b><c1>Hello1</c1><c2>Hello2</c2><c3>Hello3</c3></b></a>');
  result ('The document to modify:', ent2);
  XMLReplace (ent2, xquery_eval('//c1', ent2), 'world1', xquery_eval('//c3', ent2), 'world3');
  result ('The document after modification:', ent2);
}

Done. -- 00000 msec.

XMLReplace_demo()
DESCRIPTION                                 XMLENTITY
VARCHAR                                     VARCHAR
_______________________________________________________________________________

EXAMPLE 1                                   A simple replacement of one element with other
The document to modify:                     <a><b><q />-<c>Hello</c>-</b></a>
Node to be replaced:                        <c>Hello</c>
The document after modification:           <a><b><q />-<replacement />-</b></a>
Node to be replaced is outdated:            <c>Hello</c>
...e.g. it' not a descendant of its root:  <a><b><q />-<replacement />-</b></a>
EXAMPLE 2                                   Two replacements in parallel
The document to modify:                     <a><b><c1>Hello1</c1><c2>Hello2</c2><c3>Hello3</c3></b></a>
The document after modification:            <a><b>world1<c2>Hello2</c2>world3</b></a>
```

```
9 Rows. -- 00000 msec.
```

See Also

XMLUpdate

XMLAddAttribute

XMLAppendChildren

XMLInsertBefore

XMLInsertAfter

xslt

xml_auto

xml_auto — prepares and executes given SQL for XML string output

Synopsis

xml_auto (in *sql_text* varchar ,
in *params* any ,
in *string_output* any) ;

Description

This function prepares and executes the given SQL string, which should be a query expression with the FOR XML clause at the end of the last term. The query is passed the parameters from the params vector, which should have one element for each ? in the query text, values assigned from left to right. Consider the query: select a, b from table where a = ? and b = ?; then the params vector could reasonably be:

```
vector(1, 'myfilter')
```

.

The result set is converted to XML and appended to the *string* _output. If the *string* _output is omitted and the function executes in the context of a VSP page, the output is sent to the stream going to the user agent.

Parameters

sql_text
: Valid SQL query using the FOR XML clause. Parameterized queries can be constructed using the question mark (?) to specify a parameter place-holder that will be replaced at run time with the appropriate value from the params vector.

params
: Vector of parameters, one element per ? used in the query.

string_output
: String variable or stream for receiving the result.

Return Types

If you omit the third parameter, this function will output to the context of the calling VSP page.

Errors

Table 24.130. Errors signalled by xml_auto

SQLState	Error Code	Error Text	Description
42000		http output function outside of http context and no stream specified	From an attempt to send the output directly to a non HTTP target such as ISQL.
42000		Column 1 of the result set of the select statement should be of INTEGER type when FOR XML EXPLICIT clause is used	
42000		Column 2 of the result set of the select statement should be of INTEGER type when FOR XML EXPLICIT clause is used	

Examples

Example 24.507. Producing XML from SQL

The procedure below takes an SQL string, evaluates it - converting to XML - and produces a result set where the XML text is returned as a varchar column. Parameters are not passed in this example for the sake of simplicity.

```
create procedure xmla (in q varchar)
{
  declare st any;
  st := string_output ();
  xml_auto (q, vector (), st);
  result_names (q);
  result (string_output_string (st));
}
```

See Also

`xml_auto_schema()`, `xml_auto_dtd()`

xml_auto_dtd

xml_auto_dtd — returns an XML DTD for the result of a SQL query with a FOR XML clause

Synopsis

```
varchar xml_auto_dtd ( in query varchar ,
                       in root_element varchar ) ;
```

Description

This function returns an XML DTD for the results of a SQL query with a FOR XML clause. The returned DTD will apply to the output generated by xml_auto with the query in question after wrapping it into the specified root element.

Parameters

query
: valid SQL query

root_element
: name of root element to wrap result into

Return Types

varchar of the resultant DTD

Errors

If the *query* argument is not a valid SQL statement, i.e. SQL compiler signals an error message, the function resignals the error.

Examples

Example 24.508. Simple Use

```
SQL> select xml_auto_dtd ('  select "category"."CategoryID", "CategoryName",
    "ProductName", "ProductID"
    from "Demo".."Categories" "category", "Demo".."Products" as "product"
    where "product"."CategoryID" = "category"."CategoryID" for xml auto element', 'root');
callret
VARCHAR
_______________________________________________________________________________

<!-- dtd for output of the following SQL statement:
  select "category"."CategoryID", "CategoryName",
                "ProductName", "ProductID"
        from "Demo".."Categories" "category",

                "Demo".."Products" as "product"
        where "product"."CategoryID" = "category"."CategoryID"
        for xml auto element
-->

<!ELEMENT root (#PCDATA | category)* >
<!ELEMENT category (#PCDATA | CategoryID | CategoryName | product)* >
<!ELEMENT product (#PCDATA | ProductName | ProductID)* >
<!ATTLIST category      >
<!ATTLIST product       >
<!ELEMENT CategoryID (#PCDATA)>
<!ATTRLIST CategoryID   >
<!ELEMENT CategoryName (#PCDATA)>
<!ATTRLIST CategoryName >
<!ELEMENT ProductName (#PCDATA)>
<!ATTRLIST ProductName  >
```

```
<!ELEMENT ProductID (#PCDATA)>
<!ATTRLIST ProductID    >

1 Rows. -- 4 msec.
```

See Also

xml_auto()

xml_auto_schema()

xml_auto_schema

xml_auto_schema — returns an XML schema for the result of an SQL query with a FOR XML clause

Synopsis

```
varchar xml_auto_schema ( in query varchar ,
                          in root_element varchar ) ;
```

Description

This function returns an XML schema for the results of an SQL query with a FOR XML clause. The returned schema will apply to the output generated by `xml_auto()` with the query in question after wrapping it in the specified root element.

Parameters

query
: SQL query

root_element
: name of root element container

Return Types

varchar result containing the schema of the XML document.

Errors

If the *query* argument is not a valid SQL statement, i.e. SQL compiler signals an error message, the function resignals the error.

Examples

Example 24.509. Simple Schema Generation

```
SQL> select xml_auto_schema ('  select "category"."CategoryID", "CategoryName",
                  "ProductName", "ProductID"
          from "Demo".."Categories" "category", "Demo".."Products" as "product"
          where "product"."CategoryID" = "category"."CategoryID"
          for xml auto element', 'root');
 callret
VARCHAR
_______________________________________________________________________________

<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema">

 <xsd:annotation>
  <xsd:documentation>
   Schema for output of the following SQL statement:
   <[CDATA[  select "category"."CategoryID",
     "CategoryName", "ProductName", "ProductID"
     from "Demo".."Categories" "category",
     "Demo".."Products" as "product"
     where "product"."CategoryID" = "category"."CategoryID"
     for xml auto element]]>
  </xsd:documentation>
 </xsd:annotation>

 <xsd:element name="root" type="root__Type"/>

 <xsd:complexType name="root__Type">
  <xsd:element name="category" type="category_Type"
  minOccurs="0" maxOccurs="unbounded"/>
 </xsd:complexType>
```

```
 <xsd:complexType name="category_Type">
  <xsd:element name="CategoryID" type="xsd:int" precision="10"/>
  <xsd:element name="CategoryName" type="xsd:string" maxLength="15"/>
  <xsd:element name="product" type="product_Type" minOccurs="0" maxOccurs="unbounded"/>
 </xsd:complexType>

 <xsd:complexType name="product_Type">
  <xsd:element name="ProductName" type="xsd:string" maxLength="40"/>
  <xsd:element name="ProductID" type="xsd:int" precision="10"/>
 </xsd:complexType>

</xsd:schema>

1 Rows. -- 5 msec.
```

See Also

xml_auto()

xml_auto_dtd()

xml_create_tables_from_mapping_schema_decl

xml_create_tables_from_mapping_schema_decl — returns a vector containing strings. Each string is a command to drop a table or a foreign key or to create table.

Synopsis

xml_create_tables_from_mapping_schema_decl (in *base_uri* varchar ,
in *mapping_schema_file* varchar ,
in *content_encoding* varchar ,
in *content_language* varchar) ;

Description

`xml_create_tables_from_mapping_schema_decl` takes a file containing mapping schema and returns a vector containing strings. Each string is a command to drop a table or a foreign key or to create table. All tables and fields are mentioned in the mapping schema. If a field type is not defined in the mapping schema, the VARCHAR type is used.

Parameters

base_uri
: in HTML parser mode change all absolute references to relative from given base_uri (http://<host>:<port>/<path>)

mapping_schema_file
: well formed XML document containing a mapping schema

content_encoding
: string with content encoding type of <document>; valid are 'ASCII', 'ISO', 'UTF8', 'ISO8859-1', 'LATIN-1' etc., defaults are 'UTF-8' for XML mode and 'LATIN-1' for HTML mode

content_language
: string with language tag of content of <document>; valid names are listed in IETF RFC 1766, default is 'x-any' (it means 'mix of words from various human languages)

Return Types

A vector containing strings

Example 24.510. Extracting tables from mapping schema

Let we have the following mapping schema in the catmp.xsd file

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:sql="urn:schemas-microsoft-com:mapping-schema">
<xsd:annotation>
  <xsd:appinfo>
    <sql:relationship name="CategoryProduct"
          parent="Demo.demo.Categories"
          parent-key="CategoryID"
          child="Demo.demo.Products"
          child-key="CategoryID" />
  </xsd:appinfo>
</xsd:annotation>

  <xsd:element name="category" sql:relation="Demo.demo.Categories" type="CategoryType" />
   <xsd:complexType name="CategoryType" >
     <xsd:sequence>
        <xsd:element name="product"
                    sql:relation="Demo.demo.Products"
                    sql:relationship="CategoryProduct" >
```

```
            <xsd:complexType>
               <xsd:attribute name="ProductName" type="xsd:string" />
            </xsd:complexType>
         </xsd:element>
      </xsd:sequence>
         <xsd:attribute name="CategoryID"  type="xsd:integer" />
         <xsd:attribute name="description"  sql:field="Description"  type="xsd:string" />
    </xsd:complexType>
</xsd:schema>
```

the _result vector after the call

```
_result := xml_create_tables_from_mapping_schema_decl (
  'http://localhost.localdomain/xmlrepository',
  'catmp.xsd', 'UTF-8', 'x-any');
```

will contain the following six strings

```
drop table "Demo"."demo"."Categories",

ALTER TABLE "Demo"."demo"."Products" DROP CONSTRAINT "Demo.demo.Products_Demo.demo.Categories_FK",

drop table "Demo"."demo"."Products",

create table "Demo"."demo"."Categories"
("Description"  VARCHAR, "CategoryID" INTEGER, PRIMARY KEY ("CategoryID")),

create table "Demo"."demo"."Products" ("CategoryID"  INTEGER, "ProductName"  VARCHAR),

ALTER TABLE "Demo"."demo"."Products" ADD CONSTRAINT "Demo.demo.Products_Demo.demo.Categories_FK"
FOREIGN KEY ("CategoryID") REFERENCES "Demo"."demo"."Categories" ("CategoryID")
```

See Also

`xml_load_mapping_schema_decl()`

`xml_load_schema_decl()`

`xml_view_schema()`

xml_cut

xml_cut — creates a new XML document which contains a copy of data pointed by given XML tree- or XPER- entity

Synopsis

xml_cut (in *source_entity* any (XML entity)) ;

Description

In some special cases, some part of XML document, being pointed by a XML entity, should be copied into a new separate document with new entity pointing to the top-level element or the root of this document. One reason for doing this is optimization of XPER processing (see xper_cut). Another way to use this functionality is passing of some XML entity to a function, when function uses XPath operations with references to the "document's root".

Example 24.511. A sample of hidden bug

```
create procedure get_C (inout b any)
{
  return cast (xpath_eval('//C', b) as varchar);
}

create procedure hidden_bug_1()
{
  declare abc any;      -- some XML document
  declare b_list any;   -- a vector of all B elements of the doc
  declare c1 varchar;   -- a name of first variable
  declare c2 varchar;   -- a title of second variable
  abc := xtree_doc ('<A><B
xml:id="1_01"><C>One</C></B><B
xml:id="2_01"><C>Two</C></B></A>');
  b_list := xpath_eval ('//B', abc, 0);
  -- now b_list is a vector of two items, '<B
xml:id="1_02"><C>One</C></B>' and '<B
xml:id="2_02"><C>Two</C></B>'.
  c1 := get_C ( aref (b_list, 0));
  -- looks fine, c1 is '<C>One</C>'
  c2 := get_C ( aref (b_list, 1));
  -- looks strange fine, c2 is '<C>One</C>',
  -- while the expected value is '<C>Two</C>'.
}
```

The origin of the bug is '//C' path in get_C(), which returns not the "C element inside given b element", but "C element inside the document where given b element is located", thus get_C returns the first C element in the whole document with any of two B elements given.

There are two ways to fix this bug. It is better to correct get_C():

```
create procedure get_C (inout b any)
{
  return cast (xpath_eval('./C', b) as varchar);
}
```

If you cannot patch get_C() for some reason, xml_cut will help, but it will waste both memory and CPU time for copying a branch of XML tree:

```
create procedure hidden_bug_1()
{
...
  c1 := get_C ( xml_cut (aref (b_list, 0)));
...
}
```

The current node of the resulting entity is the node that is a copy of the current node of the source entity. In common, the top-level node of the copied subtree becomes the current node of the result. There are two special cases, however. If the source entity is an attribute entity, then the result is also an attribute entity and the attribute name remains the same. If the source entity points to the root of the document, the resulting entity also points to the root of the copied document, not to its top-level node.

With XPER entity given, xml_cut() works exactly as xper_cut().

Parameters

source_xper
: XML Entity to be converted into new document

See Also

`xper_cut()`

xml_doc_output_option

xml_doc_output_option

Synopsis

```
varchar xml_doc_output_option ( in xml_entity any ,
                                in option_name varchar ,
                                in option_value varchar ) ;
```

Description

This function reads or updates the specified XSLT output option value of the given `xml_entity` .

The function updates the option if parameter `option_value` is provided, otherwise it returns the current value of the option without any side effects.

Supported options are "method", "version", "encoding", "omit-xml-declaration", "standalone", "doctype-public", "doctype-system", "indent" and "media-type", but do not support"cdata-section-elements". When the entity is serialized, the effect is very similar to the effect of the same option specified in xsl:output element of an XSLT that created the entity.

Note that output options are properties of the whole document, not properties of some particular node in the document, so the effect of the update is document-wide. Please also note that the effect of 'preamble' options "omit-xml-declaration", "standalone", "doctype-public" and "doctype-system" is visible only when the root entity of the document is serialized, not any descendant entity. Even the serialization of top-level element of the document is not affected by these settings. The XML preamble is serialized only if the document is composed by an XSLT processor or "doctype-system" is set or at least one of two boolean properties "omit-xml-declaration" and "standalone" is set to "yes".

According to the DTD rules, "doctype-public" has no effect on serialization if "doctype-system" is not set.

Parameters

xml_entity
: An XML entity such as that returned by the xslt() function.

option_name
: A name of output option to in question.

option_value
: A new value of output option. This is a string or NULL (that reset option to default). Allowed values of boolean properties "omit-xml-declaration" and "standalone" are "yes" and "no", as it is in XSLT 1.0.

Return Types

The function returns a string that is a value of the option or NULL if the option is not set.

Errors

This function can generate the following errors:

22023 SR003 Function xml_doc_output_option needs an XML entity as argument 1, not an arg of type <type_name> (<type>)

Examples

Example 24.512. Assessing the SYSTEM DTD location

```
create function test_output_option (
  in cnt any, in opt_name any, in opt_value any)
```

```
{
  declare xt, ses any;
  declare oldval, newval varchar;
  xt := xtree_doc (cnt, 2, '', 'UTF-8');
  oldval := xml_doc_output_option (xt, opt_name);
  xml_doc_output_option (xt, opt_name, opt_value);
  newval := xml_doc_output_option (xt, opt_name);
  return concat (
    sprintf ('Old value of "%s" is "%s"\nNew value is "%s"\nThe result:\n',
       opt_name,
       cast (oldval as varchar),
       cast (newval as varchar) ),
    serialize_to_UTF8_xml (xt) );
}

select test_output_option (
  '<div xmlns="http://www.w3.org/1999/xhtml">Hello</div>',
  'doctype-system', 'http://www.example.com/xhtml.dtd')
callret
VARCHAR
_______________________________________________________________________________

Old value of "doctype-system" is "(NULL)"
New value is "http://www.example.com/xhtml.dtd"
The result:
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE http://www.w3.org/1999/xhtml:div
  SYSTEM "http://www.example.com/xhtml.dtd">
<n0:div xmlns:n0="http://www.w3.org/1999/xhtml">Hello</n0:div>

1 Rows. -- 1 msec.
```

See Also

`xtree_doc()`

xml_load_schema_decl

xml_load_schema_decl — returns a string with list of errors detected by XML Schema processor on reading given XML Schema definition document.

Synopsis

xml_load_schema_decl (in *base_uri* varchar ,
in *document_uri* varchar ,
in *content_encoding* varchar ,
in *content_language* varchar) ;

Description

Loads given file using base_uri. Try to parse file of XML Schema definition and check it for correctness to XML Schema specification.

Parameters

base_uri
: in HTML parser mode change all absolute references to relative from given base_uri (http://<host>:<port>/<path>)

content_encoding
: string with content encoding type of <document>; valid are 'ASCII', 'ISO', 'UTF8', 'ISO8859-1', 'LATIN-1' etc., defaults are 'UTF-8' for XML mode and 'LATIN-1' for HTML mode

content_language
: string with language tag of content of <document>; valid names are listed in IETF RFC 1766, default is 'x-any' (it means 'mix of words from various human languages)

dtd_validator_config
: configuration string of the validator, default is empty string meaning that DTD validator should be fully disabled and only critical errors should be reported

Return Types

Human readable list of errors if applicable as a varchar.

Validating XML Against a XML Schema

Example 24.513. Simple Use

```
declare _result varchar;
_result := xml_load_schema_decl (
  'http://localhost.localdomain/xmlrepository',
  'persons.xsd', 'UTF-8', 'x-any');

if (_result = '') _result := 'NO ERRORS DETECTED';
```

See Also

`xml_validate_schema()`

`xml_auto_schema()`

`xml_view_schema()`

xml_validate_dtd()

xml_load_mapping_schema_decl

xml_load_mapping_schema_decl — creates a xml view from mapping schema.

Synopsis

xml_load_mapping_schema_decl (in *base_uri* varchar ,
in *mapping_schema_file* varchar ,
in *content_encoding* varchar ,
in *content_language* varchar) ;

Description

`xml_load_mapping_schema_decl` takes a file containing mapping schema and creates a xml view.

Parameters

base_uri
: in HTML parser mode change all absolute references to relative from given base_uri (http://<host>:<port>/<path>)

mapping_schema_file
: well formed XML document containing a mapping schema

content_encoding
: string with content encoding type of <document>; valid are 'ASCII', 'ISO', 'UTF8', 'ISO8859-1', 'LATIN-1' etc., defaults are 'UTF-8' for XML mode and 'LATIN-1' for HTML mode

content_language
: string with language tag of content of <document>; valid names are listed in IETF RFC 1766, default is 'x-any' (it means 'mix of words from various human languages)

Example 24.514. Creating a xml view from a mapping schema

The xml view created by

```
xml_load_mapping_schema_decl (
   'http://localhost.localdomain/xmlrepository',
   'catmp.xsd', 'UTF-8', 'x-any');
```

from the file catmp.xsd containing the following mapping schema

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:sql="urn:schemas-microsoft-com:mapping-schema">
<xsd:annotation>
  <xsd:appinfo>
    <sql:relationship name="CategoryProduct"
          parent="Demo.demo.Categories"
          parent-key="CategoryID"
          child="Demo.demo.Products"
          child-key="CategoryID" />
  </xsd:appinfo>
</xsd:annotation>

  <xsd:element name="category" sql:relation="Demo.demo.Categories" type="CategoryType" />
   <xsd:complexType name="CategoryType" >
     <xsd:sequence>
        <xsd:element name="product"
                    sql:relation="Demo.demo.Products"
                    sql:relationship="CategoryProduct" >
           <xsd:complexType>
              <xsd:attribute name="ProductName" type="xsd:string" />
```

```
            </xsd:complexType>
          </xsd:element>
       </xsd:sequence>
          <xsd:attribute name="CategoryID"  type="xsd:integer" />
          <xsd:attribute name="description"  sql:field="Description"  type="xsd:string" />
      </xsd:complexType>
</xsd:schema>
```

is identical to xml view created by

```
create xml view "cat" as
{
  "Demo"."demo"."Categories" "C" as "category"
        ("CategoryID", "Description" as "description")
    {
      "Demo"."demo"."Products" "P" as "product"  ("ProductName")
        on ("P"."CategoryID" = "C"."CategoryID")
    }
}
```

See Also

`xml_load_schema_decl()`

`xml_load_schema_decl()`

`xml_view_schema()`

xml_namespace_scope

xml_namespace_scope — Returns a vector of all namespace declarations in all ancestors of the given XML entity.

Synopsis

```
vector xml_namespace_scope ( in ent XML Entity ,
                             in use_default_ns integer ) ;
```

Description

The function returns a vector of even length that consists of all declared namespace prefixes and namespace URIs from the `ent` and all its ancestors. This information is needed for processing XML documents that contains a mix of data and XPath expressions, such as BPEL documents.

Parameters

ent
: The entity to process.

use_default_ns
: Flags if the resulting array should contain declarations of default namespace. If it is zero then only declarations of namespace prefixes are listed; if non-zero then all declarations are listed.

Return Types

The function returns a vector of even length that contains narrow strings in UTF-8 encoding.

Examples

Example 24.515. Adding namespace declarations to the XPath expression

The function gets an entity whose string-value is an XPATH expression and returns the text of expression with all namespace declarations that are in scope. The resulting expression is context-independent. This is useful for BPEL-like applications and for extracting XPATH expressions from XML Schema documents.

```
create procedure xpath_add_namespace_scope (in ent any, in use_default_ns integer)
{
  declare _expn varchar;
  declare _ses any;
  declare _scope any;
  declare _ctr any;
  _expn := charset_recode (xpath_eval ('string(.)', ent), '_WIDE_', 'UTF-8');
  _scope := xml_namespace_scope (ent, use_default_ns);
  _ctr := length (_scope);
  if (_ctr = 0)
    return _expn;
  _ses := string_output ();
  http ('[', _ses);
  while (_ctr > 0)
    {
      if (_scope[_ctr-2] = '')
        http (sprintf (' xmlns="%s"', _scope[_ctr-1]), _ses);
      else
        http (sprintf (' xmlns:%s="%s"', _scope[_ctr-2], _scope[_ctr-1]), _ses);
      _ctr := _ctr - 2;
    }
  http (' ] ', _ses);
  http (_expn, _ses);
  return string_output_string (_ses);
}

select xpath_add_namespace_scope (
```

```
  xquery_eval (
'declare namespace xsd="http://www.w3.org/2001/XMLSchema";
//xsd:keyref[@name="ISBNnumber"]/xsd:field/@xpath',
    xtree_doc (
'<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.publishing.org"
        xmlns="http://www.publishing.org">
  <xsd:element name="Book"  minOccurs="1" maxOccurs="unbounded">
    <xsd:keyref name="ISBNnumber" refer="BookDB_ISBN">
      <xsd:selector xpath="."/>
      <xsd:field xpath="ISBN"/>
    </xsd:keyref>
  </xsd:element>
  <!-- The rest of the XML Schema file is skipped -->

</xsd:schema>')),
  1 );

callret
VARCHAR
_______________________________________________________________________________

[ xmlns="http://www.publishing.org" xmlns:xsd="http://www.w3.org/2001/XMLSchema" ] ISBN
```

See Also

xpath_eval

xml_add_system_path

xml_add_system_path — Adds path to the internal list of system paths.

Synopsis

xml_add_system_path (in *path* varchar) ;

Description

When validating XML parser tries to resolve system entities it searches in http_root directory. If it fails parser iterates internal list of system paths and tries to find required files there. The function adds new path in this list. List of system paths contains one directory item by default - "file://system/".

NOTE: List of system paths is not persistent. It means that you must add desired path each time when server starts. An ideal place for this operation in "autoexec.isql" file.

xml_get_system_paths

xml_get_system_paths — Returns vector of all system paths.

Synopsis

```
xml_get_system_paths (  );
```

xml_persistent

xml_persistent — returns an entity object ('XPER entity') created from given XML document

Synopsis

xml_persistent (in *document* varchar ,
in *parser_mode* integer ,
in *base_uri* varchar ,
in *content_encoding* varchar ,
in *content_language* varchar ,
in *dtd_validator_config* varchar) ;

Description

This parses the argument, which is expected to be a well formed XML fragment and returns a parse tree as a special object with underlying disk structure, named "persistent XML" or "XPER" While the result of `xml_tree()` is a memory-resident array of vectors, the XPER object consumes only a little amount of memory, and almost all data is disk-resident.

This function is equivalent to `xper_doc()` , and the only difference is in the order of arguments; xper_doc() has the same order of arguments as `xml_tree()` .

Parameters

document
: well formed XML or HTML documen

parser_mode
: 0, 1 or 2; 0 - XML parser mode, 1 - HTML parser mode, 2 - 'dirty HTML' mode (with quiet recovery after any syntax error)

base_uri
: in HTML parser mode change all absolute references to relative from given base_uri (http://<host>:<port>/<path>)

content_encoding
: string with content encoding type of <document>; valid are 'ASCII', 'ISO', 'UTF8', 'ISO8859-1', 'LATIN-1' etc., defaults are 'UTF-8' for XML mode and 'LATIN-1' for HTML mode

content_language
: string with language tag of content of <document>; valid names are listed in IETF RFC 1766, default is 'x-any' (it means 'mix of words from various human languages)

dtd_validator_config
: configuration string for the DTD validator, default is empty string meaning that DTD validator should be fully disabled. See Configuration Options of the DTD Validator for details.

Return Types

Parse tree as a structure of nested heterogeneous vectors.

Examples

Example 24.516. XML_Persistent

```
declare tree any;

tree := xml_persistent (file_to_string ('doc.html'), 1,
                'http://localhost.localdomain/', 'ISO');
```

```
...
tree := xml_persistent (file_to_string ('doc.xml'));
```

See Also

xml_tree , xper_doc , xml_view_dtd , xml_validate_dtd

xml_template

xml_template — Execute XML template from Virtuoso PL

Synopsis

xml_template (in *xml_document_entity* any ,
in *parameters* vector ,
inout *session* string_session) ;

Description

The `xml_template()` function was introduced to enable PL programming to make use of XML templates. This function expects an XML entity for the first argument, that will be the XML template file contents. Usually this entity is composed making use of the `xtree_doc()` function from the XML template file.

The second argument should be a vector of name-value pairs of the parameters for XML template.

The last argument is an output string stream for the result. If the output stream is not specified the HTTP internal stream will be used if it is available, otherwise an error will be signalled.

Parameters

xml_document_entity
: XML entity such as that returned by xtree_doc()

parameters
: Vector or name-value pairs: vector('name1', 'value1', 'name2', 'value2', ...).

session
: Output stream for handling the results of XML template. If this parameter is unspecified then the HTTP internal stream will be assumed, which if not applicable will signal an error.

Return Types

A URL of the XSLT stylesheet for further transformation will be returned. If xsl attribute was not specified in the XML template then the return value will be NULL.

xml_tree

xml_tree — Parses an XML fragment and returns the parse tree as nested vectors.

Synopsis

xml_tree (in *document* varchar ,
in *parser_mode* integer ,
in *base_uri* varchar ,
in *content_encoding* varchar ,
in *content_language* varchar ,
in *dtd_validator_config* varchar) ;

Description

This parses the argument, which is expected to be a well formed XML fragment and returns a parse tree as a structure of nested heterogeneous vectors.

Parameters

document
: (mandatory) A well formed XML or HTML document

parser_mode
: 0, 1 or 2; 0 - XML parser mode, 1 - HTML parser mode, 2 - 'dirty HTML' mode (with quiet recovery after any syntax error)

base_uri
: (optional) in HTML parser mode change all absolute references to relative from given base_uri (http://<host>:<port>/<path>)

content_encoding
: (optional) string with content encoding type of <document> valid is 'ASCII', 'ISO', 'UTF8', 'ISO8859-1', 'LATIN-1'.

content_language
: (optional) - string with language tag of content of <document>; valid names are listed in IETF RFC 1766, default is 'x-any' (it means 'mix of words from various human languages).

dtd_validator_config
: configuration string for DTD validator, default is empty string meaning that DTD validator should be fully disabled. See Configuration Options of the DTD Validator for details.

Return Types

vector of vectors representing the parsed tree of XML.

Examples

Example 24.517. Making an XML tree

```
declare tree any;

tree := xml_tree (file_to_string ('doc.html'), 1,
                'http://localhost.localdomain/', 'ISO');
...
tree := xml_tree (file_to_string ('doc.xml'));
```

See Also

`xslt`, `xml_uri_get`, `xml_validate_dtd`.

xml_tree_doc

xml_tree_doc — returns an entity object given a tree from xml_tree

Synopsis

xml_tree_doc (in *tree* any ,
in *base_uri* varchar) ;

Description

This returns an entity object given a tree of the form returned by xml_tree.

If it is given a string as an argument, it will automatically generate the parse tree and use it to make the entity instead requiring you to run the string through xml_tree first. Note that it is better to use `xtree_doc` or `xper_doc` for converting source XML text directly to an XML entity.

If the argument is an XML tree entity, the function will return it as is, so e.g. redundant calls of `xml_tree_doc` will have no effect. The only thing xml_tree_doc can alter in the returned value is base URI of the document entity: if *`base_uri`* is provided and is not NULL, and argument entity has no base URI set then the provided URI is assigned to the returned entity.

Any other type of argument is illegal, including XML persistent entity.

Parameters

tree
: either an XML tree such as that returned by xml_tree(), or a string of XML data. If a string is provided then it will automatically generate the parse tree and form an entity instead of requiring you to run the string through xml_tree() first.

base_uri
: Base URI of the original document, if known. It will be useful if the document is not "standalone" and some entity references are relative references to resources located "somewhere near" the "top-level" document passes as "tree" parameter.

Return Types

XML entity object

Errors

Table 24.131. Errors signalled by

SQL State	Error Code	Error Text	Description

Examples

Example 24.518. Transforming an XML Document

```
declare doc_base varchar;
declare doc_tree any;

doc_base := 'virt://WS.WS.SYS_DAV_RES.RES_FULL_PATH.RES_CONTENT:/DAV/docsrc/';
doc_tree := xml_tree_doc(xml_tree(
    xml_uri_get(doc_base, 'virtdocs.xml')
      ), doc_base);

http_value(
  xslt('virt://WS.WS.SYS_DAV_RES.RES_FULL_PATH.RES_CONTENT:/DAV/stylesheets/html_chapter.xsl',
```

```
            doc_tree
          )
        );
```

See Also

xtree_doc

xper_doc

xml_tree

xslt

xml_uri_get

xml_tree_doc_media_type

xml_tree_doc_media_type

Synopsis

varchar **xml_tree_doc_media_type** (in *xml_entity* any);

Description

This function returns the media type in effect for the result of the xslt() (XSL-T) transformation , based on xsl:output "media-type" and "method" attributes of the XSL-T style-sheet applied. It accepts an entity (potentially resulting from using xslt()) as a argument and will return a string containing the media-type.

Parameters

xml_entity
: An XML entity such as that returned by the xslt() function.

Return Types

If xsl:output "media-type" specified in XSL-T sheet it's value will be returned. If no "media-type" is specified, the valid results dependent on the <xsl:output "method" attribute are:

if unknown : non-string value

if 'html' : 'text/html'

if 'xhtml' : 'text/html'

if 'xml' : 'text/xml'

if 'text' : 'text/plain'.

Errors

This function can generate the following errors:

22023 SR003 Function xml_tree_doc_media_type needs an XML entity as argument 1, not an arg of type <type_name> (<type>)

Examples

Example 24.519. Assessing the media type

```
-- x.xsl --

<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
    <xsl:output method="text" />
....
</xsl:stylesheet>

-- execute a simple transformation via ISQL tool
SQL> select xml_tree_doc_media_type (xslt ('file:/x.xsl',xml_tree_doc ('<A />')));
Connected to OpenLink Virtuoso
Driver: 03.00.2310 OpenLink Virtuoso ODBC Driver
callret
VARCHAR
_______________________________________________________________________________

text/plain

1 Rows. -- 95 msec.
```

See Also

`xml_tree_doc()`

xml_uri_get

xml_uri_get — Retrieve a resource based on a URI

Synopsis

```
varchar DB.DBA. xml_uri_get ( in base varchar ,
                              in ref varchar ) ;
```

Description

This function combines a base URI and a relative URI and returns the referenced resource.

The supported protocol identifiers are http: file: and virt:. The virt: allows referencing data stored in local Virtuoso tables without passing through HTTP. See 'Entity References in Stored XML' for details.

The effective URI will be the reference if the URI of the reference is absolute. Otherwise it will be the base URI modified by the relative reference.

Authorization is derived from the SQL or DAV identification of the caller. The DAV identification is used if processing DAV content in response to a DAV request. The SQL user account is used otherwise.

xml_uri_get returns the text of the requested resource. If specific encodings or special authentication schemes are desired one may use http_get directly.

Parameters

base
: A string containing the name of the location (URI) of the resource to be referenced.

ref
: The name of the resource as a relative reference from the base URI.

Return Types

The referenced resource.

Errors

Table 24.132. Errors signalled by

SQL State	Error Code	Error Text	Description

Examples

Example 24.520. Basic Application of xml_uri_get()

```
declare doc_base varchar;
declare doc_tree any;

doc_base := 'virt://WS.WS.SYS_DAV_RES.RES_FULL_PATH.RES_CONTENT:/DAV/docsrc/';
doc_tree := xml_tree_doc(xml_tree(
    xml_uri_get(doc_base, 'virtdocs.xml')
      ), doc_base);

http_value(
  xslt('virt://WS.WS.SYS_DAV_RES.RES_FULL_PATH.RES_CONTENT:/DAV/stylesheets/html_chapter.xsl',
    doc_tree
```

```
        )
    );
```

See Also

??? xslt()

??? http_value()

??? xml_tree_doc()

??? xml_tree()

xml_validate_dtd

xml_validate_dtd — returns a string with list of errors detected by DTD validator on reading given XML document

Synopsis

xml_validate_dtd (in *document* varchar ,
in *parser_mode* integer ,
in *base_uri* varchar ,
in *content_encoding* varchar ,
in *content_language* varchar ,
in *dtd_validator_config* varchar) ;

Description

This parses the argument, which is expected to be an XML fragment (possibly with syntax errors, violations of validity conditions etc.) and returns a human-readable list of errors as a string. DTD validation may be performed during any reading of XML source in functions `xml_tree()` , `xml_persistent()` or `xper_doc()` , so that an application may check XML source on the fly; severe constraint violations in source XML will be signalled as SQL runtime errors.

Parameters

document
: XML or HTML document to check

parser_mode
: 0 or 1; 0 - XML parser mode 1 - HTML parser mode

base_uri
: in HTML parser mode change all absolute references to relative from given base_uri (http://<host>:<port>/<path>)

content_encoding
: string with content encoding type of <document>; valid are 'ASCII', 'ISO', 'UTF8', 'ISO8859-1', 'LATIN-1' etc., defaults are 'UTF-8' for XML mode and 'LATIN-1' for HTML mode

content_language
: string with language tag of content of <document>; valid names are listed in IETF RFC 1766, default is 'x-any' (it means 'mix of words from various human languages)

dtd_validator_config
: configuration string of the validator, default is empty string meaning that only critical errors should be reported. See Configuration Options of the DTD Validator for details.

Return Types

Human readable list of errors if applicable as a varchar.

Validating XML Against a DTD

Example 24.521. Simple Use

```
declare _result varchar;
_result := xml_validate_dtd (
  _text, 0, 'http://localhost.localdomain/xmlrepository', 'UTF-8', 'x-any',
  'Validation=SGML FsaBadWs=IGNORE BuildStandalone=ENABLE MaxErrors=100');

if (_res = '') _res := 'NO ERRORS DETECTED';
```

See Also

`xml_validate_schema()`

`xml_view_dtd()`

`xml_view_schema()`

xml_validate_schema

xml_validate_schema — returns a string with list of errors detected by DTD and XML Schema validator on reading given XML document.

Synopsis

```
xml_validate_schema ( in document varchar ,
                      in parser_mode integer ,
                      in base_uri varchar ,
                      in content_encoding varchar ,
                      in content_language varchar ,
                      in dtd_validator_config varchar ) ;
```

Description

This parses the argument, which is expected to be an XML fragment (possibly with syntax errors, violations of validity conditions etc.) and returns a human-readable list of errors as a string. If there is a "schemaLocation" attribute in root element, XML Schema declaration will be loaded and partial schema validation will be performed. If this attribute does not exist and the Validation option below is not set to DISABLED, then an error will be returned: ('FATAL : Schema declaration is not loaded').

The XML Schema validation routines are tightly coupled with DTD validator. If the document contains both Schema and DTD information then both validations are made in the same time in order to provide as accurate diagnostics as possible. However, it is impossible to check whether the declared DTD matches or contradicts to the declared Schema, so the parser performs two independent validations to every item of source data. E.g. if DTD contradicts to the schema in description of some particular element and data in the document does not contain this element then no errors is reported; but if sun an element occurs in the document then either DTD validator or Schema validator will log an error.

Parameters

document
: XML or HTML document to check

parser_mode
: 0 or 1; 0 - XML parser mode 1 - HTML parser mode

base_uri
: in HTML parser mode change all absolute references to relative from given base_uri (http://<host>:<port>/<path>)

content_encoding
: string with content encoding type of <document>; valid are 'ASCII', 'ISO', 'UTF8', 'ISO8859-1', 'LATIN-1' etc., defaults are 'UTF-8' for XML mode and 'LATIN-1' for HTML mode

content_language
: string with language tag of content of <document>; valid names are listed in IETF RFC 1766, default is 'x-any' (it means 'mix of words from various human languages)

dtd_validator_config
: configuration string of the validator, default is empty string meaning that DTD validator should be fully disabled and only critical errors should be reported. It is very probable that this is not the best choice for your application, so please refer Configuration Options of the DTD Validator to find out how to let validator to do better job.

Return Types

Human readable list of errors if applicable as a varchar.

Validating XML Against a DTD

Example 24.522. Simple Use

```
declare _result varchar;
_result := xml_validate_schema (
  _text, 0, 'http://localhost.localdomain/xmlrepository', 'UTF-8', 'x-any',
  'Validation=SGML FsaBadWs=IGNORE BuildStandalone=ENABLE MaxErrors=100');

if (_res = '') _res := 'NO ERRORS DETECTED';
```

See Also

`xml_validate_dtd()`

`xml_view_dtd()`

`xml_view_schema()`

`xtree_doc()`

`xper_doc()`

xml_view_dtd

xml_view_dtd — returns an XML DTD for the output of given XML VIEW

Synopsis

```
varchar xml_view_dtd ( in view_name varchar ,
                       in root_element varchar ) ;
```

Description

This function will return an XML DTD for the output of a given XML VIEW. The returned DTD will be valid if the HTTP_... output of the view is wrapped into the specified root element.

Parameters

view_name
: Name of an XML View.

root_element
: Name of the root element.

Return Types

XML DTD for the given XML VIEW

See Also

`xml_view_schema()`

`xml_validate_dtd()`

`xml_persistent()`

xml_set_ns_decl

xml_set_ns_decl — Registers the XML NS prefix as persistent or keeps it in properties of client connection depending on the persistance bits input value.

Synopsis

xml_set_ns_decl (in *prefix* varchar ,
in *iri* varchar ,
in *bits* integer) ;

Description

Registers the XML NS prefix as persistent or keeps it in properties of client connection depending on the persistance bits input value.

Parameters

prefix
: namespace prefix

iri
: namespace IRI

bits
: persistance bits: bit 1 - keeps NS decl in properties of client connection i.e stores the assignment in connection parameters so only the current connection will see the namespace prefix declaration; bit 2 - makes NS decl persistent, i.e. stores the declaration in global in-memory hashtable as well as in a system table, loaded at server startup time, visible from all connections

Examples

Example 24.523. Northwind

```
DB.DBA.XML_SET_NS_DECL ('northwind', 'http://demo.openlinksw.com/schemas/northwind#', 2);
```

See Also

xml_remove_ns_by_prefix xml_remove_ns_by_prefix;

xml_view_schema

xml_view_schema — returns an XML schema for the output of given XML VIEW

Synopsis

```
varchar xml_view_schema ( in query varchar ,
                          in root_element varchar ) ;
```

Description

This function return an XML schema for the output of given XML VIEW. The returned schema will be valid if the HTTP_... output of view wrapped into the specified root element.

Parameters

query
: a string which contains a name of XML view

root_element
: a string containing the element name in which the XML schema would be wrapped

Return Types

XML Schema

Errors

Table 24.133. Errors signalled by

SQLState	Error Code	Error Text	Description
42000		No XML view [view_name]	

Examples

Example 24.524. Creating an XML Schema

```
SQL> select xml_view_schema ('cat', 'root');
callret
VARCHAR
_______________________________________________________________________________

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <xsd:annotation>
  <xsd:documentation>
    Schema of XML view 'cat'
  </xsd:documentation>
 </xsd:annotation>

 <xsd:element name="root" type="root__Type"/>

 <xsd:complexType name="root__Type">
  <xsd:element name="category" type="root_category_Type" minOccurs="0" maxOccurs="unbounded"/>
 </xsd:complexType>

 <xsd:complexType name="root_category_Type">
  <xsd:attribute name="CategoryID" type="xsd:string"/>

<!-- <xsd:attribute name="CategoryID" type="xsd:int"/> -->
```

```
  <xsd:attribute name="description" type="xsd:string"/>
  <xsd:element name="product" type="category_product_Type" minOccurs="0" maxOccurs="unbounded"/>
 </xsd:complexType>

 <xsd:complexType name="category_product_Type">
  <xsd:attribute name="ProductName" type="xsd:string"/>
 </xsd:complexType>
</xsd:schema>

1 Rows. -- 9 msec.
```

See Also

xml_view_dtd()

xml_validate_dtd()

xml_persistent()

xmlsql_update

xmlsql_update — Performs insert/update/delete operations based on an XML updategram.

Synopsis

xmlsql_update (in *xml_grams* XML_Entity ,
in *input_parameters* vector ,
in *debug* integer) ;

Description

`xmlsql_update()` supports XML-based insert, update, and delete operations performed on an existing table in the database. See Updategrams basics in the "Web and XML section" for a detailed explanation.

Parameters

xml_grams
: Mandatory parameter containing the XML document with gram(s). This can be produced with sequential calls to the xml_tree() and xml_tree_doc() functions.

input_parameters
: Optional array or vector of parameter pairs (parameter_name, parameter_value).

debug
: The debug parameter takes values 1/0 to enable(1) or disable(0) printing of the SQL statements it generates from the updategrams on the server's standard output. If not specified the default is disabled (0).

Return Values

Zero, if no error is signalled.

Errors

Table 24.134. Errors signalled by xmlsql_update

Error Code	Error Text
XP370	xmlsql_update needs an entity as argument
42000	xmlsql_update expects a vector of even length as second argument
SX999	Empty parameters list
SX998	Too many values for query
SX998	No columns specified in updategram

See Also

Updategrams basics

XMLUpdate

XMLUpdate — Create a changed copy of given document by replacing some nodes.

Synopsis

```
any XMLUpdate ( inout source any ,
                in path1 varchar ,
                in replacement1 any ,
                in path2 varchar ,
                in replacement2 any ,
                ... ,
                in pathN varchar ,
                in replacementN any ) ;
```

Description

The function takes of the XML document referenced by `source` XML tree entity, makes a copy of that document; then it modifies the copy by finding fragments that are values of `path1` , `path2` , ..., `pathN` XPATH expressions and replacing them with values specified by `replacement1` , `replacement2` , ...,`replacementN` ; the modified copy is returned as the result of the function call.

Every `pathI` should be a string that is a correct XPATH expression. Every such expression is evaluated according to the rules for XPATH expressions in XSLT (attribute entities are not cast to their string values). The context node is `source` , context size and position are both equal to 1.

Every `replacementI` may be an XML tree entity, a NULL or a value of some other type that will be converted to varchar before use.

For simplicity, consider the case when there is only one `path1` XPATH expression parameter and only one `replacement1` . First of all, a copy of the document of the `source` XML tree entity is created. Then `path1` is evaluated.

If the result of evaluation is not an XML entity then no replacement is made and the copy of the source document is returned unchanged. If the result is an XML entity that is not in the source document (e.g. an entity returned by the call of document(...) function.) then no replacement is made either.

If the result of evaluation is an XML entity in the source document then the function finds a node in the copied document that is a copy of the node in the source that is returned by XPATH evaluation. This node in the copied document will be referred to as a "highlighted" node.

If the result of evaluation is an attribute entity then the value of the attribute in the highlighted node is changed. If `replacement1` is NULL then the attribute is removed at all, otherwise its value is set to the string value of the `replacement1` .

If the result of evaluation is not an attribute entity but a entity then the highlighted node is replaced with the value of `replacement1` . The method of replacement depends on the type of the replacement value. If `replacement1` is NULL then the highlighted node is simply removed from the result; if the node is the root or a single child of the root then function immediately returns NULL instead of an XML entity (because XML document can not be totally empty), otherwise the parent of the highlighted node will simply have one child less. If `replacement1` is an non-empty string then the highlighted node is replaced with a new text node whose string value is equal to `replacement1` . If `replacement1` is an empty string then the effect is exactly the same as in case of NULL because the string value of a text node can not be empty. If `replacement1` is an XML entity that is a root entity of some document then the highlighted node is replaced with a sequence of copies of all children of that root. If `replacement1` is an XML entity of some other sort (XML element, text, comment etc.) then the that is not a root entity of some document then the highlighted node is replaced with a sequence of copies of all children of that root.

After editing the copied document, some normalization may happen. The modified document may contain adjacent text nodes; every sequence of such nodes is replaced with one text node whose string value is a concatenation of string values of that nodes.

E.g. if the highlighted node is an element <a/> that resides between two text nodes "text1" and "text2" and the highlighted node is removed, then these text nodes become neighbors; normalization will replace them with a single text node "text1text2"

Now consider the case of more than one replacement made in one function call. If more than one pair of XPATH expression and replacement is given then the function first finds all highlighted nodes, all together, then it performs all replacements, then it performs an normalization. It is very important to understand that the order of making replacements has nothing to do with the order of pairs of parameters in the function call. If some highlighted node is an ancestor of some other highlighted node then the effect of the replacement of the descendant highlighted node is void: the descendant is replaced first; then the ancestor is replaced as a whole, no matter if some of its descendants are modified. If a node is the result of two different XPATH expressions ("highlighted more than once"), the order of parameters in the function call is used to resolve ambiguity: the last pair of parameters will have a higher priority.

Parameters

source
: A source XML tree entity. It is an error to pass persistent XML entity as an argument.

pathI
: A text of XPATH expression to be used in order to find nodes to replace.

replacementI
: A replacement value for XML nodes found by pathI ; it may be NULL or XML tree entity or a string r anything else but not an persistent XML entity.

Return Types

If the *source* parameter is an XML tree entity then the function returns an XML tree entity that points to the root of the modified copy of the source document or a NULL. If the *source* is an instance of XMLType then a non-schema based instance of XMLType is created from the resulting XML tree entity.

Examples

Example 24.525. A simple replacement of one element with other

The call of function XMLUpdate replaces element 'c' with a copy of element 'replacement'.

```
select XMLUpdate (xtree_doc('<a><b><q/>-<c>Hello</c>-</b></a>'), '//c', xtree_doc('<replacement/>'))
callret
VARCHAR
_______________________________________________________________________________

<a>
<b>
<q />-
<replacement />-
</b>
</a>
```

Example 24.526. Two replacements in parallel

Elements titled 'c1' and 'c3' are replaced with text nodes.

```
select XMLUpdate (xtree_doc('
<a>
<b>
<c1>Hello1
</c1>
<c2>Hello2
</c2>
<c3>Hello3
</c3>
</b>
```

```
</a>'), '//c1', 'world1', '//c3', 'world3')
callret
VARCHAR
_______________________________________________________________________________

<a>
<b>world1
<c2>Hello2
</c2>world3
</b>
</a>
```

See Also

XMLReplace

XMLAddAttribute

XMLAppendChildren

XMLInsertBefore

XMLInsertAfter

xslt

xpath_eval

xpath_eval — Applies an XPATH expression to a context node and returns result(s).

Synopsis

xpath_eval (in *xpath_expression* varchar ,
in *xml_tree* XML Entity ,
in *index* integer ,
in *named_params* vector) ;

Description

This function returns the result of applying the XPath expression to the context node. By default only the first result is returned, but supplying a third argument allows you to specify an index for the value; the default assumes a value of 1 here. A value of 0 returns an array of 0 or more elements, one for each value selected by the XPath expression.

When this function returns an entity in a result set, the client will see an nvarchar value containing the serialization of the entity, complete with markup. When the entity is passed as an SQL value it remains an entity referencing the node of a parsed XML tree, permitting navigation inside the tree.

The expression can be passed parameters by specifying a fourth argument to `xpath_eval()` . This will be a vector of name/value pairs. The values can be referenced from inside XPath expression by using their names with '$' prefix. You may use any Virtuoso data type. The names in the parameter vector should appear without the '$' sign. If any of the parameter values is NULL the parameter will be ignored because NULL has no XPath counterpart. If the same name appears more than once in the vector, the last name/value pair is used and all preceding pairs with this name are silently ignored. Obviously, names should be strings that are valid XPath variable names.

Parameters

xpath_expression
: A valid XPATH expression. In almost all application this is a string. There is a tricky extension that is used by BPEL-like applications: xpath_expression can be an XML entity whost string-value is a valid XPATH expression. An example of such an entity is "select" or "test" attribute in XSLT stylesheet. This trick lets XPATH processor to resolve namespace prefixes by looking at namespace declarations at the header of the stylesheet.

xml_tree
: An XML entity such as that returned from the xtree_doc() function.

index
: Result index. This parameter is optional. If omitted a value of 1 is assumed, meaning only the first result is returned. If a value of 0 is supplied then an array of 0 or more elements is returned containing one element per result.

named_params
: A vector of keyword/value parameters to be passed to the XPath processor.

Return Types

This function will return the first or *`index`* selected result of applying the XPath expression to the *`xml_tree`* input. If an *`index`* value of 0 is supplied then the output is an array.

Errors

Table 24.135. Errors signalled by xpath_eval

SQL State	Error Code	Error Text	Description
XP370	XI009		

SQL State	Error Code	Error Text	Description
		Argument 2 of xpath_eval must be an entity, not a value of type [type name] ([type code])	The *xml_tree* parameter should always be an entity. In some cases the XPath expression does not use context entity, but it should be provided anyway (e.g. some fake document).

Examples

Example 24.527. Finding the Authors of Document Titles

```
select xt_file, xpath_eval ('//author', t) from xml_text
        where xpath_contains (xt_text, '//chapter/title[. = 'Introduction']', t);
```

This will select all titles that are descendants of chapter and have a string value of 'Introduction'. This will next evaluate *//author* in the context of each, retrieving the author entities in the document of the title.

Example 24.528. xpath_eval and Repeating Nodes.

This example shows how the xpath_eval can be used to retrieve multiple node answers to queries. First to set the scene consider the following statements that create a table with XML inside.

```
CREATE TABLE t_articles (
        article_id int NOT NULL,
        article_title varchar(255) NOT NULL,
        article_xml long varchar
        );

insert into t_articles (article_id, article_title) values (1, 'a');
insert into t_articles (article_id, article_title) values (2, 'b');

UPDATE t_articles SET article_xml = '
<beatles id = "b1">
<beatle instrument = "guitar" alive = "no">john lennon</beatle>
<beatle instrument = "guitar" alive = "no">george harrison</beatle>
</beatles>'
WHERE article_id = 1;

UPDATE t_articles SET article_xml = '
<beatles id = "b2">
<beatle instrument = "bass" alive = "yes">paul mccartney</beatle>
<beatle instrument = "drums" alive = "yes">ringo starr</beatle>
</beatles>'
WHERE article_id = 2;
```

Now we make a query that will return a vector of results, each vector element corresponding to a node-set of the result.

```
SELECT xpath_eval('//beatle/@instrument', xml_tree_doc (article_xml), 0)
        AS beatle_instrument FROM t_articles WHERE article_id = 2;
```

The repeating nodes are returned as part of a vector, the third argument to xpath_eval is set to 0, which means that it is to return all nodes.

Otherwise, we can select the first node-set by supplying 1 as the third parameter to xpath_eval:

```
SELECT xpath_eval('//beatle/@instrument', xml_tree_doc (article_xml), 1)
        AS beatle_instrument FROM t_articles WHERE article_id = 2;
```

Example 24.529. Passing a named parameter to the XPath

```
SQL> select xpath_eval(
  '//phone[../name like $$n]',
  xtree_doc ('<phonebook>
```

```
      <dept><name>Installation Service</name><phone>555-1111</phone></dept>
      <dept><name>Tech Support</name><phone>555-1112</phone></dept>
    </phonebook>'),
  1,
  vector('n', 'Tech%') );
callret
VARCHAR
_______________________________________________________________________________

<phone>555-1112</phone>
```

Like in SQL queries, XPath parameters are used to avoid printing literal values into the text of the query. The listing above demonstrates how to find the phone of the first department whose name is like the specified pattern.

Note that if the text of the XPath expression is entered as a string in ISQL, dollar signs in front of parameter names should be doubled, as in this sample. ISQL uses the same syntax ($name) for macro expansion, and double dollar sign is replaced by ISQL with single dollar sign character.

See Also

`xquery_eval()`

xpath_contains SQL predicate

xcontains SQL Predicate

xper_cut

xper_cut — creates a new "persistent XML"document which contains a copy of data pointed by given XPER entity

Synopsis

xper_cut (in *source_xper* XML_Entity) ;

Description

As noted in the Storage in Database section, a subtree may be extracted from a document during writing of "persistent XML" entity into field of type LONG VARCHAR. The procedure of converting a subtree into complete document is known as "cutting". Cutting is performed only for "persistent XML" documents, it has nothing common with serialization of XML entities in form of plain XML text.

Usually it is the job of the Server itself who decides whether a cutting operation should be performed or not, without any specific activity at application level. The CPU time occupied due to cutting is up to 10 times greater than the CPU time of plain copying of LONG VARCHAR, but the amount of disk I/O is about the same, so the optimization rules discussed below are important only for time-critical, memory-located database applications.

The Virtuoso Server tries to reduce the number of cuttings to an absolute minimum. First of all, cutting is not performed when a given XML entity refers to the root of the document, or to the only child of the root, because the result of such cutting will be identical to original document. In addition, every document remembers the result of last cutting performed on data from this document, so if data of some XML entity are saved in many places without saving of other XML entities between them, cutting will be done only once and plain copying will be done for every subsequent saving.

The only situation when cutting may be seriously optimized by the application developer is in code like the following:

```
declare _blank_forms, _plain, _isdn any;
_blank_forms := xml_persistent ('file://blank_forms.xml');
_plain = xpath_eval ('/forms/form[@name = ''Phone Installation'']', _blank_forms);
_isdn = xpath_eval ('/forms/form[@name = ''ISDN Installation'']', _blank_forms);
for select ID as _id, SERVICE as _service, ADDRESS as _address from CLIENTS do
{
  if (is_isdn(_service))
    {
      insert into JOB_DETAILS (ID, QUERY_XML) values (_id, _isdn);
    }
  else
    {
      insert into JOB_DETAILS (ID, QUERY_XML) values (_id, _plain);
    }
}
```

Calls of xpath_eval are outside the loop, so it is faster than retrieval of suitable form for every selected record. But values of both _plain and _isdn shares the same underlying XML document and they will be assigned many times by the 'insert' operation. The XML document has no place to cache two results of cuttings, so new cutting will be done every time when _isdn entity is saved after _plain or _plain saved after _isdn. To optimize, it is better to cut them once outside the loop:

```
declare _blank_forms, _plain, _isdn any;
_blank_forms := xml_persistent ('file://blank_forms.xml');
_plain = xper_cut (xpath_eval ('/forms/form[@name = ''Phone Installation'']', _blank_forms));
_isdn = xper_cut (xpath_eval ('/forms/form[@name = ''ISDN Installation'']', _blank_forms));
for select ID as _id, SERVICE as _service from CLIENTS do
{
  if (is_isdn(_service))
    {
-- _isdn entity points to the root of its own document, cutting is not needed for root.
      insert into JOB_DETAILS (ID, QUERY_XML) values (_id, _isdn);
    }
  else
    {
-- similarly, _plain entity points to the root of its own document.
      insert into JOB_DETAILS (ID, QUERY_XML) values (_id, _plain);
```

```
    }
}
-- If no records found by the 'select' and no inserts done,
-- then we've made two cuts for nothing...
```

The current node of the resulting entity is the node that is a copy of the current node of the source entity. In common, the top-level node of the copied subtree becomes the current node of the result. There are two special cases, however. If the source entity is an attribute entity, then the result is also an attribute entity and the attribute name remains the same. If the source entity points to the root of the document, the resulting entity also points to the root, not to its top-level node.

Parameters

source_xper
: XML Entity

See Also

xml_doc()

xper_doc()

xper_right_sibling(), xper_left_sibling(), xper_parent(), xper_root_entity(), xper_tell(), xper_length()

xper_doc

xper_doc — returns an entity object ('XPER entity') created from an XML document

Synopsis

```
xper_doc ( in document varchar ,
           in parser_mode integer ,
           in base_uri varchar ,
           in content_encoding varchar ,
           in content_language varchar ,
           in dtd_validator_config varchar ,
           in index_attrs integer ) ;
```

Description

This parses the argument, which is expected to be a well formed XML fragment and returns a parse tree as a special object with underlying disk structure, named "persistent XML" or "XPER" While the result of xml_tree is a memory-resident array of vectors, the XPER object consumes only a little amount of memory, and almost all data are disk-resident. XPERs are better then "XML trees" for large documents and for "write once -- read many" stores such as a table with one XML document per row used as a "library" of documents. To be saved in a LONG VARCHAR column, "XML tree" entity will be converted back to plain text of XML syntax; but "XPER" entity will be saved as a ready-to-use disk structure.

Parameters

document
: well formed XML or HTML document

parser_mode
: 0, 1 or 2; 0 - XML parser mode, 1 - HTML parser mode, 2 - 'dirty HTML' mode (with quiet recovery after any syntax error)

base_uri
: in HTML parser mode change all absolute references to relative from given base_uri (http://<host>:<port>/<path>)

content_encoding
: string with content encoding type of <document>; valid are 'ASCII', 'ISO', 'UTF8', 'ISO8859-1', 'LATIN-1' etc., defaults are 'UTF-8' for XML mode and 'LATIN-1' for HTML mode.

content_language
: string with language tag of content of <document>; valid names are listed in IETF RFC 1766, default is 'x-any' (it means 'mix of words from various human languages)

dtd_validator_config
: configuration string for DTD validator, default is empty string meaning that DTD validator should be fully disabled. See Configuration Options of the DTD Validator for details.

index_attrs
: 1 or 0, indicating if additional free-text indexing information must be stored for all attributes of the document. It is 1 by default. If set to '0', it will produce a disk structure compatible with old versions of Virtuoso and will give a small benefit in disk usage but it will disable some important optimizations in free-text search operations.

Return Types

XML entity with underlying parse tree of source document; the tree will be a special sort of BLOB.

Examples

Example 24.530. Xper_Doc

```
declare tree any;

tree := xper_doc (file_to_string ('doc.html'), 1,
                'http://localhost.localdomain/', 'ISO');
...
tree := xper_doc (file_to_string ('doc.xml'));
...
-- String cannot be longer than 10 megabytes. String session can.
tree := xper_doc (file_to_string_output ('huge_doc.xml'));
...
-- A special way to read local files.
-- Strings started from characters 'file://'
-- are treated as local filesystem URIs.
tree := xper_doc ('file://doc.xml');
```

See Also

xtree_doc

xml_tree_doc

xper_cut

xper_right_sibling, xper_left_sibling, xper_parent, xper_root_entity, xper_tell, xper_length

xper_locate_words

xper_locate_words — returns a smallest fragment of persistent XML entity object ('XPER entity') such that it contains some range of words in its text

Synopsis

xper_locate_words (in *entity* any ,
in *starting_word* integer ,
in *ending_word* integer) ;

Description

This receives the XML entity and returns its fragment or signals an error.

Parameters

entity
: A 'persistent XML' entity to be searched

starting_word
: The number of the leftmost word which should be in the selected fragment

ending_word
: The number of the rightmost word which should be in the selected fragment

XPER navigation

xper_right_sibling , xper_left_sibling , xper_parent , xper_root_entity , xper_tell , xper_length — low-level navigation functions for persistent XMLs, useful for import of huge amounts of XML data

Synopsis

xper_right_sibling (in *xper_entity* XML_entity);

xper_left_sibling (in *xper_entity* XML_entity);

xper_parent (in *xper_entity* XML_entity);

xper_root_entity (in *xper_entity* XML_entity);

xper_tell (in *xper_entity* XML_entity);

xper_length (in *xper_entity* XML_entity);

Description

All these functions work with "persistent XML" (XPER) entities only, signalling errors if given entity points to "XML tree". They are useful when applications need to read a huge XML document, especially something like a datasheet dump or event log with a large number of uniform records, and is required to process all records of the document, e.g. import them into the database.

Consider a real sample of import all data from ODP's content.xml dump which contains more than 2,000,000 descriptions of various Web-sites, and the length of the file is more than 600Mb. The file has root element named 'RDF' and all descriptions are their children named either 'Topic' or 'ExternalPage'. This code looks suitable for importing these children:

```
create procedure
  import_content_xml ()
  {
    for select _frag, xpath_eval('local_name()', _frag) as _tag from SOURCE_XML
      where NAME='content.xml' and xpath_contains(XPER,'/RDF/*',_frag)
    do
    {
      if (_tag='Topic') import_topic (_frag);
      else
      {
        if (_tag='ExternalPage') import_external (_frag);
        else log_import_error (_frag, "Unknown type of record");
      }
    }
  };
  -- This should load files like content.xml, and place XPERs into SOURCE_XML table

read_source_xmls();
  -- Now SOURCE_XML is filled and data should be imported.

import_content_xml();
import_structure_xml();
import_profiles_xml();
 ...
```

It looks fine and it passes small tests but it will not work on real data!

First problem is regular checkpoints (every 1 hour by default), so import_content_xml has no chance to be completed if it takes 1.5 hours of CPU time; the function checkpoint_interval should be used to temporarily disable these checkpoints. Then, the length of transaction log become extremely large after switching checkpoints off, and it is better to insert explicit checkpoints between calls of these functions. Finally, import_content_xml will change more than 4 gigabytes of data in one transaction. This would be impossible on any 32-bit platform, because both memory available and address space become insufficient. Adding intermediate

'commit work' statements inside the loop's body will not help because both 'commit work' and 'rollback work' statements will close all opened cursors. Loop 'for select ... from ... do' uses an implicit cursor to iterate the resultset, but the 'commit work' operator will close this cursor and abort the loop.

Function xper_right_sibling is designed specially to solve the last problem. It allows you to iterate children of some element without using any cursor at all. It returns the right child of the entity passed as its argument. If the given entity is the last child of its parent, the function will return NULL. Similarly, xper_left_sibling will return the previous child or NULL for the first child, xper_parent will return parent of entity or NULL for the document's root and xper_root_entity will return the root for any entity. Using these functions, it is possible to scan the document forward (from left to right), backward and to "climb up" toward the root of the elements' tree. These functions are called *XML Navigation Functions* because they are like the statements "next record", "prev record" etc., used in cursor like navigation of databases.

There are no special functions to "go down", e.g. to find first children of given element, because xpath_eval can do any such movement very quickly.

Sometimes data import is controlled from some client application. If one operation takes hours, some sort of "progress bar" becomes very useful, at least to see whether application works or hangs. xper_length takes an XPER entity and returns whole length of XPER disk image in bytes. xper_tell takes an XPER entity and returns something like entity's offset inside the document. Their results may be used by the client application to monitor the progress and estimate the time until completion. They may be especially useful for debugging purposes, e.g. to report position of error. Unfortunately "xper_seek" is not possible for XPERs, unlike typical random access to files.

Ultimately, the working version of the procedure described above will look like this:

```
create procedure
  import_content_xml ()
  {
    declare _frag any;
    declare _tag any;
    declare _nodeid any;
    declare _r_id varchar;
    declare _about, _title, _description varchar;
    declare CurTime varchar;
    declare Frags, Inserts, Pos, Length integer;

    select _frag1 into _frag from SOURCE_XML
      where NAME = 'content.xml' and xpath_contains (XPER, '/RDF/*', _frag1);

    if (not isentity(_frag)) signal('42ODP', 'XML source "Content" is empty');

    Frags := 0;
    Inserts := 0;
    Length := xper_length(_frag);
    result_names (CurTime, Frags, Pos, Length);
    -- It's not the best way to process deadlocks, but it may be better than nothing.

    whenever sqlstate '40001' goto next_frag;

    next_frag:

    -- Server will inform client application about progress after every 10000 records.
    if (mod(Frags,10000) = 0)
    {
      -- Note that if it needs to estimate the time to completion, time should be measured
      -- on server side, because results may be returned on batches to reduce network
      -- traffic, and it may be a significant delay between call of result function on
      -- server and fetch on client side.
      CurTime := cast(now() as varchar);
      Pos := xper_tell(_frag);
      result (CurTime, Frags, Inserts, Pos, Length);
    }
    _tag := xpath_eval('local-name()', _frag);
    if (_tag = 'Topic')
    {
      import_topic(_frag);
      goto advance;
    }
    if (_tag = 'ExternalPage')
    {
```

```
    import_external(_frag);
    goto advance;
  }
  log_import_error (_frag, "Unknown type of record");
  goto nocommit_advance;

  advance:
  Inserts := Inserts + 1;
  -- Note that it may be faster to have 1 commit per 100 or 1000 records in some cases.

  commit work;

  nocommit_advance:
  Frags := Frags + 1;

  _frag := xper_right_sibling(_frag);
  if (isentity(_frag)) goto next_frag; -- If _frag is NULL, we've completed the import.

  CurTime := cast(now() as varchar);
  result (CurTime, Frags, Inserts, Length, Length);
};

checkpoint_interval(0);
checkpoint;

read_source_xmls();
checkpoint;

import_content_xml();
checkpoint;

import_structure_xml();
checkpoint;

import_profiles_xml();
checkpoint;
...

checkpoint_interval(60);
```

Parameters

xper_entity
: Persistent XML entity to operate on.

See Also

xper_doc()

xper_cut()

xpf_extension

xpf_extension — declare an XPath extension function

Synopsis

```
void xpf_extension ( in fname varchar ,
                     in procedure_name varchar ) ;
```

Description

This function is used to declare a new XPath extension function or redefine an existing function. It can be used in XPath queries and XSLT stylesheets. You should use QNames for extension functions. Note that the standard XPath functions cannot be redefined.

`xpf_extension()` stores the functions into the SYS_XPF_EXTENSIONS system table.

```
CREATE TABLE
  DB.DBA.SYS_XPF_EXTENSIONS (
      XPE_NAME VARCHAR PRIMARY KEY,
      XPE_PNAME VARCHAR
      )
```

The input parameters will be retrieved as a strings and then will be converted to the datatype of the corresponding argument of the stored procedure.

Parameters

fname
: The name of the extension function, which must be the expanded QName of the extension function

procedure_name
: The fully qualified name of the PL procedure which acts as the extension function. The procedure in question must be granted to PUBLIC, otherwise it will not be registered and error will be signalled.

Return Types

None (void).

Errors

Table 24.136. Errors signalled by

SQLState	Error Code	Error Text	Description
42001	XPE01	The function <procedure_name> does not exists	if procedure to define as a XPATH extension function is not existing one.
42001	XPE02	The <built-in XPATH|XQUERY> function " <func name>" cannot be re-defined	if XPATH or XQUERY function to be registered is a core function.

Examples

Example 24.531. Declaring a New XSLT Function

First define a PL procedure, then declare an XPath extension function and to represent it.

```
SQL> create procedure DB.DBA.str_concat (in a varchar, in b varchar)
```

```
      {
        return concat (a, ':', b);
      };

SQL> xpf_extension ('http://example.com/virtuoso/xslt:concat_strings', 'DB.DBA.str_concat');
```

The source of the ([http_root]/ext.xsl) XSLT stylesheet

```
<?xml version='1.0'?>

<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/TR/WD-xsl"
  xmlns:virt="http://example.com/virtuoso/xslt">

  <xsl:template match="/doc">
    <HTML>
     <BODY>
     <xsl:if test="function-available('virt:concat_strings')">
      <xsl:value-of select="virt:concat_strings ('foo', 'bar')"/>
     </xsl:if>
     </BODY>
    </HTML>
  </xsl:template>
</xsl:stylesheet>
```

The source of the

```
([http_root]/ext.vsp)
```

VSP page:

```
<?vsp
  http_xslt ('file:/ext.xsl');
?>
<doc>
<a/>
</doc>
```

This will produce the following HTML page:

```
<HTML><BODY>foo:bar</BODY></HTML>
```

Using the definition of the XPath extension function, we can include it in XPath expressions.

```
SQL> select p from ws..sys_dav_res
where xpath_contains (RES_CONTENT,
'[xmlns:virt=''http://example.com/virtuoso/xslt'']
virt:concat_strings (''Title '', string(/chapter/title))', p);
```

This will return the contents of any '/chapter/title' nodes, prefixed with constant string 'Title'.

See Also

`xpf_extension_remove`

xpf_extension_remove

xpf_extension_remove — discards an XPath extension function

Synopsis

```
void xpf_extension_remove ( in fname varchar ,
                            in procedure_name varchar ) ;
```

Description

Removes a user-defined XPath function.

Parameters

fname
: The expanded QName of the extension function to be removed

procedure_name
: The fully qualified name of the PL procedure which acts as the extension function.

Return Types

None (void).

Errors

Table 24.137. Errors signalled by

SQLState	Error Code	Error Text	Description
42001	XPE03	The <built-in XPATH|XQUERY> function " <func name>" cannot be removed	if XPATH or XQUERY function to be un-registered is a core function.

See Also

`xpf_extension()`

xquery_eval

xquery_eval — Applies an XQUERY expression to a context node and returns result(s).

Synopsis

xquery_eval (in *xquery_expression* varchar ,
in *xml_tree* XML Entity ,
in *index* integer ,
in *named_params* vector) ;

Description

The xquery_eval function returns the result of applying the xquery expression to the context node. By default only the first result is returned, but supplying a third argument allows you to specify an index for the value, the default assumes a value of 1 here. A value of 0 returns an array of 0 or more elements, one for each value calculated by the xquery expression.

When an entity is returned in a result set to a client the client will see an nvarchar value containing the serialization of the entity, complete with markup. When the entity is passed as a SQL value it remains an entity referencing the node of a parsed XML tree, permitting navigation inside the tree.

The expression can be passed parameters by specifying a fourth argument to `xquery_eval()` . This will be a vector of name/value pairs. The values can be referenced from inside XPath expression by using their names with '$' prefix. You may use any Virtuoso data type. The names in the parameter vector should appear without the '$' sign. If any of the parameter values is NULL the parameter will be ignored because NULL has no XPath counterpart. If the same name appears more than once in the vector, the last name/value pair is used and all preceding pairs with this name are silently ignored. Obviously, names should be strings that are valid XQuery variable names.

Parameters

xquery_expression
: A valid XQuery expression. In almost all applications this is a string. There is a tricky extension that is used by BPEL-like applications: xpath_expression can be an XML entity whost string-value is a valid XQuery expression. An example of such an entity is "select" or "test" attribute in XSLT stylesheet. This trick lets XQuery processor to resolve namespace prefixes by looking at namespace declarations at the header of the stylesheet.

xml_tree
: An XML entity such as that returned from the xtree_doc() function.

index
: Result index. This parameter is optional. If omitted a value of 1 is assumed, meaning only the first result is returned. If a value of 0 is supplied then an array of 0 or more results is returned containing one element per result. (Note that results can be in turn sequences of atomic values).

named_params
: A vector of keyword/value parameters to be passed to the XQuery processor.

Return Types

This function will return the first or index selected result of applying the xpath expression to the *`xml_tree`* input. If an index value of 0 is supplied then the output is an array.

Examples

Example 24.532. Finding the Authors of Document Titles

```
select xt_file, xquery_eval ('<authors>//author</authors>', t) from xml_text
        where xpath_contains (xt_text, '//chapter/title[. = 'Introduction']', t);
```

This will select all titles that are descendants of chapter and have a string value of 'Introduction'. This will next evaluate //author in the context of each, retrieving the author entities in the document of the title and construct an element whose name is "authors" and list of children contains all retrieved entities.

See Also

`xpath_eval()`

xpath_contains SQL predicate

xcontains SQL predicate

`processXQuery()`

xslt

xslt — returns an XML document transformed by an XSLT stylesheet

Synopsis

```
xml_entity xslt ( in sheet_uri varchar ,
                  in entity any ,
                  in sheet_params vector ) ;
```

Description

This function takes the URI of a stylesheet and an XML entity and produces an XML entity representing the transformation result of the given entity with the given stylesheet. The result tree is separate from the argument tree and the only reference to it is the returned entity. Errors occurring in the transformation will be signalled as SQL states, with XML or XSLT specific conditions beginning with XS or XP.

The stylesheet can be passed parameters by specifying a third argument to `xslt()` . This will be a vector of name/value pairs. The values can be referenced from inside XPath expressions in the stylesheet. You may use any Virtuoso data type. The names in the parameter vector should appear without the '$' sign. If any of the parameter values is NULL the parameter will be ignored because NULL has no XPath counterpart. If the same name appears more than once in the vector, the last name/value pair is used and all preceding pairs with this name are silently ignored. Obviously, names should be strings that are valid XPath variable names.

`xslt()` applies the transformation in the sheet to the specified entity. The result is the root element of the result tree, an XML entity. This entity can be used as input to another transformation, can be serialized and sent to an HTTP client or stored, etc.

The URI of the sheet is used to locate the stylesheet. The protocol can be http, file or virt. Once the stylesheet has been retrieved it is cached and not refetched until the cache is invalidated with `xslt_stale()` . Included or imported style sheets will be fetched automatically, using the initial URI as base for any relative references. The *`sheet_uri`* is the URI of a stylesheet. This should be an absolute URI resolvable with `xml_uri_get()` . If the URI has previously been used as a stylesheet and has not been marked stale with `xslt_stale()` , Virtuoso will use the cached data instead of fetching and parsing the stylesheet resource.

When a resource with a .xsl extension is stored into the WebDAV, Virtuoso marks as stale any related cached resource. The URI for such stylesheets will be

```
virt://WS.WS.SYS_DAV_RES.RES_FULL_PATH.RES_CONTENT:<path>
```

. See the section about entity references in stored text for more on this type of URI. If the URI uses the file protocol, Virtuoso will compare the date of the file against the date of the cached stylesheet, automatically reloading the sheet if the file changes. This protocol is subject to the limitations on file system access imposed by the host operating system and the virtuoso.ini file system access control settings.

Parameters

sheet_uri
: URI pointing to the location of an XSL stylesheet.

entity
: parsed XML entity such as that returned by the xtree_doc() function

sheet_params
: A vector of keyword/value parameters to be passed to the XSLT engine for use in the transformation.

Return Types

An XML entity resulting from transforming the given XML entity input with the given style sheet.

Examples

Example 24.533. Basic Use

```
declare sheet, xml varchar;
declare xml_entity, myparams any;

sheet := 'virt://WS.WS.SYS_DAV_RES.RES_FULL_PATH.RES_CONTENT:/DAV/stylesheets/mysheet.xsl';
xml := 'virt://WS.WS.SYS_DAV_RES.RES_FULL_PATH.RES_CONTENT:/DAV/xmlsource/myxml.xml';
xml_entity := xtree_doc(xml);

myparams := vector('chapter', 'overview', 'imgroot', '/DAV/images/');

http_value(xslt(sheet, xml_entity, myparams));
```

This code will send the resultant transformation to the http stream. Note that the *xml_entity* must be a parsed XML entity such as that produced by *xtree_doc*.

```
<?xml version='1.0'?>

<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/TR/WD-xsl">

  <xsl:param name="chapter">default</xsl:param>
  <xsl:param name="imgroot"></xsl:param>

  <xsl:template match="/">
    <xsl:text>Value of chapter: </xsl:text><xsl:value-of select="$chapter" />
    <xsl:text>Value of imgroot: </xsl:text><xsl:value-of select="$imgroot" />
  </xsl:template>

</xsl:stylesheet>
```

This style sheet illustrates using parameters. Note how default values can be provided in the style sheet when the parameters are declared using the *xsl:param* tag.

See Also

xtree_doc()

xper_doc()

xml_tree_doc()

xml_tree()

xslt_stale()

XMLUpdate()

xslt_format_number

xslt_format_number — returns formatted string representation of a numeric value

Synopsis

xslt_format_number (*number* any ,
format_string varchar) ;

Description

xslt_format_number is an function wrapper for the format-number() XSLT function.

It always uses the default formatting parameters described in the XSLT standard.

Parameters

number
: integer , numeric or string .

format_string
: string

Return Values

An string representation of the numeric value of *number* converted according to the format *format_string* .

Examples

Example 24.534. Simple examples

```
xslt_format_number (5351, '#,###'),        5,351
xslt_format_number ('5351', '#.00'),       5351.00
xslt_format_number (53.51, '#.0000'),      53.5100
xslt_format_number (53.51, '0000.0000'),   0053.5100
xslt_format_number (53.51, '0000.####'),   0053.51
xslt_format_number (53.56, '0.0');            53.6
```

See Also

casting

xslt_sheet

xslt_sheet — declares an XSL stylesheet for use

Synopsis

xslt_sheet (in *uri* varchar ,
in *entity* any) ;

Description

This function takes a name and the root element of a parsed XML document and defines these as a stylesheet. The unique element child of the entity object's document should be an xsl:stylesheet element. Included or imported stylesheets will be located relative to the base URI of the entity passed to `xslt_sheet()` . Once a stylesheet thus defined it can be used as the stylesheet argument of xslt.

Parameters

uri
: The location of the XSLT style sheet

entity
: A valid XSL style sheet, XML entity parsed using the xml_tree_doc() function

Examples

Example 24.535. Preparing a Style Sheet from the File System

The `xslt_view()` function first defines the style sheet from a file. The `xslt_sheet()` function is called with the name and the root element of the parsed file. `xslt_view()` next gets the string to process, parses the string as XML and converts the parse tree into an entity object. This is then passed to the `xslt()` function. The result is another entity object. This is finally serialized as XML text and written into the file *`xslt.out`* .

```
create procedure xslt_view (in v varchar, in xst varchar)
{
  declare str, r varchar;
  xslt_sheet (xst, xml_tree_doc (xml_tree  (file_to_string (xst))));
  str := xml_view_string (v);
  r := xslt (xst, xml_tree_doc (xml_tree (str)));
  declare str any;
  str := string_output ();
  http_value (r, 0, str);
  string_to_file ('xslt.out', string_output_string (str), 0);
}
```

See Also

`xml_tree_doc()`

xslt_stale

xslt_stale — force reload of XSL stylesheet

Synopsis

xslt_stale (in *uri* varchar);

Description

This function can be used to force Virtuoso to reload a cached stylesheet from the URI when next used with `xslt()` or `http_xslt()` . Using this function before every application of the stylesheet is extremely inefficient. If stylesheets are stored in the database, you can use this function in an update trigger on the table storing the stylesheets but you don't have to use it before every application of the sheet.

This function never needs to be applied to a stylesheet URI with the file:// protocol since `xslt()` and `http_xslt()` will automatically detect a stale cache entry. However if the stylesheet is stored on a remote web server, or if the stylesheet contains subdocuments (e.g. external XML entities, xsl:include or xsl:import statements) this function is needed to force a reload.

Parameters

uri
: The location of the style sheet to force a reload.

Errors

The function never signals any errors. E.g. it is not an error if the specified stylesheet does not exist or if the specified stylesheet is used by other transaction when the `xslt_stale()` is called.

Examples

Example 24.536. Simple Use

```
sheet := 'virt://WS.WS.SYS_DAV_RES.RES_FULL_PATH.RES_CONTENT:/DAV/stylesheets/document.xsl';
xslt_stale (sheet);
xslt(sheet, xml_doc_tree);
```

See Also

`xslt()`

`http_xslt()`

xtree_doc

xtree_doc — returns an entity object created from an XML document

Synopsis

xtree_doc (in *document* varchar ,
in *parser_mode* integer ,
in *base_uri* varchar ,
in *content_encoding* varchar ,
in *content_language* varchar ,
in *dtd_validator_config* varchar) ;

Description

This parses the argument, which is expected to be a well formed XML fragment and returns a parse tree as a special memory-resident object. While xper_doc creates some disk-resident data structure, xtree_doc() will work faster but it may require more memory. You may wish to use xtree_doc for small documents (e.g. less than 5 megabytes and xper_doc for larger documents.

Parameters

document
: well formed XML or HTML document

parser_mode
: 0, 1 or 2; 0 - XML parser mode, 1 - HTML parser mode, 2 - 'dirty HTML' mode (with quiet recovery after any syntax error)

base_uri
: in HTML parser mode change all absolute references to relative from given base_uri (http://<host>:<port>/<path>)

content_encoding
: string with content encoding type of <document>; valid are 'ASCII', 'ISO', 'UTF8', 'ISO8859-1', 'LATIN-1' etc., defaults are 'UTF-8' for XML mode and 'LATIN-1' for HTML mode.

content_language
: string with language tag of content of <document>; valid names are listed in IETF RFC 1766, default is 'x-any' (it means 'mix of words from various human languages')

dtd_validator_config
: configuration string for DTD validator, default is empty string meaning that DTD validator should be fully disabled. See Configuration Options of the DTD Validator for details.

Return Types

XML entity with underlying parse tree of source document; the tree will be a memory-resident structure of nested heterogeneous vectors.

Examples

Example 24.537. xtree_doc

```
declare tree any;

tree := xtree_doc (file_to_string ('doc.html'), 1,
                'http://localhost.localdomain/', 'ISO');
...
```

```
tree := xtree_doc (file_to_string ('doc.xml'));
```

See Also

xml_tree

xml_tree_doc

xper_doc

xmlStorageSystem.registerUser

xmlStorageSystem.registerUser — Register a user with the XML Storage System

Synopsis

```
structure xmlStorageSystem.registerUser ( in email string ,
                                          in name string ,
                                          in password string ,
                                          in clientPort integer ,
                                          in userAgent string ,
                                          in serialNumber string ) ;
```

Description

This method allows users to register within XML Storage System. The actions performed on the server are:

make a new Web account named <email> with the password specified.
make a home-folder in WebDAV repository as /DAV/<email>.
make a sub-folder called "blog".

Parameters

email
: For user identification, this should be a valid email account.

name
: A human readable name for the user account.

password
: The password for authentication. Note that this is the only call that sends the password in clear-text. All other functions send the password as a MD5 hash.

clientPort
: Currently ignored and reserved for future use.

userAgent
: Currently ignored and reserved for future use.

serialNumber
: Currently ignored and reserved for future use.

Return Types

This method returns a structure as follows:

```
'usernum' string    account id
'flError' boolean   0 success, 1 fault
'message' string    fault message (can be used to handle error cases)
```

Examples

Example 24.538. Registering a new user

```
SQL> SOAP_CLIENT (url=>'http://example.com/xmlStorageSystem',
                  operation=>'registerUser', parameters=>vector
                 ('email','user@example.domain', 'name', 'Somebody Name',
                  'password', 'xxx', 'clientPort', 0, 'userAgent', '' , 'serialNumber', ''));
```

```
- SOAP response
<registerUserResponse>
  <Result>
    <usernum>user@example.domain</usernum>
    <flError>0</flError>
    <message>Welcome, Somebody Name</message>
  </Result>
</registerUserResponse>
```

See Also

`xmlStorageSystem.mailPasswordToUser()`

`xmlStorageSystem.getServerCapabilities()`

`xmlStorageSystem.deleteMultipleFiles()`

`xmlStorageSystem.saveMultipleFiles()`

xmlStorageSystem.mailPasswordToUser

xmlStorageSystem.mailPasswordToUser — Mail password to user.

Synopsis

```
structure xmlStorageSystem.mailPasswordToUser ( in email string );
```

Description

This method is used to send the password for user identified by email. To work properly the default SMTP server Virtuoso INI file setting must be set properly.

Parameters

email
: For user identification, this should be a valid email account.

Return Types

This method returns a structure as follows:

```
'flError' boolean                   0 success, 1 fault
'message' string                    fault message (can be used to handle error cases)
```

Examples

Example 24.539. Mailing the XML Storage System Users Password to the User

Simple example of mailing the users password.

```
SQL> SOAP_CLIENT (url=>'http://example.com/xmlStorageSystem',
operation=>'mailPasswordToUser', parameters=>vector ('email','user@example.domain'));

SOAP response:

<mailPasswordToUserResponse>
  <Result>
    <flError>1</flError>
    <message>Mail service is not available on that server.</message>
  </Result>
</mailPasswordToUserResponse>
```

Note : in this case emailing was failed

See Also

`xmlStorageSystem.registerUser()`

`xmlStorageSystem.getServerCapabilities()`

`xmlStorageSystem.deleteMultipleFiles()`

`xmlStorageSystem.saveMultipleFiles()`

xmlStorageSystem.getServerCapabilities

xmlStorageSystem.getServerCapabilities — Retrieve server specific information.

Synopsis

```
structure xmlStorageSystem.getServerCapabilities ( in email string ,
                                                   in password string ) ;
```

Description

This method is used to obtain information about the capabilities of the server, such as files size limitations, supported types, etc...

Parameters

email
: For user identification, this should be a valid email account.

password
: The MD5 hash of the password.

Return Types

This method returns a structure as follows:

```
'flError' boolean                  0 success, 1 fault
'message' string                   fault message (can be used to handle error cases)
 ctBytesInUse integer              how many bytes are used on repository
 legalFileExtensions string array  which extensions are recognized
 maxBytesPerUser integer           total bytes available for this user
 maxFileSize integer               largest file size for upload (bytes)
 yourUpstreamFolderUrl string      base URL for user's blog on repository
```

Examples

Example 24.540. Retrieving the XML Storage System Capabilities

Simple example of fetching the server capabilities.

```
SQL> SOAP_CLIENT (url=>'http://example.com/xmlStorageSystem',
    operation=>'getServerCapabilities', parameters=>vector
    ('email','user@example.domain', 'password', md5('xxx'));

SOAP response:

<getServerCapabilitiesResponse>
   <Result>
     <ctBytesInUse>14</ctBytesInUse>
     <flError>0</flError>
       <legalFileExtensions>
         <item>ai</item>
         <item>aif</item>
         <item>aifc</item>
       </legalFileExtensions>
     <maxBytesPerUser>41943040</maxBytesPerUser>
     <maxFileSize>1048576</maxFileSize>
     <message />
     <yourUpstreamFolderUrl>http://example.com/DAV/user@example.domain/blog/</yourUpstreamFolderUrl>
   </Result>
</getServerCapabilitiesResponse>
```

See Also

`xmlStorageSystem.registerUser()`

`xmlStorageSystem.mailPasswordToUser()`

`xmlStorageSystem.deleteMultipleFiles()`

`xmlStorageSystem.saveMultipleFiles()`

xmlStorageSystem.deleteMultipleFiles

xmlStorageSystem.deleteMultipleFiles — Remove files from XML Storage System Directory

Synopsis

```
structure xmlStorageSystem.deleteMultipleFiles ( in email string ,
                                                 in password string ,
                                                 in relativepathList array ) ;
```

Description

This method is used for removing files from the repository.

Parameters

email
: For user identification, this should be a valid email account.

password
: The MD5 hash of the password.

relativepathList
: Array of strings of paths relative to the <user home>/blog folder in WebDAV to be deleted.

Return Types

This method returns a structure as follows:

```
'flError' boolean                  0 success, 1 fault
'message' string                   fault message (can be used to handle error cases)
'errorList' array of string        contains a list of errors corresponding to files
                  which cannot be removed.  Successfully deleted files have an empty string.
```

Examples

Example 24.541. Using the XML Storage System to Delete Multiple Files

Simple example of removing a file.

```
SQL> SOAP_CLIENT (url=>'http://example.com/xmlStorageSystem', operation=>'deleteMultipleFiles',
     parameters=>vector ('email','user@example.domain', 'password', md5('xxx'),
       'relativepathList', vector ('test.txt'));

- SOAP Response

<deleteMultipleFilesResponse>
  <Result>
   <flError>0</flError>
   <message />
   <errorList>
     <item />
   </errorList>
  </Result>
</deleteMultipleFilesResponse>
```

See Also

`xmlStorageSystem.registerUser()`

`xmlStorageSystem.mailPasswordToUser()`

`xmlStorageSystem.getServerCapabilities()`

`xmlStorageSystem.saveMultipleFiles()`

xmlStorageSystem.saveMultipleFiles

xmlStorageSystem.saveMultipleFiles — Upload a set of files to XML Storage System directory.

Synopsis

```
structure xmlStorageSystem.saveMultipleFiles ( in email string ,
                                               in password string ,
                                               in relativepathList array ,
                                               in fileTextList array ) ;
```

Description

This method allows users to upload a set of files. The files will be stored in WebDAV repository and will be accessible via HTTP.

Parameters

email
: For user identification, this should be a valid email account.

password
: The MD5 hash of the password.

relativepathList
: Array of strings of target paths relative to the <user home>/blog folder in WebDAV.

fileTextList
: An array of strings containing the contents of the files to be uploaded. These must be in the same sequence as the relativepathList. i.e. relativepathList[i] is name for fileTextList[i].

Return Types

This method returns a structure as follows:

```
'flError' boolean                  0 success, 1 fault
'message' string                   fault message (can be used to handle error cases)
'urllist' array of string   contains list of URLs corresponding to uploaded files
'yourUpstreamFolderUrl' string      base URL for upload
```

Examples

Example 24.542. Using the XML Storage System to Save Multiple Files

Simple example of saving a file.

```
- Invocation:

SQL> SOAP_CLIENT (url=>'http://example.com/xmlStorageSystem', operation=>'saveMultipleFiles',
    parameters=>vector ('email','user@example.domain', 'password', md5('xxx'),
      'relativepathList', vector ('test.txt'), 'fileTextList', vector ('this is a test'));

- SOAP Response

<saveMultipleFilesResponse>
  <Result>
    <flError>0</flError>
    <message />
    <urllist>
      <item>http://example.com/DAV/user@example.domain/blog/test.txt</item>
    </urllist>
    <yourUpstreamFolderUrl>http://example.com/DAV/user@example.domain/blog/</yourUpstreamFolderUrl>
```

```
    </Result>
  </saveMultipleFilesResponse>
```

See Also

`xmlStorageSystem.registerUser()`

`xmlStorageSystem.mailPasswordToUser()`

`xmlStorageSystem.getServerCapabilities()`

`xmlStorageSystem.deleteMultipleFiles()`

XMLType.XMLType

XMLType.XMLType

Synopsis

```
constructor method XMLType.XMLType ( in src any ,
                                     in schema_uri any ,
                                     in validated any ,
                                     in wellformed any ) ;
```

Description

The method creates an XMLType instance from *src* XML entity. If parameter *src* is not an XML entity then it is converted to it via internal call of `xtree_doc()` or `xml_tree_doc()` . A schema may be associated with an XML entity by passing its URI as *schema_uri* ; this schema can be used later to validate the structure of the document.

Parameters

src
: An XML entity or a value that can be converted to an XML entity.

schema_uri
: An URI of the schema of the document. The default is NULL to make result non-schema based.

validated
: An integer flag that indicates if the document is already validated against the schema of the document (this is to avoid redundant validations). The default is 0.

wellformed
: This parameter is unused and is listed solely for compatibility.

Return Types

The method returns a new instance of XMLType.

Examples

Example 24.543. Simple Use

```
create table XMLTYPE_TEST (I integer primary key, XMLVAL long xml)

Done. -- 00000 msec.

insert into XMLTYPE_TEST values (1, XMLType('<emp><empno>221</empno><ename>John</ename></emp>'))

Done. -- 00000 msec.

insert into XMLTYPE_TEST values (2, XMLType('<po><pono>331</pono><poname>PO_1</poname></po>'))

Done. -- 00000 msec.

select e.XMLVAL.extract ('//empno/text()').getNumVal() as EMPNO
  from XMLTYPE_TEST as e
  where e.XMLVAL.existsNode('/emp/empno') = 1
EMPNO
DECIMAL
_______________________________________________________________________________

221
```

```
1 Rows. -- 00000 msec.
```

See Also

`xtree_doc()`

`xper_doc()`

XMLType.createNonSchemaBasedXML

XMLType.createNonSchemaBasedXML — Returns a copy of the given instance with the schema reference removed.

Synopsis

XMLType.createNonSchemaBasedXML ( );

Description

The function returns a copy of the given instance with the schema reference removed.

Return Types

The function returns a new instance of XMLType.

Examples

Example 24.544. Converting a schema-based instance to non schema-based

The procedure creates a non schema-based copy of a sample schema-based instance.

```
create procedure schema_removal_test ()
{
  declare orig_instance XMLType;
  declare non_schema_copy XMLType;
  declare PROBE varchar;
  result_names (PROBE);
  orig_instance := XMLType ('<sample-xml/>', 'http://www.example.com/test.xsd');
  result (concat (
    '"orig_instance" is declared as schema-based, URI=',
    orig_instance.getSchemaURL() ) );
  non_schema_copy := orig_instance.createNonSchemaBasedXML();
  result (concat (
    '"non_schema_copy" is non schema-based, URI=',
    cast (non_schema_copy.getSchemaURL() as varchar) ) );
}

schema_removal_test()
PROBE
VARCHAR
_______________________________________________________________________________

"orig_instance" is declared as schema-based, URI=http://www.example.com/test.xsd
"non_schema_copy" is non schema-based, URI=

2 Rows. -- 00000 msec.
```

See Also

XMLType.createSchemaBasedXML()

XMLType.createSchemaBasedXML

XMLType.createSchemaBasedXML — Returns a copy of the given instance with new schema reference.

Synopsis

XMLType.createSchemaBasedXML (in *schema_url* varchar) ;

Description

The function creates a new instance of XMLType by copying the given one and assigning a new schema URL to it. Parameter `schema_url` specifies a new URL; if it is omitted then a non-schema based instance is created like the `XMLType.createNonSchemaBasedXML()` function.

The returned copy has an internal "validated" flag set to 0, even if the new URL is equal to the URL of the original instance.

Parameters

schema_url
: The location of a schema of a new instance.

Return Types

The member function returns a new instance of XMLType.

Examples

Example 24.545. Converting a non schema-based instance to schema-based one

The procedure creates a schema-based copy of a sample non schema-based instance.

```
create procedure schema_assignment_test ()
{
  declare orig_instance XMLType;
  declare schema_based_copy XMLType;
  declare PROBE varchar;
  result_names (PROBE);
  orig_instance := XMLType ('<sample-xml/>');
  result (concat (
    '"orig_instance" is declared as non schema-based, URI=',
    cast (orig_instance.getSchemaURL() as varchar) ) );
  schema_based_copy := orig_instance.createSchemaBasedXML (
    'http://www.example.com/test.xsd' );
  result (concat (
    '"schema_based_copy" is schema-based, URI=',
    schema_based_copy.getSchemaURL() ) );
}

schema_assignment_test()
PROBE
VARCHAR
_______________________________________________________________________________

"orig_instance" is declared as non schema-based, URI=
"schema_based_copy" is schema-based, URI=http://www.example.com/test.xsd

2 Rows. -- 00000 msec.
```

See Also

`XMLType.createNonSchemaBasedXML()`

XMLType.createXML

XMLType.createXML — Creates an XML Type instance.

Synopsis

```
static method XMLType.createXML ( in src any ,
                                  in schema_uri varchar ,
                                  in validated integer ,
                                  in wellformed any ) ;
```

Description

The static method creates an XMLType instance from the *src* XML entity. If the parameter *src* is not an XML entity then it is converted to it via an internal call of `xtree_doc()` or `xml_tree_doc()` . A schema may be associated with an XML entity by passing its URI as *schema_uri* ; this schema can be used later to validate the structure of the document.

Parameters

src
: An XML entity or a value that can be converted to an XML entity.

schema_uri
: A URI of the schema of the document. The default is NULL to make result non-schema based.

validated
: An integer flag that indicates if the document is already validated against the schema of the document (this is to avoid redundant validations). The default is 0.

wellformed
: This parameter is unused and is listed solely for compatibility.

Return Types

The method returns a new instance of XMLType.

Examples

Example 24.546. Creating instances of XMLType

The procedure creates two instances (a schema-based and a non schema-based) and demonstrates that these instances are filled with proper data.

```
create procedure createxml_test ()
{
  declare test1 XMLType;
  declare test2 XMLType;
  declare PROBE varchar;
  result_names (PROBE);
  test1 := createXML ('<test attr="value1"/>');
  test2 := createXML ('<test attr="value2"/>', 'http://www.example.com/test.xsd');
  result (concat (
    '"test1" is created as non schema-based, URI=',
    cast (test1.getSchemaURL() as varchar) ) );
  result (concat (
    'Sample data from "test1": value of test/@attr is ',
    test1.extract ('test/@attr') ) );
  result (concat (
    '"test2" is created as schema-based, URI=',
    test2.getSchemaURL() ) );
  result (concat (
```

```
    'Sample data from "test2": value of test/@attr is ',
    test2.extract ('test/@attr') ) );
}

createxml_test ()
PROBE
VARCHAR
_______________________________________________________________________________

"test1" is created as non schema-based, URI=
Sample data from "test1": value of test/@attr is value1
"test2" is created as schema-based, URI=http://www.example.com/test.xsd
Sample data from "test2": value of test/@attr is value2

4 Rows. -- 00000 msec.
```

See Also

`xtree_doc()`

`xper_doc()`

XMLType.existsNode

XMLType.existsNode — Tests node existence having evaluated given XPATH expression.

Synopsis

XMLType.existsNode (in *xpath_expn* varchar ,
in *namespace_map* varchar) ;

Description

The member function calculates the given XPATH expression *xpath_expn* using the XML entity of the instance as a context node. If a *namespace_map* parameter is given then the function adds namespace declarations from this parameter into the beginning of *xpath_expn* before the evaluation. The function returns 1 if the first result of XPATH evaluation is a node or 0 if there are no results or if the first result is not a node.

Parameters

xpath_expn
: A text of XPATH expression to evaluate.

namespace_map
: A string that is a list of namespace prefix declarations delimited by whitespace characters. If the xpath_expn expression also contains a list of declarations then these two lists are concatenated.

Return Types

The function returns integer 1 or 0

Examples

Example 24.547. A table with an XMLType column

This populates a table with XML data and then reports which rows contain 'empno' element inside 'emp'.

```
create table TEST_XMLS  (I integer primary key, XMLVAL XMLType);
insert into TEST_XMLS  values (1, XMLType('<emp><empno>221</empno><ename>John</ename></emp>'));
insert into TEST_XMLS  values (2, XMLType('<po><pono>331</pono><poname>PO_1</poname></po>'));
insert into TEST_XMLS  values (3, XMLType('<oil-rig
xml:id="14a" water="0.413"/><oil-rig
xml:id="14b" water="0.402"/>'));
select e.I, e.XMLVAL.existsNode('/emp/empno')
  from TEST_XMLS  as e
I                 callret
INTEGER NOT NULL  INTEGER
_______________________________________________________________________________

1                 1
2                 0
3                 0

3 Rows. -- 00000 msec.
```

See Also

`XMLType.extract()`

XMLType.extract

XMLType.extract — Evaluates given XPATH expression.

Synopsis

XMLType.extract (in *xpath_expn* varchar ,
in *namespace_map* varchar) ;

Description

The member function calculates the given XPATH expression *xpath_expn* using the XML entity of the instance as a context node. If a *namespace_map* parameter is given then the function adds namespace declarations from this parameter into the beginning of *xpath_expn* before the evaluation. The function returns the first result of the XPATH evaluation or NULL if there are no results. If the result is an XML entity then it is converted into a non-schema based instance of XMLType.

Parameters

xpath_expn
: A text of XPATH expression to evaluate.

namespace_map
: A string that is a list of namespace prefix declarations delimited by whitespace characters. If the xpath_expn expression also contains a list of declarations then these two lists are concatenated.

Return Types

The function returns a non-schema based XMLType instance, NULL or a value of any other type

Examples

Example 24.548. A table with an XMLType column

This populates a table with XML data and then selects 'ename' element from the appropriate record.

```
create table TEST_XMLS  (I integer primary key, XMLVAL XMLType);
insert into TEST_XMLS  values (1, XMLType('<emp><empno>221</empno><ename>John</ename></emp>'));
insert into TEST_XMLS  values (2, XMLType('<po><pono>331</pono><poname>PO_1</poname></po>'));
insert into TEST_XMLS  values (3, XMLType('<oil-rig
xml:id="14a_01" water="0.413"/><oil-rig
xml:id="14b_01" water="0.402"/>'));
select e.XMLVAL.extract ('//ename').getCLobVal() as ENAME
  from TEST_XMLS  as e
  where e.XMLVAL.existsNode('/emp/empno')
ENAME
NVARCHAR
_______________________________________________________________________________

<ename>John</ename>

1 Rows. -- 00000 msec.
```

See Also

`XMLType.existsNode()`

XMLType.getClobVal

XMLType.getClobVal — This function returns the serialization of the current node like XPATH function 'serialize()', i.e. a text in XML syntax.

Synopsis

XMLType.getClobVal ( );

Description

This function returns the serialization of the current node. This is similar to the XPATH function 'serialize()', i.e. a text in XML syntax.

Return Types

The function returns wide string.

Examples

Example 24.549. A table with an XMLType column

This populates a table with XML data and then selects string representation of 'ename' element from the appropriate record.

```
create table TEST_XMLS  (I integer primary key, XMLVAL XMLType);
insert into TEST_XMLS  values (1, XMLType('<emp><empno>221</empno><ename>John</ename></emp>'));
insert into TEST_XMLS  values (2, XMLType('<po><pono>331</pono><poname>PO_1</poname></po>'));
insert into TEST_XMLS  values (3, XMLType('<oil-rig
xml:id="14a_02" water="0.413"/><oil-rig
xml:id="14b_02" water="0.402"/>'));
select e.XMLVAL.extract ('//ename').getCLobVal() as ENAME
  from TEST_XMLS  as e
 where e.XMLVAL.existsNode('/emp/empno')
ENAME
NVARCHAR
_______________________________________________________________________________

<ename>John</ename>

1 Rows. -- 00000 msec.
```

See Also

`XMLType.getStringVal()`

`XMLType.getNumVal()`

XMLType.getNamespace

XMLType.getNamespace — Returns the namespace of the top level element providing the instance is schema-based rather than a fragment.

Synopsis

```
XMLType.getNamespace (   );
```

Description

The member function returns the namespace URI of the top level element providing that the given instance is schema-based and is well-formed. This function returns NULL if the instance is a fragment. If there are many top level elements then they may have different namespace URIs and if there is no top level element then there is nothing to return. It also returns NULL if the instance is non-schema based for compatibility.

Return Types

This function returns a wide string or NULL.

Examples

Example 24.550. Factors that affect the result of getNamespace()

The following statements demonstrate how various circumstances may force getNamespace() to return NULL.

```
-- First of all, top level element may have a name with no namespace.
select XMLType('<local />').getNamespace()
_____________________________________________________________________________
NULL

-- Compatibility: the instance should be schema-based.
select XMLType('<z:qname xmlns:z="http://www.example.com/"/>').getNamespace()
_____________________________________________________________________________
NULL

-- This is schema-based, but top level element has no namespace.
select XMLType('<local />', 'http://www.example2.com/test.xsd').getNamespace()
_____________________________________________________________________________
NULL

-- Note that for XPATH processor return an empty string, not a NULL,
-- as a namespace uri of a name with no URI.
select xpath_eval('namespace-uri(.)', XMLType('<local />', 'http://www.example2.com/test.xsd'))
_____________________________________________________________________________

1 Rows. -- 00000 msec.

-- Schema-based instance with nonempty namespace URI.
select XMLType('<z:qname xmlns:z="http://www.example.com/"/>', 'http://www.example2.com/test.xsd').getNam
_____________________________________________________________________________
http://www.example.com/

-- It also works with default namespace URIs.
select XMLType('<qname xmlns="http://www.example.com/"/>', 'http://www.example2.com/test.xsd').getNamespa
_____________________________________________________________________________
http://www.example.com/

-- Finally, it returns NULL for fragments even if all top-level elements of
-- a fragment have identical namespace URIs.
select XMLType('
    <z:qname xmlns:z="http://www.example.com/"/>
    <z:qname xmlns:z="http://www.example.com/"/>',
  'http://www.example2.com/test.xsd').getNamespace()
_____________________________________________________________________________
```

```
NULL
```

See Also

`XMLType.isFragment()`

XMLType.getNumVal

XMLType.getNumVal — This function returns the integer-value of the current node like XPATH function 'number()'.

Synopsis

```
XMLType.getNumVal (   );
```

Description

This function returns the integer-value of the current node. This is similar to the XPATH function 'number()'.

Return Types

The member function returns integer.

Examples

Example 24.551. A table with an XMLType column

This populates a table with XML data and then selects the numeric value of an 'empno' element from the appropriate record.

```
create table TEST_XMLS  (I integer primary key, XMLVAL XMLType);
insert into TEST_XMLS  values (1, XMLType('<emp><empno>221</empno><ename>John</ename></emp>'));
insert into TEST_XMLS  values (2, XMLType('<po><pono>331</pono><poname>PO_1</poname></po>'));
insert into TEST_XMLS  values (3, XMLType('<oil-rig
xml:id="14a_03" water="0.413"/><oil-rig
xml:id="14b_03" water="0.402"/>'));
select e.XMLVAL.extract ('//empno').getNumVal() as EMPNO
  from TEST_XMLS  as e
  where e.XMLVAL.existsNode('/emp/empno')
EMPNO
DECIMAL
_______________________________________________________________________________

221

1 Rows. -- 00000 msec.
```

See Also

`XMLType.getStringVal()`

`XMLType.getClobVal()`

XMLType.getRootElement

XMLType.getRootElement — Returns top-level element of the given instance (NULL for fragment)

Synopsis

```
XMLType.getRootElement (   );
```

Description

If the given instance is well-formed then this function will return the top-level element of the document that is stored within the instance. If the given instance is a fragment then NULL will be returned because there may be no top-level elements or too many of them.

Note that in spite of this functions name this function actually returns the a top-level node rather than not a root node. According to the W3C XPATH standards, the root element is an implicit node whose children are top-level elements, comments, processing instructions and maybe text nodes. E.g. if a correct HTML document is started by tag <HTML> and ended by corresponding </HTML> tag then the only top-level node is the "HTML " element node and this node is a single child of the root node.

If the given instance is well-formed then the function returns a top-level element of the document that is stored in the instance. If the given instance is fragment then NULL is returned, because there may be no top-level elements or too many of them.

If the given instance is schema-based then the returned instance is based on the same schema.

Return Types

The function returns an XMLType instance.

Examples

Example 24.552. A table with an XMLType column

First select statement lists well-formed documents; second one lists the only fragment in the table.

```
create table TEST_XMLS  (I integer primary key, XMLVAL XMLType);
insert into TEST_XMLS  values (1, XMLType('<emp><empno>221</empno><ename>John</ename></emp>'));
insert into TEST_XMLS  values (2, XMLType('<po><pono>331</pono><poname>PO_1</poname></po>'));
insert into TEST_XMLS  values (3, XMLType('<oil-rig
xml:id="14a_04" water="0.413"/><oil-rig
xml:id="14b_04" water="0.402"/>'));

select e.I, e.XMLVAL.getRootElement().getClobVal()
  from TEST_XMLS as e
  where e.XMLVAL.getRootElement() is not null
I                 callret
INTEGER NOT NULL  NVARCHAR
_______________________________________________________________________________

1                 <emp><empno>221</empno><ename>John</ename></emp>
2                 <po><pono>331</pono><poname>PO_1</poname></po>

2 Rows. -- 00000 msec.

select e.I, e.XMLVAL.getClobVal()
  from TEST_XMLS as e
  where e.XMLVAL.getRootElement() is null
I                 callret
INTEGER NOT NULL  NVARCHAR
_______________________________________________________________________________

3                 <oil-rig
xml:id="14a_05" water="0.413" /><oil-rig
xml:id="14b_05" water="0.402" />
```

```
1 Rows. -- 00000 msec.
```

See Also

`XMLType.isFragment()`

XMLType.getSchemaURL

XMLType.getSchemaURL — Returns the URL of the XML schema definition for schema based instances, NULL for non-schema based.

Synopsis

```
XMLType.getSchemaURL (  );
```

Description

The function returns the URL of the XML Types associate XML schema. This applies only to schema based instances. It will return NULL for non-schema based types.

Return Types

The function returns string.

Examples

Example 24.553. Simple test

The call of XMLType constructor creates an schema-based instance and `getSchemaURL()` returns the specified URL

```
select XMLType('<sample />', 'file://xmlschema/test0001/clean.xsd').getSchemaURL()
callret
VARCHAR
_______________________________________________________________________________

file://xmlschema/test0001/clean.xsd

1 Rows. -- 00000 msec.
```

See Also

`XMLType.isSchemaBased()`

XMLType.getStringVal

XMLType.getStringVal — The function returns the string-value of the current node like XPATH function 'string()'.

Synopsis

```
XMLType.getStringVal (   );
```

Description

The function returns the string-value of the current node. This is similar to the XPATH function 'string()'.

Return Types

The member function returns wide string.

Examples

Example 24.554. A table with an XMLType column

This populates a table with XML data and then selects the string-value of an 'ename' element from the appropriate record.

```
create table TEST_XMLS  (I integer primary key, XMLVAL XMLType);
insert into TEST_XMLS  values (1, XMLType('<emp><empno>221</empno><ename>John</ename></emp>'));
insert into TEST_XMLS  values (2, XMLType('<po><pono>331</pono><poname>PO_1</poname></po>'));
insert into TEST_XMLS  values (3, XMLType('<oil-rig
xml:id="14a_06" water="0.413"/><oil-rig
xml:id="14b_06" water="0.402"/>'));
select e.XMLVAL.extract ('//ename').getStringVal() as ENAME
  from TEST_XMLS  as e
  where e.XMLVAL.existsNode('/emp/empno')
ENAME
NVARCHAR
_______________________________________________________________________________

John

1 Rows. -- 00000 msec.
```

See Also

`XMLType.getNumVal()`

`XMLType.getClobVal()`

XMLType.isFragment

XMLType.isFragment — The function returns 1 if the instance is an XML generic entity or 0 if it is a plain document.

Synopsis

```
XMLType.isFragment (   );
```

Description

The function checks the number of children elements of the root of the document root of the given instance. If there are no such documents or there are many of them then the document is not well-formed, for example it can not be fully validated against an XML schema. However it can be used as a fragment of a larger document: it can be declared as an external generic entity and then referenced in a top-level document or in one of its subdocuments. For example a book can be stored in XML as a root document that includes subdocuments, one or more chapter per subdocument.

The function returns 1 if the given XMLType instance is a fragment and returns 0 if it is a well-formed XML document.

Return Types

Integer value 1 or 0.

Examples

Example 24.555. Basic test

The table DB.DBA.SYS_USERS always contains more than one user record so the call of XMLAGG that returns one element per record creates an valid generic entity but not a valid document.

```
select XMLAGG (XMLELEMENT ("User",U_NAME)) from DB.DBA.SYS_USERS;
callret
VARCHAR
_______________________________________________________________________________

<User>BACKUP</User><User>SOAP</User><User>dba</User><User>demo</User>

1 Rows. -- 00000 msec.
select XMLType (XMLAGG (XMLELEMENT ("User",U_NAME))).isFragment() from DB.DBA.SYS_USERS;
callret
INTEGER
_______________________________________________________________________________

1

1 Rows. -- 00000 msec.
```

See Also

`XMLType.getRootElement()`

XMLType.isSchemaBased

XMLType.isSchemaBased — Returns 1 if the given XMLType instance is schema-based, otherwise returns 0.

Synopsis

```
XMLType.isSchemaBased (   );
```

Description

Returns 1 if the given XMLType instance is schema-based, i.e. if it has a URI of an XML schema that can be used for schema validation via `XMLType.schemaValidate()` .

Return Types

The function returns 1 or 0.

Examples

Example 24.556. Simple tests

The following three calls display the behaviour of the `isSchemaBased()` . Note that it does no actual checks whether the document is valid against the declared schema or even the validity of schema URI itself.

```
select XMLType('<clean />').isSchemaBased()
callret
INTEGER
_______________________________________________________________________________

0

1 Rows. -- 00000 msec.

select XMLType('<clean />', 'file://xmlschema/test0001/clean.xsd').isSchemaBased()
callret
INTEGER
_______________________________________________________________________________

1

1 Rows. -- 00000 msec.

select XMLType('<clean />', 'http://no.such.schema.exists').isSchemaBased()
callret
INTEGER
_______________________________________________________________________________

1

1 Rows. -- 00000 msec.
```

See Also

`XMLType.getSchemaURL()`

XMLType.isSchemaValid

XMLType.isSchemaValid — Validates the given XMLType instance against an arbitrary XML schema, returns 1 if successful, 0 if errors are detected.

Synopsis

XMLType.isSchemaValid (in *schema_url* varchar ,
in *top_element_name* varchar) ;

Description

The function validates the given XMLType instance against the XML schema located at *schema_url* . The name of the current node of the XML entity should match *top_element_name* if it is specified.

The *schema_url* is optional for schema based instances: the default value is the URI of the associated schema of the instance. For non-schema based instances the *schema_url* is required, an error is signalled otherwise.

The function does not use or modify internal "validated" flag that is e.g. used by XMLType.schemaValidate() . It is true even if the given *schema_url* is equal to the URI of own schema of the instance.

Parameters

schema_url
: The URI of the schema to apply.

top_element_name
: The allowed name of top level element of the instance.

Return Types

The function returns 1 if the validation is passed, 0 if it is failed.

Examples

Example 24.557. Failed schema validation

The example creates an XMLType instance that is a document with a single element node"bad" and tries to validate it against XMLschema "file://xmlschema/test0001/clean.xsd" that does not match actual content of the document. The call of isSchemaValid() returns zero indicating failed validation.

```
select XMLType('<bad />', 'file://xmlschema/test0001/clean.xsd').isSchemaValid()
callret
INTEGER
_______________________________________________________________________________

0

1 Rows. -- 00000 msec.
```

See Also

XMLType.isSchemaValidated()

XMLType.setSchemaValidated()

XMLType.schemaValidate()

XMLType.isSchemaValidated

XMLType.isSchemaValidated — Returns 1 or 0 indicating if the XML entity has been validated against the associated schema.

Synopsis

```
XMLType.isSchemaValidated (   );
```

Description

The function returns the value of a special internal "is validated" flag of the given XMLType instance.

Return Types

An integer value 1 or 0.

Examples

Example 24.558. Sample use

These two calls demonstrate how the "is validated" flag can be set without making an actual validation in the call of XMLType constructor:

```
-- Line 75: select XMLType('<bad />', 'file://xmlschema/test0001/clean.xsd').isSchemaValidated()
callret
INTEGER
_______________________________________________________________________________

0

1 Rows. -- 00000 msec.

-- Line 77: select XMLType('<bad />', 'file://xmlschema/test0001/clean.xsd'), 1).isSchemaValidated()
callret
INTEGER
_______________________________________________________________________________

1

1 Rows. -- 00000 msec.
```

See Also

`XMLType.setSchemaValidated`

`XMLType.schemaValidate`

XMLType.schemaValidate

XMLType.schemaValidate — Validates the schema based XMLType instance against its schema and signals an error in case of failed validation.

Synopsis

```
XMLType.schemaValidate (   );
```

Description

The member function signals an error if called for a non-schema based instance. If an instance is schema-based but the validation has already been done, the call has no effect. Otherwise, a schema-based instance is validated against its schema. If the validation fails, an error is signalled. If the validation is successful then a special internal "validated"flag is set.

Return Types

The function returns a string that contains a validation log if an instance has not been validated before, otherwise it returns null or signals an error.

Examples

Example 24.559. Failed schema validation

The example creates an XMLType instance that is a document with a single element node"bad" and declared schema "file://xmlschema/test0001/clean.xsd" that does not match actual content of the document. The call of `schemaValidate()` signals an error.

```
select XMLType('<bad />', 'file://xmlschema/test0001/clean.xsd').schemaValidate();

*** Error 42000: [Virtuoso Driver][Virtuoso Server]Top-level element name <q> is unknown
at line 76 of load 'XmlType.sql':
select XMLType('<bad />', 'file://xmlschema/test0001/clean.xsd').schemaValidate()
```

See Also

`XMLType.isSchemaValidated()`

`XMLType.setSchemaValidated()`

`XMLType.isSchemaValid()`

XMLType.setSchemaValidated

XMLType.setSchemaValidated — Changes the internal "is validated" flag of the given XMLType instance.

Synopsis

`XMLType.setSchemaValidated` (in *flag* integer default 1);

Description

Every XMLType instance has a special internal "is validated" flag; The first successful call of the member function `XMLType.schemaValidate()` will set this flag to 1 indicating that next such calls are redundant and should do nothing. If a given XMLType instance is made by a procedure that guarantees the match of the result to an expected schema then one can set this flag without running actual validation. It is also possible to reset this flag to 0 in order to force the next call of `XMLType.schemaValidate()` to perform a validation.

Parameters

flag
: An integer that becomes a new value of the flag.

Return Types

The function returns a new value of the flag.

Examples

Example 24.560. Default use

This call prevents the value of myXMLinstance from being validated in the future.

```
myXMLinstance.setSchemaValidated();
```

See Also

`XMLType.isSchemaValidated`

`XMLType.schemaValidate`

XMLType.toObject

XMLType.toObject

Synopsis

```
XMLType.toObject ( in schema varchar ,
                   in element varchar ) ;
```

Description

This member function is not yet implemented.

XMLType.transform

XMLType.transform

Synopsis

XMLType.transform (*stylesheet* XMLType ,
params_map vector) ;

Description

This member function is not implemented and signals an error if called.

Use `xslt()` built-in function instead.

See Also

`xslt()`

sql:column

sql:column — Returns the value of a column from SQL result-set.

Synopsis

```
any sql:column ( constant column_name varchar ) ;
```

Description

This is actually not a function but a special macro that is converted to a reference to a global parameter.

Parameters

column_name
: A string constant (literal, not an expression) that is a column name in plain SQL syntax. It can be a column name (and it can be qualified to specify a table name or an alias name). It can be a name of a Virtuoso/PL variable.

Return Types

The value of the sql:column() invocation is the value of some SQL column or variable. If the type of the value is not supported by XQuery interpreter then it will be converted according to generic rules for XQuery parameters.

Examples

Example 24.561. Various invocations of sql:column

These two statements will produce identical results:

See Also

Pre-compilation of XPath and XQuery Expressions

and

and — Returns false if a value of some argument is false, otherwise returns true.

Synopsis

```
boolean and ( val1 boolean ,
              val2 boolean ,
              ... ,
              valN boolean ) ;
```

Description

This function calculates the values of its arguments from left to right. If the value of the calculated parameter is false, the function returns false immediately, without calculating the remaining parameters. If the list of arguments ends without any false value calculated, the function returns true (thus it will return true if called without arguments).

The name of this function is the same as the name of "and" XPATH and XQUERY operator. Thus it must be surrounded by double quotes when used in XPATH or XQUERY expressions. Moreover, this function is not a part of XPATH standard, so it cannot be used if portability is important.

Parameters

vali
: Value of boolean expression argument

Return Types

boolean

Examples

Example 24.562. Control over sequence of search operations

If two conditions must be checked, where one is simple and another is hard to calculate, then "and" may be used to calculate second condition only if first is true, to reduce average time of processing

```
"and" ( authors,
  document(concat('http://www.lib20.org/findxml.cgi?isbn=',@isbn))/status[@outofprint='NO'] )
```

See Also

not or every

append

append — Creates an sequence of all items from given sequences.

Synopsis

```
sequence append ( seq1 sequence ,
                  seq2 sequence ,
                  ... ,
                  seqN sequence ) ;
```

Description

This function calculates all given arguments from left to right, and creates a sequence which contains all items of the first calculated sequence, then all items of the second calculated sequence and so on, preserving the order of items from every sequence. The result is identical to the result of XQUERY "comma operator".

This function is not a part of XPATH 1.0 or XQUERY 1.0 libraries of standard functions.

Parameters

SeqI
: The sequence of items to be placed into the resulting sequence

Return Types

Sequence

Examples

Example 24.563.

```
append(/abstract, /chapter, /appendix/section)
```

See Also

tuple

assign

assign — Creates a local variable and assign a value to it.

Synopsis

```
null assign ( var_name string ,
              var_value any ) ;
```

Description

This function calculates the first argument, *var_name* , and converts it to the string, if needed. Then it checks if there is a local variable or parameter with such name. If not found, it checks if there is a global variable or parameter with such name. If nothing found in both cases, a new local variable is created with this name. The the value of found or created variable is changed to the value of the second argument, *var_value* .

The value of the variable may be accessed like the value of any variable created by <xsl:variable> XSL element or FLWR operator of XQUERY. The same $name should be used to get the value.

In XSLT, common rules for local variables are used for variables created by assign() function.

This function is not a part of XPATH 1.0 or XQUERY 1.0 libraries of standard functions. It may produce strange results if used in XQUERY expressions in order to change the value of local variable created by FLWR expression. It may cause infinite loop if used in XSLT expressions in order to change the value of a variable used in currently executed <xsl:apply-templates> instruction. For both safety and portability, it is much better to use <xsl:variable> and <xsl:param> XSL elements in stylesheets or LET operator in XQUERY expressions. In addition, let() function is not portable, but it may be used in XPATH or XSLT without the risk of undesired effects.

Parameters

var_name
: The name of variable which must be found or created.

var_value
: The value which should be assigned to the variable.

Return Types

Null

Examples

Example 24.564.

Increment local variable $counter by 1.

```
assign('counter', $counter+1);
```

See Also

let()

avg

avg — Returns average value of all its arguments.

Synopsis

```
number avg ( arg1 any ,
             arg2 any ,
             ... ,
             argN any ) ;
```

Description

The function returns the average of all values in all its arguments, For each node in every argument node-set, it converts the string-value of the node to a number and adds the result to the sum. If some arguments are not node-sets, they are converted to numbers first and added to the sum. Then sum is divided by number of values added and the result is returned.

This function is not a part of XPATH 1.0 standard library.

Parameters

argI
: A node-set of nodes whose string values must be converted to numbers and added to the result, or single value.

Return Types

Table 24.138. Errors signalled by

SQLState	Error Code	Error Text	Description
XP001	XPF06	Nonempty sequence of values expected as argument(s) of XPATH function avg(); avg() of nothing is senseless	This happens if the function is called without arguments, or with all arguments set to empty node-sets.

Examples

Example 24.565.

```
avg(/report[@type="daily-sales"]/total)
```

See Also

sum max min

boolean

boolean — Converts its argument to boolean

Synopsis

```
boolean boolean ( obj any );
```

Description

The function converts its argument to a boolean as follows:

- A number is true if and only if it is neither zero nor NaN.
- A node-set is true if and only if it is non-empty.
- A string is true if and only if its length is non-zero.
- An object of a type other than the four basic types is converted to a boolean in a way that is dependent on that type.

Parameters

obj
: The object to be converted into boolean.

Return Types

Boolean

Examples

Example 24.566.

The following expressions are true:

```
boolean('aaa') = true()
boolean('false') = true()
boolean(false()) = false()
boolean(.) = true()
boolean('0') = true()
boolean(0) = false()
```

See Also

string() number()

ceiling

ceiling — Returns the smallest integer that is not less than the argument.

Synopsis

```
integer ceiling ( num number );
```

Description

This function returns the smallest (closest to negative infinity) number that is not less than the argument and that is an integer. In other words, it "rounds up" the given value.

Parameters

num
: The value to be "rounded up"

Return Types

Examples

Example 24.567.

The following expressions are true:

```
ceiling(2.5) = 3
ceiling(2) = 2
ceiling(-2.5) = -2
ceiling(-2) = -2
```

See Also

floor() round()

fn:collection

fn:collection — Returns parsed documents contained in given collections.

Synopsis

```
node()* fn:collection ( uri anyURI ,
                        base_uri string ,
                        recursive_mode int ,
                        parser_mode int ,
                        encoding string ,
                        language string ,
                        dtd_config string ) ;
```

Description

This function takes one or more collection URI's and returns the parsed documents contained in these collections as a sequence.

If no uri is specified, the function returns the sequence of the nodes in the default collection in the dynamic context. The default collection is a home DAV collection. If user does not have default DAV collection, an error is signalled.

Local DAV collections
: Local DAV collections can be accessed either by providing "http://local.virt/DAV/" or "http://localhost:PORT/" URI. The "http://local.virt/DAV/" can be used even if the local http server is not enabled. The "http://localhost:PORT" URI is less efficient and takes the data over HTTP, so "http://local.virt/DAV/" is preferred.

Table Collections
: fn:collection specially recognizes URI's which beginning with "virt://". Such a URI relates to a local table, the content of a specified column is returned as the collection (table collection). The URI contains three parts: table name, id column name, xml content column name. For instance, if the table definition is:

Home path in local DAV collections
: fn:collection also supports UNIX style home paths. If URI begins with ~ the next term will treated as user name and is will be substituted by path to home DAV collection. For instance, ~john/test/test2/ can be parsed as http://local.virt/DAV/home/john/test/test2 URI.

Remote DAV collections, WEB collections
: Any other URI which begins with "http://" refers to remote DAV collections. The function uses the PROPFIND DAV method to get the list of documents contained in the collection. If the remote server does not support this method the function tries to do a HTML GET with the uri and returns all documents referenced from the result (WEB collection). In this case the function makes two http requests.
When a web page is scanned for document reference by fn:collection it searches <a href=...> tags and returns downloaded and parsed documents referenced in href. If href contains relative reference it is resolved using web page address as base uri.
In this release fn:collection does not support any collections except DAV collections, WEB collections, table collections.

Authentication
: Remote DAV and WEB collections may need authentication information. fn:collection supports two ways of providing authentication information. First one is to provide user name and password by setting appropriate connection variables:

Parameters

uri
: collection URI. If the uri is a relative xs:anyURI, it is resolved against the value of the base-URI property from the static context or from base_uri .

base_uri

If uri is an relative URI string, not an absolute one then base_uri is used to make it absolute.

parser_mode
: Sets the mode for parsing documents in collection (0 - XML parser mode, 1 - HTML parser mode, 2 - 'dirty HTML' mode (with quiet recovery after any syntax error).

encoding
: string with content encoding type of <document>; valid are 'ASCII', 'ISO', 'UTF8', 'ISO8859-1', 'LATIN-1' etc., defaults are 'UTF-8' for XML mode and 'LATIN-1' for HTML mode.

language
: string with language tag of content of <document>; valid names are listed in IETF RFC 1766, default is 'x-any' (it means 'mix of words from various human languages')

dtd_config
: configuration string for DTD validator, default is empty string meaning that DTD validator should be fully disabled. See Configuration Options of the DTD Validator for details.

Return Types

node*

Examples

Example 24.568. Example of using remote DAV collection:

```
<result>
  { for $a in fn:collection ( "http://www.somdavservice.com:8081/DAV/test1/" )
        return
                <res>{ $a/value/text() }</res>
  }
</result>
```

Example 24.569. Example of using relative URI:

```
xquery_eval('
 <pre>
    { for \044n in fn:collection("/docs", ., 2)
          return (document-get-uri(\044n), " - ", name (\044n/*[1]), "\n")
    }
 </pre> ', xtree_doc ('<stub/>', 0, 'http://www.somdavservice.com:8081/') );
```

Example 24.570. Example of using default collection:

```
xquery_eval('
 <div>
    { for \044anchor in fn:collection()//a
          return <b> { \044anchor } </b>
    }
 </div> ', xtree_doc ('<stub/>') );
```

See Also

document()

doc()

xtree_doc()

xml_uri_get()

concat

concat — Returns the concatenation of its arguments.

Synopsis

```
string concat ( strg1 string ,
                strg2 string ,
                ... ,
                strgN string ) ;
```

Description

The function converts all its arguments into strings using the same rules as XPATH function string(), then it performs concatenation and returns the resulting string.

XPATH 1.0 standard states that concat() function must have at least 2 arguments, but in Virtuoso XPATH this restriction is eliminated. concat() may be called without arguments (it will return an empty string) or with one argument (it will work like string() function). This may be useful if the text of XPATH expression must be generated by an application.

Parameters

strgI
: String or a value of any type which may be converted into a string.

Return Types

Examples

Example 24.571.

```
concat('Order ', @orderno, ' dated ', @issuedate)
```

See Also

string()

contains

contains — Returns true if the first argument string contains the second argument string, and otherwise returns false.

Synopsis

```
boolean contains ( haystack string ,
                   needle string ) ;
```

Description

For two given strings, this function checks if the first string contains the second string. If any argument is not a string, it is converted to string using rules from string() XPATH function. Thus if the second argument has no string value, the function returns true, because it will be converted to an empty string first.

Parameters

haystack
: String where the search is performed

needle
: String to search

Return Types

Boolean

Examples

Example 24.572.

```
contains ('OpenLink Virtuoso', 'Link')
```

See Also

string() starts-with() ends-with()

count

count — Returns the number of values in the sequence.

Synopsis

```
integer count ( seq any );
```

Description

Returns 1 if the argument is a single value or a count of elements in the given sequence of values.

This function must be called with an argument, it do nothing with context. To count nodes in context node-set, use last().

Parameters

seq
: Sequence of values to be counted.

Return Types

Integer

Errors

This function never returns any errors

Examples

Example 24.573.

```
count(//*)
```

See Also

empty() last()

create-attribute

create-attribute — Creates a special "attribute descriptor" object.

Synopsis

```
attribute create-attribute ( attrname string ,
                             attrvalue string ) ;
```

Description

This is an internal XQUERY function and you will probably never need to call it explicitly. It is not a part of library of standard XQUERY 1.0 functions.

This function creates an special object, called "attribute descriptor", which may be used only as argument of create-element() XQUERY function. Every attribute descriptor will force create-element() function to add an attribute with specified name and value into opening tag of the created element. If attribute with such name already exists, its value will be replaced.

Parameters

attrname
: Name of attribute which should be created or changed

attrvalue
: Value of of attribute

Return Types

Attribute descriptor

See Also

create-comment()

create-element()

create-pi()

create-comment

create-comment — Creates a "comment" XML tree entity.

Synopsis

```
entity create-comment ( comment-text string );
```

Description

This is an internal XQUERY function and you will probably never need to call it explicitly. It is not a part of library of standard XQUERY 1.0 functions.

This function creates an XML entity of sort "comment" that contains given text.

Parameters

comment-text
: The text content of the comment

Return Types

XML tree entity

See Also

create-attribute()

create-element()

create-pi()

create-element

create-element — Creates an element with specified name, attributes and children

Synopsis

```
node create-element ( head sequence ,
                      child1 any ,
                      child2 any ,
                      ... ,
                      childN any ) ;
```

Description

This is an internal XQUERY function and you will probably never need to call it explicitly. It is not a part of library of standard XQUERY 1.0 functions.

This function creates a new "XML Tree" element whose name is a string-value of the first item of *head* sequence, with attributes and children specified by the rest of *head* sequence and by the list of arguments *child1* , *child2* , ... *childN* .

First of all, a new element will be created, without attributes or children. The name of element will be taken from the first item of the *head* sequence. Then attributes will be created from the second and third items of *head* , from fourth and fifth and so on. In every pair of items will specify name and value of some attribute. Non-string items will be converted to strings first. It is an error to specify the same attribute name twice in *head* sequence.

When the "opening tag" of the element is prepared, children are added, in the same order as they are specified by arguments *child1* , *child2* , ... *childN* . If the value of some argument is a sequences (e.g. a node-set), items of the sequence are added as separate children in the same order as they are in the sequence. Nodes are added "as-is", numbers, strings and other "scalar" values are converted to strings first and these strings are converted into PCDATA (text) children.

"Attribute descriptor" objects are not converted to children elements; if descriptor item is found, one attribute in the opening tag of the created element is added or changed and the descriptor is removed from the list of children.

When all children are prepared, some normalization is performed. If there are two or more adjacent PCDATA (text) children, they are replaced with one PCDATA children whose text is a concatenation of texts of all that children.

Parameters

head
: Name of the element or a sequence of name and attributes of the element.

childI
: Children node or sequence of children node.

Return Types

Node

Errors

Table 24.139. Errors signalled by

SQLState	Error Code	Error Text	Description
XP001	XPFC0	At least one argument (name of element to be created) must be passed to create-element XPATH function.	create-element is called without arguments.

SQLState	Error Code	Error Text	Description
XP001	XPFC1	No name of element in the first argument of create-element XPATH function.	The *head* sequence is empty.
XP001	XPFC2	Last attribute has no value specified in the first argument of create-element XPATH function.	The *head* sequence is of even length.
XP001	XPFC3	Unsupported type of element of the first argument of create-element XPATH function.	Current version may create name and attributes of elements only from strings, entities and numbers.
XP001	XPFC4	Duplicate attribute names in first argument of create-element XPATH function.	Duplicate attribute names may appear in attribute descriptors but not in the *head* argument.
XP001	XPFC5	First argument of create-element XPATH function must be string, symbol or sequence of them.	The function was unable to prepare name of element.
XP001	XPFC6	Error in XPATH user extension function or internal error: sequence argument is not flat in create-element XPATH function.	A value of XQUERY "sequence" type should not contain other sequences as its items, i.e. it must be "flat".
XP001	XPFC7	Invalid special entity found in argument of create-element XPATH function.	An invalid object (probably built by XPATH extension function) is specified as child entity.
XP001	XPFCA	Persistent XML entities are not fully supported by create-element XPATH function.	The function creates "XML Tree" element, "Persistent XML" entity can not become a children of "XML Tree".
XP001	XPFCB	Unsupported type of argument in create-element XPATH function.	Some argument or some item of an argument may not be converted to a string or a node, so a children may not be created.

Examples

Example 24.574.

Two following XQUERY expressions are equivalent:

```
<H1>{'Hello, '}<B>{'world'}</B></H1>
create-element('H1', 'Hello, ', create-element('B', 'world'))
```

See Also

create-attribute()

create-comment()

create-pi()

create-pi

create-pi — Creates a "processing instruction" XML tree entity.

Synopsis

```
entity create-pi ( piname string ,
                   pivalue string ) ;
```

Description

This is an internal XQUERY function and you will probably never need to call it explicitly. It is not a part of library of standard XQUERY 1.0 functions.

This function creates an XML entity of sort "processing instruction". If *pivalue* is an empty string, the resulting processing instruction will have no value at all, exactly as if the *pivalue* is not passed at all.

Parameters

piname
: Name of processing instruction that should be created

pivalue
: String data stored in processing instruction

Return Types

XML tree entity of type "processing instruction"

See Also

create-attribute()

create-comment()

create-element()

current

current — Returns a node-set that has the current node as its only member.

Synopsis

```
node-set current ( void);
```

Description

The function returns a node-set that has the current node as its only member. For an outermost expression (an expression not occurring within another expression), the current node is always the same as the context node. For an expression occurring within another expression, e.g. within predicate in some path, the current node is the same as the context node of the first step in the path.

Please refer XSL standard before the first use of this function, to understand exact difference between "current" and "context" node.

XSLT 1.0 states that it is an error to use the current() function in a XSL "pattern", e.g. in "match" attribute of <xsl:key> element, because patterns have no value assigned for current node assigned processing. Instead of reporting the error, Virtuoso's XSLT processor uses context node if current node is not set.

Return Types

Node-set that has exactly one member.

Examples

Example 24.575. Function current() in top-level expression.

These two elements produce the same result:

```
<xsl:value-of select="current()"/>
<xsl:value-of select="."/>
```

Example 24.576. Function current() inside the path.

For current node, which is equal to context node of whole <xsl:apply-templates> key, try to find such item of glossary in the document, that the name of item is equal to "ref" attribute of current node.

```
<xsl:apply-templates select="//glossary/item[@name=current()/@ref]"/>
```

distinct

distinct — Removes double entities from the input sequence

Synopsis

```
sequence distinct ( input sequence );
```

Description

The function takes a single parameter which is sequence of XML entities (nodes or values) and returns the sequence that results from removing from input sequence all but one of a set of elements that are identical each other. If input sequence is the empty, the empty sequence is returned.

Note that the `distinct` is not a part of XPATH 1.0 or XQuery 1.0 standard library, it is rather a generalization of standard `distinct-nodes` and `distinct-values` functions.

Parameters

input
: The sequence of XML entities

Return Types

Sequence

Examples

Example 24.577. Removing double entities

In the following example the file example.xml is

```
<a>
<b/>
<b/>
</a>
```

The result of the query

```
<result>
   {
   distinct(for $r in document("example.xml")//b return $r)
   }
</result>
```

is only one 'b' element:

```
<result>
<b/>
</result>
```

doc

doc — Returns data from XML doc other than the main source document.

Synopsis

```
node-set doc ( document_uri varchar );
```

Description

The function tries to access an XML text at location specified by *document_uri* .

If the *document_uri* argument is node-set, not a string, then a node-set is returned as if document() function is applied to string-value of the first node of the node-set.

The result of call doc($uri) is similar to the call of function document() with default parameters:

```
document($uri, document-uri(.), 0, 'UTF-8', 'x-any', 'Include=ERROR IdCache=ENABLE')
```

I.e. the retrieved document should be XML or XHTML (but not an old-style HTML) in 'UTF-8' encoding, content language is 'x-any' (it means 'mix of words from various human languages'), DTD should be read but validation should be disabled; errors on including subdocuments should be reported as errors and thus should abort the processing; a dictionary of element's IDs should be created in order to support XQuery 'pointer operator'.

Parameters

document_uri
: An absolute or relative URI that points to a well formed XML or HTML document. If the URI is relative, then the base URI of context node is used to convert the given URI into absolute one.

Return Types

Node-set

Errors

The function may signal variety of errors when it reads the requested document(s) from network or from local resources. It may even cause deadlocks e.g. if documents are retrieved from the Virtuoso's own webserver and these documents must be created on the fly from data that are locked by Virtuoso/PL procedure that invokes the XPATH processor.

Examples

Example 24.578. Simple reading of a standalone XML document

Read a standalone document from http://www.example.com/sales/prices.xml

```
doc("http://www.example.com/sales/prices.xml")
```

See Also

document()

document-literal()

xtree_doc()

xper_doc()

document

document — Returns data from XML documents other than the main source document.

Synopsis

```
node-set document ( document_uri varchar ,
                    base_uri varchar ,
                    parser_mode integer ,
                    content_encoding varchar ,
                    content_language varchar ,
                    dtd_validator_config varchar ) ;
```

Description

The function tries to access an XML text at location specified by *document_uri* and optionally *base_uri* . On success, it parses the text and returns the root entity of the "XML Tree" document; the result is identical to the entity created by xtree_doc() Virtuoso/PL function.

If the *document_uri* argument is node-set, not a string, then a node-set is returned as if document() function is applied to string-value of every node of the node-set.

Note that the list of attributes of the function differs from specified in XSLT 1.0 standard. In XPATH 1.0, there is no such function at all.

Parameters

document_uri
: An absolute or relative URI that points to a well formed XML or HTML document. If the URI is relative, then the base_uri must be specified.

base_uri
: The URI that is used to resolve all relative URIs (i.e. to convert them into absolute in order to locate and load subdocuments) and to change 'local' absolute references to relative when in HTML mode.

parser_mode
: 0, 1 or 2; 0 - XML parser mode, 1 - HTML parser mode, 2 - 'dirty HTML' mode (with quiet recovery after any syntax error)

content_encoding
: string with content encoding type of document; valid are 'ASCII', 'ISO', 'UTF8', 'ISO8859-1', 'LATIN-1' etc., defaults are 'UTF-8' for XML mode and 'LATIN-1' for HTML mode.

content_language
: string with language tag of content of document; valid names are listed in IETF RFC 1766, default is 'x-any' (it means 'mix of words from various human languages')

dtd_validator_config
: configuration string for DTD validator, default is "Include=ERROR IdCache=ENABLE" meaning that DTD should be read but validation should be disabled; errors on including subdocuments should be reported as errors and thus should abort the processing; a dictionary of element's IDs should be created in order to support XQuery 'pointer operator'. See Configuration Options of the DTD Validator for details.

Return Types

Node-set

Errors

Table 24.140. Errors signalled by document()

SQLState	Error Code	Error Text	Description
XP001	XPF09	Too many arguments passed to XPATH function document()	The number of parameters may vary only from 1 to 6.
XP001	XPF10	XML entity or a string expected as \"base_uri\" argument of XPATH function document()	If presents, the second argument of the function must be either a string or an XML entity whose XML document has a suitable base URI.

The function may signal variety of errors when it reads the requested document(s) from network or from local resources. It may even cause deadlocks e.g. if documents are retrieved from the Virtuoso's own webserver and these documents must be created on the fly from data that are locked by Virtuoso/PL procedure that invokes the XPATH processor.

Examples

Example 24.579. Simple reading of a standalone XML document

Read a standalone document from http://www.example.com/sales/prices.xml

```
document("http://www.example.com/sales/prices.xml")
```

Example 24.580. Simple reading of a non-standalone document

Read a non-standalone document from local mirror but access its subdocuments as if it is retrieved directly from 'http://www.example.com/sales/prices.xml', so e.g. the subdocument that is referenced as 'termsandconditions.xml' is retrieved from http://www.example.com/sales/termsandconditions.xml but not from http://webcache.localdomain/examplesales/termsandconditions.xml .

```
document("http://webcache.localdomain/examplesales/prices.xml", "http://www.example.com/sales/prices.xml"
```

See Also

doc()

document-literal()

xtree_doc()

xper_doc()

document-literal

document-literal — Parses given XML text and returns the resulting XML data.

Synopsis

```
node-set document-literal ( document_text varchar ,
                            cache_uri varchar ,
                            parser_mode integer ,
                            content_encoding varchar ,
                            content_language varchar ,
                            dtd_validator_config varchar ) ;
```

Description

The function tries to parse an XML text as if it is obtained from a location specified by *cache_uri* . On success, it returns the root entity of the "XML Tree" document; the result is identical to the entity created by xtree_doc() Virtuoso/PL function.

When XPath processor parses a document, it saves it in a temporary cache. If a cached document is accessed again then no actual retrieval or parsing is made and a cached value is returned. The cache persists till the end of execution of a XPath query or till the end of the XSLT transformation if the XPath expression is a part of XSLT stylesheet. The *cache_uri* specifies the URI used as a key if the parsed document is cached. If *cache_uri* is not specified or the specified value is equal to an empty string, then no caching is performed. If the specified *cache_uri* is not equal to an empty string then it should distinct from URIs of other XML resources that are sources of the XPath query, to prevent confusion.

The *cache_uri* is also used by XML parser as a base URI to resolve relative URIs of external entities, so it is a good idea to specify some absolute URI if the parsed text is not a "standalone" document.

The *document_text* argument may be a sequence, not a single string. In this case a node-set is returned as if `document-literal` is applied to string-value of every element of the sequence. It is obvious that caching should not be used if there's a chance that the sequence may have more than one distinct element.

Note that the `document-literal` is not a part of XPATH 1.0 or XQuery 1.0 standard library.

Parameters

document_text
: well formed XML or HTML text

cache_uri
: The URI that is used to resolve all relative URIs (i.e. to convert them into absolute in order to locate and load subdocuments) and to change 'local' absolute references to relative when in HTML mode.

parser_mode
: 0, 1 or 2; 0 - XML parser mode, 1 - HTML parser mode, 2 - 'dirty HTML' mode (with quiet recovery after any syntax error)

content_encoding
: string with content encoding type of document_text ; valid are 'ASCII', 'ISO', 'UTF8', 'ISO8859-1', 'LATIN-1' etc., defaults are 'UTF-8' for XML mode and 'LATIN-1' for HTML mode.

content_language
: string with language tag of content of document_text ; valid names are listed in IETF RFC 1766, default is 'x-any' (it means 'mix of words from various human languages')

dtd_validator_config

configuration string for DTD validator, default is "Include=ERROR IdCache=ENABLE" meaning that DTD should be read but validation should be disabled; errors on including subdocuments should be reported as errors and thus should abort the processing; a dictionary of element's IDs should be created in order to support XQuery 'pointer operator'. See Configuration Options of the DTD Validator for details.

Return Types

Node-set

Errors

Table 24.141. Errors signalled by document-literal()

SQLState	Error Code	Error Text	Description
XP001	XPF15	Too many arguments passed to XPATH function document-literal()	The number of parameters may vary only from 1 to 6.

Examples

Example 24.581. Simple reading of a standalone XML document

Read a short document from string

```
document-literal("
<html><body><p>Hello
</p></body></html>)
```

Example 24.582. Simple reading of a non-standalone document

Read a non-standalone document from local mirror but access its subdocuments as if it is retrieved directly from 'http://www.example.com/sales/prices.xml', so e.g. the subdocument that is referenced as 'termsandconditions.xml' is retrieved from 'http://www.example.com/sales/termsandconditions.xml'

```
document-literal('
<!DOCTYPE price-list SYSTEM "http://www.example.com/dtd/sales.dtd" [
<!ENTITY t_and_c SYSTEM "termsandconditions.xml">
<!ENTITY prices_oct2002 SYSTEM "prices/2002/oct.xml">
]>
<price-list>
&t_and_c;
&prices_oct2002;
</price-list>
',
'http://webcache.localdomain/examplesales/prices.xml' )
```

See Also

document()

xtree_doc()

xper_doc()

empty

empty — Returns true if given argument is an empty sequence, false if it is any single value or nonempty sequence.

Synopsis

```
boolean empty ( seq any );
```

Description

Returns true if given argument is an empty sequence, false if it is any single value or nonempty sequence.

Parameters

seq
: Sequence of values to be checked.

Return Types

Boolean

Errors

This function never returns any errors

Examples

Example 24.583.

```
if(empty(//table),'There are no tables in context node','There is at least one table')
```

See Also

count()

ends-with

ends-with — Returns true if the first argument string ends with the second argument string, and otherwise returns false.

Synopsis

```
boolean ends-with ( strg string ,
                    suffix string ) ;
```

Description

For two given strings, this function checks if the first string ends with characters of second string. If any argument is not a string, it is converted to string using rules from string() XPATH function. Thus if the second argument has no string value, the function returns true, because it will be converted to an empty string first.

Unlike start-with() XPATH function, this function is not described in XPATH 1.0 standard. To write portable XPATH expression, use substring().

Parameters

strg
: String whose first characters must be compared

prefix
: String whose characters must be compared with first characters of strg

Return Types

Boolean

Examples

Example 24.584.

```
ends-with('OpenLink Virtuoso', 'Virtuoso')
```

See Also

string() contains() starts-with()

every

every — Returns true if all items of given sequence matches given criterion.

Synopsis

```
boolean every ( varname string ,
                test_set sequence ,
                test_expn boolean ) ;
```

Description

The function creates a temporary local variable, whose name is specified by *varname* argument. Then, for every item of *test_set* sequence it calculates the *test_expn* boolean expression having set the created variable to that "current" item. If the value of expression is false, the function immediately returns false without processing the rest of *test_set* sequence. If all items of the sequence are probed without getting false, true is returned. (So if the sequence is empty, the function returns true).

In any case, temporary variable is destroyed on return.

This function is used in the implementation of "EVERY" logical operator in XQUERY, so you will probably use that operator in XQUERY expressions, not the function. This function may be useful in XPATH expressions and in XSLT stylesheets. It is not a part of library of standard XQUERY 1.0 functions.

Parameters

varname
: Name of temporary variable

test_set
: Sequence of items; these items will be tested by test_expn

test_expn
: Boolean expression which should be calculated for items of test_set .

Return Types

Boolean

Examples

Example 24.585.

This expression returns true if all reports have positive incomes recorded.

```
every('income_value', /report/income, $income_value > 0)
```

See Also

some()

except

except — Returns a difference of two sets

Synopsis

```
sequence except ( set1 sequence ,
                  set2 sequence ) ;
```

Description

The function returns an unordered sequence that consists of all distinct items from the first sequence that are missing in the second sequence.

Duplicate values from *set1* are removed, the decision whether two values duplicate each other is made according rules used by `distinct()` function.

Parameters

set1
: Sequence that contains items that can occur in the resulting set.

set2
: Sequence that contains items that can not occur in the resulting set.

Return Types

The function returns an unordered sequence of distinct values.

See Also

`union`

`intersect`

false

false — Returns false.

Synopsis

```
boolean false ( void);
```

Description

This function returns boolean constant "false".

Return Types

Boolean

See Also

true boolean

filter

filter — Composes trees of shallow copies of given XML entities.

Synopsis

```
node-set filter ( selection sequence );
```

Description

The function takes a single parameter which can be any expression. The function evaluates its argument and returns a shallow copy of the nodes that are selected by the argument, preserving any relationships that exist among these nodes. Duplicate nodes are removed before the processing.

The structure of the resulting node-set may be explained in the following way. First of all, all input entities are grouped by their documents, so we have a set of distinct documents and for every document we have a list of entities that refers to various nodes of the document. After that, every such document is processed separately, and the result of processing is a node-set; the union of these node-sets will be returned as the result of the call of filter() function. A copy of the document is made, and a "color" is assigned to every node of the copy: it's "black if the original node is listed in the *selection* sequence, otherwise it's "white". Then "white" nodes are removed from the copy, node after node: if a "white" node may be found, it is replaced with list of its children. Finally, we have a list of one or more "black" nodes whose descendants are all "black", too, and this list is added into the resulting node-set.

(The actual algorithm is much faster and much more complicated than the described one, but the result is identical.)

Note that this function is defined in XQuery standard, not in XPath, but in Virtuoso it may be used freely in expression of any kind.

Parameters

selection
: The sequence of nodes that should be included into the result

Return Types

Node-set

Errors

Table 24.142. Errors signalled by filter()

SQLState	Error Code	Error Text	Description
XP001	XPFB0	The argument of XQUERY function filter() must be a sequence of XML entities	According to the XQuery standard, the function should signal an error if input contains values other than XML entity.

Examples

Example 24.586. Composing table of contents

The following example is from the XQuery standard and it illustrates how filter() might be used to compute a table of contents for a document that contains many levels of nested sections. The query filters the document, retaining only section elements, title elements nested directly inside section elements, and the text of those title elements. Other elements, such as paragraphs and figure titles, are eliminated, leaving only the "skeleton" of the document. The example generates a table of contents for a document named "cookbook.xml".

```
<toc>
   {
```

```
    filter(document("cookbook.xml") //
       (section | section/title | section/title/text()))
    }
</toc>
```

See Also

shallow()

floor

floor — Returns the largest integer that is not greater than the argument.

Synopsis

```
integer floor ( num number );
```

Description

This function returns the largest (closest to positive infinity) number that is not greater than the argument and that is an integer. In other words, it "rounds down" the given value.

Parameters

num
: The value to be "rounded down"

Return Types

Examples

Example 24.587.

The following expressions are true:

```
floor(2.5) = 2
floor(2) = 2
floor(-2.5) = -3
floor(-2) = -2
```

See Also

ceiling() round()

for

for — Repeats some calculation for every item of a given sequence

Synopsis

```
any for ( varname string ,
          source_set sequence ,
          mapping_expn any ) ;
```

Description

The function creates a temporary local variable, whose name is specified by *varname* argument. Then, for every item of *source_set* sequence it calculates the value of *mapping_expn* expression having set the created variable to that "current" item. It returns the "flattened" sequence of values returned by *mapping_expn* in the same order as they are calculated. "Flattened" means that if *mapping_expn* returns an sequence, items of this sequence will be added into the end of resulting sequence, one by one, instead of adding one item of type "sequence".

The temporary variable is destroyed on return.

This function is used in the implementation of "FOR" control operator in XQUERY, so you will probably use that operator in XQUERY expressions, not the function. This function may be useful in XPATH expressions and in XSLT stylesheets. It is not a part of library of standard XQUERY 1.0 functions.

Parameters

varname
: Name of temporary variable

source_set
: Sequence of items; every item will cause one call of mapping_expn

mapping_expn
: An expression which should be calculated for items of source_set .

Return Types

Sequence

Examples

Example 24.588.

These two expressions are equivalent, but first may be used in any XPATH while second is written in XQUERY syntax

```
for('itm', /bid/item, 0.20 * $itm/price)
FOR $itm IN /bid/item RETURN 0.20 * $bid_item/price
```

See Also

let()

some()

every()

format-number

format-number

Synopsis

```
string format-number (   num number ,
                         format_pattern string
                         ,
                         decimal_format
                         string ) ;
```

Description

The function converts the *num* argument to a string using the format pattern string specified by the *format_pattern* and the decimal-format named by the *decimal_format* , or the default decimal-format, if there is no third argument.

The format pattern string is in the syntax specified by the JDK 1.1 DecimalFormat class. The following describes the structure of the pattern.

- The pattern consists of one or two subpatterns, first is for positive numbers and zero, second is for negative numbers. Two subpatterns are delimited by semicolon. If there is only one subpattern, - is prefixed to the positive subpattern.
- Every subpattern consists of optional prefix characters followed by an integer part followed by an optional fraction part followed by an optional suffix characters.
- Prefix and suffix characters are any Unicode characters except special formatting characters described below, while integer and fraction part consist only from that special formatting characters. (As an exception, special characters may appear in prefix in suffix parts if enclosed in single quotes.
- If fractional present, it starts from '.' character, and only one '.' may occur in the subformat. Thus it is easy to find where each part begins.

By default, the following characters are treated as special when used in the parts of the subpattern:

Table 24.143. Sub-pattern character substitutions

Symbol	Meaning
0	A digit, zero will be printed. 0 must be the last character of integer part.
#	A digit, zero will not be printed.
.	Placeholder for decimal separator in the beginning of fraction part.
,	Placeholder for grouping separator. It may appear only in integer part. All commas except the last will be ignored.
;	Separates formats. It may appear only once in the pattern.
-	Placeholder for negative prefix.
%	Indicates that the value must be multiplied by 100 and shown as percentage.
?	Indicates that the value must be multiplied by 1000 and shown as per mille.

Note that character '¤' have a special meaning in DecimalFormat of JDK 1.1, but not in XPATH.

The format pattern string may be in a localized notation. The *decimal_format* may determine what characters have a special meaning in the pattern (with the exception of the quote character, which is not localized). The *decimal_format* must be a QName, and a search will be performed for top-level <xsl:decimal-format> key whose "key" attribute is equal to *decimal_format* ; all names will be expanded before the search. It is an error if the stylesheet does not contain a declaration of the decimal-format with the specified expanded name.

Parameters

num
: Number to format.

format_pattern
: Format pattern which must be applied to the number.

decimal_format
: Name of <xsl:decimal-format> element which must be used to parse format pattern.

Return Types

String

Errors

Table 24.144. Errors signalled by

SQLState	Error Code	Error Text	Description
XS370	XS036	Number format ... is not defined in XPath function format-number()	The function is called outside an XSLT processor or the current stylesheet contains no format definition with the required name.

Examples

Example 24.589.

If the XSLT stylesheet defines format 'financial' as '###,###.00####', then the following returns '123,456.789':

```
format(123456.789, "financial")
```

function-available

function-available — Returns true if XPATH extension function with the requested name is defined in the XPATH Processor, otherwise returns false.

Synopsis

```
boolean function-available ( funname string );
```

Description

The function returns true if XPATH Processor can execute XPATH extension function with the name specified by *funname* argument. If such function is not defined, function-available returns false.

Parameters

funname
: The name of XPATH extension function

Return Types

Boolean

Examples

Example 24.590.

This expression should return false.

```
function-available('http://example.com/Virtuoso/nosuchfunction')
```

See Also

xpf_extension()

generate-id

generate-id — Returns a string that uniquely identifies the node.

Synopsis

```
string generate-id ( place node_set );
```

Description

The function returns a string that uniquely identifies the first node in the *place* argument node-set. The unique identifier will consist of ASCII alphanumeric characters and will start with an alphabetic character. Thus, the string is syntactically an XML name. It always generates the same identifier for the same node. It always generates different identifiers from different nodes. This function is under no obligation to generate the same identifiers each time a document is transformed. There is no guarantee that a generated unique identifier will be distinct from any unique IDs specified in the source document.

If the argument node-set is empty, the empty string is returned.

If the argument is omitted, it defaults to the context node.

Parameters

place
: Node-set whose first node is used as a key to generate a resulting ID

Return Types

String which is syntactically an XML name.

Examples

Example 24.591.

```
generate_id(.)
```

id

id — Returns an entities whose ID attributes are in the given list

Synopsis

```
sequence id ( ids any );
```

Description

The function gets a list of strings, where every string is a list of IDs delimited by whitespace characters. For every listed ID, the function checks if the document that contains the context entity also contains an entity with such ID; if the entity with the listed ID exists, it is added into the sequence.

To avoid searches of every ID inside the whole document, the function uses "ID cache" that is created if "IdCache=ENABLE" DTD configuration option is specified in the call of function that parses the XML document. In some cases, it is possible to build such cache on demand if it is not built by a parser, but it consumes time and in any case it requires the saved DTD information from the XML parser, so the parser should be specially configured anyway.

Note that any attribute of types ID, IDREF or IDREFS is a good input string for the function id(). So while XQuery "pointer operator" is not available in XPath expression, the id() function is a good replacement for it.

Parameters

ids
: The list of strings that consist of IDs

Return Types

Errors

The function ignores any sort of syntax errors on input.

Examples

Example 24.592. List errors in census data

List every person who is not a spouse of his own spouse.

```
document ("census.xml")//person[@spouse and not (. = id (id (@spouse)/person/@spouse)]
```

See Also

document()

if

if — If the boolean value is true then calculates one expression, otherwise calculates another expression.

Synopsis

```
any if ( test boolean ,
         then_branch any ,
         else_branch any ) ;
```

Description

This function calculates the value of *test* argument. If the value is true, the function calculates the *then_branch* expression and returns its value. If the value is false, the function calculates the *else_branch* expression and returns its value.

Note that unlike other programming languages, *else_branch* is required argument, not optional.

This function is used in the implementation of "IF" control operator in XQUERY, so you will probably use that operator in XQUERY expressions, not the function. This function may be useful in XPATH expressions and in XSLT stylesheets. It is not a part of library of standard XQUERY 1.0 functions.

Parameters

test
: Boolean value used to choose an expression to execute

then_branch
: Expression which is calculated if test argument is true

else_branch
: Expression which is calculated if test argument is false

Return Types

Any

Examples

Example 24.593.

These two expressions are equivalent, but first may be used in any XPATH while second is written in XQUERY syntax:

```
if (2 * 2 = 4, 'I think so', 'Unbelievable!')
IF 2 * 2 = 4 THEN 'I think so' ELSE 'Unbelievable!'
```

See Also

and() or()

intersect

intersect — Returns an intersect of two sets

Synopsis

```
sequence intersect ( set1 sequence ,
                     set2 sequence ) ;
```

Description

The function returns an unordered sequence that consists of all distinct items such that every of source sequences contains every of the resulting items.

Duplicates are removed, including duplicate occurrences of same value in one sequence, the decision whether two values duplicate each other is made according rules used by `distinct()` function.

Parameters

set1, set2
: Sequence that contains items of the resulting intersect.

Return Types

The function returns a sequence of distinct values.

See Also

`union`

`except`

is_after

is_after — Returns true if the given entity is after the second given entity in document order, otherwise returns false.

Synopsis

```
boolean is_after (   ent1 entity ,
                     ent2 entity
                     ) ;
```

Description

The function ensures that both *ent1* and *ent2* are XML entities and the returns true if *ent1* and *ent2* are both in the same XML document and *ent1* is strictly after *ent2* in document order. It returns false if one of them is not an entity but an empty node-set or if their documents differ or or if they're equal or if one of them is an ancestor of other or if *ent1* is simply before *ent2* in document order.

Parameters

ent1
: An XML entity or an node-set whose first node is compared.

ent2
: An XML entity or an node-set whose first node is compared.

Return Types

Boolean

Errors

Table 24.145. Errors signalled by is_after()

SQLState	Error Code	Error Text	Description
XP001	XPFB0	First argument of XPATH function is-before() or is-after() must be XML entity	The value that is neither an XML entity nor an empty node-set may not be compared using document order.
XP001	XPFB1	Second argument of XPATH function is-before() or is-after() must be XML entity	The value that is neither an XML entity nor an empty node-set may not be compared using document order.

Examples

Example 24.594.

Find all chapters that have title after table of content

```
/book/chapter[is-after(toc,title)]
```

See Also

is_before()

is_before

is_before — Returns true if the given entity is before the second given entity in document order, otherwise returns false.

Synopsis

```
boolean is_before ( ent1 entity ,
                    ent2 entity ) ;
```

Description

The function ensures that both *ent1* and *ent2* are XML entities and the returns true if *ent1* and *ent2* are both in the same XML document and *ent1* is strictly before *ent2* in document order. It returns false if one of them is not an entity but an empty node-set or if their documents differ or or if they're equal or if one of them is an ancestor of other or if *ent1* is simply after *ent2* in document order.

Parameters

ent1
: An XML entity or an node-set whose first node is compared.

ent2
: An XML entity or an node-set whose first node is compared.

Return Types

Boolean

Errors

Table 24.146. Errors signalled by is_before()

SQLState	Error Code	Error Text	Description
XP001	XPFB0	First argument of XPATH function is-after() or is-before() must be XML entity	The value that is neither an XML entity nor an empty node-set may not be compared using document order.
XP001	XPFB1	Second argument of XPATH function is-after() or is-before() must be XML entity	The value that is neither an XML entity nor an empty node-set may not be compared using document order.

Examples

Example 24.595.

Find all chapters that have table of content before title

```
/book/chapter[is-before(title,toc)]
```

See Also

is_after()

key

key — This XSLT 1.0 function is not implemented in the current version of Virtuoso.

Synopsis

```
string key ( keyname string ,
             keyvalues any ) ;
```

lang

lang — Returns true if the language of context node matches given language name.

Synopsis

```
boolean lang ( lang_name string );
```

Description

The lang function returns true or false depending on whether the language of the context node as specified by xml:lang attributes is the same as or is a sublanguage of the language specified by the argument string. The language of the context node is determined by the value of the xml:lang attribute on the context node, or, if the context node has no xml:lang attribute, by the value of the xml:lang attribute on the nearest ancestor of the context node that has an xml:lang attribute. If there is no such attribute, then lang returns false. If there is such an attribute, then lang returns true if the attribute value is equal to the argument ignoring case, or if there is some suffix starting with "-" such that the attribute value is equal to the argument ignoring that suffix of the attribute value and ignoring case.

Parameters

lang_name
: Name of the language

Return Types

Boolean

Examples

Example 24.596.

The expression lang("en") would return true if the context node is any of these five elements:

```
<para xml:lang="en"/>
<div xml:lang="en"><para/></div>
<para xml:lang="EN"/>
<para xml:lang="en-us"/>
```

See Also

last

last — Returns the context size from expression evaluation context.

Synopsis

```
number last ( void);
```

Description

Context size is the number of nodes in the node-set where the context node comes from. For the most popular case, when last() is used inside a predicate, and the predicate relates to some axis of the path, last() is the number of elements found by that axis at once; in other words, the number of elements to be tested by predicate.

Return Types

Positive integer.

Examples

Example 24.597. Call of last() inside a predicate

Find all waybills with more than one <address> child inside the <waybill> element.

```
//waybill/address[last()>1]
```

See Also

position() count()

let

let — Creates temporary variables and calculates an expression that uses these variables

Synopsis

```
any let ( var1name string ,
          var1value sequence ,
          var2name string ,
          var2value sequence ,
          ... ,
          varNname string ,
          varNvalue sequence ,
          retval any ) ;
```

Description

For every pair of arguments, the function calculates values of these arguments and then creates a new temporary local variable whose name is the string value of the first argument in pair and the value assigned is the value of the second argument in pair. Obviously, the argument for variable name is usually a constant string of alphanumeric characters. The expression for a value of variable number I may refer to variables created during steps 1 to I-1. When all pairs of arguments are turned into temporary variables, the last argument is calculated and its value is returned as the value of the whole expression.

Temporary variables are destroyed on return.

This function is used in the implementation of "LET" control operator in XQUERY, so you will probably use that operator in XQUERY expressions, not the function. This function may be useful in XPATH expressions and in XSLT stylesheets. It is not a part of library of standard XQUERY 1.0 functions.

Parameters

varIname
: Expression for the name for the I-th temporary variable

varIvalue
: Expression for the value for the I-th temporary variable

retval
: Expression for the value to be returned

Return Types

Any, depending on the type of `retval` expression.

Errors

Table 24.147. Errors signalled by let()

SQLState	Error Code	Error Text	Description
42000	XPF02	Wrong number of arguments for XPATH function let(), maybe internal XQuery error	The function should have odd number of arguments: even number of arguments that describe variables plus one for the returned expression.

Examples

Example 24.598.

These two expressions are equivalent, but first may be used in any XPATH while second is written in XQUERY syntax

```
let('baseprice', /item/price, 'discount', 0.20, $baseprice * (1.0 - $discount))
LET  $baseprice := /item/price, $discount := 0.20 RETURN $baseprice * (1.0 - $discount)
```

See Also

for()

list

list — Selects the first item of every argument sequence and returns the sequence of the selected items

Synopsis

```
sequence list ( seq1 sequence ,
                seq2 sequence ,
                ... ,
                seqN sequence ) ;
```

Description

This function calculates all given arguments from left to right, and creates a sequence which contains the first item of the first calculated sequence, then the first item of the second calculated sequence and so on. If the value of an argument is not a sequence, but a scalar, the scalar is treated as one-element sequence so it is added into the result. If the value of an argument is an empry sequence, nothing is added into the result (unlike function tuple() that adds an empty string in this case).

This function is not a part of XPATH 1.0 or XQUERY 1.0 libraries of standard functions.

Parameters

SeqI
: The sequence of items, first of its items will be used as I-tn item of the resulting sequence

Return Types

Sequence

Examples

Example 24.599.

Compose a sequence of the first title in the document, then the first abstract, then the first introduction

```
list(/title, /abstract, /intro)
```

See Also

append() tuple()

local-name

local-name — Returns the local part of the expanded name of the argument.

Synopsis

```
string local-name ( node_obj any );
```

Description

For given node, it returns local part of the node, i.e. the name of given attribute or element with namespace prefix removed. If the argument is node-set, first node of the node-set will be considered. Empty string is returned if the argument is an empty node-set, a node without name or if the argument is not a node.

If the argument is omitted, context node is used instead as if it is a node-set of one element.

Parameters

node_obj
: Node whose name is to be returned

Return Types

String

Examples

Example 24.600.

Find all elements whose names start with "ref..." prefix

```
//*[starts-with(local-name(),'ref')]
```

See Also

name() namespace-uri()

max

max — Returns maximum value among all its arguments.

Synopsis

```
number max ( arg1 any ,
             arg2 any ,
             ... ,
             argN any ) ;
```

Description

The function returns the maximum value among all values in all its arguments, For each node in every argument node-set, it converts the string-value of the node to a number. If some arguments are not node-sets, they are converted to numbers. The maximum number found is returned.

This function is not a part of XPATH 1.0 standard library.

Parameters

argI
: A node-set of nodes whose string values must be converted to numbers and added to the result, or single value.

Return Types

Table 24.148. Errors signalled by

SQLState	Error Code	Error Text	Description
XP001	XPF05	Nonempty sequence of values expected as argument of XPATH function max(); max() of nothing is senseless	This happens if the function is called without arguments, or with all arguments set to empty node-sets.

Examples

Example 24.601.

```
max(/report[@type="daily-sales"]/total)
```

See Also

sum avg min

min

min — Returns minimum value among all its arguments.

Synopsis

```
number min ( arg1 any ,
             arg2 any ,
             ... ,
             argN any ) ;
```

Description

The function returns the minimum value among all values in all its arguments, For each node in every argument node-set, it converts the string-value of the node to a number. If some arguments are not node-sets, they are converted to numbers. The minimum number found is returned.

This function is not a part of XPATH 1.0 standard library.

Parameters

argI
: A node-set of nodes whose string values must be converted to numbers and added to the result, or single value.

Return Types

Table 24.149. Errors signalled by min()

SQLState	Error Code	Error Text	Description
XP001	XPF04	Nonempty sequence of values expected as argument of XPATH function min(); min() of nothing is senseless	This happens if the function is called without arguments, or with all arguments set to empty node-sets.

Examples

Example 24.602.

```
min(/report[@type="daily-sales"]/total)
```

See Also

sum avg max

name

name — Returns the expanded name of the argument.

Synopsis

```
string name ( node_obj any );
```

Description

For given node, it returns extended name of the node, i.e. the name of given attribute or element with namespace prefix replaced with namespace URI string. If the argument is node-set, first node of the node-set will be considered. Empty string is returned if the argument is an empty node-set, a node without name or if the argument is not a node.

If the argument is omitted, context node is used instead as if it is a node-set of one element.

Parameters

node_obj
: Node whose name is to be returned.

Return Types

String

Examples

Example 24.603.

Find all elements whose namespace URIs or local names contain "html" substring

```
//*[contains(name(),'html')]
```

See Also

local-name() namespace_uri()

namespace-uri

namespace-uri — Returns the namespace URI of the extended name of the given node

Synopsis

```
string namespace-uri ( node_obj any );
```

Description

If given argument is a node, the function returns the URI string of the namespace specified in the name of node. If the argument is node-set, first node of the node-set will be considered. Empty string is returned if the argument is an empty node-set, a node without name or if the argument is not a node.

If the argument is omitted, context node is used instead as if it is a node-set of one element.

Parameters

node_obj
: Node whose namespace URI is to be returned.

Return Types

String

Examples

Example 24.604.

Find all elements whose namespace URIs contain "html" substring

```
//*[contains(namespace-uri(),'html')]
```

See Also

local-name() name()

normalize-space

normalize-space — Returns the argument string with whitespace normalized.

Synopsis

```
string normalize-space ( strg string );
```

Description

The function returns the argument string with whitespace normalized by stripping leading and trailing whitespace and replacing sequences of whitespace characters by a single space. Whitespace characters are the same as those allowed by the S production in XML, i.e. space (#x20), carriage returns (#xD), line feeds (#xA), and tabs (#x9). If the argument is omitted, it defaults to the context node converted to a string, in other words the string-value of the context node.

Parameters

strg
: A string to be normalized

Return Types

Examples

Example 24.605.

```
normalize-space('    too     many   spaces    ')
```

See Also

translate() replace()

not

not — Returns true if its argument is false, and false otherwise.

Synopsis

```
boolean not ( obj boolean );
```

Description

This function returns true if its argument is false, and false otherwise. If the argument is not a value of boolean type, it will be converted to boolean first.

Parameters

obj
: Boolean value

Return Types

Boolean

Examples

Example 24.606.

See Also

boolean()

number

number — Converts its argument to a number.

Synopsis

```
number number ( obj any );
```

Description

The number function converts its argument to a number as follows:

- A string that consists of decimal number and optional whitespaces is converted to the number recorded. Any other string is converted to NaN ("not-a-number" value). More precisely, a string that consists of optional whitespace followed by an optional plus or minus sign followed by a Number followed by whitespace is converted to the IEEE 754 number that is nearest (according to the IEEE 754 round-to-nearest rule) to the mathematical value represented by the string. Note that it differs from XPATH 1.0 standard where plus sign is not allowed before Number part of the string.
- Boolean true is converted to 1; boolean false is converted to 0.
- A node-set is first converted to a string as if by a call to the string function and then converted in the same way as a string argument.
- An object of a type other than the four basic types is converted to a number in a way that is dependent on that type.

If the argument is omitted, it defaults to a node-set with the context node as its only member.

Parameters

obj
: Value to be converted to a number.

Return Types

Number

Examples

Example 24.607.

```
number(' 3.1415926 ')
```

See Also

string() boolean()

or

or — Returns true if a value of some argument is true, otherwise returns false.

Synopsis

```
boolean or ( val1 boolean ,
             val2 boolean ,
             ... ,
             valN boolean ) ;
```

Description

This function calculates values of its arguments, from left to right. If the value of calculated parameter is true, the function returns true immediately, without calculating of the remaining parameters. If the list of arguments ends without any true value calculated, the function returns false (Thus it returns true when called without arguments).

The name of this function is the same as name of "or" XPATH and XQUERY operator. Thus it must be surrounded by double quotes when used in XPATH or XQUERY expressions. Moreover, this function is not a part of XPATH standard, so it cannot be used if portability is important.

Parameters

valI
: Value of boolean expression argument

Return Types

boolean

Examples

Example 24.608. Control over sequence of search operations

If two conditions must be checked, where one is simple and another is hard to calculate, then "and" may be used to calculate second condition only if first is false, to reduce average time of processing

```
"or" ( empty(authors),
  document(concat('http://www.lib20.org/findxml.cgi?isbn=',@isbn))/status[@outofprint='YES'] )
```

See Also

not and some

position

position — Returns the context position from expression evaluation context.

Synopsis

```
number position ( void);
```

Description

Context position is the number of nodes in the node-set where the context node comes from. For the most popular case, when position() is used inside a predicate, and the predicate relates to some axis of the path, position() is the number of calls of the predicate, including the "current" call which is in progress when the function is called. Thus, context position cannot be greater than context size.

Return Types

Positive integer.

Examples

Example 24.609. Explicit call of position() inside a predicate

For every waybill, find the last <address> child inside the <waybill> element.

```
//waybill/address[position() = last()]
```

Example 24.610. Implicit call of position() inside a predicate

For every article, find up to three first paragraphs of the first section.

```
//article/section[1]/paragraph[position()<=3]
```

See Also

last() count()

processXQuery

processXQuery — Calls an XQuery module from XPath expression, e.g. from some XSLT or BPEL code.

Synopsis

```
any processXQuery ( module_uri string ,
                    context entity ,
                    index integer ,
                    param1_name string ,
                    param1_value any ... ,
                    paramN_name string,
                    paramN_value any) ;
```

Description

This function takes a URI of an XQuery module and an XML entity and calls the module with the entity as a context. Depending on value of *index* parameter, either the result of the module is returned 'as is' or the sequence of results is returned.

Parameters can be passed to the module by specifying additional arguments to `processXQuery()` . The names of parameters should appear in argument list without the leading '$' sign. Unlike `xquery_eval()` function, parameter can not be ignored depending on the type of its value. If the same name appears more than once in the vector, the last name/value pair is used and all preceding pairs with this name are silently ignored. Obviously, names should be strings that are valid XPath variable names.

The XQuery standard does not offer a way of calling of a module from other XQuery expression. The reason is that there's no need for such calling if the code is designed properly. If an expression is re-used in various places then it should be turned into a function and placed into an XQuery library module; one should import the module and call the function instead of calling a non-library module. It is possible to use `processXQuery()` in XQuery expressions but it is much better to use library modules instead, and to use `processXQuery()` only for tricks in XPATH expressions.

For compatibility, the `processXQuery()` function can also be called as `http://schemas.oracle.com/xpath/extension:processXQuery()` .

Parameters

module_uri
: URI pointing to the location of an XQuery module. It can be absolute or relative. A relative module_uri should be resolved before use, this requires base URI information. Base URI can be declared explicitly by "__base_uri" parameter in XPATH or "declare base-uri" setter in XQuery. If not declared but the expression is a part of some stylesheet or XQuery module then the URI of module is used as a base URI. A run-time error is signalled if the URI is relative and the expression does not contain explicit declaration and the expression is neither in a stylesheet nor in a module.

index
: Result index. If omitted a value of 1 is assumed, meaning only the first result is returned. If a value of 0 is supplied then a (flatten) sequence of all results is returned. (Note that if a non-zero value is specified then the returned value still can be a sequence).

context
: XML entity that is the context node of module call. If the function is called with only one argument then the current context node of the processXQuery() call is used as a context of module call. (In any case, context size and context position of module call are always set to 1 and not inherited from call of processXQuery() .)

paramI_name
: Name of parameter to be passed to the XSLT engine for use in the transformation.

paramI_value
: Value of parameter with name specified by paramI_name .

Return Types

The type of return value depends on type of value returned by module.

Examples

Example 24.611. Call of XQuery in XSLT

Sample templates put the result of the call of module "mymodule.xq" for context node into the resulting document. This assumes that both the stylesheet and XQuery module "mymodule.xq" reside in the same directory so relative a URI "mymodule.xq" can be resolved using the URI of the stylesheet as base URI.

```
  <xsl:template match="myelement">
  <xsl:copy-of select="processXQuery('mymodule.xq')"/>
</xsl:template>
```

This is equivalent with the following template:

```
  <xsl:template match="myelement">
  <xsl:copy-of select="processXQuery('mymodule.xq', current(), 1)"/>
</xsl:template>
```

See Also

`xquery_eval()`

`processXSLT()`

`processXSQL()`

processXSLT

processXSLT — Applies stylesheet to given XML entity and returns the result.

Synopsis

```
entity processXSLT ( stylesheet_uri string ,
                     source entity ,
                     param1_name string ,
                     param1_value any ... ,
                     paramN_name string,
                     paramN_value any ) ;
```

Description

This function takes the URI of a stylesheet and an XML entity and produces an XML entity representing the transformation result of the given entity with the given stylesheet. The result tree is separate from the argument tree and the only reference to it is the returned entity. Errors occurring in the transformation will be signalled as SQL states, with XML or XSLT specific conditions beginning with XS or XP.

The stylesheet is applied to the value of *source* parameter. Obviously, *source* must be an entity. If *source* is not specified then the stylesheet is applied to the current entity.

Parameters can be passed to the stylesheet by specifying additional arguments to `processXSLT()` . The values can be referenced from inside XPath expressions in the stylesheet. The names of parameters should appear in argument list without the leading '$' sign. Unlike `xslt()` function, parameter can not be ignored depending on the type of its value. If the same name appears more than once in the vector, the last name/value pair is used and all preceding pairs with this name are silently ignored. Obviously, names should be strings that are valid XPath variable names.

The function uses the cache of compiled stylesheets, see description of `xslt()` Virtuoso/PL function for more details.

For compatibility, the `processXSLT()` function can also be called as `http://schemas.oracle.com/xpath/extension:processXSLT()` .

Parameters

stylesheet_uri
: URI pointing to the location of an XSL stylesheet. It can be absolute or relative. A relative stylesheet_uri should be resolved before use, this requires base URI information. The base URI can be declared explicitly with the "__base_uri" parameter in XPATH or "declare base-uri" setter in XQuery. If not declared but the expression is a part of some stylesheet or XQuery module then the URI of module is used as a base URI. A run-time error is signalled if the URI is relative and the expression does not contain explicit declaration and the expression is neither in a stylesheet nor in a module.

source
: XML entity that is the source of XSL transformation. If the function is called with only one argument then the context node is used as a source.

paramI_name
: Name of parameter to be passed to the XSLT engine for use in the transformation.

paramI_value
: Value of parameter with name specified by paramI_name .

Return Types

The function returns an XML entity that is the root entity of the resulting document of an XSL transformation.

Examples

Example 24.612. XSLT inside XQuery expression

The query composes a single document that contain a list of labels for parcels. Depending on destination country and the state value of the parcel, the query applies one of three stylesheets that make a label from the address data; the query makes a label from every item of sequence $orders.

```
<parcel-labels>
for $order in $orders let $customer := $customers/customer [ @CustId = $order/@CustId ] return
  if ($customer/address/@Country = "US")
    if ($order/sum >= 50.0)
      processXSLT ("file://registered-parcel.xsl", $customer/address, "value", $order/sum)
    else
      processXSLT ("file://plain-parcel.xsl", $customer/address)
  else
    processXSLT ("file://foreign-parcel.xsl", $customer/address, "value", $order/sum)
</parcel-labels>
```

See Also

`xslt()`

`processXQuery()`

`processXSQL()`

processXSQL

processXSQL — Executes XSQL page and returns the result.

Synopsis

```
entity processXSQL ( xsql_page_uri string );
```

Description

This function takes a URI of an XSQL page, compiles the page into a Virtuoso/PL procedure (if not compiled earlier) and executes the compiled procedure. The current entity is passed to the page procedure as "context XML" argument. The function returns the XML document composed by page procedure. The result tree is separate from the argument tree and the only reference to it is the returned entity.

For compatibility, the `processXSQL()` function can also be called as `http://schemas.oracle.com/xpath/extension:processXSQL()`.

Parameters

xsql_page_uri
: URI pointing to the location of an XSQL page. It can be absolute or relative. Relative xsql_page_uri should be resolved before use, this requires base URI information. Base URI can be declared explicitly by "__base_uri" parameter in XPATH or "declare base-uri" setter in XQuery. If this is not declared but the expression is a part of some stylesheet or XQuery module then the URI of the stylesheet or module is used as a base URI. A run-time error is signalled if the URI is relative and the expression does not contain explicit declaration and the expression is neither in a stylesheet nor in a module.

Return Types

The function returns an XML entity that is the root entity of the resulting document of an XSQL page.

Examples

Example 24.613. Simple call

The query passes its argument to the XSQL page and returns its result. This is the simplest way of calling an XSQL page.

```
processXSQL ("file://sample.xsql")
```

See Also

`processXSLT()`

`processXQuery()`

progn

progn — Calculates all given expressions and returns the result produced by the last one.

Synopsis

```
any progn ( expn1 any ,
            expn2 any ,
            ... ,
            expnN any ) ;
```

Description

This function calculates its first argument, then second argument and so on. The results of these calculations are not used, except the result of the last expression which is returned. The only useful application for this function is calling XPath extension functions for side effects.

This function is not a part of library of standard XQuery 1.0 functions.

Parameters

expn1, expn2, ... expnN
: Expressions to be calculated.

Return Types

Any (according to the type of *expnN*)

Examples

Example 24.614.

```
[xmlns:fileio='http://www.example.com/file-io']
if (function-available ('fileio:printf'),
  progn (
    fileio:open('greeting.txt', 'wt'),
    fileio:printf('Hello %s', /userinfo/name),
    fileio:close(),
    'File greeting.txt saved' ),
  'File greeting.txt not saved' ) )
```

See Also

if

for

let

replace

replace — Searches for an substring and replace its occurrences with other substring.

Synopsis

```
string replace ( txt string ,
                 search_strg string ,
                 replace_strg string ) ;
```

Description

The function searches for all occurrences of a *search_strg* in *txt* and replaces every occurrence with *replace_strg* . An error is signalled if *search_strg* is empty.

Parameters

txt
: Text to process.

search_strg
: Substring to search for txt .

replace_strg
: Substring to replace search_strg in txt .

Return Types

String.

Errors

Table 24.150. Errors signalled by

SQLState	Error Code	Error Text	Description
XP001	XPF08	Empty string passed as argument 2 to XPATH function replace()	It is impossible to search for an empty substring.
XP001	XPF14	The result of XPATH function replace() is too long	If the function replaces short substring with longer one, the result is longer than the original string and may exceed 10Mb limit.

Examples

Example 24.615.

Replace all occurrences of 'http://www.example.com/' with '/webrobot/'

```
replace(
  '<A HREF="http://www.example.com/index.html">Home</A>',
  'http://www.example.com/', '/webrobot/' )
```

See Also

translate

round

round — Returns the integer that is the nearest to the argument.

Synopsis

```
integer round ( num number );
```

Description

The function returns the number that is closest to the argument and that is an integer. If there are two such numbers, then the one that is closest to positive infinity is returned. If the argument is NaN, then NaN is returned. If the argument is positive infinity, then positive infinity is returned. If the argument is negative infinity, then negative infinity is returned.

Parameters

num
: The value to be rounded

Return Types

Examples

Example 24.616.

The following expressions are true:

```
round(2.4) = 2
round(2.5) = 3
round(2.6) = 3
round(-2.4) = -3
round(-2.5) = -2
round(-2.6) = -2
```

See Also

floor() ceiling()

serialize

serialize — Serializes a value of its argument following the rules of the host RDBMS.

Synopsis

```
string serialize ( obj any );
```

Description

The serialize() function converts an object to a string as follows:

- An empty sequence is converted to an empty string.
- A nonempty node-set is converted to a string by serialization of its first node.
- A non-attribute XML entity is serialized as if it is serialized by http() function. In other words, the result is a plain text of XML syntax. For an attribute XML entity, the value of attribute is returned.
- Values of other types are converted into strings as cast(... as varchar) do this in Virtuoso/PL, exactly.

If the argument is omitted, context node is converted instead as if it is a node-set of one element.

Parameters

obj
: Value to be converted into the string

Return Types

String

Errors

Examples

Example 24.617.

```
string(//title)
```

See Also

format_number()

serialize()

concat()

shallow

shallow — Returns a shallow copy of the given XML entity

Synopsis

```
entity shallow ( ent entity );
```

Description

The function returns a shallow copy of the given XML entity, i.e. a root of a new document that consists of only one node that is a copy of the given entity but have no children.

Parameters

ent
: The XML entity to copy

Return Types

The function returns an XML entity.

Errors

Table 24.151. Errors signalled by shallow()

SQLState	Error Code	Error Text	Description
XP001	XPFD6	The argument of XPATH function shallow() is not an entity	To match XQuery standard, the function signal an error if *ent* is not an entity.
XP001	XPFD7	Persistent XML entities are not supported by XPATH function shallow()	This is an artificial restriction that is to be removed in future versions of Virtuoso.

Examples

Example 24.618. Find Martha's spouse

The idea of the following sample is from W3C's XQuery Use Cases (case 1.9.4.1.). The query should find all persons whose name is "Marta" and return entities that describes their spouses, with removed information about children.

```
-- read the document, then...
    let $doc := document("census.xml")
-- for every person named "Martha"...
      for $m in $doc//person[@name="Martha"]
-- for all (i.e. 0 or 1) her spouses...
        for $s in $m/@spouse => person
-- return <person> element with attributes preserved but with children removed
          return shallow($s)
```

See Also

filter()

some

some — Returns true if at least one item of given sequence matches given criterion.

Synopsis

```
boolean some ( varname string ,
               test_set sequence ,
               test_expn boolean ) ;
```

Description

The function creates a temporary local variable, whose name is specified by *varname* argument. Then, for every item of *test_set* sequence it calculates the *test_expn* boolean expression having set the created variable to that "current" item. If the value of expression is true, the function immediately returns true without processing the rest of *test_set* sequence. If all items of the sequence are probed without getting true, false is returned. (So if the sequence is empty, the function returns false).

In any case, temporary variable is destroyed on return.

This function is used in the implementation of "SOME" logical operator in XQUERY, so you will probably use that operator in XQUERY expressions, not the function. This function may be useful in XPATH expressions and in XSLT stylesheets. It is not a part of library of standard XQUERY 1.0 functions.

Parameters

varname
: Name of temporary variable

test_set
: Sequence of items; these items will be tested by test_expn

test_expn
: Boolean expression which should be calculated for items of test_set .

Return Types

Boolean

Examples

Example 24.619.

This expression returns true if some reports have zero incomes recorded.

```
some('income_value', /report/income, $income_value = 0)
```

See Also

every()

starts-with

starts-with — Returns true if the first argument string starts with the second argument string, and otherwise returns false.

Synopsis

```
boolean starts-with ( strg string ,
                      prefix string ) ;
```

Description

For two given strings, this function checks if the first string starts with characters of second string. If any argument is not a string, it is converted to string using rules for string() XPATH function. Thus if the second argument has no string value, the function returns true, because it will be converted to an empty string first.

Parameters

strg
: String whose first characters must be compared

prefix
: String whose characters must be compared with first characters of strg

Return Types

Boolean

Examples

Example 24.620.

```
starts-with('OpenLink Virtuoso', 'OpenLink')
```

See Also

string() contains() ends-with()

string

string — Returns a string value of its argument.

Synopsis

```
string string ( obj any );
```

Description

The string() function converts an object to a string as follows:

- An empty sequence is converted to an empty string.
- A nonempty node-set is converted to a string by returning the string-value of the first node in the node-set. Note that the XPATH standard says that 'first' means 'first in document order' here, not 'first value returned'. For almost all queries, there is no difference between these two orders; it may be important only for node-sets calculated by so-called reverse-order axis. Moreover, the rure of standard is senseless if node-set contains nodes from more than one document. That is why the standard is violated here, intentionally.
- A nonempty sequence which is not a node-set is converted to a string by returning the string-value of the first node in the sequence.
- A NaN number is converted to the string 'NaN'.
- Zero is converted to the string '0'.
- Positive infinity is converted to the string 'Infinity'
- Negative infinity is converted to the string '-Infinity'
- Integer number is represented in decimal form with no decimal point and no leading zeros, preceded by a minus sign '-' if the number is negative.
- Non-integer number is represented in decimal form including a decimal point with at least one digit before the decimal point and at least one digit after the decimal point, preceded by a minus sign '-' if the number is negative; there must be no leading zeros before the decimal point apart possibly from the one required digit immediately before the decimal point; beyond the one required digit after the decimal point there will be be as many, but only as many, more digits as are needed to uniquely distinguish the number from all other IEEE 754 numeric values.
- The boolean false value is converted to the string 'false'. The boolean 'true' value is converted to the string 'true'.
- An object of a type other than the listed above is converted to a string in a way that is dependent on that type.

If the argument is omitted, context node is converted instead as if it is a node-set of one element.

Parameters

obj
: Value to be converted into the string

Return Types

String

Examples

Example 24.621. Examples

```
SQL> select xpath_eval ('string (//*)', xtree_doc ('<a>abc</a>')) ;
xpath_eval
VARCHAR
abc

SQL> select xpath_eval ('string (//title)', xtree_doc ('<doc><title>Simple test</title></doc>')) ;
xpath_eval
VARCHAR
Simple test

SQL> select xpath_eval ('string(.) div 21', xtree_doc ('<a>100</a>'))
```

```
xpath_eval
VARCHAR
4.761904761904762
No. of rows in result: 1

SQL> select xpath_eval ('string(//title) div 21', xtree_doc ('<a>100</a>'))
xpath_eval
VARCHAR
 0
No. of rows in result: 1

SQL> select xpath_eval ('string(//a) div 20', xtree_doc ('<a>100</a>'))
Query result:
xpath_eval
VARCHAR
 5
```

See Also

format_number()

serialize()

urlify()

concat()

string-length

string-length — Returns the number of characters in the string.

Synopsis

```
integer string-length ( strg string );
```

Description

The string-length() XPATH function returns the number of characters in the string. If the argument is omitted, it defaults to the context node converted to a string, in other words the string-value of the context node.

Parameters

strg
: The string whose length must be measured.

Return Types

Integer

Examples

Example 24.622.

The following expressions returns 5.

```
string-length('ABCDE')
```

See Also

xpf_string()

substring

substring — Returns the substring of the first argument starting at the position specified in the second argument with length specified in the third argument.

Synopsis

```
string substring ( strg string ,
                   start integer ,
                   length integer ) ;
```

Description

The substring() XPATH function returns the substring of the *strg* starting at the position specified in *start* argument with length specified in *length* argument. If *length* is not specified, it returns the substring starting at the position specified in the *start* argument and continuing to the end of the string.

XPATH 1.0 defines that "each character in the string... is considered to have a numeric position: the position of the first character is 1, the position of the second character is 2 and so on. This differs from Java and ECMAScript, in which the String.substring method treats the position of the first character as 0." The returned substring contains those characters for which the position of the character is greater than or equal to *start* and, if *length* is specified, less than the sum of *start* and *length* .

If *start* and/or *length* are not integers, they are converted to integers following rules for round() XPATH function, before doing any other processing. So they will be rounded first, and the sum of rounded values will be used as "end position"

If *start* is greater than or equal to the length of string, the empty string is returned. If *length* is specified and the sum of *start* is less than or equal to 1, the empty string is returned, too. Otherwise, the result string will contains some characters even if *start* is less than 1.

If *length* *start* is greater than or equal to the length of string, the empty string is returned.

Parameters

strg
: Source string. If the argument is not a string, it is converted to string first.

start
: Position of first character of the substring in the source string.

length
: Number of characters in the substring, if specified.

Return Types

String

Examples

Example 24.623.

The following expressions are all true:

```
substring("12345", 2, 3) = "234"
substring("12345", 2) = "2345"
substring("12345", 1.5, 2.6) = "234"
substring("12345", 0, 3) = "12"
substring("12345", -2, 5) = "12"
substring("12345", -2) = "12345"
```

See Also

substring-before() substring-after()

substring-after

substring-after — Returns the substring of the first argument string that follows the first occurrence of the second argument string in the first argument string.

Synopsis

```
string substring-after ( source_strg string ,
                         sub_strg string ) ;
```

Description

If the *source_strg* does not contain *sub_strg* , the function returns the empty string. Otherwise, it finds the first occurrence of *sub_strg* and returns the part of *source_strg* that follows the occurrence. If any argument is not a string, it is converted to string using rules for string() XPATH function.

Parameters

source_strg
: String where the search is performed

sub_strg
: String to search

Return Types

String

Examples

Example 24.624.

Extract time and timezone ("23:59:59 GMT" substring) from timestamp

```
substring-after('1999-12-31 23:59:59 GMT',' ')
```

See Also

string() substring() substring-after()

substring-before

substring-before — Returns the substring of the first argument string that precedes the first occurrence of the second argument string in the first argument string.

Synopsis

```
string substring-before ( source_strg string ,
                          sub_strg string ) ;
```

Description

If the *source_strg* does not contain *sub_strg* , the function returns the empty string. Otherwise, it finds the first occurrence of *sub_strg* and returns the part of *source_strg* that precedes the occurrence. If any argument is not a string, it is converted to string using rules for string() XPATH function.

Parameters

source_strg
: String where the search is performed

sub_strg
: String to search

Return Types

String

Examples

Example 24.625.

Extract year ("1999" substring) from ISO date string

```
substring-before('1999-12-31','-')
```

See Also

string() substring() substring-after()

sum

sum — Returns sum of all its arguments

Synopsis

```
number sum ( arg1 any ,
             arg2 any ,
             ... ,
             argN any ) ;
```

Description

The function returns the sum, for each node in every argument node-set, of the result of converting the string-values of the node to a number. If some arguments are not node-sets, they are converted to numbers first.

Note that this definition differs from XPATH 1.0 standard, where sum() function must have exactly one argument of type node-set. It is important that other XPATH processors may quietly ignore all arguments except the first one, producing unexpected results.

Being called without arguments, sum() will return zero.

Parameters

argI
: A node-set of nodes whose string values must be converted to numbers and added to the result, or single value.

Return Types

Examples

Example 24.626.

```
sum(/bill[@type="sale"]/total,/bill[@type="leasing"]/income)
```

See Also

avg max min

system-property

system-property — Returns a value of the system property identified by the name

Synopsis

```
object system-property ( property_qname varchar );
```

Description

The function returns an object representing the value of the system property identified by the name. If there is no such system property, the empty string is returned.

Parameters

property_qname
: a string that is a QName. The QName is expanded into a name using the namespace declarations in scope for the expression.

Return Types

String or double precision number.

Examples

Example 24.627. xsl:version

xsl:version is a number giving the version of XSLT implemented by the processor; this version returns 1.0.

```
[xmlns:xsl='http://www.w3.org/1999/XSL/Transform'] system-property('xsl:version')
```

Example 24.628. xsl:vendor

xsl:vendor is a string identifying the vendor of the XSLT processor; this version returns 'OpenLink Software'.

```
[xmlns:xsl='http://www.w3.org/1999/XSL/Transform'] system-property('xsl:vendor')
```

Example 24.629. xsl:vendor-url

xsl:vendor-url is a string containing a URL identifying the vendor of the XSLT processor; this version returns 'http://www.openlinksw.com'.

```
[xmlns:xsl='http://www.w3.org/1999/XSL/Transform'] system-property('xsl:vendor-url')
```

Example 24.630. xsl:product-name

xsl:product-name is a string containing a name of XSLT processor; this version returns 'OpenLink Virtuoso Server'.

```
[xmlns:xsl='http://www.w3.org/1999/XSL/Transform'] system-property('xsl:product-name')
```

Example 24.631. xsl:product-version

xsl:product-version is a string containing a version of XSLT processor implementation; the returned string may depend on OS, hardware and other parameters of installation.

```
[xmlns:xsl='http://www.w3.org/1999/XSL/Transform'] system-property('xsl:product-version')
```

See Also

`function_available` ()

text_contains

text_contains — Returns true if the text value of some node in the given node-set contains the text matching the given free-text query, otherwise returns false.

Synopsis

```
boolean text_contains ( scope node-set ,
                        query string ) ;
```

Description

This function calculates text values of nodes from the *scope* , and checks whether the current text value contains any fragment that matches the *query* . When the first match is found, the rest of the node-set is ignored the boolean 'true' is returned. If the node-set ends before any match is found, 'false' is returned.

The `text_contains()` function may be used only in XPath expressions that are arguments of `xcontains()` . This restriction is for optimization purposes. When Virtuoso executes an SQL statement that uses `xcontains()` , it performs some sophisticated free-text search, and it applies the XPath expression not to all available documents but only to documents that satisfied the free-text search criterion. Moreover, the server uses the intermediate free-text data to optimize the search inside a selected document.

Parameters

scope
: The node-set where the text search is performed.

query
: The text of the query.

Return Types

Returns a boolean: True or False.

When the first match is found, the rest of the node-set is ignored the boolean 'true' is returned. If the node-set ends before any match is found, 'false' is returned.

Errors

Table 24.152. Errors signalled by text-contains()

SQLState	Error Code	Error Text	Description
XP370	XPF11	XPATH function text-contains() is allowed only in special SQL predicate xcontains()	The XPath processor is called not by xcontains() but by some other way so it has no suitable free-text data.
XP001	XPF13	Unsupported combination of arguments in XPATH function text-contains()	The XPath processor has failed to built an execution plan for the free-text search.

Examples

Example 24.632.

See Also

xcontains predicate

translate

translate — Performs char-by-char translation of given string

Synopsis

```
string translate ( strg string ,
                   search_list string ,
                   replace_list string ) ;
```

Description

The function returns the *strg* with occurrences of characters in the *search_list* replaced by the character at the corresponding position in the *replace_list* . If there is a character in the *search_list* with no character at a corresponding position in the *replace_list* (because the *replace_list* is longer than the *replace_list*), then occurrences of that character in *strg* string are removed.

If a character occurs more than once in the *search_list* , then the first occurrence determines the replacement character. If the *replace_list* is longer than the *search_list* , then excess characters are ignored.

Two popular use cases for this function are case conversion and sorting with collation. For "to-upper" case conversion, the *search_list* consists of all lowercase characters of some language and the *replace_list* consists of all uppercase characters of that language. For "to-lower" case conversion, uppercase chars are in the *search_list* and lowercase are in the *replace_list* . For sorting with collation, the function must be used in "select" string expression attribute of <xsl:sort> element; the *search_list* consists of all characters reordered by collation and the *replace_list* consists of corresponding characters from "collation string".

Parameters

strg
: String that must be translated.

search_list
: String of characters that must be edited in the strg .

replace_list
: String of characters that must be inserted in the strg .

Return Types

String

Errors

Table 24.153. Errors signalled by translate()

SQLState	Error Code	Error Text	Description
XP001	XPF07	Too long string passed as argument 1 to XPATH function translate(), the result of translation is too long	Virtuoso cannot process strings that are longer than 10M. This limit may be exceed if UTF-8 representations of the replacement characters are longer than representations of replaced characters; even if the number of encoded characters remains the same or decrements.

Examples

Example 24.633. Trivial cases

The following expressions are true:

```
translate("abracadabra","abc","ABC") = "ABrACAdABrA".
translate("ab-ra-ca-dab-ra","abc-","ABC") = "ABrACAdABrA".
```

Example 24.634. Sorting with collation in XSL

The following <xsl:sort> will sort records by its titles, making no difference between spaces and punctuation marks. In addition, spaces will be normalized after the collation processing.

```
<xsl:key select="normalize-space(translate('@title','.,:;!?','      '))"/>
```

See Also

replace() normalize-space()

true

true — Returns true

Synopsis

```
boolean true ( void);
```

Description

This function returns boolean constant "true"

Return Types

Boolean

See Also

false boolean

tuple

tuple — Selects the first item of every argument sequence and returns the sequence of the selected items

Synopsis

```
sequence tuple ( seq1 sequence ,
                 seq2 sequence ,
                 ... ,
                 seqN sequence ) ;
```

Description

This function calculates all given arguments from left to right, and creates a sequence which contains the first item of the first calculated sequence, then the first item of the second calculated sequence and so on. If the value of an argument is not a sequence, but a scalar, the scalar is treated as one-element sequence so it is added into the result. If the value of an argument is an empry sequence, the empty string is added into the result.

This function is not a part of XPATH 1.0 or XQUERY 1.0 libraries of standard functions.

Parameters

SeqI
: The sequence of items, first of its items will be used as I-tn item of the resulting sequence

Return Types

Sequence

Examples

Example 24.635.

```
tuple(/title, /abstract, /intro)
```

See Also

append list

union

union — Returns an union of two or more sets

Synopsis

```
sequence union ( set1 sequence ,
                 set2 sequence ... ,
                 setN sequence ) ;
```

Description

The function returns an unordered sequence that consists of all distinct items from all given sets.

All duplicates are removed, including duplicate occurrences of same value in one sequence, thus the function works exactly as `distinct()` when only one parameter is passed.

Parameters

setI
: Sequence that contains items of the resulting union.

Return Types

The function returns an unordered sequence of distinct values.

See Also

`intersect`

`except`

unordered

unordered — Returns the given sequence in any order.

Synopsis

```
sequence unordered ( input sequence );
```

Description

The function takes a sequence, or more typically an expression that evaluates to a sequence, and indicates that the result sequence may be returned in any order.

Parameters

input
: The sequence of nodes

Return Types

Sequence

Examples

Example 24.636. Finding pairs of books

The following example is from the XQuery standard and it illustrates how to find pairs of books that have different titles but the same set of authors (possibly in a different order).

```
<bib>
{
   for $book1 in document("bib.xml")/bib/book,
       $book2 in document("bib.xml")/bib/book
   where $book1/title > $book2/title
   and $book1/author = $book2/author
   return
      <book-pair>
         { $book1/title, $book2/title }
      </book-pair>
}
</bib>
```

unparsed-entity-uri

unparsed-entity-uri — This XSLT 1.0 function is not implemented in the current version of Virtuoso.

Synopsis

```
string unparsed-entity-uri ( unparsed_entity_name string );
```

urlify

urlify — Converts its argument to a string and returns it in URL encoding.

Synopsis

```
string urlify ( obj string );
```

Description

The urlify() function encodes its argument string according to URI rules (i.e. whitespaces are replaced with plus signs and all special characters are replaced with '%XX' notation. This is especially useful for stylesheets that compose dynamic HTML documents with calculateable values of attributes like <A HREF=...>.

Parameters

obj
: Value to be formatted as URI

Return Types

String

See Also

string()

format_number()

serialize()

concat()

xmlview

xmlview — Returns the xml document corresponding to given XML view

Synopsis

xmlview (in *view_name* varchar) ;

Description

The function returns the output produced by an XML view as it was a content of some XML document accessed via document() XPath function. The result of the function call may be used as input of some path expression that select interesting parts of the full output of the XML view. The XQuery engine will translate the XPath expression into SQL statement in order to avoid redundant data access and to build whole XML tree.

This function may be used only in the header of FOR operator of XQuery. To specify the list of values of a variable, a path expression may be used that starts with the call of xmlview() function.

Parameters

view_name
: The name of xml view

Errors

Table 24.154. Errors signalled by xmlview

SQLState	Error Code	Error Text	Description
37000		Unknown view name is passed as argument of xmlview()	

Examples

Example 24.637.

The XQUERY expressions below are queries to the following XML view:

```
create xml view "cat" as
{
  "Demo"."demo"."Categories" c as "category" ("CategoryID", "Description" as "description")
      {
            "Demo"."demo"."Products" p as "product"  ("ProductName")
                on (p."CategoryID" = c."CategoryID")
      }
};
```

The first query returns all products with the attribute ProductName starting with "L". The second query returns categories with attribute CategoryID less than 10.

```
for $r in xmlview("cat")//product[@ProductName like "L%"] return $r
for $r in xmlview("cat")/category[@CategoryID<"10"] return $r
```

BPEL.BPEL.compile_script

BPEL.BPEL.compile_script — Compile a BPEL script source

Synopsis

BPEL.BPEL.compile_script (in *scp_id* int ,
in *vdir* varchar (default null) ,
in *opts* any (default null) ,
in *no_check* int (default 0)) ;

Parameters

scp_id
: int the BPEL process identifier to be compiled

vdir
: varchar virtual directory to be created for this process (absolute path)

opts
: any options to the virtual directory if specified

no_check
: int flag to check or not partner links, bu default before compilation function will check if every partner link have corresponding WSDL uploaded

Return Types

this function has no return value

Description

`BPEL.BPEL.compile_script` is used to compile BPEL script identified by scp_id and link to it all WSDL definitions have been associated to it.

Examples

Example 24.638. Simple example

The following will compile a BPEL script with identifier equals to 1

```
SQL> create procedure echo_deploy ()
{
  declare scp int;
  BPEL.BPEL.import_script ('file:/echo/bpel.xml', 'Echo', scp);
  BPEL.BPEL.compile_script (scp, '/Echo');
};
Done. -- 0 msec.
SQL> echo_deploy ();
Done. -- 2303 msec.
```

BPEL.BPEL.copy_script

BPEL.BPEL.copy_script — Makes a copy of an existing script

Synopsis

BPEL.BPEL.copy_script (in *scp_id* int);

Parameters

scp_id
: int the ID of the BPEL script to be copied

Return Types

returns a unique integer identifier for the script copy

Description

BPEL.BPEL.copy_script makes a copy of an existing script and put it in edit mode.

Examples

Example 24.639. Simple example

This will copy a script with identifier equal to 1, the copy have Id equal to 2

```
SQL> select BPEL.BPEL.copy_script (1);
unnamed
INTEGER
_______________________________________________________________________________

2

1 Rows. -- 1672 msec.
```

BPEL.BPEL.get_partner_links

BPEL.BPEL.get_partner_links — Returns a list of partner links used in the BPEL script

Synopsis

BPEL.BPEL.get_partner_links (in *scp_id* int);

Parameters

scp_id
: int the ID of BPEL script

Return Types

returns an XML document containing partner links

Description

BPEL.BPEL.get_partner_links The function is used for partner link names retrieval. It returns an XML document containing all partner links.

Examples

Example 24.640. Simple example

```
SQL> select BPEL.BPEL.get_partner_links (4);
callret
VARCHAR
_______________________________________________________________________________

<n0:partnerLinks xmlns:n0="http://schemas.xmlsoap.org/ws/2003/03/business-process/">
    <n0:partnerLink name="client" partnerLinkType="tns:LoanFlow" partnerRole="LoanFlowRequester

    <n0:partnerLink name="creditRatingService" partnerLinkType="services:CreditRatingService" p

    <n0:partnerLink name="UnitedLoanService" partnerLinkType="services:LoanService" myRole="Loa

    <n0:partnerLink name="StarLoanService" partnerLinkType="services:LoanService" myRole="LoanS
</n0:partnerLinks>

1 Rows. -- 88 msec.
```

BPEL.BPEL.instance_delete

BPEL.BPEL.instance_delete — delete a BPEL process instance

Synopsis

BPEL.BPEL.instance_delete (in *id* int);

Parameters

id
: int instance identifier

Return Types

this function has no return value

Description

BPEL.BPEL.instance_delete Deletes the instance, regardless of completion state or such. If there are waits for the instance the waits are flagged as deleted, not physically deleted. This will cause the reply message to be received instead of going to the unexpected messages queue.

Examples

Example 24.641. Simple example

The following will delete process instance with ID equal to 5

```
SQL> BPEL.BPEL.instance_delete (5);
Done. -- 10 msec.
```

BPEL.BPEL.purge_instance

BPEL.BPEL.purge_instance — removes old BPEL process instances

Synopsis

BPEL.BPEL.purge_instance (in *completed_before* datetime ,
in *make_archive* int (default 1)) ;

Parameters

completed_before
: datetime all instances completed before this date and time will be removed

make_archive
: int flag to archive or not instances before removal

Return Types

Description

`BPEL.BPEL.purge_instance` this will erase all instances completed before the datetime specified.

Examples

Example 24.642. Simple example

The following wil lremove all process instances completed before 10 of January 2004.

```
SQL> BPEL.BPEL.purge_instance (stringdate ('2004-01-10'));
Done. -- 10 msec.
```

BPEL.BPEL.script_delete

BPEL.BPEL.script_delete — Remove a BPEL process

Synopsis

BPEL.BPEL.script_delete (in *script* varchar ,
in *delete_instances* int) ;

Parameters

script
: int the identifier of the BPEL process to be deleted

delete_instances
: int this flag specifies to delete all instances of the given process

Return Types

this function has no return value

Description

`BPEL.BPEL.script_delete` Deletes all versions of the BPEL process and all related partner links, instances, sources etc. if delete_instances is false and there exist instances of at least one version of the script the function signals an error and does nothing

Examples

Example 24.643. Simple example

The following will delete script with ID equal of 3 and all it's instances

```
SQL> BPEL.BPEL.script_delete (5, 1);
Done. -- 10 msec.
```

BPEL.BPEL.script_obsolete

BPEL.BPEL.script_obsolete — mark a BPEL script as obsoleted

Synopsis

BPEL.BPEL.script_obsolete (in *script* varchar) ;

Parameters

script
varchar

Return Types

this function has no return value

Description

BPEL.BPEL.script_obsolete Marks the script as obsolete. This means that existing instances will complete but no new instances can be made. The effect can be reversed by going to edit mode and recompiling a new version which then becomes current. Directly after this function there will be no current version but there may be a version being edited.

Examples

Example 24.644. Simple example

```
SQL> BPEL.BPEL.script_obsolete (5);
Done. -- 10 msec.
```

BPEL.BPEL.script_source_update

BPEL.BPEL.script_source_update — Updates an existing script in BPEL repository

Synopsis

```
BPEL.BPEL.script_source_update (  in scp_id int ,
                                  in url varchar ,
                                  in content any );
```

Parameters

scp_id
: int the ID of the BPEL script to be updated with new source

url
: varchar the URL to the script source

content
: any as an alternative this is to specify script source directly.

Return Types

This function does not return a value

Description

`BPEL.BPEL.script_source_update` is used to update the script source of an already uploaded script. The script must be in edit mode in order to do that. The URL and content parameters works as in script_upload.

Examples

Example 24.645. Simple example

```
SQL> BPEL.BPEL.script_source_update (1, 'file:/LoanFlow.bpel', null);
Done. -- 10 msec.
```

BPEL.BPEL.script_upload

BPEL.BPEL.script_upload — Upload a new BPEL script source

Synopsis

BPEL.BPEL.script_upload (in *name* varchar ,
in *url* varchar ,
in *content* any) ;

Parameters

name
: varchar a human readable name for script

url
: varchar the URL to the script source

content
: any as an alternative to the URL, the content can be specified.

Return Types

returns an integer unique identifier for the uploaded script

Description

`BPEL.BPEL.script_upload` uploads a new BPEL script, makes expanded version, and put into edit mode. If content is supplied the function will not try to download or check the URL specified.

Examples

Example 24.646. Simple example

```
SQL> select BPEL.BPEL.script_upload ('LoanFlow', 'file:/LoanFlow.bpel', null);
unnamed
INTEGER
_______________________________________________________________________________

1
```

BPEL.BPEL.wsdl_upload

BPEL.BPEL.wsdl_upload — Uploads a WSDL description for a given BPEL script

Synopsis

```
BPEL.BPEL.wsdl_upload (  in scp_id int ,
                         in url varchar ,
                         in content any ,
                         in pl varchar default 'wsdl' ) ;
```

Parameters

scp_id
: int the BPEL script identifier

url
: varchar the URL to the WSDL description file.

content
: any default null the source of the WSDL file, this can be omitted then the function will try downloading the source from the URL specified

pl
: varchar default 'wsdl' the partner which is described by given WSDL. When this is a file describing the process itself the default value MUST be used.

Return Types

this function has no return value

Description

`BPEL.BPEL.wsdl_upload` uploads a WSDL description for a given partner link. The function if content is not specified this will download the content via the URL specified and will associate it with the specified partner link. Also after download the WSDL will be expanded to resolve includes in WSDL and XMLSchema namespaces.

Examples

Example 24.647. Simple example

```
SQL> BPEL.BPEL.wsdl_upload (1, 'file:/LoanFlow.wsdl', null, 'wsdl');
Done. -- 10 msec.
```

BPEL.BPEL.getVariableData

BPEL.BPEL.getVariableData — gets BPEL variable data within <bpelv:exec binding="SQL"> activity

Synopsis

BPEL.BPEL.getVariableData (in *var_name* varchar ,
in *part* varchar ,
in *query* varchar) ;

Parameters

var_name
: varchar the name of BPEL variable

part
: varchar default null the part of BPEL variable in question.

query
: varchar default null XPATH query for selecting data in the BPEL variable

Return Types

this function returns the selected data from the BPEL variable. This can be either an XML tree or a varchar or an integer.

Description

`BPEL.BPEL.getVariableData` Returns select by query expression data from part (if applicable) of the BPEL variable. If the variable is unknown, or if the XPATH expression signals an error, the appropriate error is signalled. This procedure may only be used in Virtuoso/PL code invoked from a BPEL process.

Examples

Example 24.648. Simple example

```
declare city, country varchar;
city := cast (BPEL.BPEL.getVariableData ('request',
                'req_payload',
                '/destRequest/city/text()');
```

BPEL.BPEL.setVariableData

BPEL.BPEL.setVariableData — sets BPEL variable data within code invoked from <bpelv:exec binding="SQL"> activity

Synopsis

```
BPEL.BPEL.setVariableData ( in var_name varchar ,
                            in value any ,
                            in part varchar ,
                            in query varchar ) ;
```

Parameters

var_name
: varchar the name of BPEL variable

value
: any the value to be set. Can be XML tree or varchar

part
: varchar default null the part of BPEL variable in question.

query
: varchar default null XPATH query for selecting data in the BPEL variable

Return Types

this function has no return value.

Description

`BPEL.BPEL.setVariableData` Sets the data selected by "query" in the part (named by "part" argument) of the BPEL variable. If there is no variable with such name or the query contains syntax errors or if its evaluation signals a run time error, the appropriate error is signalled. This procedure may only be used in Virtuoso/PL code invoked from a BPEL process.

Examples

Example 24.649. Simple example

```
declare country any;

...

if (country is null)
  BPEL.BPEL.setVariableData ('res',
        xtree_doc (']]&gt;<destResponse><country>Unknown</country></destResponse><![CDATA['),
        'repl_payload');
 else
  BPEL.BPEL.setVariableData ('res',
        country,
        'repl_payload',
        '/destResponse/country');
```

BPEL.BPEL.plink_get_option

BPEL.BPEL.plink_get_option — retrieving a partner link option

Synopsis

BPEL.BPEL.plink_get_option (in *script* varchar ,
in *plink* varchar ,
in *opt* varchar) ;

Parameters

script
: varchar name of a process that partner link belongs to

plink
: varchar name of the partner link

opt
: varchar name of the options to be retrieved

Return Types

The function returns a string or array of strings depending of the requested option.

Description

`BPEL.BPEL.plink_get_option` This function can be used to retrieve the options assigned to a particular partner link. These options are used to configure various Web Service protocols (see Protocol Support section in BPEL reference chapter). For available options see `plink_set_option()` function.

Examples

Example 24.650. Simple example

```
SQL> select BPEL.BPEL.plink_get_option ('WSSecho', 'service', 'wss-in-encrypt');
callret
VARCHAR
_______________________________________________________________________________

NONE
```

BPEL.BPEL.plink_set_option

BPEL.BPEL.plink_set_option — set value of a partner link option

Synopsis

```
BPEL.BPEL.plink_set_option (  in script varchar ,
                              in plink varchar ,
                              in opt varchar ,
                              in val any ) ;
```

Parameters

script
: varchar name of a process that partner link belongs to

plink
: varchar name of the partner link

opt
: varchar name of the options to be set

val
: any value of the option to be set

Return Types

The function has no return value

Description

`BPEL.BPEL.plink_set_option` This function is used to set an option of the specified partner link. The options are used to configure WS-Security, WS-Addressing and WS-Reliable Messaging protocols for sending and receiving messages to the given partner.

- *wsa* the version of namespace value for WS-Addressing protocol.
- *http-auth-uid* user name for basic HTTP authentication
- *http-auth-pwd* password for HTTP authentication
- *wss-priv-key* name of the private key to be used for signing the requests
- *wss-pub-key* name of the partner's public key to be used for encryption of the requests
- *wss-in-encrypt* incoming message XML data encryption policy
- *wss-in-signature* incoming messages XML signature policy
- *wss-in-signers* list of public key names which are trusted (empty list mean all are trusted)
- *wss-out-encrypt-key* type of the session key for outgoing message encryption
- *wss-out-signature-type* XML signature template type for signing the outgoing messages
- *wss-out-signature-function* in case of custom signature this option is a name of the PL function producing the template for XML signature
- *wsrm-in-type* WS-Reliable Messaging protocol usage for incoming messages
- *wsrm-out-type* WS-Reliable Messaging protocol usage for outgoing messages

Examples

Example 24.651. Simple example

```
SQL> BPEL.BPEL.plink_set_option  ('WSSecho', 'service', 'wss-in-encrypt', 'Mandatory');

Done. -- 16 msec.
```

BPEL.BPEL.import_script

BPEL.BPEL.import_script — Import a new BPEL process

Synopsis

BPEL.BPEL.import_script (in *base_uri* varchar ,
in *base_name* varchar ,
in *scp_id* int) ;

Parameters

base_uri
: varchar the URL to the bpel.xml which contains information for bpel and wsdl for process and its partnerlinks wsdl's urls.

base_name
: varchar a human readable name for process

scp_id
: int id of the process.

Description

`BPEL.BPEL.import_script` imports a new BPEL process or refetches the content of bpel, wsdl files and wsdl for partner links for a given process. In both cases the process is turned to mode "Edit". Urls of wsdl files must have absolute paths.

Examples

Example 24.652. Simple example

```
SQL>create procedure echo_import ()
   {
    declare scp int;
    BPEL.BPEL.import_script ('file:/echo/bpel.xml', 'Echo', scp);
   }
  ;

SQL> echo_import ();
Done. -- 10 msec.
```

GeometryType

GeometryType — returns EWKT type name of a shape.

Synopsis

GeometryType (in *shape* any);

Description

GeometryType function returns EWKT type name of a given shape.

Parameters

shape
: Geometry.

Return Types

Returns varchar.

Examples

Example 24.653. Simple example

```
SQL> DB.DBA.TTLP ('
<point2d>               <shape> "POINT(1 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<point4d>               <shape> "POINTZM(0 1 2 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry>
<box1>                  <shape> "BOX(0 0, 2 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<linestring>            <shape> "LINESTRING(10 21, 12 23, 10 25)"^^<http://www.openlinksw.com/schemas/vir
<polygon1>              <shape> "POLYGON((1 3,2 4,1 5,0 4,1 3))"^^<http://www.openlinksw.com/schemas/virt
<polygon2>              <shape> "POLYGON((1.5 3.5,2.5 4.5,1.5 5.5,0.5 4.5,1.5 3.5))"^^<http://www.openlin
<polygon3>              <shape> "POLYGON((1 3,2 4,1 5,0 4,1 3),(1 3.5,1.5 4,1 4.5,0.5 4,1 3.5))"^^<http:/
<multipolygon>          <shape> "MULTIPOLYGON(((1 3,2 4,1 5,0 4,1 3)),((1 2,2 1,1 0,0 1,1 2)))"^^<http://
<collection>            <shape> "GEOMETRYCOLLECTION(POINT(0 0),MULTILINESTRING((1 5,0 1,4 0,5 4),(1 5,1 0
', 'http://geo-api-demo/', 'http://geo-api-demo/', 0)

SQL>
 SPARQL
 SELECT ?s,
        bif:GeometryType(?o)
 FROM <http://geo-api-demo/>
 WHERE
   {
     ?s ?p ?o
   }
 ORDER BY ASC(str(?s))
s                                callret-1
VARCHAR                          VARCHAR
_______________________________________________________________________________

http://geo-api-demo/box1         BOX2D
http://geo-api-demo/collection   GEOMETRYCOLLECTION
http://geo-api-demo/linestring   LINESTRING
http://geo-api-demo/multipolygon MULTIPOLYGON
http://geo-api-demo/point2d      POINT
http://geo-api-demo/point4d      POINTZM
http://geo-api-demo/polygon1     POLYGON
http://geo-api-demo/polygon2     POLYGON
http://geo-api-demo/polygon3     POLYGON
```

See Also

`ST_GeometryN()`

`ST_NumGeometries()`

http_st_dxf_entity

http_st_dxf_entity — writes a DXF (Data Exchange Format) representation of shape into the given output session.

Synopsis

http_st_dxf_entity (in *shape* any ,
in *attrs* vector ,
in *sess* any) ;

Description

This function writes a DXF (Data Exchange Format) representation of shape into the given output session.

Parameters

shape
: If shape is null, the function returns without writing anything to the shape.

attrs
: vector of arbitrary DXF properties in form (tag1, value1, tag2, value2...) where tags are integer codes according to DXF specification; related values are not validated and are printed to the session as-is. Current version does not support ARCSTRINGs, CURVEs, nor CURVEPOLYGONs.

sess
: Output session.

Examples

Example 24.654. Simple example

```
create procedure Example_http_st_dxf_entity ()
{
  declare ses, ctx any;
  ses := string_output ();

  for (select a1, deserialize (Example_GEOM) as geom from Demo.Demo."Example_Countries") do
    http_st_dxf_entity (geom, vector (62, 0)), ses);
  string_to_file ('examples/Example_http_st_dxf_entity.dxf', ses, -2);
}
;
```

http_st_ewkt

http_st_ewkt — Writes a EWKT representation of a shape to the given session.

Synopsis

http_st_ewkt (in *shape* any ,
inout *sess* any) ;

Description

Writes a EWKT representation of a shape to the given session, a fast replacement for http (st_astext (shape), ses).

Parameters

shape
: A given shape.

sess
: Session.

Return Types

Returns any.

Examples

Example 24.655. Simple example

```
create procedure Example_http_st_ewkt ()
{
  declare g, ses, ctx, g_ewkt any;
  ses := string_output ();

  g := st_ewkt_read ('POLYGON
                      (
                        (1 3,2 4,1 5,0 4,1 3),
                        (1 3.5,1.5 4,1 4.5,0.5 4,1 3.5)
                      )
                     ');
  ses := string_output ();
  http_st_ewkt (g, ses);
}
;
```

See Also

`http()`

`st_ewkt_read()`

`http_st_dxf_entity()`

st_ewkt_read

st_ewkt_read — Parses the given text as a EWKT and returns the parsed shape.

Synopsis

st_ewkt_read (in *ewkt_text* any);

Description

Parses the given text as a EWKT and returns the parsed shape.

Parameters

ewkt_text
: A given text to be parsed as EWKT.

Return Types

This function returns shape.

Examples

Example 24.656. Simple Use

```
select st_translate (
         st_ewkt_read ('POLYGON((1 3,2 4,1 5,0 4,1 3),
                                (1 3.5,1.5 4,1 4.5,0.5 4,1 3.5))'),
                       0.1, 0.2);
Query result:
geometry
 POLYGON((1.100000 3.200000,2.100000 4.200000,1.100000 5.200000,0.100000 4.200000,1.100000 3.200000),
         (1.100000 3.700000,1.600000 4.200000,1.100000 4.700000,0.600000 4.200000,1.100000 3.700000))
No. of rows in result: 1
```

See Also

`http_st_ewkt()`

postgis_proj_version

postgis_proj_version — Returns the version of v7proj4 plugin in use, as a string, for compatibility with PostGIS.

Synopsis

```
postgis_proj_version (  );
```

Description

Returns the version of the v7proj4 plugin in use, as a string, for compatibility with PostGIS.

Return Types

any

dist_from_point_to_line_segment

dist_from_point_to_line_segment — returns the distance between a point and a segment on a plane.

Synopsis

dist_from_point_to_line_segment (in *Xpoint* any ,
in *Ypoint* any ,
in *Xsegment1* any ,
in *Ysegment1* any ,
in *Xsegment2* any ,
in *Ysegment2* any) ;

Description

returns the distance between a point and a segment on a plane.

Parameters

Xpoint
: A X point

Ypoint
: A Y point

Xsegment1
: X Segment

Ysegment1
: Y Segment

Xsegment2
: X Segment

Ysegment2
: Y Segment

Return Types

Returns any.

Examples

Example 24.657. Simple Use

```
SQL> select dist_from_point_to_line_segment (5, 5, 0, 3, 4, 0)

4.6

SQL> select dist_from_point_to_line_segment (5, 0, 0, 3, 4, 0)

1

SQL> select dist_from_point_to_line_segment (5, -1, 0, 3, 4, 0)

1.414213562373095

SQL> select dist_from_point_to_line_segment (5, -5, 0, 3, 4, 0)
```

```
5.099019513592784
```

See Also

haversine_deg_km()

st_point

st_x

st_y

ST_Z

st_distance

ST_SetSRID

st_astext

st_geomfromtext

st_intersects

st_contains

st_within

isgeometry

geo_insert

geo_delete

DB.DBA.RDF_GEO_ADD

DB.DBA.RDF_GEO_FILL

earth_radius

earth_radius — returns geom.mean of radius of Earth

Synopsis

```
earth_radius (   );
```

Description

returns geom.mean of radius of Earth in kilometers, 6367.43568

Return Types

Returns any.

Examples

Example 24.658. Simple Use

```
SELECT earth_radius ()
earth_radius
ANY
 6367.43568
```

dist_from_point_to_line_segment()()

st_point()

st_x()

st_y()

ST_Z()

st_distance()

ST_SetSRID()

st_astext()

st_geomfromtext()

st_intersects()

st_contains()

st_within()

isgeometry()

geo_insert()

geo_delete()

DB.DBA.RDF_GEO_ADD()

```
DB.DBA.RDF_GEO_FILL()
```

ST_ExteriorRing

ST_ExteriorRing — Returns an external (the very first) ring of a polygon.

Synopsis

ST_ExteriorRing (in *poligon* any);

Description

Returns an external (the very first) ring of a polygon.

Parameters

poligon
: A given poligon from a type shape.

Return Types

Returns any.

Examples

Example 24.659. Simple Use

```
SQL> select ST_ExteriorRing (st_ewkt_read ('POLYGON((1 3,2 4,1 5,0 4,1 3),(1 3.5,1.5 4,1 4.5,0.5 4,1 3.5)
_____________________________________________________________________________

RING(1.0 3.0,2.0 4.0,1.0 5.0,0.0 4.0,1.0 3.0)
```

See Also

`ST_InteriorRingN()`

`ST_NumInteriorRings()`

`st_point`

`st_x`

`st_y`

`ST_Z`

`st_distance`

`ST_SetSRID`

`st_astext`

`st_geomfromtext`

`st_intersects`

`st_contains`

`st_within`

```
isgeometry

geo_insert

geo_delete

DB.DBA.RDF_GEO_ADD

DB.DBA.RDF_GEO_FILL
```

ST_GeometryN

ST_GeometryN — Given a 1-based index of a member of a MULTI... or ...COLLECTION shape, returns the member.

Synopsis

ST_GeometryN (in *shape* any ,
in *idx* any) ;

Description

Given a 1-based index of a member of a MULTI... or ...COLLECTION shape, returns the member.

Parameters

shape
: Shape.

idx
: Index.

Return Types

Returns any.

Examples

Example 24.660. Simple Use

```
SQL> select st_geometryn (
       st_geometryn (
         st_ewkt_read ('
           GEOMETRYCOLLECTION(
              POINT(0 0),
              MULTILINESTRING(
                (1 5,0 1,4 0,5 4),
                (1 5,1 0,4 0,4 5)),
              MULTIPOLYGON(
               ((1 3,2 4,1 5,0 4,1 3)),
               ((1 2,2 1,1 0,0 1,1 2))))'),
        3),1)
_______________________________________________________________________________

POLYGON((1.0 3.0,2.0 4.0,1.0 5.0,0.0 4.0,1.0 3.0))
```

See Also

`ST_NumGeometries()`

`GeometryType()`

`st_point`

`st_x`

`st_y`

`ST_Z`

st_distance

ST_SetSRID

st_astext

st_geomfromtext

st_intersects

st_contains

st_within

isgeometry

geo_insert

geo_delete

DB.DBA.RDF_GEO_ADD

DB.DBA.RDF_GEO_FILL

st_get_bounding_box

st_get_bounding_box — returns BOX2D that is a bounding box of a shape.

Synopsis

st_get_bounding_box (in *shape* any);

Description

returns BOX2D that is a bounding box of a shape.

Parameters

shape
: Shape.

Return Types

Returns any.

Examples

Example 24.661. Simple Use

```
SQL> SPARQL
SELECT bif:st_get_bounding_box(?o) ?o
FROM <http://geo-api-demo/>
WHERE { ?s ?p ?o }
ORDER BY ASC(str(?s))
o                                                                    callret-3

BOX2D(0.0 0.0,2.0 3.0)                          BOX2D(0.0 0.0,2.0 3.0)
BOX2D(0.0 0.0,5.000001 5.000001)                GEOMETRYCOLLECTION(POINT(0.0 0.0),MULTILINESTRING((1.0 5.0
BOX2D(9.999999 20.999997,12.000001 25.000003)   LINESTRING(10.0 21.0,12.0 23.0,10.0 25.0)
BOX2D(0.0 0.0,2.0 5.000001)                     MULTIPOLYGON(((1.0 3.0,2.0 4.0,1.0 5.0,0.0 4.0,1.0 3.0)),(
BOX2D(1.0 3.0,1.0 3.0)                          POINT(1 3)
BOXZM(0.0 1.0 2.0 3.0,0.0 1.0 2.0 3.0)          POINTZM(0.0 1.0 2.0 3.0)
BOX2D(0.0 3.0,2.0 5.000001)                     POLYGON((1.0 3.0,2.0 4.0,1.0 5.0,0.0 4.0,1.0 3.0))
BOX2D(0.500000 3.500000,2.500000 5.500001)      POLYGON((1.500000 3.500000,2.500000 4.500000,1.500000 5.50
BOX2D(0.0 3.0,2.0 5.000001)                     POLYGON((1.0 3.0,2.0 4.0,1.0 5.0,0.0 4.0,1.0 3.0),(1.0 3.5
```

See Also

`st_get_bounding_box_n()`

`st_point`

`st_x`

`st_y`

`ST_Z`

`st_distance`

`ST_SetSRID`

`st_astext`

st_geomfromtext

st_intersects

st_contains

st_within

isgeometry

geo_insert

geo_delete

DB.DBA.RDF_GEO_ADD

DB.DBA.RDF_GEO_FILL

st_get_bounding_box_n

st_get_bounding_box_n — Given a 1-based index of a member of a MULTI... or ...COLLECTION shape, returns the bounding box of a member.

Synopsis

st_get_bounding_box_n (in *shape* any ,
in *idx* any) ;

Description

Given a 1-based index of a member of a MULTI... or ...COLLECTION shape, returns the bounding box of a member. This is a fast equivalent of `st_get_bounding_box_n(ST_GeometryN (shape,idx))`.

Parameters

shape
: Shape

idx
: Index

Return Types

Returns any.

Examples

Example 24.662. Simple Use

```
SQL> select st_get_bounding_box_n (
               st_ewkt_read ('
                 GEOMETRYCOLLECTION(
                   POINT(0 0),
                   MULTILINESTRING(
                     (1 5,0 1,4 0,5 4),
                     (1 5,1 0,4 0,4 5)),
                   MULTIPOLYGON(
                     ((1 3,2 4,1 5,0 4,1 3)),
                     ((1 2,2 1,1 0,0 1,1 2))))'), 1)
unnamed
VARCHAR NOT NULL
_______________________________________________________________________________

BOX2D(0.0 0.0,0.0 0.0)

SQL> select st_get_bounding_box_n (
               st_ewkt_read ('
                 GEOMETRYCOLLECTION(
                   POINT(0 0),
                   MULTILINESTRING(
                     (1 5,0 1,4 0,5 4),
                     (1 5,1 0,4 0,4 5)),
                   MULTIPOLYGON(
                     ((1 3,2 4,1 5,0 4,1 3)),
                     ((1 2,2 1,1 0,0 1,1 2))))'), 2)
unnamed
VARCHAR NOT NULL
_______________________________________________________________________________

BOX2D(0.0 0.0,5.000001 5.000001)
```

See Also

st_get_bounding_box()

st_point

st_x

st_y

ST_Z

st_distance

ST_SetSRID

st_astext

st_geomfromtext

st_intersects

st_contains

st_within

isgeometry

geo_insert

geo_delete

DB.DBA.RDF_GEO_ADD

DB.DBA.RDF_GEO_FILL

haversine_deg_km

haversine_deg_km — Given latitudes and longitudes of two points in degrees, calculates the approximate distance between that points in kilometers

Synopsis

haversine_deg_km (in *lat1* double precision ,
in *long1* double precision ,
in *lat2* double precision ,
in *long2* double precision) ;

Description

The function calculates the approximate distance between two points, using haversine formula with an adjustment for case when points are ends of some diameter.

Note that the order of arguments is latitude-longitude, not longitude-latitude.

Parameters

lat1
: latitude

long1
: longitude

lat2
: latitude

long2
: longitude

Return Types

The function returns distance in kilometers as double precision.

Examples

Example 24.663. Simple example

```
-- select haversine_deg_km (-90,0,90,0)
haversine_deg_km
VARCHAR
_______________________________________________________________________________

20003.88915449353

-- select haversine_deg_km (0,0,0,180) - haversine_deg_km (-90,0,90,0)
temp
VARCHAR
_______________________________________________________________________________

33.61918829571368
```

See Also

earth_radius()

dist_from_point_to_line_segment()

st_point

st_x

st_y

ST_Z

st_distance

ST_SetSRID

st_astext

st_geomfromtext

st_intersects

st_contains

st_within

isgeometry

geo_insert

geo_delete

DB.DBA.RDF_GEO_ADD

DB.DBA.RDF_GEO_FILL

ST_InteriorRingN

ST_InteriorRingN — Given a 1-based index of an interior ring of a polygon, returns the ring. Wrong index is not reported as an error and NULL is returned.

Synopsis

ST_InteriorRingN (in *poligon* any ,
in *idx* any) ;

Description

Given a 1-based index of an interior ring of a polygon, returns the ring. Wrong index is not reported as an error and NULL is returned.

Parameters

poligon
: A given poligon of a type shape.

idx
: Index.

Return Types

Returns any.

Examples

Example 24.664. Simple Use

```
SQL> select ST_InteriorRingN (st_ewkt_read ('POLYGON((1 3,2 4,1 5,0 4,1 3),(1 3.5,1.5 4,1 4.5,0.5 4,1 3.5
unnamed
VARCHAR NOT NULL
_______________________________________________________________________________

RING(1.0 3.500000,1.500000 4.0,1.0 4.500000,0.500000 4.0,1.0 3.500000)

SQL> DB.DBA.TTLP ('
<point2d>       <shape> "POINT(1 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<point4d>       <shape> "POINTZM(0 1 2 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<box1>          <shape> "BOX(0 0, 2 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<linestring>    <shape> "LINESTRING(10 21, 12 23, 10 25)"^^<http://www.openlinksw.com/schemas/virtrdf#Geo
<polygon1>      <shape> "POLYGON((1 3,2 4,1 5,0 4,1 3))"^^<http://www.openlinksw.com/schemas/virtrdf#Geom
<polygon2>      <shape> "POLYGON((1.5 3.5,2.5 4.5,1.5 5.5,0.5 4.5,1.5 3.5))"^^<http://www.openlinksw.com/
<polygon3>      <shape> "POLYGON((1 3,2 4,1 5,0 4,1 3),(1 3.5,1.5 4,1 4.5,0.5 4,1 3.5))"^^<http://www.ope
<multipolygon>  <shape> "MULTIPOLYGON(((1 3,2 4,1 5,0 4,1 3)),((1 2,2 1,1 0,0 1,1 2)))"^^<http://www.open
<collection>    <shape> "GEOMETRYCOLLECTION(POINT(0 0),MULTILINESTRING((1 5,0 1,4 0,5 4),(1 5,1 0,4 0,4 5
', 'http://geo-api-demo/', 'http://geo-api-demo/', 0)

SQL> SPARQL
SELECT ?s, bif:ST_InteriorRingN (?o, bif:ST_NuminteriorRings (?o))
FROM <http://geo-api-demo/>
WHERE
  {
    ?s ?p ?o .
    FILTER (bif:ST_NumInteriorRings (?o) > 0)
  }
ORDER BY ASC(str(?s))
s                           callret-1
VARCHAR                     VARCHAR
_______________________________________________________________________________

http://geo-api-demo/polygon3  RING(1.0 3.500000,1.500000 4.0,1.0 4.500000,0.500000 4.0,1.0 3.500000)
```

See Also

ST_ExteriorRing()

ST_NumInteriorRings()

st_point

st_x

st_y

ST_Z

st_distance

ST_SetSRID

st_astext

st_geomfromtext

st_intersects

st_contains

st_within

isgeometry

geo_insert

geo_delete

DB.DBA.RDF_GEO_ADD

DB.DBA.RDF_GEO_FILL

st_linestring

st_linestring — returns a linestring in default SRID

Synopsis

st_linestring (in *item1* any ,
in *item2* any ,
in *itemN* any) ;

Description

returns a linestring in default SRID, the coordinates of vertices are specified by arguments that are points, 2-, 3- or 4-item vectors of coordinates, linestrings, arcstrings or vectors of the above mentioned values. Repeating vertices are automatically removed, except the case of repeating vertices in the middle of a linestring/arcstring argument.

Parameters

item1
: Geometry.

item2
: Geometry.

itemN
: Geometry.

Return Types

Returns varchar.

Examples

Example 24.665. Simple Use

```
SQL> select st_linestring (st_point (11,22),
                           st_point (13, 24),
                           st_point (13,24),
                           st_point (15,26),
                           st_point (17,28))
unnamed
VARCHAR NOT NULL
_______________________________________________________________________________

LINESTRING(11.0 22.0,13.0 24.0,15.0 26.0,17.0 28.0)

SQL> select st_linestring (st_linestring (vector(vector (11,22),
                                                 vector (13, 24))),
                           st_linestring (vector (13,24), vector(vector (15,26),
                                                                 vector (17,28))))
st_linestring
VARCHAR
_______________________________________________________________________________

LINESTRING(11.0 22.0,13.0 24.0,15.0 26.0,17.0 28.0)
```

See Also

ST_X()

ST_Y()

ST_Z()

ST_M()

ST_XMin()

ST_XMax()

ST_YMin()

ST_YMax()

ST_ZMin()

ST_ZMax()

ST_MMin()

ST_MMax()

ST_M

ST_M — returns the M coordinate.

Synopsis

ST_M (in *g* any);

Description

returns the M coordinate. M is for mileage or the like. M equal to 245.3 could be 245.3 km of road or 0.3 distance from buoy 245 to buoy 246 or similar.

Parameters

g
: Geometry point

Return Types

Returns double precision.

Examples

Example 24.666. Simple Use

```
SELECT st_m( st_point (10, 20, 30, 40));
unnamed
DOUBLE PRECISION
40
```

See Also

`ST_X()`

`ST_Y()`

`ST_Z()`

`ST_XMin()`

`ST_XMax()`

`ST_YMin()`

`ST_YMax()`

`ST_ZMin()`

`ST_ZMax()`

`ST_MMin()`

`ST_MMax()`

st_may_intersect

st_may_intersect — checks whether bounding boxes of two shapes intersect or some of its points are within the specified proximity.

Synopsis

st_may_intersect (in *shape1* any ,
in *shape2* any ,
in *proximity* any) ;

Description

checks whether bounding boxes of two shapes intersect or some of its points are within the specified proximity. This is much faster than full `st_intersects()` check.

Parameters

shape1
: Given shape

shape2
: Given shape

proximity
: A proximity.

Return Types

Returns any.

Examples

Example 24.667. Simple Use

```
SQL> SPARQL
SELECT ?s1 ?s2
FROM <http://geo-api-demo/>
WHERE
  {
    ?s1 ?p1 ?o1 .
    ?s2 ?p2 ?o2 .
    FILTER (bif:st_may_intersect (?o1, ?o2))
  }
ORDER BY ASC(str(?s1)) ASC(str(?s2));

s1                              s2
ANY                             ANY
_______________________________________________________________________________
http://geo-api-demo/box1                http://geo-api-demo/box1
http://geo-api-demo/box1                http://geo-api-demo/collection
http://geo-api-demo/box1                http://geo-api-demo/multipolygon
http://geo-api-demo/box1                http://geo-api-demo/point2d
http://geo-api-demo/box1                http://geo-api-demo/point4d
http://geo-api-demo/box1                http://geo-api-demo/polygon1
http://geo-api-demo/collection  http://geo-api-demo/box1
http://geo-api-demo/collection  http://geo-api-demo/collection
http://geo-api-demo/collection  http://geo-api-demo/multipolygon
```

See Also

st_intersects()

st_point

st_x

st_y

ST_Z

st_distance

ST_SetSRID

st_astext

st_geomfromtext

st_contains

st_within

isgeometry

geo_insert

geo_delete

DB.DBA.RDF_GEO_ADD

DB.DBA.RDF_GEO_FILL

ST_MMax

ST_MMax — returns boundaries of a bounding box around a shape in cordinates.

Synopsis

ST_MMax (in *shape* any) ;

Description

returns boundaries of a bounding box around a shape. Bounding boxes around arcs are calculated in assumption that no one arc is longer than a half of full circle.

Parameters

shape
: A bounding box.

Return Types

Returns double precision.

Examples

Example 24.668. Simple Use

```
DB.DBA.TTLP ('
<point2d>           <shape> "POINT(1 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<point4d>           <shape> "POINTZM(0 1 2 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<box1>              <shape> "BOX(0 0, 2 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<linestring>        <shape> "LINESTRING(10 21, 12 23, 10 25)"^^<http://www.openlinksw.com/schemas/virtrdf
<polygon1>          <shape> "POLYGON((1 3,2 4,1 5,0 4,1 3))"^^<http://www.openlinksw.com/schemas/virtrdf#
<polygon2>          <shape> "POLYGON((1.5 3.5,2.5 4.5,1.5 5.5,0.5 4.5,1.5 3.5))"^^<http://www.openlinksw.
<polygon3>          <shape> "POLYGON((1 3,2 4,1 5,0 4,1 3),(1 3.5,1.5 4,1 4.5,0.5 4,1 3.5))"^^<http://www
<multipolygon>      <shape> "MULTIPOLYGON(((1 3,2 4,1 5,0 4,1 3)),((1 2,2 1,1 0,0 1,1 2)))"^^<http://www.
<collection>        <shape> "GEOMETRYCOLLECTION(POINT(0 0),MULTILINESTRING((1 5,0 1,4 0,5 4),(1 5,1 0,4 0
', 'http://geo-api-demo/', 'http://geo-api-demo/', 0)

--
SQL>
SPARQL
SELECT ?s, bif:st_mmax(?o) as ?mmax
FROM <http://geo-api-demo/>
WHERE { ?s ?p ?o };
s                                   mmax
ANY
_____________________________________________________________________________
 http://geo-api-demo/box1                BOX2D(0.000000 0.000000,2.000000 3.000000)      1e+38
 http://geo-api-demo/linestring    LINESTRING(10.000000 21.000000,12.000000 23.000000,10.000000 25.000000
 http://geo-api-demo/point4d         POINTZM(0.000000 1.000000 2.000000 3.000000)        3
```

See Also

ST_X()

ST_Y()

ST_Z()

ST_M()

ST_XMin()

ST_XMax()

ST_YMin()

ST_YMax()

ST_ZMin()

ST_ZMax()

ST_MMin()

ST_MMin

ST_MMin — returns boundaries of a bounding box around a shape in cordinates.

Synopsis

ST_MMin (in *shape* any) ;

Description

returns boundaries of a bounding box around a shape. Bounding boxes around arcs are calculated in assumption that no one arc is longer than a half of full circle.

Parameters

shape
: A bounding box.

Return Types

Returns double precision.

Examples

Example 24.669. Simple Use

```
DB.DBA.TTLP ('
<point2d>           <shape> "POINT(1 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<point4d>           <shape> "POINTZM(0 1 2 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<box1>              <shape> "BOX(0 0, 2 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<linestring>        <shape> "LINESTRING(10 21, 12 23, 10 25)"^^<http://www.openlinksw.com/schemas/virtrdf
<polygon1>          <shape> "POLYGON((1 3,2 4,1 5,0 4,1 3))"^^<http://www.openlinksw.com/schemas/virtrdf#
<polygon2>          <shape> "POLYGON((1.5 3.5,2.5 4.5,1.5 5.5,0.5 4.5,1.5 3.5))"^^<http://www.openlinksw.
<polygon3>          <shape> "POLYGON((1 3,2 4,1 5,0 4,1 3),(1 3.5,1.5 4,1 4.5,0.5 4,1 3.5))"^^<http://www
<multipolygon>      <shape> "MULTIPOLYGON(((1 3,2 4,1 5,0 4,1 3)),((1 2,2 1,1 0,0 1,1 2)))"^^<http://www.
<collection>        <shape> "GEOMETRYCOLLECTION(POINT(0 0),MULTILINESTRING((1 5,0 1,4 0,5 4),(1 5,1 0,4 0
', 'http://geo-api-demo/', 'http://geo-api-demo/', 0)

--
SQL>
SPARQL
SELECT ?s, bif:st_mmin(?o) as ?mmin
FROM <http://geo-api-demo/>
WHERE { ?s ?p ?o };
s                                   mmin
ANY
________________________________________________________________________________
 http://geo-api-demo/box1                BOX2D(0.000000 0.000000,2.000000 3.000000)      1e+38
 http://geo-api-demo/linestring    LINESTRING(10.000000 21.000000,12.000000 23.000000,10.000000 25.000000
 http://geo-api-demo/point4d         POINTZM(0.000000 1.000000 2.000000 3.000000)        3
```

See Also

ST_X()

ST_Y()

ST_Z()

ST_M()

ST_XMin()

ST_XMax()

ST_YMin()

ST_YMax()

ST_ZMin()

ST_ZMax()

ST_MMax()

ST_NumGeometries

ST_NumGeometries — returns number of members of a MULTI... or ...COLLECTION shape, 1 for other sorts of shapes.

Synopsis

ST_NumGeometries (in *shape* any);

Description

returns number of members of a MULTI... or ...COLLECTION shape, 1 for other sorts of shapes.

Parameters

shape
: MULTI... or ...COLLECTION shape

Return Types

Returns int.

Examples

Example 24.670. Simple Use

```
SQL> DB.DBA.TTLP ('
<point2d>           <shape> "POINT(1 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<point4d>           <shape> "POINTZM(0 1 2 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<box1>              <shape> "BOX(0 0, 2 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<linestring>        <shape> "LINESTRING(10 21, 12 23, 10 25)"^^<http://www.openlinksw.com/schemas/virtrdf
<polygon1>          <shape> "POLYGON((1 3,2 4,1 5,0 4,1 3))"^^<http://www.openlinksw.com/schemas/virtrdf#
<polygon2>          <shape> "POLYGON((1.5 3.5,2.5 4.5,1.5 5.5,0.5 4.5,1.5 3.5))"^^<http://www.openlinksw.
<polygon3>          <shape> "POLYGON((1 3,2 4,1 5,0 4,1 3),(1 3.5,1.5 4,1 4.5,0.5 4,1 3.5))"^^<http://www
<multipolygon>      <shape> "MULTIPOLYGON(((1 3,2 4,1 5,0 4,1 3)),((1 2,2 1,1 0,0 1,1 2)))"^^<http://www.
<collection>        <shape> "GEOMETRYCOLLECTION(POINT(0 0),MULTILINESTRING((1 5,0 1,4 0,5 4),(1 5,1 0,4 0
', 'http://geo-api-demo/', 'http://geo-api-demo/', 0)

SQL>
SPARQL
SELECT ?s, bif:GeometryType(?o),
       bif:ST_NumGeometries(?o)
FROM <http://geo-api-demo/>
WHERE
 {
   ?s ?p ?o
 }
ORDER BY ASC(str(?s))
s                                 callret-1           callret-2
_______________________________________________________________________________

http://geo-api-demo/box1          BOX2D               1
http://geo-api-demo/collection    GEOMETRYCOLLECTION  3
http://geo-api-demo/linestring    LINESTRING          1
http://geo-api-demo/multipolygon  MULTIPOLYGON        2
http://geo-api-demo/point2d       POINT               1
http://geo-api-demo/point4d       POINTZM             1
http://geo-api-demo/polygon1      POLYGON             1
http://geo-api-demo/polygon2      POLYGON             1
http://geo-api-demo/polygon3      POLYGON             1
```

See Also

`ST_GeometryN()`

`GeometryType()`

ST_NumInteriorRings

ST_NumInteriorRings — returns number of interior rings of the given polygon, NULL if shape is not a polygon.

Synopsis

ST_NumInteriorRings (in *poligon* any) ;

Description

returns number of interior rings of the given polygon, NULL if shape is not a polygon.

Parameters

poligon
: Given polygon.

Return Types

Returns int.

Examples

Example 24.671. Simple Use

```
SQL> DB.DBA.TTLP ('
<point2d>           <shape> "POINT(1 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<point4d>           <shape> "POINTZM(0 1 2 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<box1>              <shape> "BOX(0 0, 2 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<linestring>        <shape> "LINESTRING(10 21, 12 23, 10 25)"^^<http://www.openlinksw.com/schemas/virtrdf
<polygon1>          <shape> "POLYGON((1 3,2 4,1 5,0 4,1 3))"^^<http://www.openlinksw.com/schemas/virtrdf#
<polygon2>          <shape> "POLYGON((1.5 3.5,2.5 4.5,1.5 5.5,0.5 4.5,1.5 3.5))"^^<http://www.openlinksw.
<polygon3>          <shape> "POLYGON((1 3,2 4,1 5,0 4,1 3),(1 3.5,1.5 4,1 4.5,0.5 4,1 3.5))"^^<http://www
<multipolygon>      <shape> "MULTIPOLYGON(((1 3,2 4,1 5,0 4,1 3)),((1 2,2 1,1 0,0 1,1 2)))"^^<http://www.
<collection>        <shape> "GEOMETRYCOLLECTION(POINT(0 0),MULTILINESTRING((1 5,0 1,4 0,5 4),(1 5,1 0,4 0
', 'http://geo-api-demo/', 'http://geo-api-demo/', 0)

SQL> SPARQL
SELECT ?s bif:ST_NumInteriorRings (?o)
FROM <http://geo-api-demo/>
WHERE { ?s ?p ?o }
ORDER BY ASC(str(?s))
s                                callret-1
VARCHAR                          VARCHAR
_______________________________________________________________________________

http://geo-api-demo/box1         NULL
http://geo-api-demo/collection   NULL
http://geo-api-demo/linestring   NULL
http://geo-api-demo/multipolygon 0
http://geo-api-demo/point2d      NULL
http://geo-api-demo/point4d      NULL
http://geo-api-demo/polygon1     0
http://geo-api-demo/polygon2     0
http://geo-api-demo/polygon3     1
```

See Also

`ST_InteriorRingN()`

`ST_ExteriorRing()`

ST_Transform

ST_Transform — Transforms the given shape from its current spatial reference system to one specified by dest_srid.

Synopsis

ST_Transform (in *shape* any ,
in *dest_srid* any ,
in *origin_proj4_string* varchar ,
in *dest_proj4_string* varchar) ;

Description

Transforms the given shape from its current spatial reference system to one specified by dest_srid. Two optional arguments: the SRID of the resulting shape is set to dest_srid but the conversion is done by proj4 using origin_proj4_string for projection of original shape and dest_proj4_string for the result. If orig_proj4_string or dest_proj4_string argument is passed but is NULL instead of string, the projection corresponding to original or destination SRID is used. ST_Transform is provided by a separate plugin named v7proj4, as described below. When the plugin is loaded, functions like ST_Intersects() support pairs of arguments with different SRIDs by converting coords of second argument into the system of the first one, as required by OGC and GeoSPARQL.

Parameters

shape
: Geometry.

dest_srid
: Destination SRID.

origin_proj4_string
: String to be used for the projection.

dest_proj4_string
: String used for the destination result.

Return Types

Returns varchar.

Examples

Example 24.672. Simple Use

```
SQL> SELECT * FROM DB.DBA.SYS_V7PROJ4_SRIDS;
SR_ID              SR_FAMILY          SR_TAG    SR_ORIGIN              SR_IRI    SR_PROJ4_STRING
INTEGER NOT NULL   VARCHAR NOT NULL   VARCHAR   VARCHAR NOT NULL       VARCHAR   VARCHAR NOT NULL
_________________  _________________  _______   ____________________   _______   ____________________

2005               EPSG               2005      /usr/share/proj/epsg   NULL      +ellps=clrk80 +k=0.99950
2249               EPSG               2249      /usr/share/proj/epsg   NULL      +datum=NAD83 +ellps=GRS8
4326               EPSG               4326      /usr/share/proj/epsg   NULL      +datum=WGS84 +ellps=WGS8

3 Rows. -- 1 msec.
SQL> SELECT
st_transform (
    st_geomfromtext ('POLYGON((-16 20.25,
                               -16.1 20.35,
                               -15.9 20.35,
                               -16 20.25))'),
                     1,
                     '+proj=latlong +ellps=clrk66',
```

```
                    '+proj=merc +ellps=clrk66 +lat_ts=33');
unnamed
VARCHAR NOT NULL
_____________________________________________________________________________

SRID=1;POLYGON((-1495284.211473 1920596.789917,
                -1504629.737795 1930501.842961,
                -1485938.685152 1930501.842961,
                -1495284.211473 1920596.789917))

1 Rows. -- 0 msec.

SQL> SELECT ST_AsText(
              ST_Transform(
                ST_GeomFromText('POLYGON((743238 2967416,
                                          743238 2967450,
                                          743265 2967450,
                                          743265.625 2967416,
                                          743238 2967416))',
                                2249),4326)) AS wgs_geom;
wgs_geom
VARCHAR NOT NULL
_____________________________________________________________________________

POLYGON((-71.177685 42.390290,
         -71.177684 42.390383,
         -71.177584 42.390383,
         -71.177583 42.390289,
         -71.177685 42.390290))

1 Rows. -- 1 msec.
SQL>
```

See Also

st_transform_by_custom_projection()

ST_Translate()

ST_TransScale()

ST_X()

ST_Y()

ST_Z()

ST_M()

ST_XMin()

ST_XMax()

ST_YMin()

ST_YMax()

ST_ZMin()

ST_ZMax()

ST_MMin()

ST_MMax()

st_transform_by_custom_projection

st_transform_by_custom_projection — Performs a custom projection of shape, using the specified algorithm and algorithm-specific arguments.

Synopsis

st_transform_by_custom_projection (in *shape* any ,
in *algorithm_id* any) ;

Description

Performs a custom projection of shape, using the specified algorithm and algorithm-specific arguments.

Parameters

shape
: Geometry.

algorithm_id
: Algorithm -- Current version supports only one algorithm, st_transform_by_custom_projection (shape, 'OLAEAPS', long_of_center, lat_of_center) for Oblique Lampert Azimuthal Equal-Area Projection System with the specified center point.

Return Types

Returns varchar.

Examples

Example 24.673. Simple Use

```
select st_transform_by_custom_projection (
  st_ewkt_read ('POLYGON((1 3,2 4,1 5,0 4,1 3),
                         (1 3.5,1.5 4,1 4.5,0.5 4,1 3.5))'),
  'OLAEAPS', 20, 45)
unnamed
VARCHAR NOT NULL
_______________________________________________________________________________

SRID=0;POLYGON((-0.352161 -0.627582,
                -0.332392 -0.617674,
                -0.348956 -0.599730,
                -0.368752 -0.609534,
                -0.352161 -0.627582),
               (-0.351386 -0.620657,
                -0.341498 -0.615717,
                -0.349784 -0.606732,
                -0.359678 -0.611647,
                -0.351386 -0.620657))
```

See Also

`ST_Translate()`

`ST_TransScale()`

`ST_X()`

`ST_Y()`

ST_Z()

ST_M()

ST_XMin()

ST_XMax()

ST_YMin()

ST_YMax()

ST_ZMin()

ST_ZMax()

ST_MMin()

ST_MMax()

ST_Translate

ST_Translate — returns a copy of a shape with all coordinates shifted by the provided dX, dY and dZ.

Synopsis

ST_Translate (in *shape* any ,
in *dX* any ,
in *dY* any ,
in *dZ* any) ;

Description

returns a copy of a shape with all coordinates shifted by the provided dX, dY and dZ.

Parameters

shape
: Geometry.

dX
: X coordinate.

dY
: Y coordinate.

dZ
: Z coordinate.

Examples

Example 24.674. Simple Use

```
SQL> DB.DBA.TTLP ('
<point2d>               <shape> "POINT(1 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<point4d>               <shape> "POINTZM(0 1 2 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry>
<box1>                  <shape> "BOX(0 0, 2 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<linestring>            <shape> "LINESTRING(10 21, 12 23, 10 25)"^^<http://www.openlinksw.com/schemas/vir
<polygon1>              <shape> "POLYGON((1 3,2 4,1 5,0 4,1 3))"^^<http://www.openlinksw.com/schemas/virt
<polygon2>              <shape> "POLYGON((1.5 3.5,2.5 4.5,1.5 5.5,0.5 4.5,1.5 3.5))"^^<http://www.openlin
<polygon3>              <shape> "POLYGON((1 3,2 4,1 5,0 4,1 3),(1 3.5,1.5 4,1 4.5,0.5 4,1 3.5))"^^<http:/
<multipolygon>          <shape> "MULTIPOLYGON(((1 3,2 4,1 5,0 4,1 3)),((1 2,2 1,1 0,0 1,1 2)))"^^<http://
<collection>            <shape> "GEOMETRYCOLLECTION(POINT(0 0),MULTILINESTRING((1 5,0 1,4 0,5 4),(1 5,1 0
', 'http://geo-api-demo/', 'http://geo-api-demo/', 0)

SQL>
SPARQL SELECT ?s, bif:ST_Translate(?o, 0.1, 0.2)
FROM <http://geo-api-demo/>
WHERE
  {
    ?s ?p ?o
  }
ORDER BY ASC(str(?s))
s                               callret-1
VARCHAR                         VARCHAR
_______________________________________________________________________________

http://geo-api-demo/box1        BOX2D(0.10 0.20,2.10 3.20)
http://geo-api-demo/collection  GEOMETRYCOLLECTION(POINT(0.10 0.20),MULTILINESTRING((1.10 5.20,0.10 1.2
http://geo-api-demo/linestring  LINESTRING(10.10 21.20,12.10 23.20,10.10 25.20)
http://geo-api-demo/multipolygon MULTIPOLYGON(((1.10 3.20,2.10 4.20,1.10 5.20,0.10 4.20,1.10 3.20)),((1.
http://geo-api-demo/point2d     POINT(1.1 3.2)
http://geo-api-demo/point4d     POINTZM(0.10 1.20 2.0 3.0)
```

```
http://geo-api-demo/polygon1      POLYGON((1.10 3.20,2.10 4.20,1.10 5.20,0.10 4.20,1.10 3.20))
http://geo-api-demo/polygon2      POLYGON((1.60 3.70,2.60 4.70,1.60 5.70,0.60 4.70,1.60 3.70))
http://geo-api-demo/polygon3      POLYGON((1.10 3.20,2.10 4.20,1.10 5.20,0.10 4.20,1.10 3.20),(1.10 3.70,
```

See Also

ST_TransScale()

ST_X()

ST_Y()

ST_Z()

ST_M()

ST_XMin()

ST_XMax()

ST_YMin()

ST_YMax()

ST_ZMin()

ST_ZMax()

ST_MMin()

ST_MMax()

ST_TransScale

ST_TransScale — returns a copy of a shape with all coordinates shifted by the provided dX, dY and then multiplied by Xfactor and Yfactor.

Synopsis

ST_TransScale (in *shape* any ,
in *dX* any ,
in *dY* any ,
in *Xfactor* any ,
in *Yfactor* any) ;

Description

returns a copy of a shape with all coordinates shifted by the provided dX, dY and then multiplied by Xfactor and Yfactor. Different values for Xfactor and Yfactor will result in distorted arcs.

Parameters

shape
: Geometry.

dX
: X coordinate.

dY
: Y coordinate.

Xfactor
: X value to multiple by.

Yfactor
: Y value to multiple by.

Return Types

Returns any.

Examples

Example 24.675. Simple Use

```
SQL> DB.DBA.TTLP ('
<point2d>               <shape> "POINT(1 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<point4d>               <shape> "POINTZM(0 1 2 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry>
<box1>                  <shape> "BOX(0 0, 2 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<linestring>            <shape> "LINESTRING(10 21, 12 23, 10 25)"^^<http://www.openlinksw.com/schemas/vir
<polygon1>              <shape> "POLYGON((1 3,2 4,1 5,0 4,1 3))"^^<http://www.openlinksw.com/schemas/virt
<polygon2>              <shape> "POLYGON((1.5 3.5,2.5 4.5,1.5 5.5,0.5 4.5,1.5 3.5))"^^<http://www.openlin
<polygon3>              <shape> "POLYGON((1 3,2 4,1 5,0 4,1 3),(1 3.5,1.5 4,1 4.5,0.5 4,1 3.5))"^^<http:/
<multipolygon>          <shape> "MULTIPOLYGON(((1 3,2 4,1 5,0 4,1 3)),((1 2,2 1,1 0,0 1,1 2)))"^^<http://
<collection>            <shape> "GEOMETRYCOLLECTION(POINT(0 0),MULTILINESTRING((1 5,0 1,4 0,5 4),(1 5,1 0
', 'http://geo-api-demo/', 'http://geo-api-demo/', 0)

SQL> SPARQL
SELECT ?s, bif:ST_TransScale(?o, 0.1, 0.2, 0.1, 0.1)
FROM <http://geo-api-demo/>
WHERE
  {
```

```
    ?s ?p ?o
  }
ORDER BY ASC(str(?s))
s                                                                                callret-1
VARCHAR                                                                          VARCHAR
_______________________________________________________________________________

http://geo-api-demo/box1          BOX2D(0.010 0.020,0.210 0.320)
http://geo-api-demo/collection    GEOMETRYCOLLECTION(POINT(0.010 0.020),MULTILINESTRING((0.110 0.520,0.01
http://geo-api-demo/linestring    LINESTRING(1.010 2.120,1.210 2.320,1.010 2.520)
http://geo-api-demo/multipolygon  MULTIPOLYGON(((0.110 0.320,0.010 0.420,0.110 0.520,0.210 0.420,0.110 0.
http://geo-api-demo/point2d       POINT(0.11 0.32)
http://geo-api-demo/point4d       POINTZM(0.010 0.120 2.0 3.0)
http://geo-api-demo/polygon1      POLYGON((0.110 0.320,0.010 0.420,0.110 0.520,0.210 0.420,0.110 0.320))
http://geo-api-demo/polygon2      POLYGON((0.160 0.370,0.060 0.470,0.160 0.570,0.260 0.470,0.160 0.370))
http://geo-api-demo/polygon3      POLYGON((0.110 0.320,0.010 0.420,0.110 0.520,0.210 0.420,0.110 0.320),(
```

See Also

ST_X()

ST_Y()

ST_Z()

ST_M()

ST_XMin()

ST_XMax()

ST_YMin()

ST_YMax()

ST_ZMin()

ST_ZMax()

ST_MMin()

ST_MMax()

ST_XMax

ST_XMax — returns maximum x boundaries of a bounding box around a shape.

Synopsis

ST_XMax (in *shape* any) ;

Description

returns boundaries of a bounding box around a shape. Bounding boxes around arcs are calculated in assumption that no one arc is longer than a half of full circle.

Parameters

shape
: A bounding box.

Return Types

Returns double precision.

Examples

Example 24.676. Simple Use

```
DB.DBA.TTLP ('
<point2d>           <shape> "POINT(1 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<point4d>           <shape> "POINTZM(0 1 2 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<box1>              <shape> "BOX(0 0, 2 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<linestring>        <shape> "LINESTRING(10 21, 12 23, 10 25)"^^<http://www.openlinksw.com/schemas/virtrdf
<polygon1>          <shape> "POLYGON((1 3,2 4,1 5,0 4,1 3))"^^<http://www.openlinksw.com/schemas/virtrdf#
<polygon2>          <shape> "POLYGON((1.5 3.5,2.5 4.5,1.5 5.5,0.5 4.5,1.5 3.5))"^^<http://www.openlinksw.
<polygon3>          <shape> "POLYGON((1 3,2 4,1 5,0 4,1 3),(1 3.5,1.5 4,1 4.5,0.5 4,1 3.5))"^^<http://www
<multipolygon>      <shape> "MULTIPOLYGON(((1 3,2 4,1 5,0 4,1 3)),((1 2,2 1,1 0,0 1,1 2)))"^^<http://www.
<collection>        <shape> "GEOMETRYCOLLECTION(POINT(0 0),MULTILINESTRING((1 5,0 1,4 0,5 4),(1 5,1 0,4 0
', 'http://geo-api-demo/', 'http://geo-api-demo/', 0)

--
SQL>
SPARQL
SELECT ?s,
       bif:st_xmax(?o) as ?xmax
FROM <http://geo-api-demo/>
WHERE { ?s ?p ?o };
s                                   xmax
ANY                                 DOUBLE PRECISION
_____________________________________________________________________________
 http://geo-api-demo/box1                 2
 http://geo-api-demo/collection     5
 http://geo-api-demo/linestring     12
 http://geo-api-demo/multipolygon         2
 http://geo-api-demo/point2d          1
 http://geo-api-demo/point4d          0
 http://geo-api-demo/polygon1         2
 http://geo-api-demo/polygon2         2.5
```

See Also

`ST_X()`

`ST_Y()`

ST_Z()

ST_M()

ST_XMin()

ST_YMin()

ST_YMax()

ST_ZMin()

ST_ZMax()

ST_MMin()

ST_MMax()

ST_XMin

ST_XMin — returns minimum x boundaries of a bounding box around a shape.

Synopsis

ST_XMin (in *shape* any) ;

Description

returns boundaries of a bounding box around a shape. Bounding boxes around arcs are calculated in assumption that no one arc is longer than a half of full circle.

Parameters

shape
: A bounding box.

Return Types

Returns double precision.

Examples

Example 24.677. Simple Use

```
DB.DBA.TTLP ('
<point2d>           <shape> "POINT(1 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<point4d>           <shape> "POINTZM(0 1 2 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<box1>              <shape> "BOX(0 0, 2 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<linestring>        <shape> "LINESTRING(10 21, 12 23, 10 25)"^^<http://www.openlinksw.com/schemas/virtrdf
<polygon1>          <shape> "POLYGON((1 3,2 4,1 5,0 4,1 3))"^^<http://www.openlinksw.com/schemas/virtrdf#
<polygon2>          <shape> "POLYGON((1.5 3.5,2.5 4.5,1.5 5.5,0.5 4.5,1.5 3.5))"^^<http://www.openlinksw.
<polygon3>          <shape> "POLYGON((1 3,2 4,1 5,0 4,1 3),(1 3.5,1.5 4,1 4.5,0.5 4,1 3.5))"^^<http://www
<multipolygon>      <shape> "MULTIPOLYGON(((1 3,2 4,1 5,0 4,1 3)),((1 2,2 1,1 0,0 1,1 2)))"^^<http://www.
<collection>        <shape> "GEOMETRYCOLLECTION(POINT(0 0),MULTILINESTRING((1 5,0 1,4 0,5 4),(1 5,1 0,4 0
', 'http://geo-api-demo/', 'http://geo-api-demo/', 0)

--
SPARQL
SELECT ?s,
       bif:ST_XMin(?o) as ?xmin
FROM <http://geo-api-demo/>
WHERE { ?s ?p ?o }
s                                 xmin
_______________________________________________________________________________

http://geo-api-demo/linestring    10
http://geo-api-demo/polygon2      0.5
http://geo-api-demo/collection    0
http://geo-api-demo/multipolygon  0
http://geo-api-demo/box1          0
http://geo-api-demo/point4d       0
http://geo-api-demo/point2d       1
http://geo-api-demo/polygon1      0
```

See Also

ST_X()

ST_Y()

ST_Z()

ST_M()

ST_XMax()

ST_YMin()

ST_YMax()

ST_ZMin()

ST_ZMax()

ST_MMin()

ST_MMax()

ST_YMax

ST_YMax — returns maximum y boundaries of a bounding box around a shape.

Synopsis

ST_YMax (in *shape* any) ;

Description

returns boundaries of a bounding box around a shape. Bounding boxes around arcs are calculated in assumption that no one arc is longer than a half of full circle.

Parameters

shape
: A bounding box.

Return Types

Returns double precision.

Examples

Example 24.678. Simple Use

```
DB.DBA.TTLP ('
<point2d>           <shape> "POINT(1 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<point4d>           <shape> "POINTZM(0 1 2 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<box1>              <shape> "BOX(0 0, 2 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<linestring>        <shape> "LINESTRING(10 21, 12 23, 10 25)"^^<http://www.openlinksw.com/schemas/virtrdf
<polygon1>          <shape> "POLYGON((1 3,2 4,1 5,0 4,1 3))"^^<http://www.openlinksw.com/schemas/virtrdf#
<polygon2>          <shape> "POLYGON((1.5 3.5,2.5 4.5,1.5 5.5,0.5 4.5,1.5 3.5))"^^<http://www.openlinksw.
<polygon3>          <shape> "POLYGON((1 3,2 4,1 5,0 4,1 3),(1 3.5,1.5 4,1 4.5,0.5 4,1 3.5))"^^<http://www
<multipolygon>      <shape> "MULTIPOLYGON(((1 3,2 4,1 5,0 4,1 3)),((1 2,2 1,1 0,0 1,1 2)))"^^<http://www.
<collection>        <shape> "GEOMETRYCOLLECTION(POINT(0 0),MULTILINESTRING((1 5,0 1,4 0,5 4),(1 5,1 0,4 0
', 'http://geo-api-demo/', 'http://geo-api-demo/', 0)

--
SQL>
SPARQL
SELECT ?s,
       bif:st_ymax(?o) as ?ymax
FROM <http://geo-api-demo/>
WHERE { ?s ?p ?o };
s                                 ymax
ANY                               DOUBLE PRECISION
_______________________________________________________________________________

 http://geo-api-demo/box1                3
 http://geo-api-demo/collection     5
 http://geo-api-demo/linestring     25
 http://geo-api-demo/multipolygon          5
 http://geo-api-demo/point2d          3
 http://geo-api-demo/point4d          1
 http://geo-api-demo/polygon1         5
 http://geo-api-demo/polygon2         5.5
```

See Also

ST_X()

ST_Y()

ST_Z()

ST_M()

ST_XMin()

ST_XMax()

ST_YMin()

ST_ZMin()

ST_ZMax()

ST_MMin()

ST_MMax()

ST_YMin

ST_YMin — returns minimum y boundaries of a bounding box around a shape.

Synopsis

ST_YMin (in *shape* any) ;

Description

returns boundaries of a bounding box around a shape. Bounding boxes around arcs are calculated in assumption that no one arc is longer than a half of full circle.

Parameters

shape
: A bounding box.

Return Types

Returns double precision.

Examples

Example 24.679. Simple Use

```
DB.DBA.TTLP ('
<point2d>           <shape> "POINT(1 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<point4d>           <shape> "POINTZM(0 1 2 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<box1>              <shape> "BOX(0 0, 2 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<linestring>        <shape> "LINESTRING(10 21, 12 23, 10 25)"^^<http://www.openlinksw.com/schemas/virtrdf
<polygon1>          <shape> "POLYGON((1 3,2 4,1 5,0 4,1 3))"^^<http://www.openlinksw.com/schemas/virtrdf#
<polygon2>          <shape> "POLYGON((1.5 3.5,2.5 4.5,1.5 5.5,0.5 4.5,1.5 3.5))"^^<http://www.openlinksw.
<polygon3>          <shape> "POLYGON((1 3,2 4,1 5,0 4,1 3),(1 3.5,1.5 4,1 4.5,0.5 4,1 3.5))"^^<http://www
<multipolygon>      <shape> "MULTIPOLYGON(((1 3,2 4,1 5,0 4,1 3)),((1 2,2 1,1 0,0 1,1 2)))"^^<http://www.
<collection>        <shape> "GEOMETRYCOLLECTION(POINT(0 0),MULTILINESTRING((1 5,0 1,4 0,5 4),(1 5,1 0,4 0
', 'http://geo-api-demo/', 'http://geo-api-demo/', 0)

--
SQL>
SPARQL
SELECT ?s,
       bif:st_ymin(?o) as ?ymin
FROM <http://geo-api-demo/>
WHERE { ?s ?p ?o };
s                                   ymin
ANY                                 DOUBLE PRECISION
_____________________________________________________________________________
 http://geo-api-demo/box1                 0
 http://geo-api-demo/collection     0
 http://geo-api-demo/linestring     21
 http://geo-api-demo/multipolygon         0
 http://geo-api-demo/point2d          3
 http://geo-api-demo/point4d          1
 http://geo-api-demo/polygon1         3
 http://geo-api-demo/polygon2         3.5
```

See Also

`ST_X()`

`ST_Y()`

ST_Z()

ST_M()

ST_XMin()

ST_XMax()

ST_YMax()

ST_ZMin()

ST_ZMax()

ST_MMin()

ST_MMax()

ST_Z

ST_Z — Retrieves the z coordinate of a geometry.

Synopsis

ST_Z (in *g* any);

Description

Retrieves the x coordinate of a geometry.

Parameters

g
: Geometry

Return Types

Returns double precision.

Examples

Example 24.680. Simple Use

```
SELECT st_z( st_point (10, 20, 30, 40));
DOUBLE PRECISION
 30
```

See Also

ST_X()

ST_Y()

ST_M()

ST_XMin()

ST_XMax()

ST_YMin()

ST_YMax()

ST_ZMin()

ST_ZMax()

ST_MMin()

ST_MMax()

ST_ZMax

ST_ZMax — returns maximum z boundaries of a bounding box around a shape.

Synopsis

ST_ZMax (in *shape* any) ;

Description

returns boundaries of a bounding box around a shape. Bounding boxes around arcs are calculated in assumption that no one arc is longer than a half of full circle.

Parameters

shape
: A bounding box.

Return Types

Returns double precision.

Examples

Example 24.681. Simple Use

```
DB.DBA.TTLP ('
<point2d>           <shape> "POINT(1 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<point4d>           <shape> "POINTZM(0 1 2 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<box1>              <shape> "BOX(0 0, 2 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<linestring>        <shape> "LINESTRING(10 21, 12 23, 10 25)"^^<http://www.openlinksw.com/schemas/virtrdf
<polygon1>          <shape> "POLYGON((1 3,2 4,1 5,0 4,1 3))"^^<http://www.openlinksw.com/schemas/virtrdf#
<polygon2>          <shape> "POLYGON((1.5 3.5,2.5 4.5,1.5 5.5,0.5 4.5,1.5 3.5))"^^<http://www.openlinksw.
<polygon3>          <shape> "POLYGON((1 3,2 4,1 5,0 4,1 3),(1 3.5,1.5 4,1 4.5,0.5 4,1 3.5))"^^<http://www
<multipolygon>      <shape> "MULTIPOLYGON(((1 3,2 4,1 5,0 4,1 3)),((1 2,2 1,1 0,0 1,1 2)))"^^<http://www.
<collection>        <shape> "GEOMETRYCOLLECTION(POINT(0 0),MULTILINESTRING((1 5,0 1,4 0,5 4),(1 5,1 0,4 0
', 'http://geo-api-demo/', 'http://geo-api-demo/', 0)

--
SQL>
SPARQL
SELECT ?s,
       bif:st_zmax(?o) as ?zmax
FROM <http://geo-api-demo/>
WHERE { ?s ?p ?o };
s                                   zmax
ANY                                 DOUBLE PRECISION
_______________________________________________________________________________
 http://geo-api-demo/box1                 1e+38
 http://geo-api-demo/collection     1e+38
 http://geo-api-demo/linestring     1e+38
 http://geo-api-demo/multipolygon         1e+38
 http://geo-api-demo/point2d          1e+38
 http://geo-api-demo/point4d          2
 http://geo-api-demo/polygon1         1e+38
 http://geo-api-demo/polygon2         1e+38
```

See Also

ST_X()

ST_Y()

ST_Z()

ST_M()

ST_XMin()

ST_XMax()

ST_YMin()

ST_YMax()

ST_ZMin()

ST_MMin()

ST_MMax()

ST_Zmflag

ST_Zmflag — returns bits indicating presence of Z and/or M coordinates.

Synopsis

ST_Zmflag (in *g* any);

Description

Returns bits indicating presence of Z and/or M coordinates. 0 means that the shape has only X and Y, 1 means that M is also in use (but not Z), 2 means Z but ont M, 3 is for both M and Z.

Parameters

g
: Geometry

Return Types

Returns integer.

Examples

Example 24.682. Simple Use

```
SELECT ST_Zmflag( st_point (10, 20, 30, 40));
unnamed
INTEGER
 3
```

See Also

`ST_X()`

`ST_Y()`

`ST_Z()`

`ST_M()`

`ST_XMin()`

`ST_XMax()`

`ST_YMin()`

`ST_YMax()`

`ST_ZMin()`

`ST_ZMax()`

`ST_MMin()`

`ST_MMax()`

ST_ZMin

ST_ZMin — returns minimum z boundaries of a bounding box around a shape.

Synopsis

ST_ZMin (in *shape* any) ;

Description

returns boundaries of a bounding box around a shape. Bounding boxes around arcs are calculated in ssumption that no one arc is longer than a half of full circle.

Parameters

shape
: A bounding box.

Return Types

Returns double precision.

Examples

Example 24.683. Simple Use

```
DB.DBA.TTLP ('
<point2d>           <shape> "POINT(1 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<point4d>           <shape> "POINTZM(0 1 2 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<box1>              <shape> "BOX(0 0, 2 3)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry> .
<linestring>        <shape> "LINESTRING(10 21, 12 23, 10 25)"^^<http://www.openlinksw.com/schemas/virtrdf
<polygon1>          <shape> "POLYGON((1 3,2 4,1 5,0 4,1 3))"^^<http://www.openlinksw.com/schemas/virtrdf#
<polygon2>          <shape> "POLYGON((1.5 3.5,2.5 4.5,1.5 5.5,0.5 4.5,1.5 3.5))"^^<http://www.openlinksw.
<polygon3>          <shape> "POLYGON((1 3,2 4,1 5,0 4,1 3),(1 3.5,1.5 4,1 4.5,0.5 4,1 3.5))"^^<http://www
<multipolygon>      <shape> "MULTIPOLYGON(((1 3,2 4,1 5,0 4,1 3)),((1 2,2 1,1 0,0 1,1 2)))"^^<http://www.
<collection>        <shape> "GEOMETRYCOLLECTION(POINT(0 0),MULTILINESTRING((1 5,0 1,4 0,5 4),(1 5,1 0,4 0
', 'http://geo-api-demo/', 'http://geo-api-demo/', 0)

--
SQL>
SPARQL
SELECT ?s,
       bif:st_zmin(?o) as ?zmin
FROM <http://geo-api-demo/>
WHERE { ?s ?p ?o };
s                                   zmin
ANY                                 DOUBLE PRECISION
_____________________________________________________________________________
 http://geo-api-demo/box1                 1e+38
 http://geo-api-demo/collection     1e+38
 http://geo-api-demo/linestring     1e+38
 http://geo-api-demo/multipolygon         1e+38
 http://geo-api-demo/point2d          1e+38
 http://geo-api-demo/point4d          2
 http://geo-api-demo/polygon1         1e+38
 http://geo-api-demo/polygon2         1e+38
```

See Also

ST_X()

ST_Y()

ST_Z()

ST_M()

ST_XMin()

ST_XMax()

ST_YMin()

ST_YMax()

ST_ZMax()

ST_MMin()

ST_MMax()